Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Electrical, Electronics and Communications Engineering
(36^{th}cycle)

Machine Learning for Perception and Autonomous Navigation of Service Mobile Robots

Mauro Martini
******

Supervisor:
Prof. Marcello Chiaberge

Politecnico di Torino
2024

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Mauro Martini
Torino, 2024

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Abstract

Autonomous service robots are expected to revolutionize multiple industries and social contexts in the following decades. Service robots perform helpful tasks for humans or equipment, excluding industrial automation applications. Potential contexts that can benefit from adopting service robots include precision agriculture, search and rescue, indoor cleaning, social assistance, inspection, and exploration of hazardous environments. In the last years, research has significantly investigated mobile robots' robust perception and autonomous navigation capabilities to address complex service tasks in dynamic environments. In particular, perception refers to the ability of a robot to acquire, process, and understand information from its surroundings, such as images, sounds, and tactile sensory data. Autonomous navigation refers to the ability of a robot to plan and execute its motion, achieving its target goal without collisions or failures. In this context, the most recent data-driven approaches can empower service mobile robots with these required capabilities. Machine learning and Deep Learning are successful branches of Artificial Intelligence (AI) that have been widely applied to various complex tasks such as Pattern Recognition, Computer Vision, and Natural Language Processing, outperforming classical methods. Robotics represents another challenging application field for Machine Learning, as it involves complex and dynamic environments and high-dimensional and multimodal sensory inputs to solve a wide range of diverse and interactive tasks. This thesis aims to study novel methodologies to efficiently integrate AI in service robotics perception and autonomous navigation tasks. Precision agriculture and social indoor assistance are the applicative contexts for all the methods discussed in this dissertation. A complete Deep Learning pipeline for robot navigation in row-based crops is presented, combining different low-cost visual controller solutions with a way-point generator for global path construction. Various methodologies have been developed for indoor social assistance tasks, from online odometry correction to omnidirectional person following and monitoring. Consistent simulation and synthetic data adoption have

enabled fast data collection to train the models, analyzing and mitigating the deriving sim-to-real gap with new techniques. Generalization is a fundamental aspect of robust AI-based perception in the real world, and a great focus has been devoted to investigating it from different perspectives. Domain Adaptation and Domain Generalization methodologies have been applied to image classification and segmentation tasks in the agricultural field, as well as a general study of backbone generalization properties. Extensive experiments and results obtained are presented for each presented methodology, together with a theoretical description of the methods and a complete review of related works. Moreover, a great endeavor has been spent to study and optimize the inference time of Deep Learning models on constrained resource hardware, such as the onboard computational units of mobile robots. Different approaches have been considered to allow for real-time prediction performances required by the robotic tasks. Overall, this dissertation aims to advance the efficient integration of AI and mobile robots, paving the way for further improvements in this rapidly evolving research field.

Contents

Introduction

Service robotics

Service robots, defined as *"semi-automatic or fully automatic machines that perform tasks beneficial to humans"*, are spreading across various domains, offering innovative solutions to everyday challenges. Nowadays, the field of service robotics stands at the forefront of innovation and represents a dynamic sector that intersects advanced technology from many disciplines and conveys them to practical applications. The market has witnessed substantial growth in the last years, with projections indicating an increase from USD 19.08 billion in 2023 to USD 62.35 billion by 2030, reflecting a compound annual growth rate (CAGR) of 18.4% [1, 2]. The recent expansion of the market sector is also fueled by the integration of Artificial Intelligence (AI) and big data analytics, enhancing the capabilities of robots to perform a myriad of tasks ranging from domestic to industrial applications. Among the most promising fields we find industrial inspection [3], search and rescue [4], cultural [5] and environmental [6] heritage protection, planetary exploration [7], and logistics [8].

In this thesis, we seek for a genuine integration between Deep Learning and service robotics applications, focusing on the specific contexts of precision agriculture and indoor well-being. In precision agriculture, service robots are revolutionizing traditional farming practices, offering solutions that boost efficiency and sustainability [9]. The integration of advanced sensors and data analytics enables these robots to perform tasks such as crop monitoring, harvesting, soil analysis, and targeted pesticide application with unprecedented precision. This not only optimizes resource management but also opens a new era where farmers make decisions based on data and environmental awareness.

(a) (b)

Fig. 1 Service robotics solutions: Marvin prototype (**a**) while monitoring a patient in a domestic environment, and Husky rover (**b**) navigating through a vineyard.

Indoor well-being applications of service robots are also rapidly evolving, particularly in healthcare and domestic settings [10]. Autonomous robots can assist the elderly or isolated individuals with disabilities and enhance patient care through continuous monitoring and interaction inside hospitals. These applications underscore the potential of service robots to improve the everyday quality of life and support independent living. Indoor robots can also support cleaning and transport activities in vast environments like airports, hospitals, and offices.

However, the deployment of service robots is challenging, particularly in the realm of autonomous navigation [11]. Navigating diverse and dynamic environments requires sophisticated algorithms capable of real-time decision-making and obstacle avoidance. Integrating Machine Learning (ML) is pivotal in addressing these challenges, enabling robots to learn from experience, adapt to new scenarios, and execute tasks with greater autonomy and flexibility. Hence, the advancement of AI and ML for service robotics paves the way for the evolution of collaborative robots that enhance human capabilities in our everyday lives. As research advances, the potential applications and benefits of service robots will probably expand, stepping towards more innovative and more reliable technology.

The research conducted in the thesis spans from the improvement of the underlying technologies and algorithms for robot localization, perception, and navigation to almost complete solutions. For example, a prototype for indoor social assistance and smart home management is presented, grounding all the intelligence required for person detection, interaction, and motion control in a unique platform. Then,

an algorithmic pipeline for navigating row-based crops like vineyards and orchards is also presented. Figure 1 shows the Marvin prototype for indoor assistance (left) while checking the user's status on the bed and the Husky rover in a vineyard row.

Contributions

This doctoral thesis focuses on the investigation of Machine Learning solutions to enhance autonomous robots in service applications. The project focuses on different problems and applications of AI in mobile robots, from perception in real-world contexts to control and planning tasks. A particular focus is devoted to mobile robot navigation aspects. Besides, relevant challenges related to adopting Deep Learning methods in real-world tasks are considered and tackled, analyzing both generalization and optimization of Artificial Neural Network execution. The overall contributions of this dissertation can be therefore summarized in the following aspects:

- A study on autonomous navigation and control of indoor social-assistive robots is discussed in Part II. An advanced method mixing classic local planning and Deep Reinforcement Learning is first presented to tackle adaptive social navigation [12]. A position-tracking problem with wheeled robots is then analyzed, and an online learning method to mitigate it is introduced [13]. Finally, an omnidirectional platform and human-aware control are presented as a practical application case study [14–16].

- A Deep Learning pipeline for row-based crops autonomous navigation is presented in Part III. The complete solution extensively explores Deep Learning solutions to enable a low-cost, flexible navigation approach in vineyards and orchards, tackling practical problems such as poor localization signals and expensive sensors [17]. Deep semantic segmentation and Deep Reinforcement Learning are used to generate position-agnostic commands inside the rows of crops [18–21]. A global waypoint generator is also presented [22].

- Generalization and optimization of Deep Learning models have been further investigated in Part IV for their usage in real-world problems. The ability of Deep Neural Networks to adapt to unseen conditions have been deeply explored firstly considering the role of the backbone in standard multi-domain datasets for image classification [23]. Then, a great focus has been devoted to Computer Vision tasks in the agricultural context, such as crop classification

and segmentation, proposing methods to obtain reliable performance in real-world settings [24, 25]. Moreover, novel inference optimization methods have been explored for a Single Image Super Resolution task on low-power edge devices [26].

This dissertation aggregates the research I conducted during my PhD at PIC4SeR (Politecnico di Torino Interdepartmental Center for Service Robotics). The thesis gathers the studies and the projects I have carried out as principal investigator [24, 19, 18, 27, 12], together with the ones I conducted in a joint effort with other PhD students and researchers at PIC4SeR. Most of the works presented have been published in both peer-reviewed journals [24, 14, 15, 26, 23] and conferences [27, 18, 19, 12, 13, 22, 25, 16]. Nonetheless, some side projects have not been included in the thesis [28, 29]. Many research studies presented in the chapters still leave open challenges and unsolved problems to be investigated in future works. Certainly, all of them contributed significantly to my doctoral path.

Thesis organization

The thesis is structured in four main parts, with the aim of aggregating the diverse contents according to the application context or research problem.

Part I contains all the theoretical foundations of the methodologies adopted in the thesis. Hence, Chapter 1 opens the thesis introducing Machine Learning and Deep Learning principal concepts and algorithms, spanning from the definition of Artificial Neural Networks to the most recent architectures and optimization practices. Chapter 2 continues framing the Deep Reinforcement Learning paradigm and describing the main algorithms used in the thesis. Chapter 3 instead briefly summarizes the architecture of a mobile robot autonomous navigation system and the most popular localization and local planning algorithms.

Part II contains the studies that enhances indoor social context. Chapter 4 discuss the findings of my latest study on social navigation: how to efficiently mix up standard controllers and reinforcement learning for crowded environments. Then, Chapter 5 considers a Machine Learning approach to mitigate the position tracking error accumulated by wheels encoders, the most popular sensor for indoor platforms positioning. Finally, Chapter 6 concludes this part of the thesis describing the concepts behind the realization of Marvin: a prototype for domestic assistance .

Part III aggregates the diverse studies conducted to realize a navigation pipeline for vineyards and orchards. Chapter 7 introduces and describe the overall pipeline, and then focuses on local controller techniques to traverse the rows of the crop without a reliable localization of the robot, using only a camera. The approaches investigated consist in Semantic Segmentation and Deep Reinforcement Learning. Experimental results in simulation and on the field are reported. As complementary module of the navigation pipeline, Chapter 8 describes the waypoints generator conceived to estimate and cluster each starting/ending point of the crop field.

Part IV presents insights and deeper analyses on adopting Deep Learning solutions for practical perception tasks, sometimes isolated from a specific robotics application. The keywords of this section of the thesis are *generalization* and *optimization*: the two main problems faced to deploy AI algorithms in real-world settings. Chapter 9, 10 and 11 are devoted to define the Domain Generalization and Adaptation problems and propose meaningful analyses and solutions to obtain robust models to out-of-distribution data. Chapter 9 investigates the properties of most recent backbone architectures in a Domain Generalization framework for image classification, introducing a novel rigorous benchmark to compare models and algorithms on popular datasets. In similar problem settings, Chapter 10 proposes instead a novel method to boost the generalization properties of a lightweight model for the crop segmentation task. This findings aims to mitigate the challenges encountered during the experimental sessions on the field with the robot (Chapter 7). A multi-crop robust model is the learning goal of the study, which also leverages a new realistic synthetic crops dataset (AgriSeg). Chapter 11 applies a different method for the restricted problem of Domain Adaptation, this time considering a separate task in precision agriculture: land crop classification from satellite multi-spectral imagery. In this case, the multi-temporal satellite data perfectly fits with the promising Vision Transformer architecture analysed in Chapter 9. To conclude, an isolated study on Deep Learning model optimization for real-time execution at the edge is proposed in Chapter 12, targeting the Single-Image Super-Resolution task, useful for many robotics applications including image efficient transmission.

Part I

Fundamentals

Chapter 1

Basics of Machine Learning

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) that provides systems the ability to automatically learn patterns and extract information from data without being explicitly programmed. In this chapter a thorough introduction to ML foundational concepts is provided to favour an easier understanding of Part II, III and IV. Firstly, the rich landscape of ML research can be framed based on the possible categories of ML algorithms. Different paradigms of ML have been developed so far:

- *supervised learning*: the model is trained on a labeled dataset, i.e. each sample in the training dataset is paired with the correct output value for the task of interest. The model learns to map inputs to outputs and can be used to predict the output for unseen inputs after successful training.
- *unsupervised learning*: unlike supervised learning, the model is trained on an unlabeled dataset. The model learns the inherent structure of the data without any guidance, only leveraging inner structures or patterns in the data.
- *semi-supervised learning*: refer to those cases when labeled data are only partially available or human supervision is injected in the learning process also without a fully labeled dataset.
- *self-supervised learning*: this paradigm is usually adopted when the goal is learning the structure, a vectorial representation of the data, regardless of a specific task. In this case, supervision is directly taken from the data, trying to

teach the models how to recompose the original structure or extract meaningful knowledge from the data without labels.

- *reinforcement learning*: reinforcement learning is a special paradigm in which an agent learns a decision-making policy by acting in an environment, with the aim of maximizing a reward signal.

Besides, a taxonomy of ML based on the specific task to solve can also be built. ML can now be used to solve a wide range of tasks. Here the most relevant ones are reported:

- *classification*: the model predicts discrete variables as output, identifying a set of categories in the input data.
- *regression*: in this task the model predicts instead a continuous output variable that can assume any specific meaning according to the data and application at hand.
- *clustering*: it is an unsupervised learning task where the model groups similar instances together.

In this chapter, ML concepts are exposed starting from the basic working principles of Artificial Neural Networks in Section 1.1 to the most recent architectures. A short overview of optimization techniques and typical hardware devices for networks execution at the edge is then proposed in Section 1.2.

1.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) have become an essential part of the current technological landscape, driving advancements in many areas of computer science: computer vision, natural language processing, autonomous vehicles and cybersecurity [30–35]. Their ability to learn complex patterns from data makes them incredibly powerful tools for tackling intricate tasks and problems in a wide range of contexts. ANNs are complex models originally inspired by the biological neurons that constitute animal brains. In the last years, after decades of research, the fast advancement of computational devices and the collection of huge datasets [36, 37] have favoured the rise of Deep Learning [38], a further subfield of ML characterized by deep networks architectures with multiple layers. In this section a brief introduction to ANNs elements is provided, explaining the principles behind ANN basic models

and training, finally illustrating successful architectures such Convolutional Neural Networks [39] and Transformers [40].

1.1.1 The artificial neuron

ANNs are parametric models used to approximate a generic function $y = f(x)$, being x the input variable and y the output. The atomic element of ANNs is the artificial neuron. A first model of an artificial neuron was studied by McCulloch and Pitts in 1943 [41]. Clearly, it was inspired by a biological neuron, trying to model the passage of electrical signal through neural cells, axons and synapses.

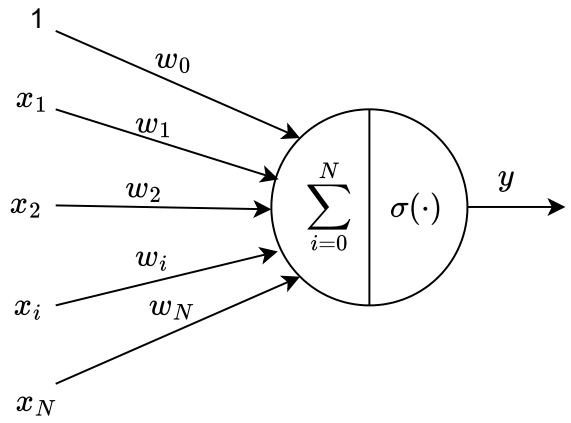


Fig. 1.1 Artificial neuron schematic model.

A simple artificial neuron model is shown in Fig. 1.1. The neuron receives an array of input signals $\mathbf{x}$ and process them through a weighted sum and a non-linear activation function $\sigma(\cdot)$ to obtain the output signal $\mathbf{y}$. The complete mathematical operation it performs consists in:

$$y = \sigma(\mathbf{w}\mathbf{x} + b) = \sigma(\sum_{i=0}^{N} w_i x_i + b) = \sigma(z) \tag{1.1}$$

Where $\mathbf{w}$ is the vector of weights of the neuron and b the bias term of the neuron. Weights and biases are the parameters of an ANNs, that regulate the importance of each input component to obtain the desired output. In general, an artificial neuron presents $N+1$ parameters, with N being the dimension of the input signal. $\sigma(\cdot)$ is a generic activation function used to introduce non-linearity in the model, similarly to what happen in biologic neurons.

1.1.2 Multi-layer perceptron

The passage from the artificial neuron to ANNs is quite straightforward. It is sufficient to stack multiple neurons in layers and connect layers among them. In the Fully Connected (FC) or Dense layer architecture each neuron receives in input all the output signals of the neurons in the previous layer. This layered architecture composes the *Multi-Layer Perceptron* (MLP), originally proposed in the study [42]. In a MLP, the signal flows from the left (input layer) to the right (output layer). Fig. 1.2 shows a schematic example of a MLP with two intermediate or *hidden* dense layers H_1 and H_2 that process the input vector $\mathbf{x}$ of dimension M to obtain the output $\mathbf{y}$ of size P. This architecture is generally known as Feed-forward Neural Network (FNN), the most basic version of a ANN. The number of hidden layers and neurons in each layer is a design choice of the network, according to the complexity of the function to approximate, while the input and the output sizes are dictated by the task to solve. The overall mathematics of a FC layer can be expressed in a matrix form

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) = \sigma(\mathbf{z}) \tag{1.2}$$

with $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_K]^T$ and $\mathbf{b} = (b_1, b_2, ..., b_K)$ the matrix of the weights and bias vector for a layer with K neurons.

MLPs can theoretically approximate whatever non-linear function $f(x)$ [43]. However, in practice they are not the best option to efficiently approach any ML tasks. For this reasons, different architectures have been intensively studied, as discussed in the following part of the chapter.

1.1.3 Activation functions

The learning process of a neural network consists in the adaptation of its weights and bias. Activation functions play a crucial role in the learning process of a complex non-linear function, being responsible of introducing the non-linearity in each neuron's operation. However, using an activation function with binary output, as done in the perceptron with a step function, lead small changes of the parameters to significantly modify the output. In other words, the output can switch from 0 (or -1 according to the activation function used) to 1, with a small variation of the weights.

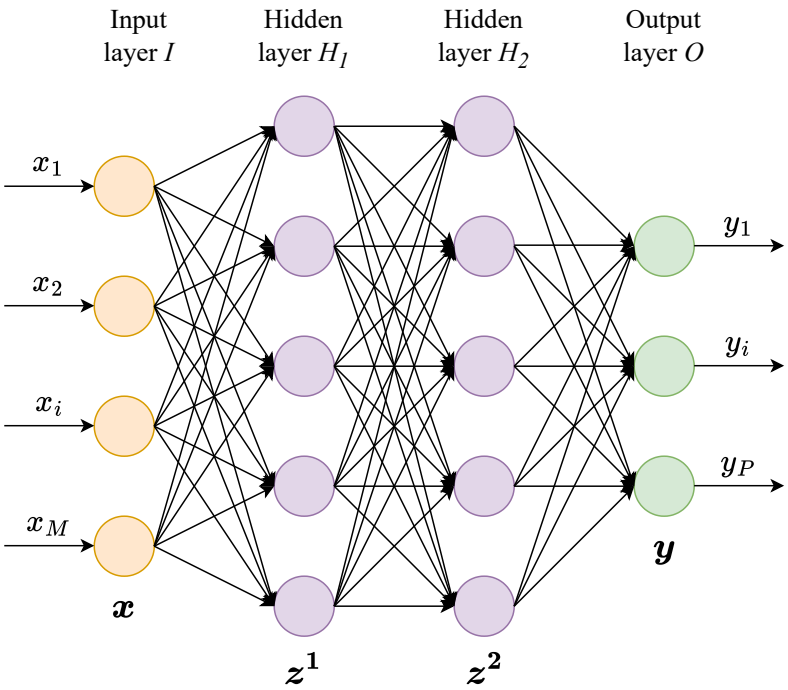


Fig. 1.2 Multi-layer perceptron schematic model with M inputs, P outputs and two hidden layers. The signal **x** enters from the input layer and flows to the right until the output **y** is obtained.

Sigmoid function The solution to overcome this limitation is the introduction of a smoother activation trend, the *sigmoid function*.

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{1.3}$$

A more specific formulation for neurons will be:

$$\sigma(wx+b) = \frac{1}{1+exp(-\sum_{i=1}^{n} w_i x_i - b)} \tag{1.4}$$

It provides a smoother variation of the output with respect to modification of the parameters. This mitigates the learning process with respect to a binary step functions. Nonetheless, the output still remain bounded in a limited range.

Linear activation function A linear activation function of the form $\sigma(z) = cz$, produces an output proportional to the input. Graphically, it results in a simple line

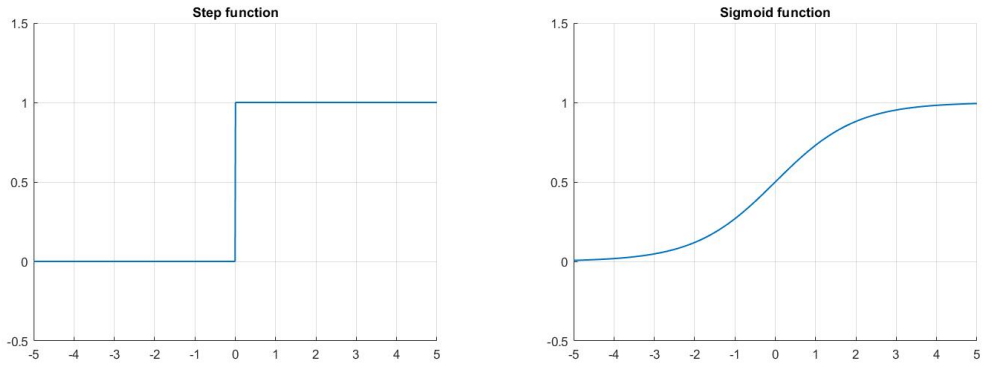


Fig. 1.3 On the left a step activation function: the output is binary as in the perceptron due to the sharp variation from 0 to 1. On the right the sigmoid activation function showing its smooth trend in the same interval.

passing for the origin. Although the output is not binary, it produces some problems in neural network's training. In fact, in a network having linear activation in all neurons the output signal will merely be a linear signal as well, making possible to replace multiple layers with just an equivalent one. Hence, linear activation function is generally adopted in the output neurons of regression tasks, when there are no strict signal bounds.

Tanh activation function The hyperbolic tangent activation function presents a behaviour similar to sigmoid functions. The main difference between the two of them is the range of the output values. $\tanh(z)$ presents an output bounded within -1 and 1. The hyperbolic tangent can be preferred to the sigmoid for reasons strictly related to the specific application. The expression of the function is reported below, whilst its graphical representation is shown on the left in Fig. 1.4.

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{1.5}$$

The expression for the activation function of the neurons considering $\tanh(wx+b)$ can be formulated with:

$$\tanh(z) = \frac{1 + \tanh(z/2)}{2} \tag{1.6}$$

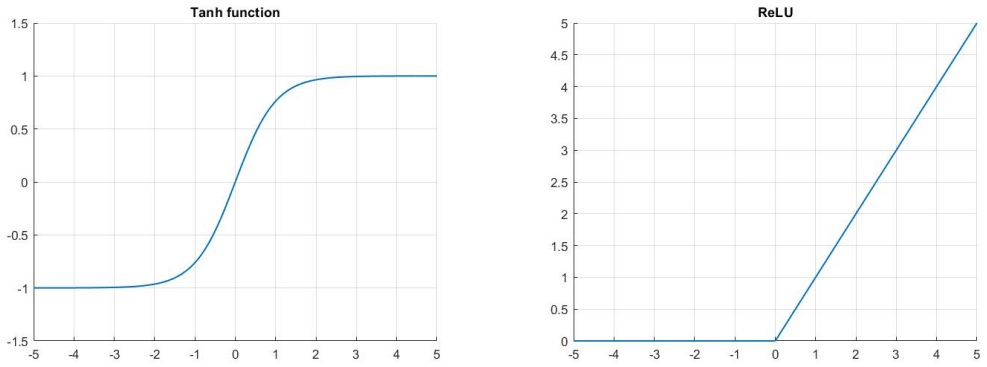


Fig. 1.4 On the left the tanh activation function: it presents a smooth shape in the output range (-1,1). On the right the ReLU activation function.

Rectified Linear Unit (ReLU) The Rectified Linear Unit (ReLU) [44, 45] fuction is defined according to the expression:

$$\sigma(z) = max(0, z) \tag{1.7}$$

Its output signal will be a classic ramp for positive inputs, 0 otherwise. Although it seems very similar to a linear unit activation function, ReLU presents several advantages. First of all, its non-linearity gives it good approximation properties, differently from the simple linear unit. Moreover, due to its nature, it allows to a restricted part of neurons to fire and let the signal pass. In this way the network is lighter from a computational point of view, since ReLU is an easier mathematical operation compared to sigmoid like functions. ReLU is probably the most popular activation function in Deep Learning, not only for the already mentioned benefits. It resulted to be an effective solution for more complex issues such as the *vanishing or exploding gradient*. Many variants of ReLU have been developed to further improve its performance. Among them, Leaky ReLU [46] allows small negative signal passage with an additional linear trend with reduced slope. Other variants like the Exponential Linear Unit (ELU) [47] and the Gaussian Error Linear Unit (GELU) [48] focuses on make ReLU continuous in $z = 0$ and smoother.

Softmax activation function The *softmax* function, also known as *normalized exponential function*, is a widely used activation unit. It is especially chosen for the output layer of neural networks for classification tasks. The standard softmax

expression is:

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{1.8}$$

The main property of softmax function is that the resulting signal can be interpreted as a probability distribution. In other words, the output of the network tells us which is the probability of a sample to be classified with the label of a certain category. This becomes clear by looking at the expression of the function: the exponential of each component of the vector $\mathbf{z}$ is divided by the sum of all the exponentials.

$$\sum_j \sigma(z_j) = \frac{e^{z_j}}{\sum_k e^{z_k}} = 1 \tag{1.9}$$

Moreover, it provides a confidence score related to the network's prediction, which is a precious information about its performance.

1.1.4 Training Artificial Neural Networks

At this point, it is possible to explain the main concepts about the learning process of ANNs. Training an ANNs means to obtain the optimal set of weights and bias for the model that provide the desired output, according to the task. Thus, a cost expression is needed to set up the optimization problem and to evaluate how the network is adapting its weights. For this purpose a cost function, that in ML domain is usually called *loss function* is introduced.

$$\mathcal{L}(\mathbf{w}, \mathbf{b}) = ||\mathbf{y} - \hat{\mathbf{y}}||_l = \sum_{j=1}^{d} (|y_j - \hat{y}_j|^l)^{1/l} \tag{1.10}$$

In the expression above, $\mathbf{w}$ and $\mathbf{b}$ are the weights and bias of the network. With $\mathbf{y}$ we refer to the desired output and with $\hat{\mathbf{y}}$ to the output prediction of the ANN. The loss function reported in the equation above is an l-norm loss function, usually adopted in regression tasks with continuous numerical output variables. For $l = 1$ we have a Mean Absolute Error (MAE) loss, while for $l = 2$ we have a Mean Squared Error (MSE) loss. Since the number of variables involved in an ANN is huge, an iterative algorithm called *Gradient Descent* is therefore used for the optimization. A generic n-dimensional input vector v is considered. For small variations of each variables v_j

it is possible to express the variation of the loss function in the following way:

$$\Delta\mathcal{L} \approx \frac{\partial\mathcal{L}}{\partial v_1}\Delta v_1 + \frac{\partial\mathcal{L}}{\partial v_2}\Delta v_2 + ... + \frac{\partial\mathcal{L}}{\partial v_j}\Delta v_j + ... + \frac{\partial\mathcal{L}}{\partial v_n}\Delta v_n \tag{1.11}$$

A more compact form of the expression above can be rewritten by exploiting the concept of gradient of $\mathcal{L}$:

$$\nabla\mathcal{L} = \left(\frac{\partial\mathcal{L}}{\partial v_1}, \frac{\partial\mathcal{L}}{\partial v_2}\right)^T \Delta\mathcal{L} \approx \nabla\mathcal{L}\cdot\Delta\mathbf{v} \tag{1.12}$$

where $\Delta\mathbf{v}$ is the vector representation of the variations of $\mathbf{v}$.

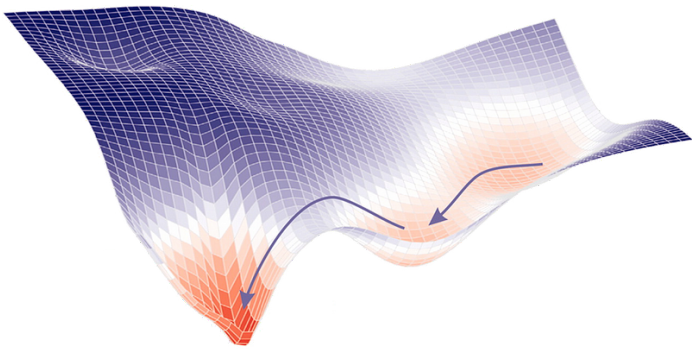


Fig. 1.5 Gradient descent visualization on a 3D surface. Image from [49].

The gradient's notation is useful because it directly relates the variations of $\mathbf{v}$ with the ones of $\mathcal{L}(\mathbf{v})$. At this point, we are interested in finding a set of $\Delta\mathbf{v}$ such that $\Delta\mathcal{L}$ is negative. The reason behind that can be easily explained with a visual metaphor. It it sufficient to imagine the loss function as a deep valley. Starting from a random point on its surface, the goal of the process is to reach its bottom (see Fig. 1.5). Hence, it is necessary to choose the appropriate movements $\Delta\mathbf{v}$ to go down correctly, which corresponds to a negative $\Delta\mathcal{L}$. This concept can be expressed with the following, considering also a small positive parameter called *learning rate*:

$$\Delta\mathbf{v} = \mathbf{v}' - \mathbf{v} = -\eta\nabla\mathcal{L} \tag{1.13}$$

The representative equation of gradient descent can be obtained by combining the last two equations:

$$\Delta\mathcal{L} \approx -\eta\nabla\mathcal{L}\cdot\nabla\mathcal{L} = -\eta||\nabla\mathcal{L}||^2 \tag{1.14}$$

Therefore, an update rule for the parameters $\mathbf{v}$ is provided by the algorithm:

$$\mathbf{v} \to \mathbf{v}' = \mathbf{v} - \eta \nabla \mathcal{L} \tag{1.15}$$

To sum up, by choosing a suitable set of changes in the parameters, it is possible to minimize a loss function $\mathcal{L}(\mathbf{v})$ with gradient descent. A correct choice of the learning rate is also crucial to tune the process. It has to be small enough to guarantee a good approximation of $\Delta\mathcal{L}$ especially in the final steps of the algorithm, when the goal is close and fine adjustments are needed. On the other hand, when the global minimum of the loss function is still far, it should not speed down the process too much. For this reasons it is often modified during the process according to an update rule.

Finally, it is possible to express the update rule provided by the gradient descent algorithm using the weights and biases of a neural network. This formulation describes how ANNs are actually trained. For a $j'h$ weight w_j and bias b_j it looks like

$$\begin{aligned} w_j \to w'_j &= w_j - \eta \frac{\partial \mathcal{L}}{\partial w_j} \\ b_j \to b'_j &= b_j - \eta \frac{\partial \mathcal{L}}{\partial b_j} \end{aligned} \tag{1.16}$$

Other Loss functions

The l-norm loss described above is usually adopted for regression tasks with continuous outputs. A popular loss function for classification task is instead the *cross-entropy* loss function.

For the binary case, when we have only two classes in the dataset, the binary cross-entropy loss is defined as:

$$\mathcal{L} = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y})) \tag{1.17}$$

In the case of K different classes in the dataset, the cross-entropy loss expression can be generalized as:

$$\mathcal{L} = -\sum_{c=1}^{K} y \log(\hat{y}) \tag{1.18}$$

1.1.5 Stochastic Gradient Descent

The basic gradient descent can be improved to train ANNs. The loss function $\mathcal{L} = 1/n \sum_x \mathcal{L}_x$ is an average over all the cost contribution $\mathcal{L}_x$ of each training input x, with $\mathcal{L}_x = ||y(x) - a||^2/2$. This means we need to compute gradients $\nabla \mathcal{L}_x$ for each training input, resulting in a slow convergence of the algorithm. To speed it up, a modified version is usually chosen. It is known as *Stochastic Gradient Descent* (SGD) and it consists in considering a limited subset of m out of n samples to compute the gradient $\nabla \mathcal{L}$.

$$\nabla \mathcal{L} = \frac{1}{n} \sum_x \nabla \mathcal{L}_x \approx \frac{1}{m} \sum_j \nabla \mathcal{L}_j \tag{1.19}$$

The small subset of m samples is usually called *mini-batch*. It is possible to express the update rule of each i^{th} weight and bias taking care of this.

$$\begin{aligned} w_i \rightarrow w_i' &= w_i - \frac{\eta}{m} \frac{\partial \mathcal{L}}{\partial w_i} \\ b_i \rightarrow b_i' &= b_i - \frac{\eta}{m} \frac{\partial \mathcal{L}}{\partial b_i} \end{aligned} \tag{1.20}$$

The stochastic gradient descent selects in a random way a mini-batch from the training data for each iteration of the algorithm. When all have been used once, a *training epoch* is finished and a new cycle is started. The number of epochs required to finish a training process depends on the specific network's size and on the other parameters influencing the process, such as the learning rate and the dimension of the mini-batches.

1.1.6 The backpropagation algorithm

The SGD method allows to find a suitable set of weights and biases. However, computing the gradients to update each parameter of a deep multi-layer ANN can represent a computational bottleneck of the training process. The *backpropagation* algorithm [42] has been introduced to boost the efficiency of the gradient computation for the entire network's model, and today the approach is a pillar of Deep Learning's success. It merely consists in computing partial derivatives of the loss function $\mathcal{L}$ with respect to all the parameters of the network. As the name suggests, it works

starting from the output layer going backwards to the first layer of the model. Given the operation performed by a generic neuron of the last layer L of the ANN:

$$a_j^L = \sigma(z_j^L) = \sigma(\mathbf{w}\mathbf{x} + b) = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{1.21}$$

where a_j^L is the output of the activation function $\sigma(\cdot)$, whereas z_j^L is the result of the weighted sum over the n inputs received by the neuron, before the activation. It is possible to define the quantity δ_j^L as

$$\delta_j^L = \frac{\partial \mathcal{L}}{\partial a_j^L} \tag{1.22}$$

The partial derivative δ_j^L can be used to define the dependence of the overall loss from the error committed by the single j^th neuron in layer L in computing the activation signal a_j^L. The chain rule allows to concatenate the partial derivatives and compute the gradient of the loss with respect to each neuron's weight $w_{i,j}^L$ and bias $b_{i,j}^L$:

$$\frac{\partial \mathcal{L}}{\partial w_{i,j}^L} = \frac{\partial \mathcal{L}}{\partial a_j^L}\frac{\partial a_j^L}{\partial w_{i,j}} = \delta_j^L \frac{\partial a_j^L}{\partial z_j^L}\frac{\partial z_j^L}{\partial w_{i,j}^L} \tag{1.23}$$

$$\frac{\partial \mathcal{L}}{\partial b_{i,j}^L} = \frac{\partial \mathcal{L}}{\partial a_j^L}\frac{\partial a_j^L}{\partial b_{i,j}} = \delta_j^L \frac{\partial a_j^L}{\partial z_j^L}\frac{\partial z_j^L}{\partial b_{i,j}^L} \tag{1.24}$$

Following the chain rule, we can easily express the derivative of whatever parameter of the ANN. For example, considering the previous layer $l-1$

$$\delta_j^{l-1} = \frac{\partial \mathcal{L}}{\partial a_j^{l-1}} = \frac{\partial \mathcal{L}}{\partial a_j^l}\frac{\partial a_j^l}{\partial a_j^{l-1}} = \delta_j^l \frac{\partial a_j^l}{\partial a_j^{l-1}} \tag{1.25}$$

In this way, we save computation using previously computed partial derivatives. Moreover, the activation function can also be a different one for each layer. Finally, a schematic summary of a training epoch is reported below. It shows how a SGD optimization algorithm works in combination with backpropagation, with a mini-batch of m training samples for a single training epoch.

Algorithm 1 A training epoch with SGD optimizer.

Input: a feed-forward ANN with parameters **w**, **b** and L layers, with input layer $l = 0$ and output layer $l = L$, a mini-batch size m, a learning rate η.

Randomly split the training dataset in mini-batches of size m.
for each mini-batch in the training set **do**
  for each input x in the mini-batch **do**
    Feed-forward: x passes from the input through all the hidden layers.
    for each hidden layer $l = 1, 2, ..., L$ **do**

$$z^{x,l} = w^l a^{x,l-1} + b^l$$
$$a^{x,l} = \sigma(z^{x,l})$$

    end for
    Output error $\delta^{x,L}$: compute the error in the output layer

$$\delta_j^L = \frac{\partial \mathcal{L}}{\partial a_j^L}$$

    Backpropagate the error computing the partial derivatives for each parameter
    for each layer $l = L-1, L-2, ..., 1$ **do**

$$\frac{\partial \mathcal{L}}{\partial w_{i,j}^L} = \frac{\partial \mathcal{L}}{\partial a_j^L}\frac{\partial a_j^L}{\partial w_{i,j}} = \delta_j^L \frac{\partial a_j^L}{\partial z_j^L}\frac{\partial z_j^L}{\partial w_{i,j}^L}$$
$$\frac{\partial \mathcal{L}}{\partial b_{i,j}^L} = \frac{\partial \mathcal{L}}{\partial a_j^L}\frac{\partial a_j^L}{\partial b_{i,j}} = \delta_j^L \frac{\partial a_j^L}{\partial z_j^L}\frac{\partial z_j^L}{\partial b_{i,j}^L}$$

    end for
    Update: update the weights and biases with SGD rule.
    for each parameter in the layer $l = L, L-1, ..., 2$ **do**

$$w_i^l \to w_i^l - \frac{\eta}{m}\frac{\partial \mathcal{L}}{\partial w_i^l}$$
$$b_i^l \to b_i^l - \frac{\eta}{m}\frac{\partial \mathcal{L}}{\partial b_i^l}$$

    end for
  end for
end for

This is the basic algorithm to train a generic ANN. Further interventions can be thought when the learning process does not work properly. A selected list of possible actions and methods to improve the training of an ANN are briefly discussed in the following section.

1.1.7 Adam optimizer

Beside Stochastic Gradient Descent (SGD), many different algorithm have been developed to solve the optimization problem in the ANNs learning process. Among them, the Adaptive moment estimation (Adam) [50] optimizer algorithm deserves a greater focus due to its popularity and advantages. Adam is an adaptive learning rate method. In other words, learning rates are modulated specifically for each different parameter during the update step. This is done by estimating the first and the second moments of the gradient. Before looking at the entire algorithm, it is convenient to introduce the concept of *moment*. It can be defined as the expected value of a random variable to the power of n.

$$m_n = \mathcal{E}[X^n] \tag{1.26}$$

Hence, the first moment of a random variable is equal to its mean, whilst the second one is the uncentered variance. For the estimation of such quantities for the gradient, Adam makes use of exponentially moving averages m and v computed with the gradient obtained from the current mini-batch:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{1.27}$$

Where g is the gradient on the the mini-batch, β_1, β_2 constant hyper-parameters usually fixed at 0.9 and 0.999. These estimators are biased, hence they need a proper correction. The final expression for the estimator results to be:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{1.28}$$

Therefore, the update rule for the weight during training will be:

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \tag{1.29}$$

Where η is the step size and ε is necessary for numerical stability. According to the definition given by Kingma and Ba in the original paper of 2015 [50], Adam algorithm is reported below.

Algorithm 2 Adam optimizer algorithm. Default values for the constants are $\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 10^{-8}$.

Input: stepsize η, ε for numerical stability, $\beta_1, \beta_2 \in [0,1)$ exponential decay rates for the moment estimates, $f(\theta)$ the stochastic objective function, the initial vector of parameters θ_0.

Output: Resulting parameters θ_t.

$m_0 \leftarrow 0$ ▷ First moment estimate vector set to 0

$v_0 \leftarrow 0$ ▷ Second moment estimate vector set to 0

$t \leftarrow 0$ ▷ Timestep set to 0

while θ_t not converged **do**

$t \leftarrow t + 1$

Compute gradient w.r.t parameters θ: $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$

Update of first-moment and second-moment estimates, biased

$$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$$

$$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$$

Bias-correction of estimates

$$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$$

Update parameters

$$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

end while

Weights initialization

The initial value assigned to weights and biases in the layers plays a fundamental role in the learning process [51]. A simple common practice consists in initializing the weights with a normal probability distribution with 0 mean and standard deviation

equal to $\frac{1}{\sqrt{n_i}}$, where n_i is the number of input connections. A popular initialization strategy has been developed by Xavier Glorot and Yoshua Bengio, known as *Xavier initialization* or *Glorot initialization* [52]. According to this theory, the values for weights are picked from a random uniform distribution bounded between $\pm\frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}}$, where n_i is the number of input connections and n_{i+1} the number of output connections. A good initialization is usually necessary to mitigate training slowdown issues.

1.1.8 Regularization methods

Overfitting

Overfitting is a well known issue affecting the performance of data-driven models such as ANNs. It occurs when a model is designed to fit extremely well on some data, increasing its level of specificity. It is common to incur in overfitting when using models with a high number of parameters such as ANNs. Overfitting must be strongly avoided in ML, since an ANN is useless if provides good results only using data contained in the training set. On the other hand, the model should be complex enough to correctly fit the data and avoid bias errors (*underfitting*). The ability to generalize the performance of ANNs on different data can be improved with several methods.

Splitting data

A fist possible method to reduce overfitting is to split the available data in three different subsets. The *training set* usually includes only a subgroup of the original amount of data and it is used for the proper learning process, together with a *validation set* to monitor the performance of the network at the end of each epoch. This is a key procedure, overfitting can be detected by looking at the loss gap between the training and the validation set. The K-fold cross-validation is a simple regularizing practice to avoid bias errors [53, 38]. This is especially true for non-parametric algorithms in ML such as *K-nearest neighbours*, which do not explicitly make use of a loss function. Validation is also helpful to select the best model found over the training time. If the number of training epochs is not accurately chosen, we could pick up the model that reaches the best metrics score on the validation set. Finally, the selected model can be run on a *test set*. A grid-search procedure is often adopted to look for a suitable combination of the the *hyper-parameters*:

learning rate, mini-batch size, number of epochs, learning rate's decay policy. A simple strategy called *early stopping* consists in interrupting the training when the loss in the validation set becomes stable or starts to increase.

Data augmentation

The availability of a rich dataset is not always guaranteed in real-world applications of Deep Learning. An accurate tuning of the hyper-parameters sometimes is not enough to avoid overfitting. *Data augmentation* is a particular strategy developed for these purposes. In the specific case of image classification, it allows to expand the training dataset at runtime with artificial samples. A set of different transformation such as rotation, cropping, flipping or filtering can be applied to images in order to artificially create new ones. This is particularly useful when the number of samples for each class in the training set is strongly unbalanced, often because some kind of data is more difficult to collect.

L_p Regularization

Overfitting and model's generalization can be also tackled acting directly on the loss function. The L_p regularization methods aims at constraining the values of the parameters in allows boundaries to forbid them to capture fluctuations of the training data distribution. The two most popular methods of L_p regularization are known as L_1 and L_2 regularization. The key concept of both methods is to add a penalty or term to the loss function used in the training process. For example, a cross-entropy loss function can be considered. With the L_2 regularization the loss function assumes the following shape:

$$\mathcal{L} = \mathcal{L}_0 + \frac{\lambda}{2} \sum_{i=1}^{n} w_i^2 \tag{1.30}$$

As shown, the regularization term for the L_2 method is composed of the sum of the squared weights and of a multiplicative factor. It is responsible of reducing the value of the selected variables. Basically, during the learning process the network has to choose certain weights such that a good trade-off between the two terms of the new loss function is found. To highlight the concept better, it is possible to rewrite the expression using the notation $\mathcal{L}_0$ to indicate the original loss function.

The role of λ, which is a positive *regularization parameter*, is central to make things work. According to its value the relevance of the second term with respect to the

first one can be tuned. A small λ makes the regularization term negligible, a large λ increases the importance of learning small weights. Considering a SGD optimization algorithm, a new update rule for the weights can be computed with the L_2 regularized loss function.

$$w \rightarrow w' = w\left(1 - \frac{\eta\lambda}{n}\right) - \eta\frac{\partial\mathcal{L}_0}{\partial w} \tag{1.31}$$

Differently, in the L_1 regularization technique the extra term contains the sum of the absolute values of the weights in the networks and the regularized loss function is expressed with

$$\mathcal{L} = \mathcal{L}_0 + \lambda\sum_{i=1}^{n}|w_i|. \tag{1.32}$$

In an analogue way, the resulting update rule will be:

$$w \rightarrow w' = w\frac{\eta\lambda}{n}sgn(w) - \eta\frac{\partial\mathcal{L}_0}{\partial w} \tag{1.33}$$

Both L_1 and L_2 regularization techniques penalize large weights in the network. However, in L_1 regularization weights are reduced by a constant amount toward 0 value, whilst in L_2 regularization the reduction is proportional to the weight's value. This means L_2 is particularly effective with larger weights. On the contrary the impact of L_1 regularization is much bigger when $|w|$ is very small, shrinking to 0 less important connections. The result is that it selects a selected group of important features and connections.

Activation normalization

A regularization approach to control the evolution of the parameters value during training consists in normalizing the activation signal. Batch Normalization (BN) [54] is the most popular technique. BN performs the normalization channel-wise on the mini-batch, and then uses a linear transformation to re-center and re-scale the resulting activation signal. For each channel dimension of the input tensor $\mathbf{x}$ the empirical mean $\mu^{(k)}$ and standard deviation $\sigma_B^{(k)}$ over the mini-batch B composed of m samples are computed as:

$$\mu_B^{(k)} = \frac{1}{m}\sum_{i=1}^{m}x_i^{(k)}, \quad \sigma_B^{(k)} = \frac{1}{m}\sum_{i=1}^{m}(x_i^{(k)} - \mu_B^{(k)}) \quad \forall k \in 1,...,C \tag{1.34}$$

where C is the number of total channels of the input tensor. The single input tensor $\mathbf{x}$ is therefore normalized channel-wise:

$$\hat{\mathbf{x}}^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{(\sigma_B^{(k)})^2 + \varepsilon}} \quad \forall k \in 1, ..., C \tag{1.35}$$

The tensor is finally re-scaled and re-centered:

$$\mathbf{z}^{(k)} = \gamma^{(k)} \hat{\mathbf{x}}^{(k)} + \beta^{(k)} \quad \forall k \in 1, ..., C \tag{1.36}$$

where γ and β are parameters learned during the training. Hence, each BN layer adds a total of 2C learnable parameters and 2C non-learnable parameters, the empirical mean and standard deviation. However, at inference time, usually there is not a batch dimension in the input tensor. Hence, the empirical mean $\mu^{(k)}$ and standard deviation $\sigma_B^{(k)}$ cannot be computed. For this reason, they are estimated with a running average during the training to enable a deterministic inference at test time:

$$E[\mathbf{x}^{(k)}] = E_B[\mu_B^{(k)}], \quad Var[\mathbf{x}^{(k)}] = \frac{m}{m-1} E_B[\left(\sigma_B^{(k)}\right)^2] \tag{1.37}$$

The transformation performed by the BN layer in the inference step thus becomes a linear transformation of the activation tensor:

$$\mathbf{z}_{inf}^{(k)} = \gamma^{(k)} \frac{\mathbf{x}^{(k)} - E[\mathbf{x}^{(k)}]}{\sqrt{Var[\mathbf{x}^{(k)}] + \varepsilon}} + \beta^{(k)} \quad \forall k \in 1, ..., C \tag{1.38}$$

Dropout

Dropout [55, 56] is a totally different way of acting on the learning process to regularize it. With respect to L_1 and L_2 regularization, it does not act on the loss function. For each training step it randomly selects a certain percentage of neurons contained in a layer and it turn them off, usually the 50%. For these neurons the parameters will not be updated. Hence, only the remaining active neurons are able to create connections with the neighboring layers. The dropout can be also thought as an averaging process among different ANNs. In some sense, this peculiar approach avoid the network to rely on a restricted number of connections, making its performance more robust.

1.1.9 Convolutional Neural Networks

In the previous sections, ANNs have been generally introduced. The described architecture only includes FNNs with dense layers. In this case, each neuron inside a hidden layer is connected to all the neurons of the next layer and of the previous one. This architecture is not very efficient when dealing with high-dimensional input data such as images. Dense layers become too computationally expensive with a huge number of parameters to train, and, furthermore, the spatial structure of the image is not considered in the operation performed by classic neurons. For this reason, inspired by human vision system, *Convolutional Neural Networks* (CNNs) have been studied [39]. The architecture of CNNs is optimized to process images extracting meaningful features to efficiently solve computer vision tasks. The three pillars of CNN can be identified in the following concepts:

- *local receptive field*;
- *shared weights*;
- *pooling*.

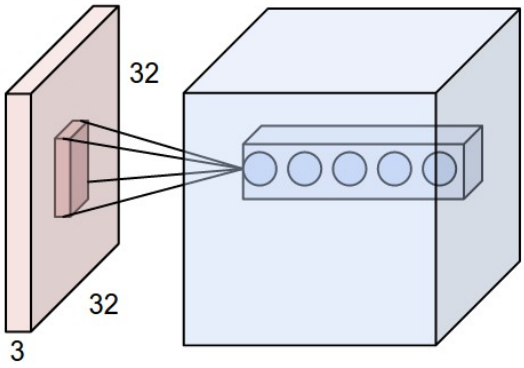


Fig. 1.6 A representative scheme of a convolutional layer. An input image with dimensions $32 \times 32 \times 3$ is processed by the kernel and mapped to a new data volume with 5 feature maps. The receptive field is represented by the smaller volume on the image tensor.

From a mathematical perspective, an image is a 3D tensor of shape $[H, W, C]$, being H the height, W the width in pixels, and C the number of channels. Colour images usually have 3 channels (RGB). In order to avoid a full connection between input pixels and neurons of a hidden layer, each neuron is associated to a small rectangular region of the image of size $K_h \times K_w$ in the spatial dimension and C channels. For example a 5×5 receptive field covers a square region of 25 pixels on the image.

This is called the local receptive field of the neuron. The convolutional layer learns a weight for each connection and a general unique bias. The ensemble of weights and bias identify a convolutional *kernel* or *filter*, which is shared among all the neurons of the layers. This is a fundamental aspect of a convolutional layer. The operation performed by the single neuron unit $(i,j)^l$ is the same weighted sum of MLP, being $\sigma(\cdot)$ a generic activation function and using rectangular filter of size $K_h \times K_w$:

$$a^l_{j,k} = \sigma\left(b + \sum_{u=0}^{K_h-1}\sum_{v=0}^{K_w-1} w_{u,v}\, a^{l-1}_{i+u,j+v}\right) \tag{1.39}$$

The filter is then slided over the image by a quantity called *stride* $[S_h, S-w]$. This operation is repeated horizontally and vertically until complete coverage of the image tensor. The complete 3D output tensor of a given filter is called a *feature map*. The number of feature maps F is a design parameter for the CNN that depend on the task and on the data. Moreover, the spatial dimension of the output map will be:

$$H' = \frac{H - K_h + 2P_h}{S_h} + 1 \quad , \quad W' = \frac{W - K_w + 2P_w}{S_w} + 1 \tag{1.40}$$

where $[P_h, P_k]$ indicates the padding quantity. Padding consist in simmetrically adding 0 values to the border of the input tensor to compensate the reduction of spatial resolution caused by non-unitary kernel's size. Conversely, the stride is responsible of a spatial sub-sampling operation, avoiding a piwel-wise sliding of the kernel. The number of total parameters in a convolutional layer is given by $K_h \cdot K_w \cdot C \cdot F + F$ biases, that is much lower compared to a FC architecture, enabling a faster training process.

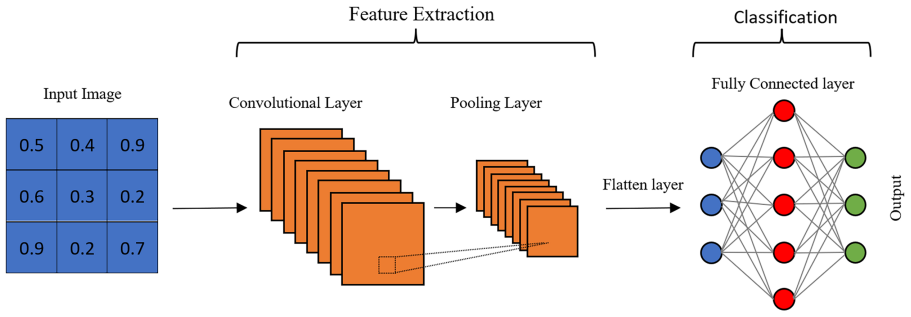


Fig. 1.7 A typical pipeline of a CNN for image classification. Image from [57].

An additional operation of CNNs is *pooling*. With the same sliding mechanism described for the convolution, pooling filters can be employed to simplify the in-

formation contained in small regions of the image. For example, the *max-pooling* operation can output the maximum activation values over a 2×2 input region. Pooling allows to further reduce the number of neurons and parameters in the ANN. To sum up, a basic complete architecture of a CNN is composed of a first stack of convolutional and pooling layers devoted to extract meaningful features from the image. This first part of the CNN is usually called *backbone*. The data volume is squeezed all along this convolutional section. Extracted features are then flattened, passed through FC layers and a final task-specific output layer, as shown in Fig. 1.7. Optimizers such as SGD and Adam, together with backpropagation, work in the same way for CNN.

This kind of overall network architecture has been adopted by a wide range of successful models, from AlexNet [58] to ResNet [59]. Several methods have been also studied to enhance deep CNN models with more advanced operations: residual connections [59], inception blocks [60], depth-wise separable convolutions [61], squeeze-and-excitation [62], channel and spatial attention [63].

1.1.10 Self-attention and Vision Transformer

The most recent architecture that has become a standard in Deep Learning is the Transformer [40]. Originally thought for Natural Language Processing (NLP) problems with sequence data, in the last years it has reached state-of-the-art performance in a wide variety of tasks. Hence, Transformer has been conceived for sequential data processing, potentially evolving the already existing Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks [64–66]. However, they have been adapted also to classic computer vision tasks such as image classification, as explained below.

Multi-head self-attention

The Transformer is based on the key operation of self-attention computation, computed with a multi-head encoder-decoder architecture inspired by autoencoders [68]. Given the input sequence $\mathbf{X}$ with shape $T \times d_{model}$, self-attention is computed through a scaled dot-product attention operation, using h parallel independent heads with dimensionality $d_h = d_{model}/h$. Firstly, three matrix representations of the input sequence are computed using FC layers, namely query $\mathbf{Q}$, key $\mathbf{K}$ and value $\mathbf{V}$:

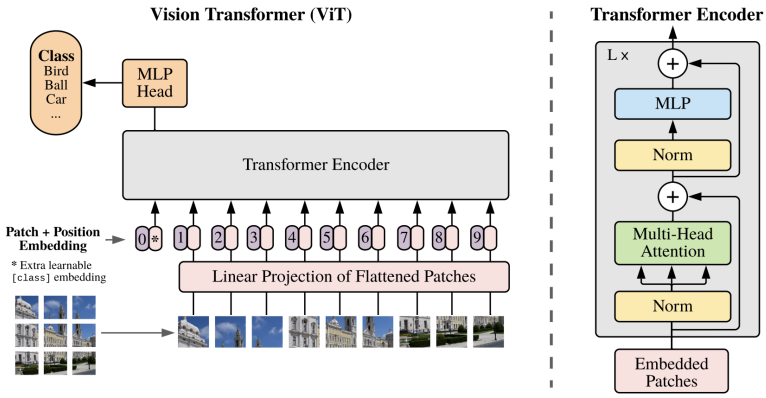


Fig. 1.8 The Vision Transformer (ViT) schematic architecture. On the left, the data flow of the model is depicted, on the right the Encoder residual scheme with MSA and MLP modules is shown. Image from [67].

$$\mathbf{Q}_i = \mathbf{X}\mathbf{W}_i^q \ , \ \mathbf{K}_i = \mathbf{X}\mathbf{W}_i^k \ , \ \mathbf{V}_i = \mathbf{X}\mathbf{W}_i^v \quad \forall i \in 1,...,h \tag{1.41}$$

where $\mathbf{W}_i^q$, $\mathbf{W}_i^k$ and $\mathbf{W}_i^v$ are the weights matrix of the FC layers with shape $d_{model} \times h$. Then, the self-attention tensor A_i is computed for each head h of the Transformer Encoder:

$$\mathbf{A_i} = \text{Self-Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_h}}\right)\mathbf{V} \tag{1.42}$$

where $\sqrt{d_h}$ is the dimension of the key vector $\mathbf{K}$ and query vector $\mathbf{Q}$.

Finally, the multi-head output is obtained by concatenating the H heads and a linear projecting with an output FC layer:

$$MSA(\mathbf{X}) = Concat(\mathbf{A_1},...\mathbf{A_h})\mathbf{W}^o \tag{1.43}$$

where W^o has shape $d_{model} \times d_{model}$, since $h \cdot d_h = d_{model}$.

Vision Transformer

In this chapter, we introduced the most popular version of Transformer architecture for computer vision applications, the Vision Transformer (ViT) [67]. ViT is based on the key intuition that an image of resolution $H \times W$ can be decomposed and processed as a sequence of N patches $P \times P$, such that $N = HW/P^2$. Each patch

is flattened and projected into a latent space of dimensionality d_{model} through FC layers. A sequence is then obtained concatenating the embeddings of the patches. A positional information is encoded into each patch. The positional encoding can be learnable or made of constant geometric progressions. An extra learnable class token embedding is also inserted at the beginning of the sequence. The obtained sequence is then processed with a Transformer Encoder, composed of a stack of L residual layers. Each layer subsequently computed the MSA and then re-project the sequence vector with MLPs. A normalization layer is present before the MSA and the MLP modules in each residual layer. A final classification head is used to predict the output distribution. Fig. 1.8 shows the ViT model architecture and the schematic operation flow of an Encoder layer.

1.2 Optimized execution of ANN at the Edge

This thesis is mainly devoted to the study and development of Deep Learning models for service robotics tasks. Thus, the execution of ANNs directly on-board the robotic platform is strongly preferred in this context, avoiding cloud-based servers and only relying on the low-power computational hardware already present on the robot. This setting offers competitive advantages for the final application of the robot, improving latency and consumes, and, moreover, avoiding privacy issues and connectivity requirements. In the core of this thesis, we find many examples where ANNs are optimized for on-board off-line execution. Among the most representative examples, Chapter 6 offers a complete case study in the sector of robotic domestic assistance, while Chapter 12 deals with real-time requirements for image processing and transmission processes. The specific set of techniques studied to deploy AI algorithms on low-power-embedded devices is commonly titled *Edge-AI* [69].

- *Pruning*: it consists in removing the low-weights connections in the model's graph, that does not contribute significantly to the predicted outcome. Pruning can be performed at the level of the single weights or removing entire neurons or layers.
- *Quantization*: it reduces the numerical precision of the model's parameters. Quantization can be performed directly on the model post-training, or diversely a quantization-aware training can be done considering the effect of reduced precision already in the learning phase. For example, weights and activations

can be quantized from 32-bit floating-point numbers to lower-precision formats such as 16-bit floats or 8-bit integers, reducing the computational load required by the feed-forward inference.

- *Knowledge Distillation*: it is a transfer learning techniques that aims to distill the knowledge from a bigger model to a more compact model with reduced size. The target small model, the *student*, is trained to match the output distribution or feature representations of the *teacher* model.

In this thesis, we mainly adopt techniques of post-training quantization and knowledge distillation. The principle of both of them are explained directly in Chapter 12. The implementation of optimization and quantization methods is supported by the TFLite library[1]. TFLite basic conversion already provides some advantages in the memory allocation for the graph execution, mitigating the inference performance of models on CPU. Then, it offers different level of quantization. Float16 quantization halves the precision and memory requirement of weights and activations, allowing for execution on both CPUs and GPUs. 8-bit weights quantization is another option, limiting the execution on CPU, Edge TPU (Tensor Processing Unit), and Microcontrollers. The full quantization of the model reduce both the weights and outputs to 8-bit precision. This final step can drastically boost the inference and memory performance of the model, but also significantly reduce its accuracy. From the hardware perspective, graphic processing units (GPUs) are the standard for parallel matrix computation through ad-hoc libraries such as Nvidia CUDA. GPUs allows for float-16 and float-32 precision, while CPUs only admit float-32 or 8-bit integer operations. Beside classic CPUs, some ad-hoc computational devices have been realized to boost fast inference at the edge. Among them, the most popular are the Nvidia Jetson[2] platforms (Nano, Xavier, Orin), the Intel Movidius VPU (Visual Processing Unit)[3] and the Google Coral Edge TPU[4]. Nvidia Jetson platforms are single-board computers with dedicated embedded GPUs, with power consumption going from 5-10 W for the Jetson Nano, and 10-30 W for the Xavier version. Differently, Coral Edge TPU and Intel Movidius VPU are tiny graphics accelerator, usually bought as USB stick devices, for computer vision and AI models that consume up

[1] https://tensorflow.org/lite
[2] https://www.nvidia.com/it-it/autonomous-machines/embedded-systems/
[3] https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu/movidius-myriad-x/products.html
[4] https://coral.ai/

to 2 W. Coral only allows for 8-bit integer operations, hence, for fully quantized models. Most recently, the Intel Movidius Myriad X VPU have also been integrated on camera devices for robotics like the OAK-D[5], to directly provide practitioners with a complete set-up for fast computer vision applications at the edge.

[5] https://shop.luxonis.com/products/oak-d

Chapter 2

Deep Reinforcement Learning

In this section the Reinforcement Learning (RL) framework is explored, starting from an introduction of the main concepts. The chapter is focused on Deep Reinforcement Learning (DRL) algorithms that are used in the core of the thesis. Hence, DRL concepts are gradually, briefly explained starting from tabular RL such as Dynamic Programming and Monte Carlo methods, touching the idea of Temporal-Difference learning and finally getting to approximate solution methods. Some examples of the principal algorithms are also provided for a more comprehensive and clear analysis of RL. In the last part of the chapter, approximate solution methods are treated with a great focus on actor-critic paradigms, deepening the discussion for the Deep Deterministic Policy Gradient and the Soft Actor-Critic methods. The whole chapter is written extracting the RL concepts from the reference RL book [70] and specific articles cited in the text.

2.1 Introduction to Reinforcement Learning

The foundational stone of many learning theories is that we learn through the interaction with the environment where we are placed in, as represented in Fig. 2.1. Many recurrent events during the childhood confirm this theory: a child continually learn through a trial and error procedure, from walking to riding a bike. Framing the problem in a more general scene, according to the action an agent chooses in a certain condition, the environment gives it back a response, for example, the hurting sensation perceived when falling. A good consciousness of the environment is

necessary to successfully address a task, and more in depth, to be aware of the result of a chosen action. This causal connection is effectively learnt through experience. Reinforcement Learning (RL) can be defined as a computational approach focused

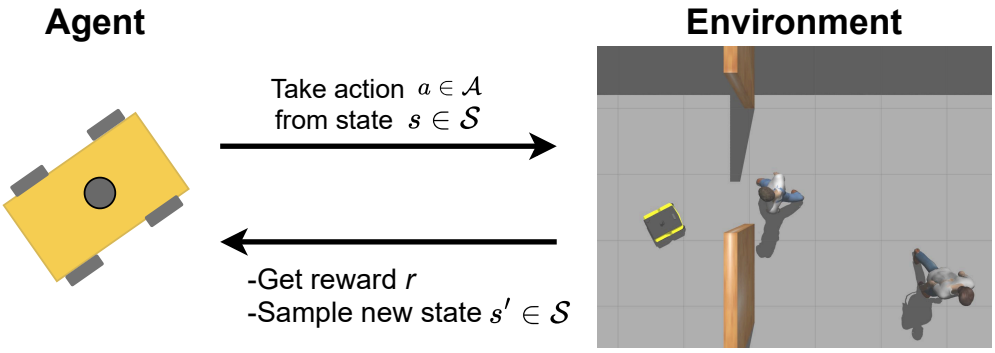


Fig. 2.1 The interaction between an agent and the environment in reinforcement learning.

on learning how to reach a task-specific desired behaviour by interacting with the environment. More specifically, it is associated to the problem of how to map situations to actions in order to maximize a numerical reward signal. The main characteristics of RL can be identified in the following:

- it can be described as a *closed-loop* problem because the selected action influence the successive inputs.
- the action is chosen by the learning agent according to a trial and error procedure.
- choosing an action in a certain situation determine the immediate reward as well as the consequent situations and rewards.

Due to these peculiar aspects, RL can be identified as a separate, different paradigm in Machine Learning. The differences with supervised and unsupervised learning are quite evident. In RL, no labeled dataset is exploited, and the process merely focuses on maximizing the reward signal rather than in finding hidden structure in unlabeled data.

As already explained, the agent and the environment play the two pivotal roles in RL. Besides, it is possible to identify a set of elements that are present in almost every RL system:

The policy of a learning agent defines its behaviour, i.e. its way of selecting actions at a given instant of time. It is therefore the core of a RL agent, since it defines how the agent will tackle a given task. In other words, the policy is responsible of the mapping from a state in the environment to a specific action to perform in that situation. Generally speaking, a policy can be both stochastic or deterministic.

The reward signal defines the goodness of a certain event for the agent. At each time step, the environment sends to the agent a numerical evaluation of its behaviour, a *reward*. The agent tries to maximize the total reward over the entire training period. This means the reward signal is responsible of guiding the agent to the desired behaviour by indicating good and bad actions.

The value function can be roughly defined as a long-term advisor for the agent. It associates to a state a *value* which is correlated to the total reward which is possible to gain in the future starting from that state. Hence, if rewards give the agent an indication of what is good to do in an immediate sense, values take care of the potential development of taking a decision in a certain situation. This is certainly a fundamental role in a RL system. For example, an agent can decide to move into a new state gaining a low immediate reward. Nevertheless, this can still be a good choice if it gives the agent the chance to reach next states that yield high rewards. When selecting an action, it should be considered the one that allow to reach states with the associated highest value, not the highest reward, because the total reward accumulated over the long run will be much greater. Unfortunately, an efficient estimation of the value function is not trivial. For this reason this task is a crucial component of almost every RL algorithm.

A model of the environment can be defined as an approximation of the environment dynamics and it can be useful to make predictions about its future evolution for planning purposes. Methods that exploits such model are referred as *model-based*; on the contrary, *model-free* methods do not make use of any model and they are based on a pure trial and error learning. In this thesis only model-free RL methods are explained and used in the project.

2.1.1 Markov Decision Process

RL aims to frame the problem of goal-based learning from interaction. The *Markov decision process* (MDP) is a way to formally define this problem. As already introduced in the previous section, RL is based on the interaction between:

- the *agent*: the learner and decision-maker;
- the *environment*: what is outside the agent and interact with it.

This continuous interaction can be formalized in a closed-loop dynamics as shown in Fig. 2.2. At each discrete time step $t = 0, 1, 2, 3...$ the agent receives in input the *state*

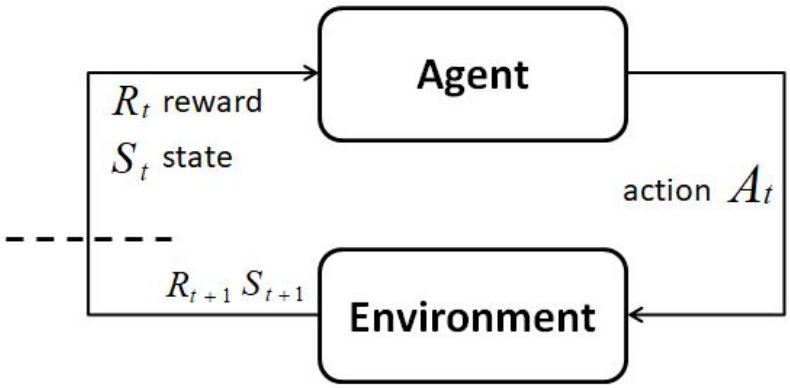


Fig. 2.2 Markov Decision Process schematic.

of the environment $s_t \in \mathcal{S}$, which should contain all relevant information about the environment. The agent chooses an action $a_t \in \mathcal{A}$ based on that. At the next time step the environment sends back a new state s_{t+1} and a reward $r_{t+1} \in \mathcal{R} \subset \mathbb{R}$. Therefore, the process generates an alternated sequence of signals exchanged between the agent and the environment as the following:

$$(s_0), (a_0), (r_1, s_1), (a_1), (r_2, s_2), (a_2), ...$$

A graphical representation can be useful to better understand a generic set of transitions. An example is reported in Fig. 2.3, where circles contains the states and arrows represent the transition from a state to another one based on the selected action. A generic response of the environment at time $t+1$ can be expressed by a probability distribution that takes in consideration all the previous events:

$$P_r\{r_{t+1} = r, s_{t+1} = s' | s_0, a_0, R_1, ..., s_{t-1}, A_{t-1}, r_t, s_t, a_t\}$$

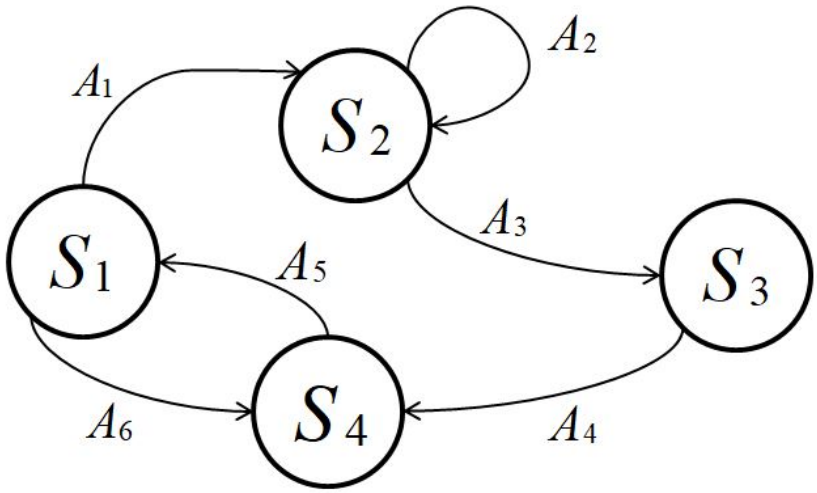


Fig. 2.3 Finite Markov Decision Process: a simplified transition schematic.

However, if the state contains all the relevant information about past transitions the environment's step at time $t+1$ depends only on the state and the action at time t. When this is true, the state signal has the *Markov property*. In this case the previous expression becomes:

$$p(s',r|s,a) = P_r\{s_{t+1} = s', r_{t+1} = r|s_t = s, a_t = a\}$$

Moreover, the Markov property can be extended to the whole environment if the previous assumption holds for every s', r. This property has a tremendous relevance for the whole RL conceptual framework. In fact, it means that given the actual state and action, we are able to predict all future states and possible rewards. Markov property is therefore extremely advantageous in RL to efficiently choose good actions. Hence, it is possible to refer to a RL task as a Markov decision process whenever the Markov property is satisfied. In the case of finite state and action spaces, the process is called a *finite Markov decision process*.

Goals, Reward and Returns

As already introduced, a RL agent has the final goal of maximizing a numerical signal called reward. At each time, the environment assign a reward to the agent. For example, a robot can learn how to collect empty bottles for recycling simply by assigning a +1 for each bottle collected and a -1 every time it collect a wrong object or miss a bottle. A wide variety of rewards can be designed according to the specific

application. The final amount of reward obtained can be formally called *return* and it can be indicated as G_t. The mathematical expression of G_t depends on the specific problem to tackle, a basic case could be the sum of all the rewards:

$$G_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T$$

A final time instant T concludes the sequence of rewards. A finite number of agent-environment interaction steps makes the learning process evolve in separated subsequences called *episodes*. This episodic interaction can be easily compared to games levels that finishes with different scores. Differently, some processes may need to be represented by a continuous agent-environment interaction. In this scenario, it would be $T = inf$ and the return could diverge. A *discount factor* is therefore introduced to prioritize the latest steps performed in the training. In this case, the expected discounted return will be:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + = \sum_{k=0}^{\inf} \gamma^k r_{t+k+1},$$

where the parameter γ is $0 \leq \gamma \leq 1$ and it is called the *discount rate*. With $\gamma = 1$ the result is unchanged with respect to the previous expression. When instead $\gamma < 1$ the infinite sum will converge to a finite value, given all bounded reward contributions. A peculiar case occurs with $\gamma = 0$, since the only immediate reward r_{t+1} is maximized.

Optimal Value Functions and Policies

A formal definition of *value functions* can be given at this point. In particular, it possible to define different value functions for the RL framework. A *state-value function* express how good is a certain state for the agent, according to the associated expected return. In an analogue logic, a *state-action pair value function* can be defined. This last function specifies how good an action is for the state in account. The concept of *policy* can be formulated as the function that associate the states to the probabilities of selecting the possible actions. Hence, the policy describes agent's behaviour also account for uncertainty in the action selection. According to this definition, an agent which follows a policy π has the probability $\pi(a|s)$ to choose the action a in the state s. The value function under the policy π is indicated with $v_\pi(s)$. It expresses the expected return that the agent should get when starting

from state s having a policy π. In a MDP this can be written as:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\inf} \gamma^k R_{t+k+1}\middle| s_t = s\right]$$

where $\mathbb{E}_\pi[\cdot]$ is the expected value of a random variable given the policy π of the agent. v_π is the *state-value function for policy* π.
In an analogue way the *action-value function for policy* π, q_π can be defined:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\inf} \gamma^k R_{t+k+1}\middle| s_t = s, a_t = a\right]$$

Hence, $q_\pi(s,a)$ formally expresses the value of choosing the action a in state s under the policy π.

At this point it is easy to give a definition of optimal policy and optimal value function. It can be said that a policy π is better than another policy π' if its expected return is the greatest among the two of them, for all states:

$$\pi > \pi' \iff v_\pi(s) > v_{\pi'}(s), \forall s \in \mathcal{S}$$

A policy is said to be *optimal* if it is better than or equal to all other policies. An optimal policy is usually denoted as π_*. There could be more than one optimal policy, but all of them will share the *optimal state-value function*, v_*:

$$v_*(s) = \max_\pi v_\pi(s), \forall s \in \mathcal{S},$$

as well as a *optimal action-value function*, q_*:

$$q_*(s,a) = \max_\pi v_\pi(s,a), \forall s \in \mathcal{S},$$

Optimal policies and value functions cannot be found in non-finite MDPs due to practical constraints in the implementation (such as the amount of available memory), however useful approximations can be used.

2.2 Tabular methods

In this section, a brief discussion about the simplest RL approaches is carried out. These include tabular methods applied only considering finite MDPs. Dynamic Programming and Monte Carlo methods are shortly introduced. Then, the key concept of Temporal-Difference Learning will be explained.

2.2.1 Dynamic programming

Dynamic programming (DP) consists in a variety of algorithms used to compute optimal policies. Usually, they can be implemented only when a perfect model of the environment is given. In real control and robotics applications, a perfect environment usually cannot be found and DP may result to be inappropriate or too computationally expensive. Nonetheless, DP can be successfully used in the financial field.

Iterative policy evaluation is one of the possible method in DP literature. It allows to obtain the state-value function v_π given an arbitrary policy π. This algorithm exploits different results that can be obtained combining the equations described in the previous section. Among them, the starting point for policy evaluation is the *Bellman equation*, here used for computing v_π:

$$v_\pi = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')], \ \forall s \in \mathcal{S}.$$

In this equation, the term $\pi(a|s)$ is the probability of choosing an action a in state s under the policy π. For $\gamma < 1$ the existence and uniqueness of v_π are guaranteed. However, an iterative computational procedure is more appropriate for the purposes of RL. Hence, by considering an arbitrary initial v_0, the state-value function can be approximated through successive computational steps using the Bellman equation as an update rule:

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi[r_{t+1} + \gamma v_k(s_{t+1})|s_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[r + \gamma v_k(s')\right], \ \forall s \in \mathcal{S} \end{aligned}$$

Once a value function v_π has been computed, looking for an optimal policy can be a good advancement, since an arbitrary one has been used to compute v_π. The result is a new *greedy* policy called π' that can be obtained combining equations already

seen. Here it is directly reported without discussing the whole derivation. It can be computed with:

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

The greedy policy takes the action that maximizes the action-value function in the short-term. No further details are provided about further possible optimization of π and v_π since the result is out of the scope of this thesis. A deeper analysis of DP can be found at [38].

2.2.2 Monte Carlo Methods

With respect to DP algorithm, Monte Carlo methods are not strictly dependent on the environment's knowledge. In fact, they only require to know the transitions composed of states, actions and rewards obtained through the interaction with a real or a simulated environment. These samples are also called *experience*. An approximate model of an environment is required to simulate the interaction and obtain the sample transitions, but the associated probability distributions are not requested as in DP. Only episodic tasks are considered for Monte Carlo methods. This allows to have well-defined returns that can be easily averaged over all the episodes of the task. Indeed, the approach used for RL problem consists in sampling and averaging returns associated to state-actions pairs.

As first goal, we always aim to estimate the value function of a state s under the policy π, $v_\pi(s)$. A particular state can occur several times inside the same episode. The term *visit* is usually used to refer to the occurrence of the state. Based on this definition, it is possible to identify two main Monte Carlo methods:

- The *first-visit MC method* estimates $v_\pi(s)$ averaging the returns coming after the first visit to s.
- The *every-visit MC method* takes in account the returns following all visits to s for the average.

Both the MC algorithms converge to the value function. The pseudo-code of first-visit MC method is reported in Algorithm 3. The every-visit version is basically the same except for the check of state s_t. Monte Carlo methods allow also to estimate the state-action pair value function $q(s,a)$. More in detail, this is particularly useful

Algorithm 3 First-visit Monte Carlo method for the estimation of $V \approx v_\pi(s)$

Input: a policy π, a positive number of episodes n_e
Output: the estimated value function V

Initialization of returns $R(s) = 0, \forall s \in \mathcal{S}$
Initialization of $N(s) = 0, \forall s \in \mathcal{S}$
▷ N is a counter of the number of visits to each state s
for *episodes* in n_e **do**
    Generate the sequence according to π: $s_0, a_0, r_1, s_1, a_1, r_2, .., s_{T-1}, a_{T-1}, r_T$
    $G \leftarrow 0$
    for each time step of the episode $t = T-1, T-2, ..., 0$ **do**
        $G \leftarrow G + R_{t+1}$
        if *state* s_t *is* ***not*** *present in the sequence* $s_0, ..., s_{t-1}$ **then**
            $R(s_t) \leftarrow R(s_t) + G_t$
            $N(s_t) \leftarrow N(s_t) + 1$
        end if
    end for
end for
$V(s) \leftarrow \frac{R(s)}{N(s)}, \forall s \in \mathcal{S}$

when a model of the environment is not known. In this case the state values are not sufficient to determine the actions associated with highest rewards. It is possible to talk about a *visit* of a state-action pair when the agent takes the action a when it is in the state s. The Monte Carlo methods illustrated before work in the very same way and they can estimate the expected values with an increasing number of visits. However, a problem arises when using a deterministic policy π to choose the actions. In this case many state-action pairs will never be visited. This means that the agent will not choose among all the possible actions associated to a state, which is the basic purpose of learning action values. In other words, we need to keep a sufficient rate of *exploration* in the policy for a better learning process. A possible approach for this problem is called *exploring starts*. It consists in starting the episode from a specific state-action pair and then assign a non-null probability to each possible action. It guarantees a complete exploration of state-action pairs in an infinite number of episodes but it could be practically not feasible.

At this point it is possible to briefly describe how Monte Carlo methods are able to approximate optimal policy for control purposes. The main idea of the procedure is very similar to DP, i.e. the *generalized policy iteration* (GPI) is followed. As graphically shown in Fig. 2.4 the iterative process consists in updating an approxi-

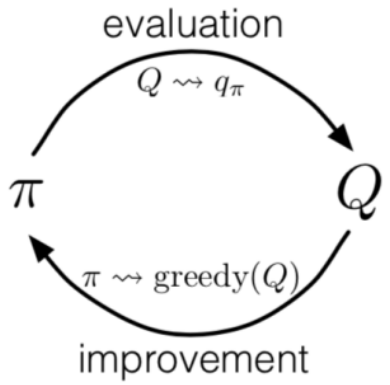


Fig. 2.4 Policy improvement scheme.

mate value function for the current policy, and the policy is modified at each step according to the value function. This adversarial behaviour results in an approximate optimal policy. As seen before, with an action-value function no models are needed. Policy improvement constructs each policy π_{k+1} as the greedy policy based on q_{π_k}. This is a direct application of the *policy improvement theorem*, which state:

$$q_{\pi_k}(s, \pi_{k+1}(s)) \geq v_{\pi_k}(s)$$

Basically, whenever a better policy is found by considering its future returns through the value function, the actions start to be chosen according to π_{k+1} instead of following π_k.

Practical considerations

Two basic assumptions have been considered so far for policy improvements: an infinite number of episodes for the policy evaluation and the usage of exploring starts. For a practical implementation of the method, these have to be removed. Firstly, a limited number of steps is enough to guarantee a good approximation of the solution within certain bounds. Secondly, when the exploring start is not feasible, it is possible to use a particular exploration policy called $\varepsilon - greedy$ policy:

$$\pi(a|s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|}$$

This means that a greedy action is usually picked up from the policy π such that it maximizes the action-value. However, with a probability ε a random action is

selected and it is used to explore non promising state-action pairs that would never be visited otherwise. In practice, an exponential decrement of ε is usually adopted, to decrease the level of random exploration along the learning process.

It is now convenient to introduce the concept of *on-policy* and *off-policy* methods. Basically, on-policy methods aims to maintain and improve the policy used to make decisions. On the contrary, off-policy methods tries to improve a different policy. This difference will be useful to understand the following sections. The main idea behind on-policy Monte Carlo methods are based on GPI. Differently, off-policy methods focus on the usage of two different policies. The first one, that we improve to become the optimal policy and it is called the *target policy*. On the other side, a second policy will be mainly devoted to exploration and it is called *behaviour policy*.

2.2.3 Temporal-Difference Learning

Temporal-difference learning (TD) can be considered the central innovative idea behind RL. TD presents some elements of both Monte Carlo and Dynamic Programming methods. Indeed TD methods are able to learn without the need of a precise model of the environment in the same way of MC. On the other hand, similarly to DP, they bootstrap, i.e. they update estimates using also previously learned quantities. We start introducing the problem of prediction, briefly illustrating how TD estimates the value function v_π for a policy π. A basic every-visit MC uses the obtained return of the visit as target for $V(s_t)$, according to

$$V(s_t) \leftarrow V(s_t) + \alpha \left[G_t - V(s_t) \right],$$

where G_t is the actual return and α is a constant. Differently, TD simplest method makes the update waiting only for the next time step using the obtained reward r_{t+1} and the estimate $V(s_{t+1})$:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]$$

This method is called TD(0) or also *one-step* TD. The quantity $r_{t+1} + \gamma V(s_{t+1})$ represents the target for TD. Moreover, it is possible to define the TD *error* as follows:

$$\delta = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

The advantages of TD with respect to DP and MC are easy to be noticed. The combination of bootstrapping and independence from a model are precious characteristics especially with long episodes. Convergence is usually guaranteed with TD, making these methods convenient in the majority of the situations. Here, the main TD method are introduced and briefly discussed.

2.2.4 SARSA: on-policy TD method

SARSA algorithm takes its name from the quintuple composed by the state-action pair transition sequence $s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}$. Being an on-policy control method, in SARSA the action-value function $q_\pi(s,a)$ is continually estimated for the unique policy π, pushing it toward greediness. The assumption of an infinite number of visits for all state-action pairs ensures the convergence of SARSA algorithm. The pseudo-code of SARSA algorithm is shown below in Algorithm 4.

Algorithm 4 SARSA algorithm for estimating $Q \approx q_*$

Input: a small $\varepsilon > 0$ for ε-greedy policy π, step size $\alpha \in (0,1]$

Output: the estimated action-value function Q

Initialize arbitrarily $Q(s,a), \forall s \in \mathcal{S}^+, a \in \mathcal{A}(s)$,except $Q(terminalstate,) = 0$
for *each episode* **do**
    Initialize s
    Choose a from s using ε-greedy policy
    for each time step of the episode **do**
        Take action a, observe r, s'
        Choose a' from s' using ε-greedy policy
        $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$; $a \leftarrow a'$;
        if s is terminal **then**
            break loop
        end if
    end for
end for

2.2.5 Q-Learning: off-policy TD method

Q-learning is an off-policy TD control method. It can be considered a real innovation in RL. In Q-Learning, the optimal action-value function q_* is approximated by directly learning Q according to:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t) \right]$$

In such a way, the algorithm results to be immediate, as shown in pseudo-code in Algorithm 5.

Algorithm 5 Q-learning algorithm for estimating $\pi \approx \pi_*$

Input: a small $\varepsilon > 0$ for ε-greedy policy π, step size $\alpha \in (0,1]$

Output: the estimated action-value function Q

Initialize arbitrarily $Q(s,a), \forall s \in \mathcal{S}^+, a \in \mathcal{A}(s)$,except $Q(terminalstate,) = 0$
for *each episode* **do**
    Initialize s
    for each time step of the episode **do**
        Choose a from s using ε-greedy policy
        Take action a, observe r, s'
        Choose a' from s'
        $Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$
        $s \leftarrow s'$;
        if s is terminal **then**
            break loop
        end if
    end for
end for

2.3 Deep Reinforcement Learning

The general framework of RL has been introduced in this chapter, and principal RL methods such as Dynamic Programming, Monte Carlo, and Temporal Difference learning have been briefly discussed. To make a further step towards the algorithms employed in the core of the thesis, we need to consider that state spaces can be frequently huge according to the specific task. Hence, tabular methods present strong

limitations due to the cost of updating accurately tables with data in the required time. In this scenario an optimal policy and an optimal value function cannot be found and approximate solutions must be considered according to the available computational resources. What usually happens is that many encountered states will be totally new, and the algorithm should be able to generalize the knowledge that has already learnt in order to make sensible decisions. In practice, it has to learn how to behave correctly on a wide number of situations, experiencing a much smaller subset. The desired function to approximate is usually a value function.

The Deep Reinforcement Learning (DRL) framework is considered at this point. Although several methods exist for function approximation, in this thesis only solutions based on artificial neural networks will be explained. An essential difference in this new framework is that value functions are no more represented using tables. Instead, they are shaped with a parametric functional form. For ANNs the parameters coincide with the weights of the network $\mathbf{w}(\mathbf{w} \in \mathbb{R}^d)$. Since both the state-value and the action-value functions are now dependent on the vector $\mathbf{w}$, they can be written as $\hat{v}(s,\mathbf{w}) \approx v_\pi(s)$ and $\hat{q}(s,a,\mathbf{w}) \approx q_*(s,a)$.

Experience Replay

A popular practice in DRL is the method of *experience replay*. It has been initially studied by Lin in 1992. However, its recent success is mainly related to its application in the DQN algorithm proposed by Mnih et al. [71] (2013) to learn playing ATARI games with DRL. The DQN algorithm will be described later, here we briefly focus on experience replay. The method is based on saving in a memory buffer called *replay memory*, at each time step, the tuple $(s_t, a_t, r_{t+1}, s_{t+1})$. It contains all the information about the transition of the agent from a state s_t to the next one, choosing a certain action a_t and receiving a reward r_t. Once the replay memory reaches a sufficient number of stored transitions, mini-batches can be sampled uniformly at random. Hence, experience is directly used to train the ANN controlling the agent. Experience replay provides several advantages. First of all, it increases the data efficiency of the algorithm, since an experienced event can be used to update the agent's weights multiple times. Another fundamental effect of experience replay is to remove the instability in the learning process caused by temporally correlated training samples. Consecutive samples should be always avoided in RL.

Prioritized Experience Replay

In 2015 Schaul et al. [72] have introduced a new formulation of the experience memory replay buffer which is called Prioritized Experience Replay buffer (PER). It is based on the theory that some experiences are more instructive than others. For example, situations in which the agent made a significant error or received a large reward may be more instructive than other events. The agent can learn faster if certain events receive priority in the learning process. PER can be put into practice by giving each experience a priority value determined by an index of relevance. One way to measure relevance is to look at the TD-error or the absolute error between the target Q-value and the estimated Q-value. During the learning phase, experiences with higher priority values are more likely to be sampled. PER has the potential to significantly boost RL algorithms' sample-efficiency. It has been demonstrated that it can improve the agent's overall performance and accelerate learning's convergence and stability, especially with sparse reward and dynamic environments.

2.3.1 Deep Q-Learning algorithm

Deep Q-Learning algorithm is an advanced version of Q-learning method. Similarly, this method focuses on the approximation of the optimal action-value function $Q^*(s,a)$. The definition of optimal action-value function remains the same:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[r_t | s_t = s, a_t = a, \pi]$$

As explained in the previous sections, it can be exactly computed thanks to the Bellman equation for action-value function:

$$Q^*(s,a) = \mathbb{E}\left[r_{t+1} + \gamma \max_{a'} Q^*(s',a') | s,a \right]$$

Practically, an ANN is used as non-linear function approximator to obtain $Q(s,a;\theta) \approx Q^*(s,a)$, where θ is used to indicate the weights of the network. In this particular case, the neural network is often called *Q-networks* or *Deep Q-network* (DQN), due to its purpose. The Q-network is usually trained with stochastic gradient descent over a set of transitions (s,a,r,s') collected in the replay buffer $\mathcal{D}$. The training process aims to minimize the loss function $L_Q(\theta)$ at each time step:

$$L_Q(\theta) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a,\theta_i))^2 \right)],$$

where y_i is the target at step i

$$y_i = \mathbb{E}_{s' \sim P}\left[r + \gamma \max_{a'} Q(s', a'; \theta) | s, a \right] \tag{2.1}$$

and $\rho(s,a)$ is the probability distribution over the action given the state, i.e. the distribution of the behaviour policy used to sample the action, and P is the transition probability of the environment. This loss is basically a mean-squared Bellman error (MSBE) function, which indicates how close the predicted Q-value $Q(s,a,\theta_i)$ satisfies the Bellman equation. The action is then obtained by the trained DQN selecting the one that maximizes the estimated maximum Q-value:

$$a = \arg\max_{a \in \mathcal{A}} Q(s,a;\theta) \tag{2.2}$$

The pseudo-code of the algorithm is reported below in Algorithm 6. Deep Q-learning is a model-free algorithm, since it does not need to know the probability distribution of the environment dynamics. Moreover, it is an off-policy method, because it uses a target greedy policy for the optimization of the action-value function, together with an ε-greedy policy for the behaviour distribution $\rho(s,a)$. It also exploits the experience replay, training the networks with mini-batches sampled from the a total of N transitions stored in the memory buffer $\mathcal{D}$. The replay buffer should be defined large enough to contain various previous experience and avoid overfitting on most recent transitions, balancing the its size at the cost of learning speed.

Algorithm 6 Deep Q-learning algorithm with experience replay

Input: a small $\varepsilon > 0$ for exploratory ε-greedy policy, replay memory D
Output: the function approximator of the action-value function Q

Initialize the replay memory $\mathcal{D}$
Initialize the Q function approximator with random weights
for each episode **do**
    Initialize s
    for each time step $t = 1, T$ **do**
        Generate a random number $0 \leq h \leq 1$
        if $h \leq \varepsilon$ **then**
            Pick a random action a_t
        else
            Select $a_t = \arg\max_a Q(s_t, a; \theta)$
        end if
        Perform selected action in the environment and observe r_t, s_{t+1}
        Set next state $s_{t+1} = s_t$
        Store experience transition (s_t, a_t, r_t, s_{t+1}) in replay memory $\mathcal{D}$
        Sample mini-batch of transitions (s, a, r, s') from replay memory $\mathcal{D}$
        Set target: $y = \begin{cases} r \text{ if final state} \\ r + \gamma \max_{a'} Q(s', a'; \theta) \text{ otherwise} \end{cases}$
        Perform a gradient descent step on loss $(y - Q(s, a, \theta))^2$
    end for
end for

Target Network

A problematic source of instability is that the expression of the target y_i depends on the same network's parameters θ we are trying to optimize. This makes the loss minimization unstable. For this reason, Mnih et al. [71] suggested to use another network called *target network* to break the dependence of the target expression on the weights θ. The target network is updated with a certain delay with respect to the original one. This improves convergence of the algorithm when using TD-error.

2.3.2 Actor-Critic architecture

A key concept for the algorithms used in this thesis is the actor-critic architecture. Differently from the classic MDP agent, this DRL framework involves the usage of two separate entities. The *actor* is responsible of selecting the action, hence it represents the function approximator of the policy. The *critic* evaluates the goodness

of what the actor has decided to do. For this reason, it usually approximates an action-value function $Q(s,a,\theta)$ using the TD error. The critic loss is therefore based on the TD error, while the actor network will be updated according to Policy Gradient algorithm, i.e. exploiting a gradient computed from the critic's prediction. In practice, the critic tries to indicate to the actor which are good or bad choices. A schematic is reported in Fig. 2.5 for a better visualization. During the training process, both the actor and critic networks are updated. However, the actor will be the only entity employed in the desired task, the presence of the critic is limited to the training phase.

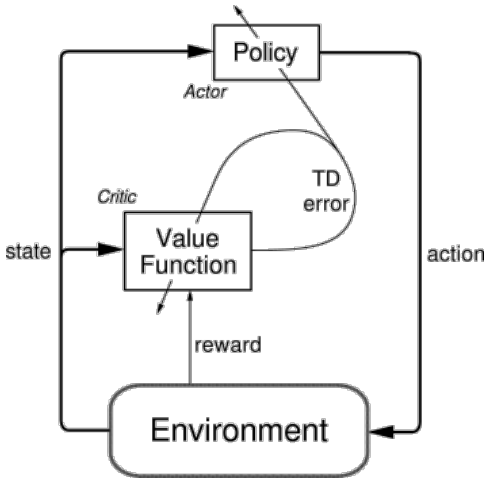


Fig. 2.5 Actor-Critic architecture scheme.

2.3.3 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is a model-free off-policy actor-critic algorithm. The DDPG algorithm was originally proposed by Lillicrap et al. in 2015 [73], mixing the principle of Deterministic Policy Gradient (DPG) [74] methods with Deep Q-learning. From a broader perspective, DDPG can be considered an extension of DQN from discrete to continuous action space. This is not a trivial passage considering how to compute the maximum Q-value over actions $\max_a Q^*(s,a)$. With discrete action space, it is easy to compute the Q-value for each action in the set and compare them directly. Instead, DDPG deals with the continuous space leveraging the actor-critic architecture to directly approximate $\max_a Q^*(s,a) \approx Q(s,\mu(s))$, where $\mu(s)$ is the deterministic policy to be learned. Hence, a gradient-based rule is derived directly relating the function $Q^*(s,a)$ to the action predicted. The actor-critic

architecture enables this mechanism training two networks concurrently: the actor network approximates the deterministic policy μ_θ with weights θ, and the critic one the $Q^*(s,a)$ with Q_ϕ. Similarly to DQN, experience replay and target networks are used also in DDPG, hence we will have target networks $\mu_{\theta_{targ}}$ and $Q_{\phi_{targ}}$ for both actor and critic.

Critic update: training the critic represents the Deep Q-learning nature of DDPG. The loss function is indeed defined with a mean-squared Bellman error (MSBE) computed over a batch B of transitions sampled from the replay buffer $\mathcal{D}$:

$$L(Q_\phi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[(Q_\phi(s_t,a_t) - y_t)^2\right] \tag{2.3}$$

where y_t is the target computed with actor's prediction,

$$y_t = r(s_t,a_t) + \gamma Q_{\phi_{targ}}(s_{t+1}, \mu_{\theta_{targ}}(s_{t+1})) \tag{2.4}$$

The critic Q-network is updated with a gradient descent step using the gradient obtained differentiating the MSBE over the weights ϕ.

Actor update: the goal of the actor training is to maximize the Q-function estimated by the critic. The gradient to update the deterministic policy network can be obtained differentiating the Q-value with respect to the actor's weights θ,

$$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} Q_\phi(s, \mu_\theta(s)) \tag{2.5}$$

The actor's θ weights are then updated with a gradient ascent step. A soft update is then performed as follows for the two target networks using τ as temperature parameter:

$$\phi_{targ} \leftarrow \tau\phi + (1-\tau)\phi_{targ}$$

$$\theta_{targ} \leftarrow \tau\theta + (1-\tau)\theta_{targ}$$

Moreover, in the original implementation of the paper by Lillicrap et al., exploration is carried out by adding to the actor policy a noise $\mathcal{N}$, generated with a particular process (Ornstein-Uhlenbeck). A detailed explanation is avoided here, the resulting

expression of the policy is shown below.

$$\mu(s_t) = \mu_\theta(s_t|\theta) + \mathcal{N}$$

The resulting action value is then clipped in the allowed ranges $[a_{min}, a_{max}]$.

2.3.4 TD3

The Twin Delayed Deep Deterministic Policy Gradient (TD3) [75] is a more recent extension of DDPG designed to improve the stability and performance of the actor-critic algorithm. Indeed, DDPG results to be brittle with respect to hyperparameters and tuning process. A typical issue in the DDPG convergence is caused by the overestimation of the Q-value. Similar to DDPG, TD3 trains an actor policy network $\mu_\theta(s)$ and its target copy. However, TD3 employs some tricks to boost the convergence of the learning:

- Clipped Double-Q Learning: TD3 uses two "twins" critic networks $Q_{\phi_1}(s,a)$ and $Q_{\phi_2}(s,a)$, and it picks up the minimum of the two estimated Q-values to compute the targets in the Bellman error loss;
- Delayed Policy Updates: TD3 updates the policy and the target networks less frequently than the Q-function;
- Target policy smoothing: TD3 adds noise to the target action used to compute the Q-learning target, to regularize the learning of the algorithm.

Critic Update: The critic loss is the same MSBE computed for the two Q-functions using a unique minimum target

$$y_t = r(s_t, a_t) + \gamma \min_{i=1,2} Q_{\phi_{i,targ}}(s_{t+1}, \tilde{\mu}_{\theta_{targ}}(s_{t+1})) \tag{2.6}$$

$$L(Q_{\phi_i}) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}} \left[\left(Q_{\phi_i}(s_t, a_t) - y_t \right)^2 \right] \tag{2.7}$$

Notice that in the target computation TD3 adopts a noisy target policy

$$\tilde{\mu}_{\theta_{targ}} = \mu_{\theta_{targ}}(s_{t+1}) + \mathcal{N}_t \tag{2.8}$$

$\mathcal{N}_t$ is the clipped target policy smoothing noise. The resulting action is also clipped in the allowed ranges $[a_{min}, a_{max}]$. The noise is applied to regularize the algorithm preventing overestimation of the Q-function.

Actor Update: TD3 updates the actor in the same way of DDPG, computing a gradient using the first critic Q-value Q_{ϕ_1}:

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s)) \tag{2.9}$$

The actor parameters θ are updated by performing gradient ascent on $L(\theta)$. However, TD3 updates the actor policy and the target networks less frequently than the Q-functions. This acts as a further regularization action preventing continuous sudden changes of the target due to policy updates. The pseudo-code of TD3 is reported in Algorithm 7.

Algorithm 7 Twin Delayed Deep Deterministic Policy (TD3)

Initialize actor network μ_θ and two Q-networks Q_{ϕ_1}, Q_{ϕ_2}
Initialize target networks $Q_{1_{targ}}$, $Q_{2_{targ}}$, and μ_{targ} with the same weights
Initialize target policy smoothing noise $\varepsilon \sim \mathcal{N}(0, \sigma)$
Initialize replay buffer $\mathcal{D}$
for each episode **do**
    Initialize state s
    for each step in the episode **do**
        Select action a from the current policy with noise: $a = \text{clip}(\mu(s) + \varepsilon, a_{min}, a_{max})$
        Execute action a, observe reward r and next state s'
        Store (s, a, r, s') in $\mathcal{D}$
        if update step **then**
            for number of updates **do**
                Sample a mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
                Compute target actions with policy smoothing noise:
                $\tilde{a} = \text{clip}((\mu_{\theta_{targ}}(s') + \text{clip}(\varepsilon, -c, c), a_{min}, a_{max}))$
                Compute target Q-values:
                $y = r(s,a) + \gamma \min_{i=1,2} Q_{\phi_{i,targ}}(s', \tilde{a})$
                Update Q-networks minimizing the loss:
                $L(Q_{\phi_i}) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[(Q_{\phi_i}(s,a) - y)^2 \right]$
                if policy update step **then**
                    Update the actor network with gradient ascent step:
                    $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$
                    Update target networks with soft updates:
                    $\phi_{Q_{1_{targ}}} = \tau \phi_1 + (1 - \tau) \phi_{Q_{1_{targ}}}$
                    $\phi_{Q_{2_{targ}}} = \tau \phi_2 + (1 - \tau) \phi_{Q_{2_{targ}}}$
                    $\theta_{targ} = \tau \theta + (1 - \tau) \theta_{targ}$
                end if
            end for
        end if
    end for
end for

2.3.5 Soft Actor-Critic

Soft Actor-Critic (SAC) is a model-free off-policy DRL algorithm that optimizes a stochastic policy. The core concept of SAC is entropy regularization, it finds an optimal trade-off between entropy of the policy and expected cumulative return. Entropy $H(P) = \mathbb{E}_{x\sim P}[-\log P(x)]$ is a way to measure the randomness of the policy. In this case, we have a natural regulation of exploration and exploitation without adding externally generated noise into actions. SAC was introduced by [76], here the version for continuous action spaces is presented. SAC maintains a stochastic policy $\pi_\theta(a|s)$ and two value functions: $Q_{\phi_1}(s,a)$ and $Q_{\phi_2}(s,a)$. Hence, SAC adopts the same clipped double Q-value of TD3, with some small differences in the target computation and policy update. In an entropy-regularized RL setting, the optimal policy π^*_θ is obtained maximizing a discounted term which also includes the entropy term $H(\pi(\cdot|s_t))$:

$$\pi^*_\theta = \arg\max_\pi \mathbb{E} \sum_{t=0}^{\infty} \gamma^t [r_t + \alpha H(\pi(\cdot|s_t))] \tag{2.10}$$

Where α is the temperature parameter which regulates the trade-off between return optimization and policy stochasticity. The stochastic policy is approximated in SAC algorithm with a neural network regressor through a reparametrization trick. To do so, the output of the network are the the mean μ_θ and standard deviation σ_θ parameters of a Gaussian that we use to sample actions. Then, action are squashed with a *tanh* function to bound it in a finite range.

$$\tilde{a}_\theta(s,\xi) = tanh(\mu_\theta(s) + \sigma_\theta(s) \odot \xi), \quad \xi \sim \mathcal{N}(0,I). \tag{2.11}$$

Critic Update: The critic networks are updated to minimize the mean squared Bellman error

$$L(Q_{\phi_i}) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}} \left[(Q_{\phi_i}(s_t,a_t) - y_t)^2 \right], \tag{2.12}$$

with the entropy-regularized target Q-value given by

$$y_t = r(s_t,a_t) + \gamma \left(\min_{i=1,2} Q_{\phi_{i,targ}}(s_{t+1}, \tilde{a}_{t+1}) - \alpha \log \pi_\theta(\tilde{a}_{t+1}|s_{t+1}) \right), \quad \tilde{a}_{t+1} \sim \pi_\theta(\cdot|s_{t+1}) \tag{2.13}$$

Actor Update: The actor is trained to maximize both expected return and entropy, encouraging exploration. The objective function for the actor is:

$$L(\theta) = \mathbb{E}_{s_t \sim D}\left[\mathbb{E}_{a_t \sim \pi(\cdot|s_t)}\left[\min_{i=1,2} Q_{\phi_i}(s_t, a_t) - \alpha \log(\pi_\theta(a_t|s_t))\right]\right] \tag{2.14}$$

where the temperature parameter α can be a static value, or fine tuned with an update rule along the course of the training. Differently from TD3, SAC uses the minimum Q-value instead of the first one in the actor loss. Algorithm 8 report the pseudo-code of SAC.

Algorithm 8 Soft Actor-Critic (SAC)

Initialize critic networks Q_{ϕ_1}, Q_{ϕ_2}, and actor network π_θ with random weights
Initialize target networks $Q_{\phi_{1,targ}}$, $Q_{\phi_{2,targ}}$, and $\pi_{\theta_{targ}}$ with same weights
Initialize replay buffer $\mathcal{D}$ and temperature parameter α
for each episode **do**
    Initialize state s
    for each step in the episode **do**
        Select action a from the current policy: $a \sim \pi_\theta(\cdot|s)$
        Execute action a, observe reward r and next state s'
        Store (s, a, r, s') in $\mathcal{D}$
        if update step **then**
            for number of updates **do**
                Sample a mini-batch B of transitions (s, a, r, s') from $\mathcal{D}$
                Compute target Q-values:
                $y = r + \gamma \min_{j=1,2} Q_{\phi_{i,targ}}(s', \tilde{a}') - \alpha \log(\pi(\tilde{a}'|s')), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$
                Update critic networks by minimizing the mean squared loss:
                $L(Q_{\phi_i}) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}}\left[(Q_{\phi_i}(s, a) - y)^2\right]$
                Update the actor network:

$$\nabla_\phi \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}) - \alpha \log \pi_\theta(\tilde{a}|s) \right)$$

                Update target networks with soft updates:
                $\phi_{1_{targ}} = \tau \phi_1 + (1 - \tau)\phi_{1_{targ}}$
                $\phi_{2_{targ}} = \tau \phi_2 + (1 - \tau)\phi_{2_{targ}}$
            end for
        end if
    end for
end for

Chapter 3

Autonomous Navigation of Mobile Robots

3.1 Autonomous navigation: localization, planning and control

In robotics, navigation describes the capability of mobile robots to move from the starting point to a specified destination by choosing a viable path made up of a series of configurations. Autonomous navigation in a given environment is a fundamental and difficult robotics matter. Numerous methods have been created, which also allows the systems that are now in place to be categorized based on their primary characteristics. Generally, three primary components of a navigation system can be identified:

1. a *localization system* to identify the robot position and orientation with respect to a reference frame;
2. a *path planner* to compute a suitable sequence of configurations for the robot to reach the goal in the map;
3. a *motion controller* to select the actions for the robot to make it follow the desired computed trajectory. Commands are typically expressed as velocity pairs, or throttle plus angle.

Given the target position, the robot should be able to localize itself and plan a suitable path in the environment. Then, the motion controller, or local planner, will be responsible of moving the robot generating commands that fit the points on the global trajectory, and, at the same time, handle obstacles not present in the map. Fig. 3.1 shows a schematic flow diagram of a navigation system with pre-built map. Regarding this last point, navigation tasks can be classified as map-based or mapless. Before completing the localization and path planning tasks, most conventional approaches require mapping the environment where the robot moves. Simultaneous Localization and Mapping (SLAM) [77, 78] is a fundamental technique for localization that enables the concurrent construction of the environment map. However, creating an accurate map is frequently difficult in a variety of real-world situations. In addition, intricate settings with dynamic obstacles, most notably people, remain a barrier for conventional local controller techniques. Two important characteristics of an autonomous navigation system are computational costs and flexibility, i.e. the ability to generalize the performance regardless of the commands designed by the programmers. To this extent, learning methods and

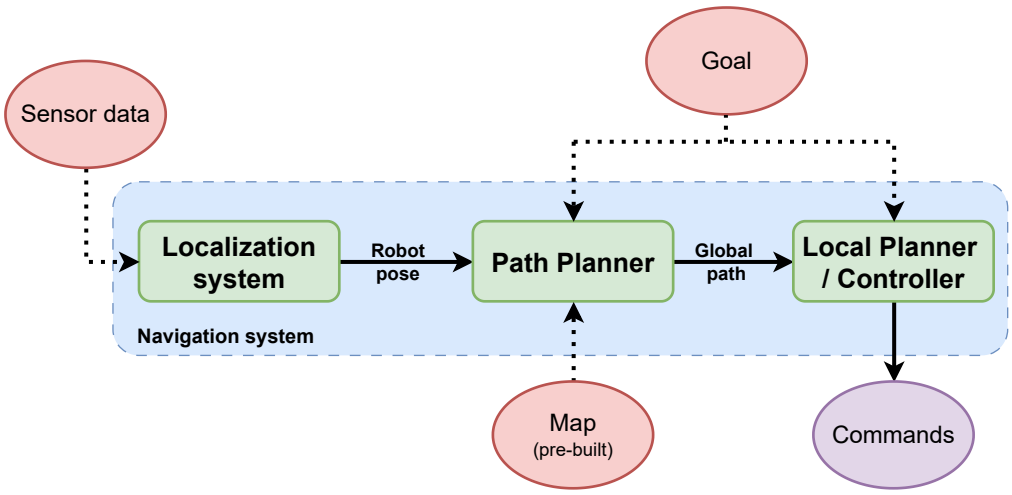


Fig. 3.1 Schematic diagram of a navigation system.

AI-driven navigation systems could be an interesting mapless solution to increase both flexibility and autonomy. In the next sections of the chapter a selection of classic localization and local planning approaches is briefly explained, choosing the ones that are directly used or heavily cited in the following chapters of the thesis.

3.2 Localization Approaches

Localization is the first necessary phase of autonomous navigation. In this step, the robot needs to estimate its current position based on motion commands and sensor data. The probabilistic estimate of the robot state in a time instant is usually indicated as *belief*. Based on the initial conditions available for the robot, it is possible to define the following taxonomy of the localization problem:

1. **Local position tracking**: the initial pose of the robot is known and the robot has to track its position while moving in the map. The previous state, commands and odometry data are usually enough to carry our position tracking. However, uncertainty in the belief estimation may increase during time if other sensor measurements are not used. The belief is modeled as a unimodal Gaussian distribution in this case.

2. **Global localization**: the initial pose of the robot in the environment is not known, hence a multimodal belief should be used since there exist multiple possible poses of the robot. In this case, sensor data are necessary for solving the state estimation problem.

3. **Kidnapped robot**: an extreme case of global localization in which the robot may wrongly believe to be placed in a certain pose, this happen for example when it is physically teleported to another position. Truly autonomous robots should be able to recover from kidnap conditions using sensors for place recognition in the map.

There exist many localization approaches in literature, ranging from probabilistic algorithms to map-based and radio frequency identification (RFID) approaches [79]. In this dissertation we only make reference to position tracking problems, hence the most common Extended Kalman Filter localization is briefly described.

3.2.1 Kalman Filter Localization

The Kalman Filter (KF) is one of the most popular mathematical tools for stochastic estimation from noisy sensor measurements. The KF is essentially a set of mathematical equations that implement a predictor-corrector type estimator. It is designed to manage discrete-time controlled processes that are governed by linear stochastic difference equations, with the further assumption of Gaussian belief distribution.

The multivariate normal distribution for k-dimensional random variables $\mathbf{x}$ can be expressed as:

$$p(\mathbf{x}) = \frac{1}{\sqrt[k]{2\pi}|\sqrt{\Sigma}|} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right) \tag{3.1}$$

where μ is the mean vector and Σ the covariance matrix of the Gaussian. However, real-world and robotics processes are usually described by non-linear equations, breaking the linearity assumption of the KF. To tackle this problem, linearization about the current mean and covariance can be performed. The Extended Kalman Filter (EKF) is the most common algorithm obtained with this principle, using the Taylor expansion linearization.

Theoretical Overview

The general problem of robot localization consists in the estimation of the robot state $\mathbf{x} \in \mathbf{R}^n$. The standard process of robot's position estimation and filtering is composed of two main steps [80, 81]:

- *prediction or action update*: the belief of the robot's position is estimated based on the previous knowledge and the last action given;
- *correction or measurement update*: the predicted belief is corrected according to the information extracted from perception data.

The belief prediction and correction steps can be described for discrete states by the equations:

$$\begin{aligned} \overline{bel}(x_k) &= \sum_{x_{k-1}} p(x_k|u_k, x_{k-1}) bel(x_{k-1}) \\ bel(x_k) &= \eta\, p(z_k|x_k)\overline{bel}(x_k) \end{aligned} \tag{3.2}$$

where $\overline{bel}(x_k)$ is the predicted belief, u_k the command signal and η a normalization factor. The corrected belief $bel(x_k)$ is obtained applying the Bayes rule. The discrete-time stochastic evolution of the state $\mathbf{x} \in \mathbf{R}^n$ is described by a non-linear function $\mathbf{g}()$, also called *transition model* dependent on the previous state x_{k-1}, the command received u_k and the process noise w_{k-1}:

$$x_k = g(x_{k-1}, u_k) + w_{k-1} \tag{3.3}$$

Differently, the non-linear function $\mathbf{h}()$ describes the *measurement model* according to the specific set of sensor data used in the perception stage. The measurement $\mathbf{z} \in \mathbf{R}^m$ is therefore obtained knowing the actual state estimate by:

$$z_k = h(x_k) + v_k \tag{3.4}$$

where the random variable v_k represents the measurement noise. The process and measurement noises are assumed to be independent of each other, white, and with normal probability distribution.

$$\begin{aligned} p(w) &\sim \mathcal{N}(0, \mathbf{R}) \\ p(v) &\sim \mathcal{N}(0, \mathbf{Q}) \end{aligned} \tag{3.5}$$

$\mathbf{R}$ and $\mathbf{Q}$ represent the process noise covariance matrix and the measurement noise covariance matrix.

In the EKF the transition and measurement model are approximated using the Taylor linearization for multi-dimensional vectors. The resulting linearized models can be expressed as follow:

$$\begin{aligned} g(x_{k-1}, u_k) &\approx g(\mu_{k-1}, u_k) + \mathbf{G}_k (x_{k-1} - \mu_{k-1}) \\ h(x_k) &\approx h(\overline{\mu}_k) + \mathbf{H}_k (x_k - \mu_{k-1}) \end{aligned} \tag{3.6}$$

Where

- x_k and z_k are the actual state and measurement vectors,
- $\overline{\mu_k}$ and μ_k are the predicted and corrected estimate of the state at time k
- $\mathbf{G}$ is the Jacobian matrix of $g()$ with respect to the state vector x, whose (i,j)-th entry is

$$\mathbf{G_{i,j}} = \frac{\partial g_i(\mu_{k-1},\ u_k,)}{\partial x_j} \tag{3.7}$$

- $\mathbf{V}$ is the Jacobian matrix of $g()$ with respect to the command input u, whose (i,j)-th entry is

$$\mathbf{V_{i,j}} = \frac{\partial g_i(\mu_{k-1},\ u_k,)}{\partial u_j} \tag{3.8}$$

- **H** is the Jacobian matrix of h with respect to the state vector x, whose (i,j)th entry is

$$\mathbf{H_{i,j}} = \frac{\partial h_i(\overline{\mu}_k,)}{\partial x_j} \tag{3.9}$$

Table 3.1 EKF equations for prediction and correction steps.

EKF Prediction
1) Predict the new mean vector of the state
$$\overline{\mu}_k = g\ (\mu_{k-1},\ u_k,) \tag{3.10}$$
2) Predict the new uncertainty covariance matrix
$$\overline{\mathbf{P}}_\mathbf{k} = \mathbf{G_k P_{k-1} G_k^T} + \mathbf{W_k R_{k-1} W_k^T} \tag{3.11}$$
EKF Correction
1) Compute Kalman Gain **K**
$$\mathbf{Z_k} = \mathbf{H_k}\overline{\mathbf{P}}_\mathbf{k}\mathbf{H_k^T} + \mathbf{Q_k} \tag{3.12}$$
$$\mathbf{K_k} = \overline{\mathbf{P}}_\mathbf{k}\mathbf{H_k^T Z_k^{-1}} \tag{3.13}$$
2) Correct estimate with measurement z_k
$$\mu_k = \overline{\mu}_k + \mathbf{K}(z_k - h(\overline{\mu_k})) \tag{3.14}$$
3) Correct the uncertainty covariance with measurement innovation
$$\mathbf{P_k} = (\mathbf{I} - \mathbf{K_k H_k})\overline{\mathbf{P}}_\mathbf{k} \tag{3.15}$$

Jacobians are necessary to map the covariance matrix **P** into the correct linearized space. The same is done for the covariance matrix **Z** associated to the measurement's prediction. The equations of the EKF are reported in Table 3.1. Further details can be found in the book [82].

3.2.2 Robot model

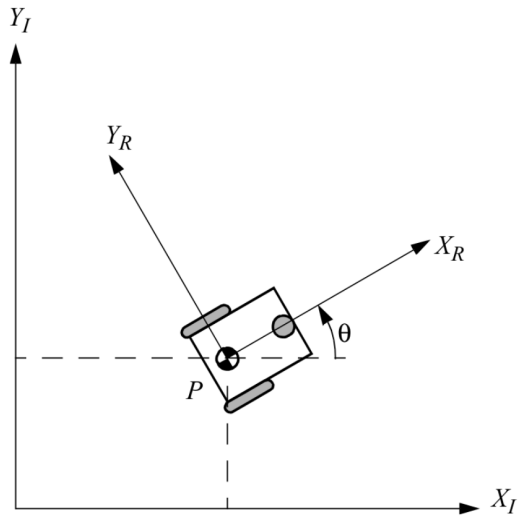


Fig. 3.2 Model of a differential drive robot. Image from [81].

The robot model proposed in this section is based on a differential drive robot. Differential drive is one of the most popular kinematics architectures for mobile robots, especially for indoor applications. Many experimental tests in this thesis have been conducted using this kind of robots. Fig. 3.2 shows a model of differential robot and its reference frames. The global inertial reference frame is defined by X_I and Y_I, while X_R and Y_R defines the local mobile reference frame of the robot. The states that describe the robot's pose in 2D space in the global frame can be expressed as $[x, y, \theta]$, where, x and y are its cartesian coordinates on the ground plane, and θ its rotation angle about the vertical axis.

$$\mathcal{R}_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} \tag{3.16}$$

A differential drive robot spin two wheels at a different rotational velocity. The robot motion can be modeled as a unicycle with constant velocity inputs v_k and ω_k during the sampling interval $T_s = [t_k, t_{k+1}]$. In this interval, the robot follows a circular trajectory of radius $\frac{v_k}{\omega_k}$.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.17}$$

where (v, w) is the velocity control input. The unicycle model reported below is equivalent to the differential drive as long as we are only interested in the position of the platform given the overall control input (v, w) instead of the specific wheels velocity $(\dot{\phi}_R, \dot{\phi}_L)$:

$$\begin{aligned} \dot{\phi}_R &= \omega(r + d/2) \\ \dot{\phi}_L &= \omega(r - d/2) \end{aligned} \tag{3.18}$$

The platform displacement and rotation in the time interval T_s can be expressed as:

$$\begin{aligned} \Delta s &= \frac{r(\Delta\phi_R + \Delta\phi_L)}{2} \\ \Delta\theta &= \frac{r(\Delta\phi_R + \Delta\phi_L)}{d} \end{aligned} \tag{3.19}$$

where r is the radius of the wheel and d is the wheelbase, $\Delta\phi_i$ is the angular rotation of each wheel.

Given as known the previous state of the robot $\mathcal{R}_k$ and the velocities v_k and ω_k, the next state $\mathcal{R}_{k+1}$ can be computed by integration over the time interval $[t_k, t_{k+1}]$. Different techniques can be used for the discrete integration, considering that the trade-off should be made between precision and computational complexity.

$$g(x_k, u_k) = \begin{cases} x_{k+1} = & x_k + \frac{v_k}{\omega_k}(\sin\theta_{k+1} - \sin\theta_k) \\ y_{k+1} = & y_k + \frac{v_k}{\omega_k}(\cos\theta_{k+1} - \cos\theta_k) \\ \theta_{k+1} = & \theta_k + \omega_k T_s \end{cases} \tag{3.20}$$

A more precise model can be obtained using the Runge-Kutta integration method, which consider the average value of the angle θ in the short time interval T_s.

$$g(x_k, u_k) = \begin{cases} x_{k+1} = & x_k + v_k T_s \cos\left(\theta_k + \frac{\omega_k T_s}{2}\right) \\ y_{k+1} = & y_k + v_k T_s \sin\left(\theta_k + \frac{\omega_k T_s}{2}\right) \\ \theta_{k+1} = & \theta_k + \omega_k T_s \end{cases} \tag{3.21}$$

3.3 Local Planning: Dynamic Window Approach

Dynamic Window Approach (DWA) is a local motion planning algorithm commonly used in mobile robotics. The original paper was presented in 1997 [83] and, despite some improvement in terms of additional cost function introduced by most recent version (DWB)[84], it still holds as a milestone of local planning. DWA is based on the concept of dynamic window, i.e. it selects velocity commands (v, w) for a differential robot among the ones that can be reached in the next time step according to the acceleration limits of the roobot. Overall, DWA optimizes a cost function with the twofold objective of guaranteeing a collision-free and kinematically feasible trajectory generation.

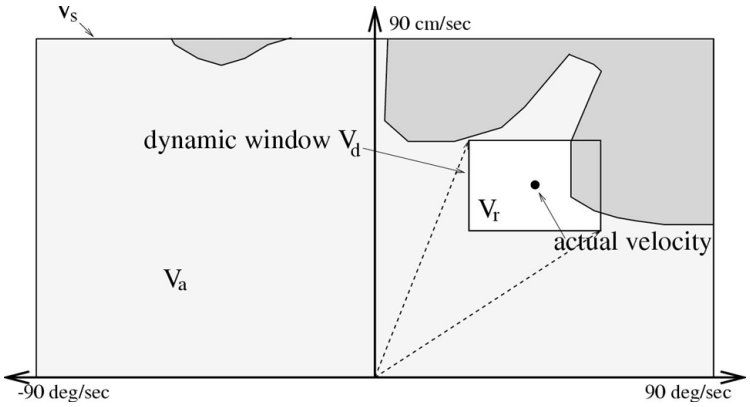


Fig. 3.3 The velocity search space in Dynamic Window Approach: it is limited to the admissible (collision-free) and reachable velocities. [83]

3.3.1 Velocity search space

The velocity search space and the dynamic window are shown in Fig. 3.3, taken directly from the paper. The dynamic window approach considers only circular trajectories uniquely determined assuming the pair (v, ω) of translational and rotational velocities constant for a small time step dt. This results in a two-dimensional velocity search space.

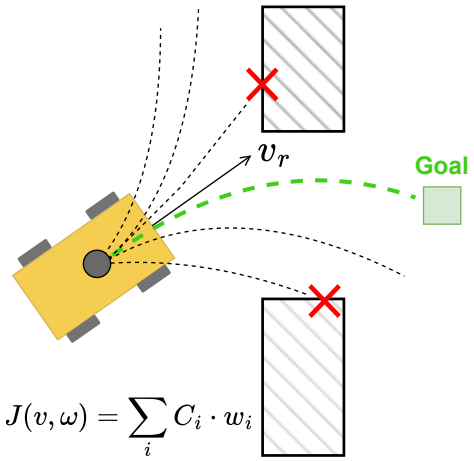


Fig. 3.4 The Dynamic Window Approach evaluates trajectories obtained by simulating admissible velocities for a small time interval, using a cost function $J(v,\omega)$ composed of a weighted sum of different costs.

The search space of the possible velocities is then reduced only considering:

1. **Admissible velocities**: only safe, collision-free trajectories are considered. A pair (v,ω) is considered admissible, if the robot is able to stop before it reaches the closest obstacle on the corresponding curvature.
2. **Reachable Velocities** (Dynamic window): the velocities that can be reached within a short time interval dt given the limited accelerations of the robot, respecting the kinematic limit of the platform.

3.3.2 Optimization

The candidate velocity pairs are then used to simulate the resulting trajectories for a given simulation time interval, as depicted in Fig. 3.4. An objective function is then used to score the obtained trajectories:

$$J(v,\omega) = \sigma\left(\alpha \cdot heading + \beta \cdot dist + \gamma \cdot vel\right) \tag{3.22}$$

This function trades off the following aspects:

1. **Target heading**: *heading* is a metric to indicate that the robot is moving towards the goal.

2. **Obstacles**: *dist* is the distance to the closest obstacle on the trajectory. This term should be maximized for a safe command selection.

3. **Velocity**: *vel* is the forward velocity of the robot, faster motion allows for reduced traversal time to reach the goal.

The function σ smooths the weighted sum of the three components. More complex cost function can be designed to describe advanced target behaviours. For example, following an available global path while avoiding obstacles. The standard DWA presented some critic aspect when dealing with velocity regulation in sharp angles or narrow passages. Dynamic obstacles also require for a high reactivity and re-planning capability. Despite that, still many recent local planners leverage the principle of the DWA. For example, a new implementation called DWB present in the open-source navigation framework of ROS 2 [84] improves the limitation of the classic DWA by adopting a more detailed and rich cost function, together with inserting some smoothing sub-policy to mitigate the overall navigation quality. However, there is still space to improve the flexibility and reliability of autonomous navigation algorithms for mobile robots. Exploring new solutions and approaches to challenging application contexts is one of the core objectives of this thesis, relying on novel Machine Learning methods.

Part II

Autonomous Robots for Indoor Social Assistance

Chapter 4

Adaptive social navigation with Deep Reinforcement Learning

In recent years, service robots have emerged as a promising automation solution in various social contexts, ranging from domestic assistance [15, 14] to healthcare [10]. These advancements have opened up new avenues for robotics research, particularly in human-aware navigation. The robotics community has proposed different benchmarks to evaluate the existing social navigation algorithms [85, 86]. However, social navigation is a complex problem that poses contrasting objectives and is often difficult to formulate with an analytical expression, as is usually done in classic navigation cost functions. This complexity arises from the intricate dynamics of human behavior and the multitude of social rules not considered in standard path planning. Different social navigation scenarios have been partially categorized in the literature to build consistent research and benchmarks [87, 88], highlighting unique challenges in each situation. Standard social planners struggle to perform properly in all of them, considering that environmental geometry and features are crucial in constraining navigation in cluttered, narrow passages or wide open spaces. Therefore, the diversity and unpredictability of social scenarios necessitate a more flexible and adaptive approach.

In this context, Machine Learning (ML) techniques represent a potential solution to this problem. ML models can leverage data to learn behaviors that enhance mobile robots' adaptability to new situations[89] without being explicitly programmed for a specific task. Diverse end-to-end learning approaches [90, 91] have been directly

applied to make navigation control adaptive. Deep Learning can also select the most suitable social navigation strategy according to the context, as done in [92], which provides the robot with adaptive behavior. Among existing ML paradigms, Deep Reinforcement Learning (DRL) is particularly suited for learning behavioral policies and, hence, for navigation [93]. On the other hand, end-to-end learning approaches may often present weaker performance in unseen testing conditions. The authors in [94] proposed a precious comparison between end-to-end and parameter-learning approaches, highlighting the improved performance of hybrid solutions combining standard controllers and learning. For example, a DRL approach is employed in [95] to evaluate the projected trajectories of the classical Dynamic Windows Approach local planner (DWA), learning a reward function for navigating in dynamic environments. The parameter-learning presented in the family of APPL approaches (Adaptive Planner Parameter Learning) [96] results in a promising direction to convey robustness and versatility in a unique solution [94]. APPL aims to learn a parameter management policy that can dynamically adjust the hyper-parameters of classical navigation algorithms according to the environment geometry. The authors proposed different ML techniques, including RL [97]. However, the adaptive parameter approach is applied only in static environments. Finally, an adaptive DWA implementation has been proposed in [98], dynamically changing the basic cost terms of the algorithm with a Q-table approach.

In this chapter an adaptive parameter-learning approach for social navigation is proposed. This study is an improvement and extension of the most promising previous works: we frame the adaptive control method in a social navigation problem, adopting a DRL agent working with continuous action space. A social controller is designed by adding a social cost to the Dynamic Window Approach (DWA). This cost is computed considering the robot-pedestrian interaction according to the Social Force Model (SFM) [99, 100]. The proposed solution leverages the advantage of a DRL agent to dynamically adapt the cost weights of the human-aware local planner to different social scenarios. From a general perspective, this research aims to enhance the performance and versatility of service mobile robots in general social contexts. The contribution of this study can be summarized in (i) a social-aware local planner based on SFM, used as a baseline solution, that we refer to as Social Force Window (SFW) planner; (ii) an adaptive cost optimization of the SFW planner with a DRL agent, managing the cost terms dynamically according to the context.

4.1 Methodology

4.1.1 Social Force Window Planner

The Dynamic Window Approach (DWA) is an extremely popular local path planning method in mobile robotics [83]. A brief overview of DWA has been provided in Chapter 3. DWA generates velocity commands that comply with the robot's kinematics constraints. The search space is, therefore, restricted to velocities that can be reached quickly and avoid collisions with obstacles. The classic objective function used to evaluate the trajectories comprises three terms associated with the goal, the velocity, and the obstacles.

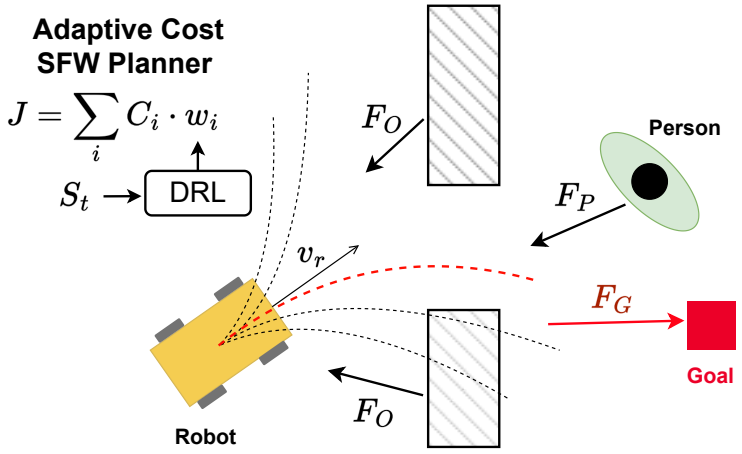


Fig. 4.1 The Social Force Window (SFW) Planner combines standard Dynamic Window Approach and Social Force Model. The trajectory scoring process is optimized by a DRL agent that dynamically adjust the cost weights based on local environmental conditions.

A human-aware local planner has been proposed, adding a social cost to the classic DWA trajectory scoring function. For the social cost, a "social work" quantity is adopted by using the Social Force Model (SFM) of interaction between a crowd of agents proposed by [99, 101, 100]. A social work C_s is computed at each time step for the robot according to the following expression:

$$C_s = W_r + \sum_i W_{p,i} \tag{4.1}$$

where W_r is the sum of the modulus of the robot social force (F_P) and the robot obstacle force (F_O) along the trajectory according to the SFM, while W_p is the sum of the modulus of the social forces generated by the robot for each pedestrian i along

the trajectory. A schematic representation of forces acting on the robot is shown in Fig. 4.1. The goal produces an attractive force while the obstacles and pedestrians generate different repulsive forces.

The overall cost function for trajectory scoring can be formulated as:

$$J = C_s \cdot w_s + C_o \cdot w_o + C_v \cdot w_v + C_d \cdot w_d + C_h \cdot w_h \tag{4.2}$$

where we have a single cost term for social navigation C_s, obstacles in the costmap C_o, robot velocity C_v, distance C_d and heading C_h from a local waypoint on a given global path. The costs are combined using weights w that regulate the impact of each term in the velocity command selection. We refer to this advanced social version of the DWA as a Social Force Window (SFW) planner which is publicly available in Github[1]. This local planner aims to generate safe, efficient, and human-aware paths. However, finding an optimal trade-off between all those desired aspects in every environmental context is not easy. Hence, we tackle the challenge by using a Reinforcement Learning approach to dynamically handle the weights of the costs.

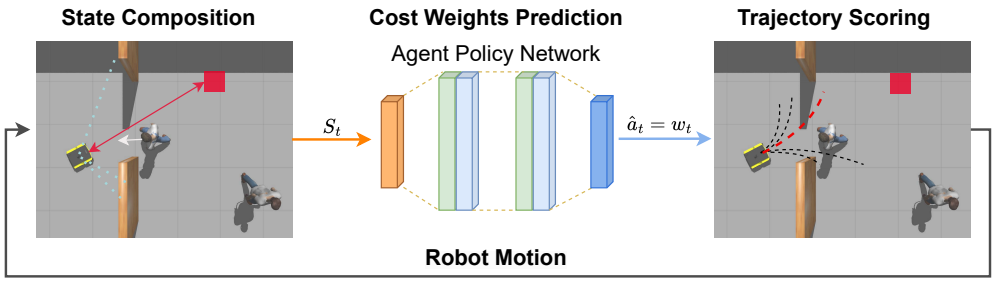


Fig. 4.2 Workflow of the main step performed by the proposed adaptive Social Force Window (SFW-SAC) Planner with DRL. The policy network learns to set the weights of the social cost used by the DWA for each situation.

4.1.2 Deep Reinforcement Learning framework

As better explained in Chapter 2, a typical Reinforcement Learning (RL) framework can be formulated as a Markov Decision Process (MDP) described by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$. An agent starts its interaction with the environment in an initial state s_0, drawn from a pre-fixed distribution $p(s_0)$ and then cyclically selects an action

[1] https://github.com/robotics-upo/social_force_window_planner

$\mathbf{a_t} \in \mathcal{A}$ from a generic state $\mathbf{s_t} \in \mathcal{S}$ to move into a new state $\mathbf{s_{t+1}}$ with the transition probability $\mathcal{P}(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t})$, receiving a reward $r_t = R(\mathbf{s_t}, \mathbf{a_t})$.

In RL, a parametric policy π_θ describes the agent behavior. In the context of autonomous navigation, we usually model the MDP with an episodic structure with maximum time steps T. Hence, the agent's policy is trained to maximize the cumulative expected reward $\mathbb{E}_{\tau\sim\pi}\sum_{t=0}^{T}\gamma^t r_t$ over each episode, where $\gamma \in [0,1)$ is the discount factor. More in detail, we aim at obtaining the optimal policy π_θ^* with parameters θ through the maximization of the discounted term:

$$\pi_\theta^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi} \sum_{t=0}^{T} \gamma^t [r_t + \alpha\mathcal{H}(\pi(\cdot|s_t))] \tag{4.3}$$

where $\mathcal{H}(\pi(\cdot|s_t))$ is the entropy term, which increases robustness to noise through exploration, and α is the temperature parameter which regulates the trade-off between reward optimization and policy stochasticity. In this work, a parameter-learning approach has been adopted to develop an adaptive social navigation system. The DRL agent learns a policy to dynamically set the weights of the cost function that governs the robot's control algorithm. In particular, a Soft Actor-Critic (SAC)[76] off-policy algorithm has been used to train the agent in simulation.

4.1.3 SFM Adaptive Cost Approach

The key idea of the proposed method lies in learning an optimal policy to dynamically set the weights of each objective function term used by the SFM local planner to score the simulated circular trajectories and select the next velocity command (v, w). A DRL agent is trained to learn such policy given the local features of the surrounding environment and induce the local planner to choose optimal velocity commands. DRL is considered a competitive approach to tackle this complex task since it is not straightforward for a human to find an optimal trade-off between all the cost terms of a social controller in each situation. On the other hand, the overall methodology represents a robust hybrid solution that efficiently integrates the flexibility of the DRL agent policy with the reliability of a classical navigation algorithm. Moreover, the agent allows the planner to extend its adaptability to different social scenarios by learning the map between task-related and perception data to suitable cost weights. Fig. 4.2 shows the main working steps of the proposed methodology.

4.1.4 Reward function

Reward shaping is a fundamental and controversial practice in model-free RL. A specific reward function, similar to the cost function of the SFW planner, has been designed to let the agent learn an optimal cost weights regulation policy among all the desired components of the overall navigation behavior.

Goal distance First, a distance advancement reward term is defined to encourage the approach of the next local goal on the global path, always placed at $2m$ from the robot's actual pose:

$$r_d = d_{t-1} - d_t \tag{4.4}$$

where d_{t-1} and d_t are Euclidean distances between the robot and the local goal. Local goal and final goal coincide in the final section of the trajectory.

Path alignment Then, we define a reward contribution r_h to keep the robot oriented towards the next local goal:

$$r_h = \left(1 - 2\sqrt{\left|\frac{\phi}{\pi}\right|}\right) \tag{4.5}$$

where ϕ is the heading angle of the robot, namely the angle between its linear velocity and local goal on the global path.

Robot velocity A velocity reward is defined to promote faster motion when allowed by the environment:

$$r_v = \frac{v - v_{max}}{v_{max}} \tag{4.6}$$

Obstacle avoidance An obstacle reward is included to encourage safe trajectory scoring and avoid collisions:

$$r_o = \frac{d_{o,min} - lidar_{max}}{lidar_{max}} \tag{4.7}$$

where $d_{o,min}$ is the lowest distance measured by the LiDAR ranges at the current time step and $lidar_{max}$ is the saturation distance of LiDAR points, which is set to $3m$ to perceive only local environmental features.

Social penalty The main reward contribution has been assigned to provide the agent with a socially compliant navigation policy. In particular, two different social terms have been considered: proxemics-based reward and social work. The proxemics term penalizes the robot when intruding into the personal space of a person:

$$r_p = \frac{1}{d_{p,min}} \tag{4.8}$$

where $d_{p,min}$ is the minimum person distance from the robot. The social work generated by robot and people interaction according to the Social Force Model has been used as reward r_s, as done in the cost of the SFW planner.

We also include a reward contribution for end-of-episode states, assigning $r_c = -400$ if a collision occurs. The final reward signal is finally obtained linearly, combining the described terms. More in detail, $c_d = 10.0$, $c_h = 0.4$, $c_o = 2.0$, $c_p = 2.0$ and $c_s = 2.5$ are the numerical coefficients chosen to balance the diverse reward contributions in the final signal.

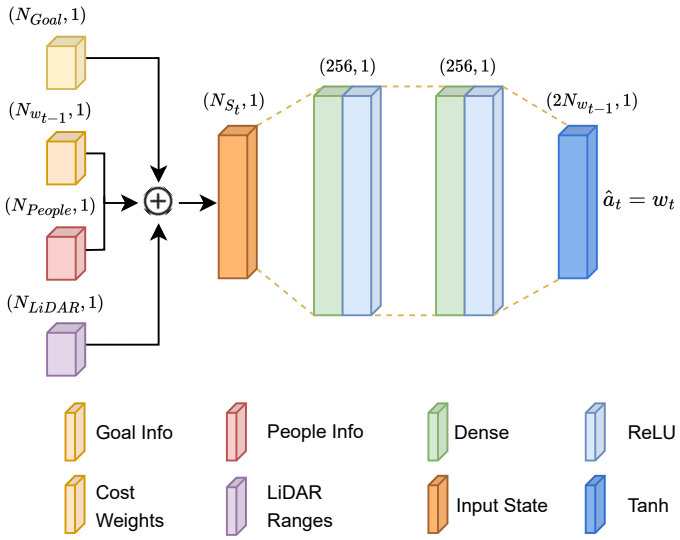


Fig. 4.3 Schematic of the policy network architecture. State composition is illustrated with separate inputs: goal distance and angle, previous cost weights, people position and velocity, and LiDAR ranges. The new cost weights are predicted as output action of the policy network.

4.1.5 Policy Neural Network and Training Design

We define the parametrized agent policy with a deep neural network. We train the agent with the Soft Actor-Critic (SAC) algorithm presented in [76], which allows for a continuous action space and a fast convergence. In particular, we instantiate a stochastic Gaussian policy for the actor and two Q-networks for the critics. The neural network architecture of the agent, represented in Fig. 4.3, is composed of two dense layers of 256 neurons each. A random initial exploration phase has been performed. Random actions are then sampled with a probability that is exponentially reduced with the increase of episodes to maintain a proper rate of exploration during the whole training. Moreover, SAC uses a stochastic Gaussian policy that outputs the mean and the standard deviation of each action distribution, which are used to sample the action value at the training phase. Differently, the mean value of actions' distribution is directly used at test time. The critic networks' structure presents no differences, except they include the predicted action vector in the inputs and predict the Q values.

State definition

The information included in the input state of the policy network has been selected to be a synthetic but complete description of the main environmental and task-specific aspects. The state s_t has been, therefore designed as the ensemble of:

- Goal information: $[goal_{angle}, goal_{distance}]$ with respect to the robot expressed in polar coordinate.
- The set of cost weights predicted at the previous time step, $[w_d, w_h, w_v, w_o, w_s]_{t-1}$ to provide information about the actual state of the SFW costs used for trajectory scoring.
- People position and velocity information is embedded in the state for the closest $K = 4$ people to the robot. Position is computed in polar coordinates $[person_{angle}, person_{distance}]$, while velocity with module and orientation, both expressed in the robot frame. People are perceived at a maximum distance of $5m$, and if people are detected to be less than $K = 4$, padding at the maximum distance is used to fill the empty input features and guarantee a constant input dimension.

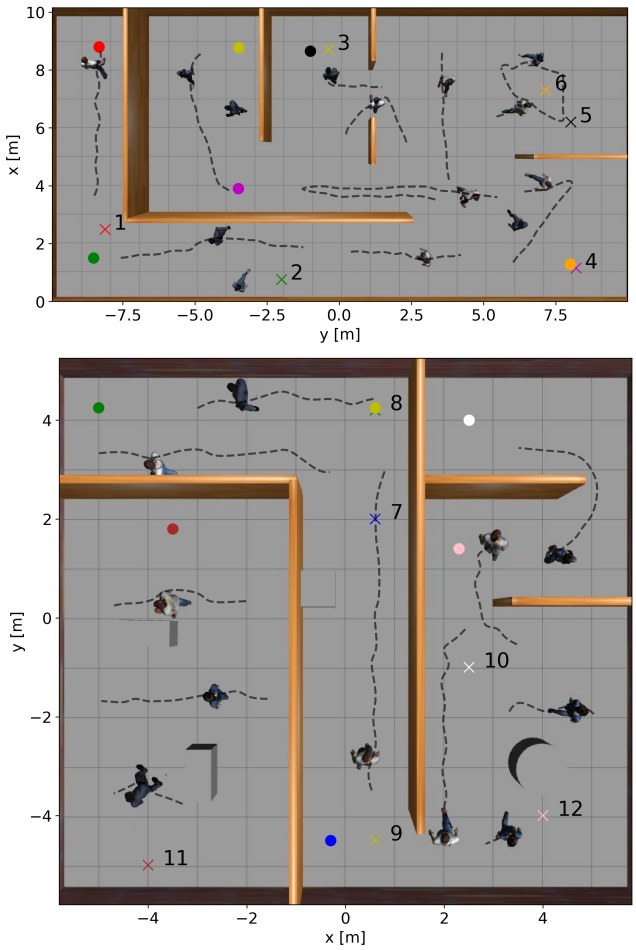


Fig. 4.4 Gazebo simulation environments where the agent has been trained (top) and tested (top and bottom). People trajectories are indicated with dotted lines, robot starting poses with a circle, goals with a cross and the associated episode's number.

- A set of 36 LiDAR 2D points saturated at $3m$ to provide the agent with the necessary awareness of local environmental geometry and spaces and the presence of obstacles.

Output actions

The policy network predicts an action $a_t = [w_d, w_h, w_v, w_o, w_s]$ at each time step, directly representing the new set of cost weights for the Social Force Window local planner. The weights are chosen in the interval of values $[0.1, 5.0]$, and they are set with a frequency of $2Hz$, which has been considered a proper choice for a robot

moving at a maximum translational velocity of $v_{max} = 0.6[m/s]$ in dynamic social scenarios.

4.2 Experiments and Results

4.2.1 Experimental settings

The adaptive social navigation system has been trained and tested in Gazebo simulation in diverse scenarios. The HuNavSim plugin [86] has been adopted to instantiate people moving according to the SFM with customized trajectories and behaviors; HuNavSim has also been used to collect the metrics of interest for the evaluation of the algorithms. Differently, the PIC4rl-gym [27] has been used as ROS 2 framework for DRL agent training and testing. Fig. 4.4 shows the environments realized to carry out challenging experiments categorized in different social challenges. The first Gazebo world is used for both training and testing. A wide set of diverse episodes is defined for training the agent in various conditions involving pedestrians passing, overtaking, and crossing tasks in narrow and open spaces. The agent has also been partially tested in the same world, changing the starting pose of the robot and its goals, indicated in Fig. 4.4, scenarios $[1-6]$. Diversely, testing episodes $[7-12]$ have been performed in a separate different world to evaluate the system in diverse scenarios, always considering crossing, passing, overtaking, and mixed miscellaneous tasks.

For general and reproducible experimentation, we set a basic pedestrian behavior that considers the robot an obstacle. A global path is computed once at the beginning of each episode with the standard grid-based search planner of the Nav2 framework. The main controller parameters of the SFW algorithm are the ones of the classic DWA. Besides the kinematics limits of the robots, the waypoint position and the trajectory simulation time are important factors for a social controller, regulating the alignment to the global path and the predicting horizon. Controller parameters and cost weight values used for the experimentation are reported in Table 4.1. The static cost weights used are the results of the fine-tuning process carried out by a human expert. We use the same implementation of the SFW planner for the DWA baseline, setting the social cost to zero value.

Table 4.1 Controller parameters. On the left the kinematic and classic DWA parameters, on the right the cost function weights used by the SFW controller and modified by the SFW-SAC in the range reported. The DWA uses the same cost weights except for the social term.

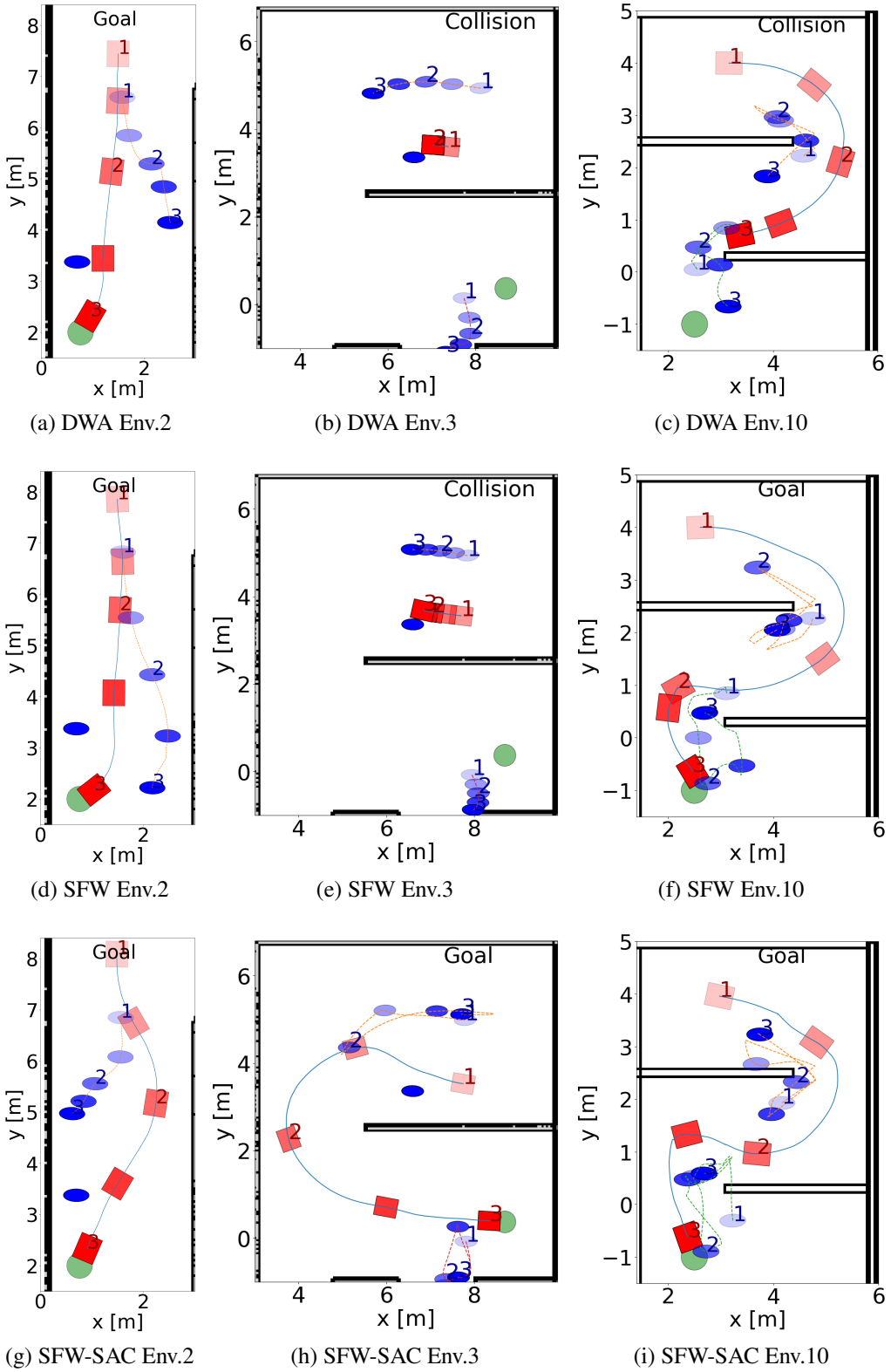
DWA parameter	Value	Cost weight	SFW	SFW-SAC
max linear vel	0.6 [m/s]	social weight	2.0	[0.5 - 3.0]
min linear vel	0.08 [m/s]	costmap weight	2.0	[0.5 - 3.0]
max angular vel	1.5 [rad/s]	velocity weight	0.8	[0.1 - 1.0]
waypoint tol	2.0 [m]	angle weight	0.6	[0.1 - 1.0]
sim time	2.5 [s]	distance weight	1.0	[0.1 - 1.5]

4.2.2 Results

In this section, we describe the metrics chosen to evaluate the proposed social navigation system, and we discuss the obtained results. Considering the difficulty of strictly judging the performance of a social navigation algorithm without adopting human rating, the adaptive social planner SFW-SAC has been analyzed from different perspectives. First, we compared it to the baselines DWA and Social Force Window Planner (SFW) with static costs using relevant quantitative metrics.

Quantitative evaluation Standard navigation metrics such as clearance time $[s]$, path length (PL) $[m]$ and average linear velocity $v_{avg}[m/s]$ are employed to evaluate the effectiveness of the planner from a classic perspective. On the other hand, the social work (SW) metric is included in the quantitative results to show the social impact of the navigation, measuring the social forces generated by the robot and on the robot by pedestrians during its motion. The social work has been taken into account as the average value SW_{step} over the number of trajectory steps to consider the duration of the episode, and avoiding metrics biases caused by a fast execution of the navigation task.

A thorough inspection of the performance is presented, reporting both resulting metrics in Table 4.2. Results show that the DRL method enables the planner to overcome the baseline performance in multiple environments. The basic DWA fails to complete the navigation task in a high percentage of scenarios, colliding with obstacles or pedestrians. On the other hand, the SFW baseline demonstrates an improved ability to handle social navigation tasks. Even though the cost weights combination found by a human offers safe behavior in most situations, it still presents some limitations. For example, in cluttered scenarios, SFW can be hindered by high



(a) DWA Env.2 (b) DWA Env.3 (c) DWA Env.10

(d) SFW Env.2 (e) SFW Env.3 (f) SFW Env.10

(g) SFW-SAC Env.2 (h) SFW-SAC Env.3 (i) SFW-SAC Env.10

Fig. 4.5 Trajectory plots of Env 2, 3, 10 comparing DWA, SFW and the proposed SFW-SAC adaptive planner with DRL. Goals are represented with green circles, the robot with a red rectangle and people with blue ellipses. Transparency and indexes 1,2,3 represent temporal evolution of the motion of both robot and people.

social costs, which can cause the algorithm to get stuck. Diversely, the SFW-SAC proposes a more general performance, finding a better trade-off of costs in different situations. This advantage is proved by the higher success rate obtained in almost all the environments, sometimes being the unique solution able to complete it (Env 3). A more detailed analysis of results tells us that SFW-SAC often finds a compromise performance between the more aggressive DWA and the SFW with high static social cost values. This trend can be noticed by looking at the time, path length, and average velocity results. On average, the adaptive planner generally chooses higher velocities than the SFW but lower than the DWA. Social work embeds all the navigation effects on humans, considering distances, approaching velocities, and time spent close to people. Thus, it often presents alternating results that are difficult to interpret without a visual inspection of the navigation. Indeed, DWA often reduces the duration of the episode thanks to abrupt motion and brief transitions close to people that can lead to a collision with a high risk. The SW_{step}, relating the social impact to the duration of the task, shows more clearly the socially compliance of SFW and SFW-SAC compared to DWA. The agent-based solution is often able to mitigate the social work improving or remaining comparable with the SFW, without compromising the success rate or strongly violating social rules.

Proxemics According to this, the human awareness of navigation is also measured through the level of intrusion of proxemics spaces of people. Fig. 4.6 illustrates the percentage of time spent by the robot in the intimate, personal, social, and public space of people in each testing episode. It can be noticed that even though the SFW-SAC planner develops a more risky navigation policy, it can keep people's distances respected and comparable with the SFW baseline. It should be noted that the proxemics results reported should be paired with the success rate of the algorithms on each episode for a clear perspective. DWA often computes aggressive trajectories that do not take humans into account, although the temporal intrusion of social spaces is sometimes limited to short time intervals.

Trajectory visual comparison Resulting trajectories of some relevant scenarios where the proposed adaptive SFW-SAC show significant improvements and interesting differences, i.e., Environments $[2, 3, 10]$, are plotted in Fig. 4.5 for a better understanding of the performance of the algorithms. In Env. 2, only the SFW-SAC can properly perform the overtaking, passing to the left of the person, while the other algorithms cross the person's path. In Env. 3, DWA and SFW are not able to

handle the presence of a static person on the path and collide; the agent learns how to deviate the motion from the path and avoid the person. In Env. 10, a narrow curved passage with people passing is successfully handled by the SFW and the SFW-SAC, with a smoother trajectory.

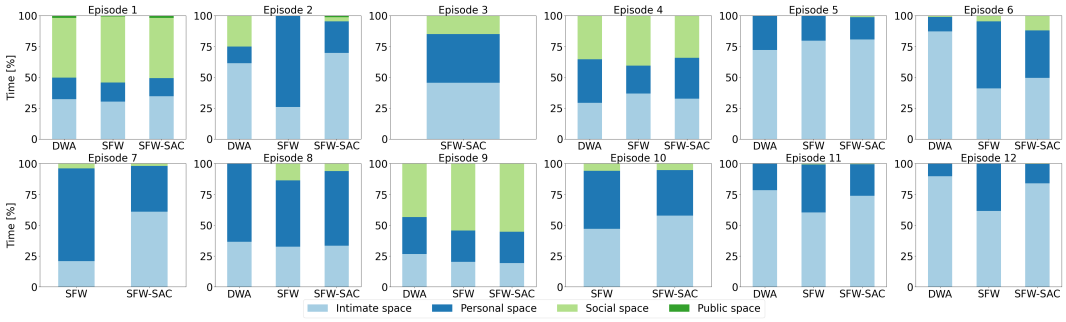


Fig. 4.6 Average temporal percentage of pedestrians space intrusion according to the proxemics standard in 10 different scenarios. Proxemics data must be coupled with success rate reported in Table 4.2 for a clear performance frame.

Table 4.2 Results obtained testing the proposed adaptive controller SFW-SAC on different environments. For each environment we report average metric results over 10 runs, comparing the agent with DWA and SFW baselines.

Env	Method	Success%	Time [s]	PL [m]	v_{avg} [m/s]	SW_{step}
	DWA	100.00	11.65	5.68	0.49	0.03
1	SFW	100.00	12.48	6.04	0.49	0.04
	SFW-SAC	100.00	12.21	5.91	0.48	0.05
	DWA	20.00	12.32	6.21	0.50	0.17
2	SFW	70.00	20.43	6.42	0.31	0.11
	SFW-SAC	100.00	13.08	6.36	0.49	0.22
	DWA	0.0	-	-	-	-
3	SFW	0.0	-	-	-	-
	SFW-SAC	70.00	23.26	11.29	0.48	0.12
	DWA	20.00	21.71	12.21	0.56	0.25
4	SFW	60.00	37.91	12.65	0.38	0.16
	SFW-SAC	70.00	26.20	12.60	0.48	0.20
	DWA	50.00	19.53	9.47	0.49	0.28
5	SFW	60.00	29.34	9.54	0.34	0.34
	SFW-SAC	70.00	33.28	9.47	0.30	0.24
	DWA	50.00	19.57	8.87	0.46	0.26
6	SFW	40.00	44.24	10.75	0.25	0.16
	SFW-SAC	90.00	23.60	8.75	0.37	0.18
	DWA	0.0	-	-	-	-
7	SFW	90.00	19.83	6.38	0.32	0.08
	SFW-SAC	100.00	16.59	6.25	0.39	0.11
	DWA	90.00	10.35	5.21	0.50	0.10
8	SFW	100.00	15.64	5.95	0.35	0.13
	SFW-SAC	100.00	12.32	5.42	0.44	0.16
	DWA	80.00	15.75	8.32	0.53	0.14
9	SFW	100.00	19.77	8.84	0.45	0.12
	SFW-SAC	100.00	18.62	8.98	0.48	0.12
	DWA	0.0	-	-	-	-
10	SFW	70.00	31.96	8.91	0.29	0.13
	SFW-SAC	90.00	32.84	9.99	0.29	0.14
	DWA	50.00	15.45	6.98	0.45	0.29
11	SFW	80.00	48.58	8.37	0.19	0.16
	SFW-SAC	80.00	30.60	7.72	0.27	0.17
	DWA	90.00	13.98	6.09	0.43	0.21
12	SFW	40.00	53.04	6.48	0.14	0.16
	SFW-SAC	90.00	32.01	6.29	0.24	0.20
	DWA	58.57	15.84	8.00	0.50	0.17
Avg	SFW	76.67	25.73	8.39	0.35	0.14
	SFW-SAC	89.00	21.20	8.50	0.42	0.15

Chapter 5

Online Learning of Wheel Odometry Correction for Mobile Robots with Attention-based Neural Network

Wheel odometry (WO) and inertial odometry (IO) are the simplest forms of self-localization for wheeled mobile robots [102]. However, extended trajectories without re-localization, together with abrupt kinematic and ground changes, drastically reduce the reliability of wheel encoders as the unique odometric source. For this reason, visual odometry (VO) has recently emerged as a more general solution for robot localization [103], relying only on the visual features extracted from images. Nonetheless, service and assistive robotics platforms may often encounter working conditions that forbid the usage of visual data. Concrete scenarios are often related to the lack of light in indoor environments where GPS signals are denied, as occurs in tunnels exploration [104, 105] or in assistive nightly routines [14, 15, 106]. Repetitive feature patterns in the scene can also hinder the precision of VO algorithms, a condition that always exists while navigating through empty corridors [107] or row-based crops [19]. Therefore, an alternative or secondary localization system besides VO can provide a substantial advantage for the robustness of mobile robot navigation. Wheel-inertial odometry is still widely considered a simple but effective option for localization in naive indoor scenarios. However, improving its precision in time would extend its usage to more complex scenarios. Previous works tackle the problem with filters or simple neural networks. Learning-based solutions demonstrate

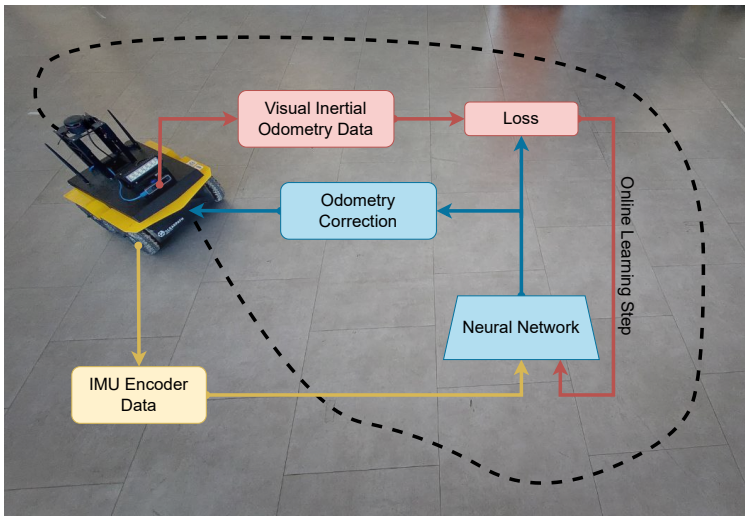


Fig. 5.1 Diagram of the proposed approach. Red blocks and arrows refer to the online training phase, blue ones to the model inference stage, and yellow ones to the odometric input data.

to mitigate the odometric error at the cost of a time-consuming data collection and labeling process. Recently, online learning has emerged as a competitive paradigm to efficiently train neural networks on-the-fly avoiding dataset collection [108]. In this context, this work aims at paving the way for a learning-based system directly integrated into the robot and enabling a seamless transition between multiple odometry sources to increase the reliability of mobile robot localization in disparate conditions. Fig. 5.1 summarizes the proposed methodology schematically.

Several studies have explored using machine learning techniques to estimate wheel odometry (WO) in mobile robotics applications. Approaches include different feed-forward neural networks (FFNN) [109], of which, in some cases, the output has been fused with other sensor data [110], and long short-memory (LSTM) NN, which have been applied to car datasets [111]. These approaches show a promising improvement in WO accuracy, which is crucial for mobile robotics applications. Many works have focused on using Inertial Measurement Unit (IMU) data in mobile robots or other applications, such as person tracking using IMU data from cell phones [112]. One system was improved by implementing a zero-velocity detection with Gate Recurrent Units (GRU) neural network [113]. Another study used an Extended Kalman Filter (EKF) to estimate positions and velocities in real-time in a computationally lightweight manner [114]. Additionally, a custom deep Recurrent Neural Network (RNN) model, IONet, was used to estimate changes in position

and orientation in independent time windows [115]. Some studies used a Kalman Filter (KF) to eliminate noise from the accelerometer, gyroscope, and magnetometer sensor signals and integrate the filtered signal to reconstruct the trajectory [116]. Another KF approach has been combined with a Neural Network to estimate the noise parameters of the filter [117]. Several neural network architectures have been proposed to predict or correct IO odometry over time. For example, a three-channel LSTM was fed with IMU measurements to output variations in position and orientation and tested on a vehicle dataset [118]. Another LSTM-based architecture mimics a kinematic model, predicting orientation and velocity given IMU input data. Studies have investigated the role of hyper-parameters in IO estimation [119].

Sensor fusion of wheel encoder and IMU data is a common method for obtaining a robust solution. One approach involves fusing the data with a Kalman Filter, which can assign a weight to each input based on its accuracy [120]. A Fully Connected Layer with a convolutional layer has been employed for estimating changes in position and orientation in a 2D space over time in an Ackermann vehicle, along with a data enhancement technique to improve learning efficiency [121]. Additionally, a GRU RNN-based method has been proposed to compensate for drift in mechanum wheel mobile robots, with an in-depth fine-tuning of hyper-parameters to improve performance [122].

In this chapter, we tackle the problem of improving wheel-inertial odometry by learning how to correct it online with an efficient artificial neural network [13]. At this first stage, the study has been conceived to provide the robot with a more reliable, secondary odometric source in standard indoor environments where the working conditions for VO can temporarily vanish, as in the case of robots for domestic night surveillance or assistance.

5.1 Methodology

5.1.1 Problem Formulation

A theoretical introduction to the problem of mobile robot localization is provided in Chapter3. Here, some pills are reported to frame the main concepts and the notation used in this chapter. The position of a robot at time t referred to the starting reference frame $\mathbf{R}_0$ can be calculated by accumulating its increments during time segments δt.

The time stamp n refers to the generic time instant $t = n\delta t$. The state of the robot $\mathbf{x}_n$ is defined by the position and orientation of the robot, such as:

$$\mathbf{x}_n = (x_n, y_n, \theta_n)^T, \tag{5.1}$$

where (x_n, y_n) is the robot's position in the 2D space and θ_n is its heading angle. Given the state, it is possible to parametrize the transformation$\mathbf{T}_0^m$ matrix from the robot's frame $\mathbf{R}_m$ to the global frame $\mathbf{R}_0$. Its first two columns represent the axes of the robot frame, and the last one is its position with respect to the origin. The

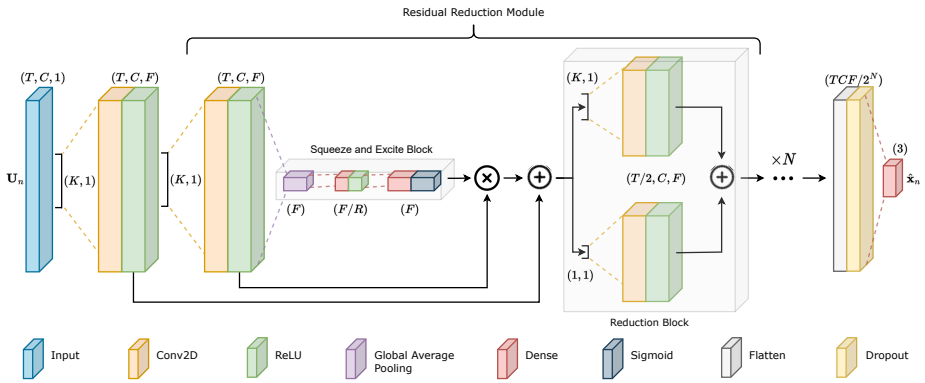


Fig. 5.2 Architecture of the proposed model. The batch dimension is omitted for better clarity.

robot employed to develop this work is equipped with an IMU, which includes a gyroscope and an accelerometer, and two wheel encoders. Therefore, $\mathbf{u}_n$ is defined as the measurement array referred to instant n, i.e.:

$$\mathbf{u}_n = \left(v_l, v_r, \ddot{x}, \ddot{y}, \ddot{z}, \dot{\theta}_x, \dot{\theta}_y, \dot{\theta}_z\right)^T, \tag{5.2}$$

where (v_l, v_r) are the wheels' velocities, $(\ddot{x}, \ddot{y}, \ddot{z})$ are the linear accelerations and $(\dot{\theta}_x, \dot{\theta}_y, \dot{\theta}_z)$ are the angular velocities. The input $\mathbf{U}_n$ to the proposed model consists in the concatenation of the last N samples of the measurements $\mathbf{U}_n = (\mathbf{u}_{(n)}, \mathbf{u}_{(n-1)}, \ldots, \mathbf{u}_{(n-N)})^T$. At each time sample, the state is updated as a function of the measurements $f(\mathbf{U}_n)$: first, the change of the pose $\delta\hat{x} = f(\mathbf{U}_n)$ of the robot is estimated, relative to the previous pose $\hat{\mathbf{x}}_{n-1}$. Then, the updated state is calculated, given the transformation matrix obtained before, as:

$$\hat{\mathbf{x}}_n = \hat{\mathbf{x}}_{n-1} \boxplus f(\mathbf{U}_n) = \mathbf{T}_{0(n-1)}^m \delta\hat{\mathbf{x}}_n, \tag{5.3}$$

where the operator $\boxplus$ symbolizes the state update.

5.1.2 Neural Network Architecture

As formalized in the previous section, the prediction of $\hat{\mathbf{x}}_n \in \mathbb{R}^3$ from $\mathbf{U}_n \in \mathbb{R}^{T \times C}$ is framed as a regression problem. The architecture we propose to solve this task is inspired to REMNet [123, 124], though it uses 2D convolutions instead of the original 1D convolutional blocks (Fig. 5.2). This modification aims at exploiting temporal correlations without compressing the channel dimension throughout the backbone. In particular, we keep the channel dimension C separated from the filter dimension F. In this way, the first convolutional step with kernel $(K, 1)$ and F filters outputs a low-level feature map $f_1 \in \mathbb{R}^{T \times C \times F}$. Then, a stack of N Residual Reduction Modules (RRM) extracts high-level features while reducing the temporal dimension T. Each RRM consists of a residual (*Res*) block followed by a reduction (*Red*) module:

$$RRM(x) = Red(Res(x)) \tag{5.4}$$

The *Res* block comprises a 2D convolution with kernel $K \times 1$ followed by a Squeeze-and-Excitation (SE) block [62] on the residual branch. The SE block applies attention to the channel dimension of the features with a scaling factor learned from the features themselves. This operation improves the representational power of the network by enabling it to perform dynamic channel-wise feature recalibration. First, the block applies average pooling to dimensions T and C. Then, it reduces the channel dimensionality with a bottleneck dense layer of F/R units. Finally, another dense layer restores the original dimension and outputs the attention weights. After multiplying the attention mask for the features, the result is used as a residual and added to the input of the residual block. The *Red* block halves the temporal dimension by summing two parallel convolutional branches with a stride of 2. The layers have kernels $K \times 1$ and 1×1, respectively, to extract features at different scales. After N RRM blocks, we obtain the feature tensor $f \in \mathbb{R}^{T \times C \times F/2^N}$, which is flattened to predict the output through the last dense layer. We also include a dropout layer to discourage overfitting.

5.1.3 Training Procedure

The goal of this work consists of learning the positioning error of the robot using wheel odometry. Nonetheless, it is important to remark that, nowadays, visual-inertial

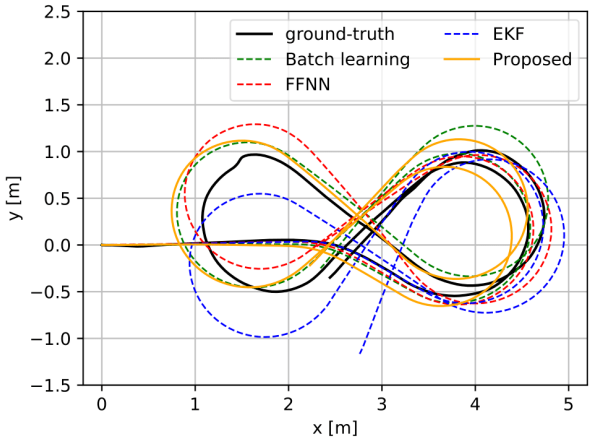


Fig. 5.3 Infinite-shaped trajectories estimated by different methods. The data are collected during a total navigation time of about 60s.

odometry (VIO) is a standard approach on robotic platforms. This work does not aim to propose a more precise localization system but to learn wheel-inertial odometry as a second reliable localization algorithm available whenever visual approaches fail. We exploit a basic VIO system on the robot for the only training process since it enables a competitive online learning paradigm to train the model directly on the robot. Batch learning, the most used training paradigm, requires all the data to be available in advance. As long as the data are collected over time, the proposed method consists in training the network in a continuous way when a batch of N data is available. This approach has been tested extensively in [125], demonstrating a negligible loss in accuracy compared to the batch-learning paradigm.

The proposed model's training consists of two main steps, which are repeated as long as new data are available. First, a batch of N elements is collected, respectively, the input of the network $\mathbf{U}_n$ and the expected output δx. Then, an update step is carried out using an SGD-based optimizer algorithm adopting a Mean Absolute Error loss function, which does not emphasize the outliers or the excessive noise in the training data.

5.2 Experiments and Results

In this section, the proposed approach is tested through extensive experimental evaluations. The model presented in Section 5.1.2 has been trained with an incremental

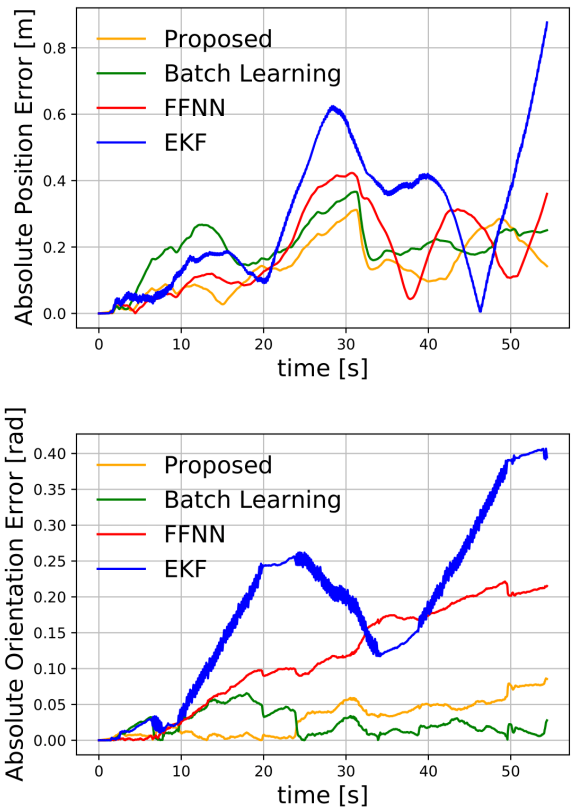


Fig. 5.4 Absolute error of position and orientation of different methods during the test performed on a subset of infinite-shaped trajectories. The considered subset is the same as figure 5.3.

learning method and a classical batch training approach. Results obtained with a simple FFNN model and a standard localization solution based on an EKF are also discussed in the comparison. For this sake, both training processes have been accomplished on the same dataset, and all the tests have been executed on the same test set.

5.2.1 Experimental Setting

The dataset used for the experiments was collected in a generic indoor environment. The employed robotic platform was a Clearpath Jackal[1], a skid-steer driving four-wheeled robot designed for indoor and outdoor applications. All the code was developed in a ROS 2 framework and is tested on Ubuntu 20.04 LTS using the ROS

[1] https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/

2 Foxy distro. Since an indoor environment was considered, the linear velocity of the robot was limited to $0.4m/s$ and its angular velocity to $1rad/s$. The data from the embedded IMU and wheel encoders were used as inputs to the model. According to these assumptions, we used the robot pose provided by an Intel Realsense T265 tracking camera as ground truth. As the testing environment is a single room, the precision of the tracking camera is guaranteed to provide a drift of less than 1% in a closed loop path[2]. All the data have been sampled at $1/\delta t = 25Hz$. The data were collected by teleoperating the robot around the room and recording the sensor measurements. For the training dataset, the robot has been moved along random trajectories. For the test dataset, critical situations when the skid-steer drive robot's odometry is known to lose the most accuracy were reproduced, such as tight curves, hard brakings, strong accelerations, and turns around itself. The obtained training dataset consists of 156579 samples; 80% have been used for training and 20% for validation and hyperparameter tuning. The test dataset consists of 61456 samples. The model hyperparameters have been tuned by performing a grid search using a batch learning process, considering a trade-off between accuracy and efficiency. In the identified model, we adopted $F = 64$ filters, $N = 2$ reduction modules, and a ratio factor $R = 4$. Kernel size $K = 3$ is used for all the convolutional layers, including the backbone. The input dimensions were fixed to $T = 10$ and $C = 8$. The former corresponds to the number of temporal steps, and it has been observed how a higher value appears to be superfluous. In contrast, a lower value leads to performance degradation. The latter value, C, corresponds to the number of input features, i.e., sensor measurements as described in 5.1. We adopted Adam [50] as the optimizer for the training. The exponential decay rate for the first-moment estimates is fixed to $\beta_1 = 0.9$, and the decay rate for the second-moment estimates is fixed to $\beta_2 = 0.999$. The epsilon factor for numerical stability is fixed to $\varepsilon = 10^{-8}$. The optimal learning rate η was experimentally determined as 1×10^{-4} for batch learning. Conversely, the incremental learning process showed how a value of $\eta = 7 \times 10^{-5}$ avoided overfitting since the data were not shuffled. In both learning processes, a batch size of $B = 32$ was used.

[2] https://www.intelrealsense.com/tracking-camera-t265/

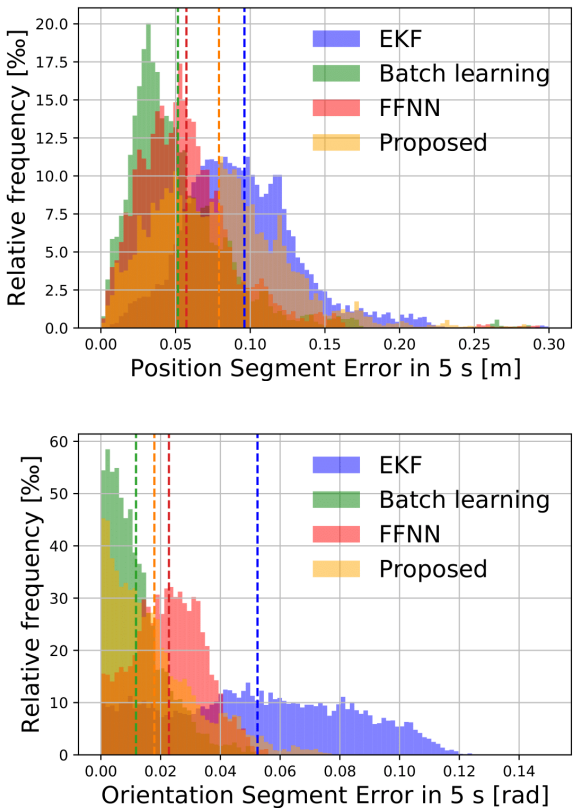


Fig. 5.5 Histograms of the SE error in position and orientation in section B of the test set.

5.2.2 Evaluation Metrics

To evaluate the performance of the proposed model, two different metrics were used [126]:

- *Mean Absolute Trajectory Error (m-ATE)*, which averages the magnitude of the error evaluated between the estimated position and orientation of the robot and its ground truth pose in the same frame. Sometimes, it can lack generalization due to possible error compensations along the trajectory.
- *Segment Error (SE)*, which averages the errors along all the possible segments of a given length s, considering multiple starting points. It is strongly less sensitive to local degradation or compensations than the previous metrics.

Table 5.1 Performance comparison on the different test scenarios and the overall test set with the respective standard deviation. When trained with batch learning, the proposed architecture performs better than the FFNN proposed in [121]. If trained online, it outperforms the common EKF-based localization method and achieves the results of the model trained offline.

Test	T$[s]$	Method	$m-ATE_{(x,y)}[m]$	$m-ATE_{\theta}[rad]$	$SE_{(x,y)}[m]$	$SE_{\theta}[rad]$
A	998	EKF	0.692 ± 0.213	0.821 ± 0.334	0.099 ± 0.043	0.069 ± 0.032
		Online Learning	0.292 ± 0.098	0.118 ± 0.079	0.071 ± 0.038	0.020 ± 0.013
		Batch Learning	0.208 ± 0.061	0.084 ± 0.072	0.062 ± 0.039	0.013 ± 0.010
		FFNN [121]	0.354 ± 0.124	0.326 ± 0.125	0.063 ± 0.036	0.027 ± 0.012
B	668	EKF	1.118 ± 0.586	0.380 ± 0.126	0.096 ± 0.041	0.052 ± 0.030
		Online Learning	0.330 ± 0.081	0.117 ± 0.097	0.079 ± 0.042	0.017 ± 0.015
		Batch Learning	0.197 ± 0.059	0.067 ± 0.030	0.051 ± 0.034	0.011 ± 0.009
		FFNN [121]	0.513 ± 0.223	0.38 ± 0.181	0.057 ± 0.034	0.022 ± 0.011
C	802	EKF	0.572 ± 0.207	0.343 ± 0.174	0.088 ± 0.045	0.049 ± 0.034
		Online Learning	0.270 ± 0.104	0.112 ± 0.053	0.081 ± 0.043	0.033 ± 0.030
		Batch Learning	0.178 ± 0.095	0.086 ± 0.062	0.047 ± 0.031	0.019 ± 0.016
		FFNN [121]	0.326 ± 0.102	0.183 ± 0.058	0.050 ± 0.031	0.019 ± 0.013
All	2458	EKF	0.738 ± 0.385	0.553 ± 0.338	0.094 ± 0.043	0.058 ± 0.033
		Online Learning	0.292 ± 0.100	0.115 ± 0.075	0.076 ± 0.041	0.023 ± 0.021
		Batch Learning	0.195 ± 0.076	0.081 ± 0.062	0.054 ± 0.036	0.014 ± 0.012
		FFNN [121]	0.377 ± 0.160	0.285 ± 0.145	0.057 ± 0.034	0.023 ± 0.012

5.2.3 Quantitative Results

The proposed method was tested by training the neural network from scratch using the stream of sensor data in real-time, brought by the ROS 2 topics. The data were first collected in mini-batches of 32 elements. After completion, backpropagation is performed on the model to update all the weights. The data stream is recorded to provide the aforementioned 5.2.1 training dataset, which was later used to evaluate other methods. The results of the methods are compared to different state-of-the-art solutions, which are i) the same network trained with a traditional batch learning, ii) a Feed-Forward neural network, as in [121], and iii) an Extended Kalman Filter based method, which can be considered one of the most common wheel-inertial odometry estimators. All the models were evaluated offline using a test set composed of 19 sequences of various lengths, comprised between 60s and 280s, which aim to recreate different critical situations for wheel inertial odometry. In particular, the sequences can be separated into three main trajectory types:

- *Type A*, comprises round trajectories which do not allow fortunate error compensation during the time. Therefore, they may lead to fast degradation of the estimated pose, and especially of the orientation.
- *Type B* comprises an infinite-shaped trajectory. This test allows partial error compensations, but possible unbalanced orientation prediction may lead to fast degradation of the position accuracy. A partial sequence of type B trajectories are shown in Fig. 5.3.
- *Type C* comprises irregular trajectories, including hard brakings and accelerations, and aims to test the different methods' overall performance.

Table 5.1 presents the numeric results of the different tests, considering the proposed model (Online Learning) and the selected benchmarks. All the leaning-based approaches show a significant error reduction compared to the EKF results, which can be considered a baseline for improvement. Considering both the neural network architectures trained offline, the proposed convolutional one achieves an average improvement of 73.5% on the position $m-ATE_{(x,y)}$ and 85.3% on the orientation $m-ATE_{\theta}$. In comparison, the FFNN model achieves 49.0% and 48.4%, respectively. The Segment Error improves in both cases: the proposed model improves by 42.6% on the position $SE_{(x,y)}$ and 75.8% on the orientation SE_{θ}. The FFNN architecture improves by 39.3% and 60.3%, respectively. Compared with the EKF baseline, the online learning model shows almost the same improvement as batch learning. The improvement on the m-ATE equals 60.4% on the position and 79.2% on the orientation. The Segment Error also appears to be lower, showing an improvement of 19.1% on position and 60.3% on orientation. The observed difference between the two training paradigms is an acceptable trade-off between the slight loss of accuracy of the online training compared to the batch training and the possibility of training the model without a pre-collected dataset. Fig. 5.5 reports the histograms of the distribution of the Segment Errors, in position and orientation, respectively, for test scenario B. It emerges how learning-based methods achieve, on average, a smaller error than the EKF method. Fig. 5.4 shows the error trend during time related to the trajectory of figure 5.3. It is evident how the batch-trained and online-trained models perform similarly to the other methods.

5.2.4 Latency Evaluation

Since all the training and inference processes are tested online, firm real-time performance is needed to avoid missing data for training or producing late odometry data. The trained neural network has been converted into a TensorFlow Lite *float32* model, which allows the development of models on edge devices and performs inference on CPU devices. Using the Jackal's Board computer, based on an i3-4330TE @ 2.4 GHz chip, a mean odometry estimation time of 4 ms was achieved on 100 measurements, which is 10% of the sampling frequency of 25 Hz. The training process on an external PC with 32-GB RAM on a 12^{th}-generation Intel Core i7 @ 4.7 GHz took an average time of 25 ms per batch, considering 100 measurements.

Chapter 6

Domestic assistance with an omidirectional service robot

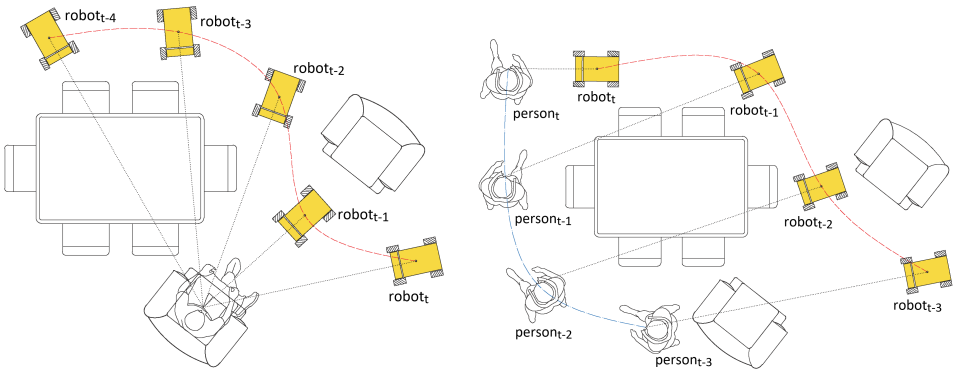
Robot assistants have recently emerged as a promising solution for elderly care and monitoring in the indoor domestic environment. The increasing demand for service robotic platforms for indoor assistance has paved the way for the development of diverse robotic solutions, especially devoted to elderly care [127, 128]. According to the World Population Prospects (2019) provided by the United Nations [129], life expectancy reached 72.6 years, with a future expectation of 77.1 in 2050. Furthermore, projections reveal that there will be more people aged 65 years or over than young aged 15 to 24 years by 2050 [129]. Population ageing dramatically impacts our society's organization, exacerbating delicate issues such as the isolation of numerous vulnerable subjects and elderly people in their homes for most of the day. Moreover, the recent emergency related to the COVID-19 outbreak has further increased the need for a reliable and automatic assistance tool in both hospitals and patients' residential environments. In this scenario, robots demonstrated to be a key technological ally in fighting the pandemic and its dramatic social effects, such as people isolation [130, 131]. Indeed, they can offer support to both medical staff and families whenever the services of dedicated assistive operators or volunteers are not available due to the intensive demand generated by the pandemic. Although the specific role and objectives of a robotic assistant for elderly care need to be concurrently discussed from an ethical perspective according to Abdi et al. [132] diverse robotic platforms for social assistance already exist. However, these studies have been limited to the scope of human-machine interaction, realizing companion

robots with humanoid [133] or pets-like architectures [134, 135]. These robots have been particularly studied for what concerns dementia, aging, and loneliness problems [136, 137]. Different studies specifically focus on detailed monitoring tasks, for example, heat strokes [138] and fall detection [139]. Besides the healthcare and elderly monitoring purposes, the potential scope of application of an indoor robot assistant is wide: house and air quality management [140, 141] according to the Internet of Things (IoT) paradigm [142], surveillance and security and many other service tasks [143].

In this chapter, a novel robotic assistive platform is presented: **Marvin** [14]. The goal of this mobile robot is to provide basic domestic assistance to the user. More in detail, we identify a set of service functions for Marvin within the overall research scope of socially assistive robots: *user monitoring*, *night assistance*, *remote presence, and connectivity*. A layered modular design is adopted to conceive Marvin, resulting in a system indifferent to small modifications of the domestic environment and features required by the specific application. Differently from previously presented robots for home assistance, discussed in Section 6.1, we chose a tiny omnidirectional base platform [144]. In particular, omnidirectional mobility can be exploited to monitor the user while navigating and avoiding obstacles efficiently. Thus, a great focus is devoted to development of a human-centered autonomous navigation system for a robotic assistant, which aims at fulfilling the user assistance requirement in two key scenarios: goal-based navigation and person following (Fig. 6.1). Indeed, person following [145, 146] is the primary challenging task to enable any visual or vocal interaction with the robot while the user is moving around. On the other hand, the robot should be also capable to accomplish desired services in the room, moving around towards different destinations while keep monitoring the person.

Overall, Marvin is a novel robotic solution for domestic and, more generally, indoor user assistance. The contributions of this work can be summarized as follows:

- an agile tiny platform for user monitoring, night assistance, and remote presence, by adopting an omnidirectional wheeled base and a controllable telescopic positioning device (Section 6.2);
- a real-time AI-based vision system (Section 6.3) to constantly detect and track the users and check potential critical conditions based on their pose, triggering an emergency call;



(a) *The rover has to reach different goals while keep monitoring the user.*

(b) *The robot must follow a different path keeping active the monitoring of the person.*

Fig. 6.1 Visualization of human-centered navigation and person following task for domestic assistance: the omnidirectional capability allows the rover to follow the user maintaining its orientation towards them while avoiding obstacles.

- an integrated navigation system that exploits the omnidirectional platform separately handling obstacle avoidance and orientation control for person monitoring (Section 6.4);
- the PIC4Speech vocal control system (Section 6.5) to provide a reliable, offline vocal interface for the user to express commands to the robot easily.

6.1 Assistive Service Robots

In the last years, the robotics research community is focusing its effort on the study of an effective design for an indoor assistant, and different proposals have recently emerged. Diverse researchers based their study on the human-machine interaction, realizing humanoid companion robots such as NAO [133] or pets-like architectures such as Aibo [134] and PARO [135]. These kinds of robots have been particularly studied for research on dementia, aging, and loneliness problems [136, 137], although their usage cannot be extended to home assistance without a mobile platform. Different studies specifically focus on detailed monitoring tasks, for example, heat strokes [138] and fall detection [139]. However, although their usual expensive cost, they often result to be unused for a long time horizon due to the complex healthcare task they try to accomplish. Indeed, for the pure purpose of a companion robot, marginal differences exist with the more competitive commercial vocal assistants like Alexa, with a much lower cost. Jibo [147] (Fig. 6.2a) is another

example of a social robot for the home which falls in this category. Hence, an assistive domestic robot should go beyond the conversational skills of common vocal assistants and we decided to choose a mobile platform, trying to identify a clear, helpful role for the robot in a domestic scenario.

Fig. 6.2 Commercial robots developed or suitable for service home-care applications.

There are already a good amount of research prototypes and few commercial mobile platforms for home-care robotics today. Among them, the HOBBIT robot [148] and the Toyota Home Service Robot (HSR) [149] are the results of different research projects and they present a similar architecture composed of a wheeled main body equipped with manipulators for grasping objects. The Pepper robot (Fig. 6.2b) is one of the most popular humanoid robots and it has been also used for nursing and rehabilitative care of the elderly [150]. TIAGo [151] (Fig. 6.2c)is another comparable platform developed for robotics research groups and in general for indoor applications. Even though they are standard differential drive wheeled platforms, all these robots aim at reproducing a human-like overall shape and presence. However, a large footprint and a standard steering system represent strong disadvantages to navigating in a realistic cluttered household environment. The same limitations hold for the SMOOTH robot [152], the resulting prototype of a research study that aims at developing a modular assistant robot for healthcare with a participatory design process. Three use cases for the SMOOTH robot have been identified: laundry and garbage handling, water delivery, and guidance. In agreement with the authors of the SMOOTH robot, we decided to avoid a robotic arm on the robot, due to the higher control complexity it requires and stability issues it causes when mounted on a tiny lightweight mobile platform. Instead, a simpler positioning device is preferred to lift the camera and to allow the user to access the robot visual interface easily. The

abilities to carry objects without a manipulator and offer physical support to elderly people are the advantages of the SMOOTH robot design.

An agile and flexible motion of the platform is considered a crucial design choice to enable the introduction of robot assistants in real-world domestic environments on a large scale. Omniwheels and mecanum wheels have already been studied in many prototypes [153, 154] and they are particularly used in industrial robotics applications [155], where the flexibility and the optimization of trajectories are a priority. Recently, our design considerations have been partially confirmed by the lastly emerging commercial proposals. Indeed, Amazon has recently launched its commercial home assistant Astro [156] (Fig. 6.2d). Astro can surely be considered an enhanced design thought for end-users, which can visually recognize people and interact with them through a visual interface that aims at conveying expressive reactions and thanks to the Alexa vocal assistant. Robotic platforms such as Astro aim at totally managing the house, also providing surveillance and telepresence services. Astro presents a reduced size compared to the typical humanoid platforms to guarantee agile movements in the house and does not represent an oppressive figure for the users, at the cost of not being able to carry or manipulate items. Moreover, the robot is thought to be integrated within the full house automation system, exploiting Alexa as a vocal interface. However, this choice exposes Astro to high privacy risks and issues, handling both vocal and visual data of domestic private environments.

6.2 Marvin robot design

Researchers at Pic4SeR Center (Interdepartmental Centre for Service Robotics) of the Politecnico di Torino, in association with the researchers at Officine Edison, developed a personal assistant mobile robot called Marvin considering the landscape of existing solutions described. The robot has been conceived as a proof of concept to explore the possibilities of autonomous assistive robots in domestic environments, designed for people owning reduced motility, like elderly or people with disabilities. To such an aim, the robot must be able to perform the following service functions:

1. *User Monitoring*: the robot should be able to detect a potentially dangerous situation for the user and call for help.

2. *Night Assistant*: one of the most critical moments in the daily life of elders is the night-time bedroom-to-toilet journey. This service proposes to accompany the user in any desired location of the domestic environment, enlightening the path and raising alarms in the case of need.

3. *Remote presence and connectivity*: the robot must be provided with the ability to access commonly used communication platforms (mobile phone, video-call services).

These tasks, addressed to provide a service to the user, in turn require a series of robotic capabilities described in the next sections. The architecture of the mobile robot has been developed with a modular approach to support different environment and potential new features to be added, without completely rethinking the robot's structure. The overall system can be divided into three main layers, as presented in Fig. 6.3:

- A *Low Layer System* consists of the mechanical structure, the control electronics, and firmware. This layer is responsible for the actuation and control of the system motion given the desired state of the system which is computed by the Upper Layer System.
- A *Upper Layer System* collects the Upper Layer sensors such as lidar, cameras and remote controller, the autonomous navigation stack, and the visual perception sub-system.
- A *Human-Machine Interface* consists of a vocal control interface and a manual control interface.

The interaction between the different layers is coordinated by predefined communication protocols. In this thesis, we briefly mention the core elements of the low layer system, providing a major focus on the perception, navigation and vocal command features developed. Further details can be found in the realted article [14].

An omnidirectional platform is considered to overcome the limitations of differential drive locomotion systems and fulfill the flexible and agile mobility requirements. A Nexus 4WD Mecanum robot [157] has been chosen as starting base platform for Marvin's development. It is characterized by overall dimensions of 400 mm $\times$ 360 mm $\times$ 100 mm and a limited mass (5.4kg), with a passive roll joint between the front wheels and the rear wheels to deal with the presence of four contact points

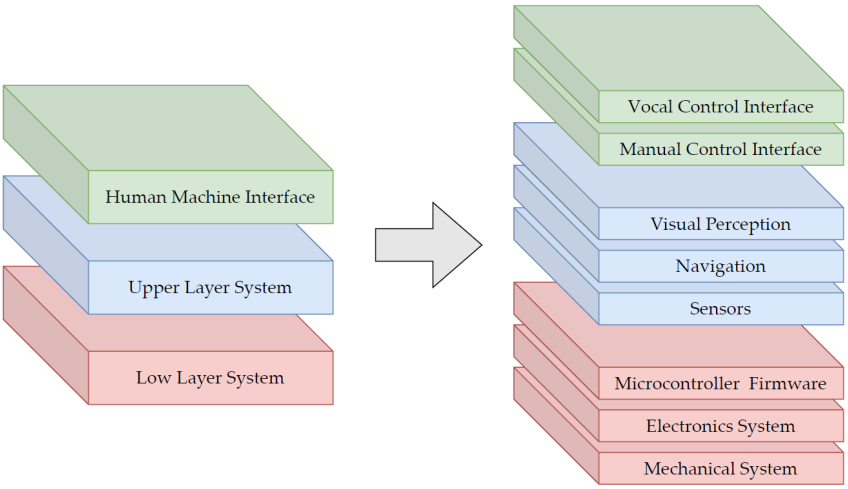


Fig. 6.3 Schematic representation of the modular platform architecture. The three main layers of the architecture (**left**) are decomposed in their respective principal components (**right**).

with the ground. Aside from its ability to exhibit full planar mobility, Marvin has the capability to deploy its sensors and user interface, exploiting its integrated positioning device designed in accordance with the typical domestic workspace. Such aspect is crucial for different reasons:

- it allows improving the perception of the robot of the external environment improving the range of view of the sensors;
- a re-orientable and deployable head enhances the usability of the touch interface for the users, giving a chance also to bedridden or handicapped people to easily interact with the robot.

In Fig. 6.4, the mobile assistive robot is represented in two configurations: on the left, the telescopic mechanism is deployed for better standing usage, while on the right, the custom mechanism is retracted and inclined forward for better-seated usage. The retracted configuration is also very effective at keeping the center of gravity low during the motion of the robot.

6.2.1 Sensors and computational resources

For the robot to effectively work in the domestic environment, a whole series of sensors are required to perceive the surroundings adequately. Marvin mounts the following sensor devices. also shown in Fig. 6.5:

(a) (b)

Fig. 6.4 Final prototype of the mobile assistive robot in two different working configurations: (**a**) Deployed configuration for standing usage, HMI height = 1.1 m, mechanism tilting angle = 0°, (**b**) Retracted end angled configuration for better-seated usage, HMI height = 0.80 m, mechanism tilting angle = 20°.

- Intel RealSense T265 Tracking Camera[1], with VIO technology for self-localization of the platform. It is placed in the front of the rover, to better exploit its capability;
- Intel RealSense D435i Depth Camera [2], that provide aligned color and depth images at 30 frame-per-second (fps). It is mounted on the appropriate support, on the positioning device, which provides a convenient elevated position for the camera;
- RPLIDAR A1 [3] provides a 2D point cloud for obstacle avoidance navigation and mapping of the environment.
- Jabra 710 [4], with a panoramic microphone and speaker. It is particularly useful for voice command. Can be placed on the rover or used wireless from a distance.
- Furthermore, a wireless gamepad is employed for manual control operations.

[1] https://www.intelrealsense.com/tracking-camera-t265/
[2] https://www.intelrealsense.com/depth-camera-d435i/
[3] https://www.slamtec.com/en/Lidar/A1
[4] https://www.jabra.com/business/speakerphones/jabra-speak-series/jabra-speak-710#7710-409

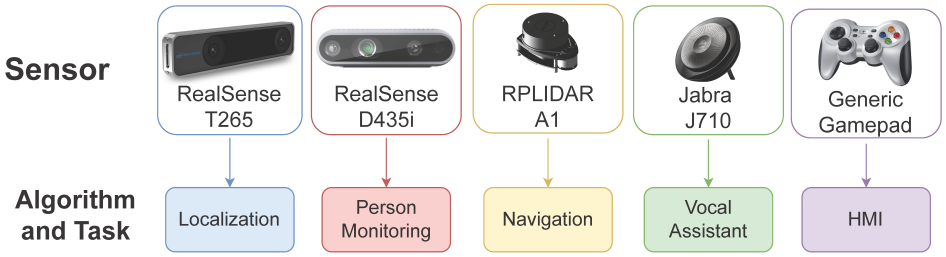


Fig. 6.5 Sensors employed on Marvin robotic platform, associated with the corresponding task they serve.

The robot relies on two computational units: a PJRC Teensy 4.1[5] microcontroller unit (MCU), which manages the low layer system software and an Intel NUC11TNHv5[6] that executes the main Upper Layer system applications. A Coral Edge TPU Accelerator[7] is also employed to run optimized neural network models without the necessity of a full-size graphics processing unit. At software level, the Robot Operating System 2 (ROS2) [158] middleware framework is adopted to integrate all the different high-level application nodes. ROS 2 is preferred over the original ROS [159] as it is more suitable for real-time systems, it is actively supported by the robotics community and it has access to more advanced applications [160, 161].

6.3 Visual Perception for Person Monitoring

Person detection is the pillar of every visual-based Human-Robot interaction. Classic Deep Learning-based one-stage object detectors such as YOLO [162] and SSD [163] estimate a bounding box in the image where the target object is contained. Pose estimation models may represent a more competitive alternative for human-aware robotics tasks since they also offer the information about person's pose status. State-of-the-art models for human pose estimation [164, 165] provide a skeleton schematic graph of the person.

The monitoring task is carried out through a double-step computing pipeline. Firstly, the person-detection is obtained with PoseNet [165], a lightweight neural network able to detect humans in images and videos. As output, it gives 17 key joints (like

[5] https://www.pjrc.com/store/teensy41.html
[6] https://www.intel.com/content/dam/support/us/en/documents/intel-nuc/NUC11TNH_L6_UserGuide.pdf
[7] https://coral.ai

elbows, shoulders, or feet) of each person present in the scene. At this point, a second simple Convolutional Neural Network (CNN) receives the key points to classify the pose of the person as standing, sitting, or laying. A persisting laying condition can automatically activate an emergency call to an external agent (a relative or a healthcare operator). A custom labeled dataset of images has been collected in a house environment to train the CNN for the pose classification. The total number of images used for this dataset is $25,009$. The images are divided into three classes: standing, sitting, and lying, containing 7849, $11,400$, and 5760 images, respectively. The classification model reaches an accuracy of almost 99% on the test set, obtained retaining the 20% of the original dataset. The performance of the model is definitely high, probably due to the common background scene of the collected images. A randomized background with scenes of diverse domestic environments may allow for a more challenging testing condition, leading also to improving the generalization performance of the model. Moreover, the overall pipeline runs on the Google Coral Edge TPU device for a faster inference, reaching 30 frame-per-second (FPS), that is the maximum frame rate allowed with the Realsense D435i camera. Moreover, as shown in Fig. 6.6, the key points predicted by PoseNet are then translated into a bounding box that can be exploited for human-centered navigation tasks. The resulting bounding box is tracked with SORT [166], a very simple online and real-time tracking algorithm based on the Kalman filter. SORT also keeps track of the subject when they leave the frame for a few moments, and associates an ID to each person in the image. This ID is maintained as long as the person does not leave the frame for several time instants. At this point, a depth map aligned with the RGB frame, can be used to extrapolate the relative position of the detected individual in the robot reference frame (x_P, y_P). A subset of the skeleton-pose key points is selected to find the person's center point C in the image. It is computed as the average coordinate of the shoulders joints if recognized recognized with high confidence, otherwise hips are considered.

6.4 Navigation System

Domestic environments are highly dynamic environments, where obstacles' position could be changed over time (chairs, bins) or they can move on their own (people, animals, other autonomous platforms). Therefore, beside a map-based global path planner, these scenarios pose the need for a reactive navigation system. Fig. 6.7

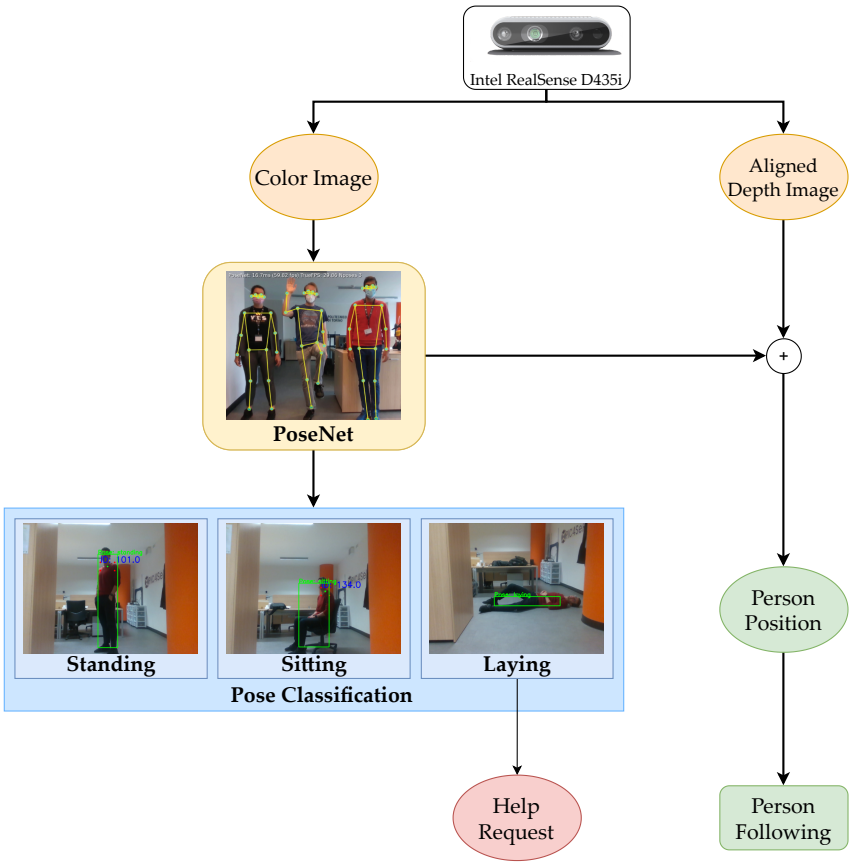


Fig. 6.6 Representation diagram of the person identification system: the estimated pose of the person is continuously classified as standing, sitting, or laying, generating a help emergency request if necessary. Moreover, it is used to extract the dynamic goal coordinates for the person following task.

resumes the complete proposed human-centered navigation system. The upper blue section of the scheme contains the extraction of the person position (x_P, y_P) in the robot reference frame through the visual perception pipeline. The yaw controller then processes this position to obtain the angular velocity command ω needed to keep the platform oriented towards the person. On the lower red section of the scheme, the local planner receives the LiDAR range points and the goal coordinate (x_G, y_G) to produce a collision-free trajectory and provide linear velocities $[v_x, v_y]$. The full velocity command for the robot is therefore obtained by combining linear and angular velocities in the vector $[v_x, v_y, \omega]$.

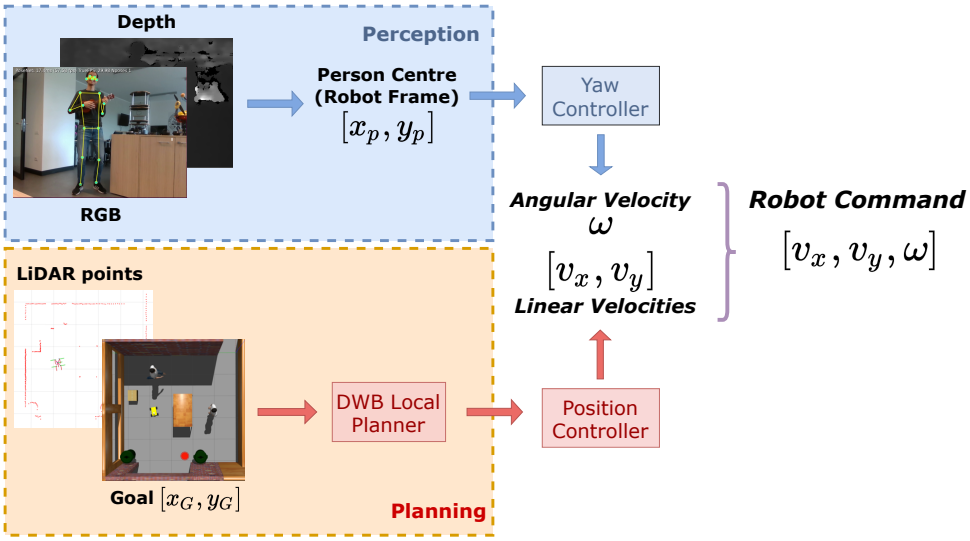


Fig. 6.7 Human-Centered navigation methodology pipeline scheme. Linear and angular velocity $[v_x, v_y, \omega]$ are generated separately to successfully carry out obstacle avoidance through local trajectory planning together with person monitoring through yaw control.

6.4.1 Omnidirectional Motion Planner and Obstacle Avoidance

An integrated navigation system is developed tailoring the Navigation 2 algorithmic stack [8] for the specific use case of assistance and person monitoring. Hence, the DWB local planner, an optimized revisited version of the Dynamic Window Approach (DWA) presented in Chapter 3, generates an obstacle-free trajectory towards the goal and drives the rover along it. To detach the control of linear and angular velocities for the double-objective navigation task at hand, DWB plans safe trajectories using only the two linear velocities $[v_x, v_y]$, along x and y axes of the horizontal plane. The goal of the navigation task (x_G, y_G) coincides with the person's position (x_P, y_P) in the specific case of the person following, diversely it is a separate target point to be reached while monitoring the person in the service navigation scenario.

6.4.2 Person-focused Orientation Control

The angular velocity ω is provided by another system node, which at any instant computes the angular difference $\Delta\theta$ between the orientation of the rover and the orientation of the vector connecting the rover's center of rotation with the person

[8] https://navigation.ros.org/

position, retrieved from the perception module:

$$\Delta\theta = \arctan(y_P, x_P) \tag{6.1}$$

The exact yaw velocity is then calculated as follow:

$$\omega = sign(\Delta\theta) \cdot min(\|k \cdot \Delta\theta\|, \omega_{max}) \tag{6.2}$$

Where k is a parameter used to linearly increase ω as $\Delta\theta$ grows, and ω_{max} is the maximum angular speed value allowed. After some tests on our indoor application, we found optimal values of these parameters respectively at 1.3 and $1.5 rad/s$, but they can be changed depending on the specific operating scenario.

An alternative orientation control strategy has been investigated in [16]. A Soft Actor-Critic (SAC) algorithm [76] has been adopted to train a Reinforcement Learning agent to directly compute the ω control for the robot.

Input features The input features of the policy network embed the necessary information about the dynamic goal: 1) d_t: the distance of the goal from the rover 2) $\Delta\theta_t$: the angular difference between the orientation of the rover and the orientation of the vector connecting the rover's center of rotation with the goal 3) ω_{t-1}: yaw velocity command assigned to the platform at the previous time instant

Reward Reward shaping is the typical process that leads researchers to analytically specify the desired behavior to the agent thanks to a dense reward signal assigned at each time step. To this end, a reward r_h is defined as the arithmetic sum of two distinct contributions:

$$r_{yaw} = \left(1 - 2\sqrt{\left|\frac{\Delta\theta_t}{\pi}\right|}\right) \tag{6.3}$$

$$r_{smooth} = -|\omega_{t-1} - \omega_t| \tag{6.4}$$

The first contribute r_{yaw} teaches the agent to maintain its orientation towards the goal, while the second contribute r_{smooth} is used to obtain a smooth transition between the current agent's yaw velocity output and that at the next time instant.

Neural network architecture The simple neural network used for the orientation control policy comprises three dense layers, respectively with 512, 256, and 256 units each.

The DRL policy showed slightly improved performance compared to the proportional error-based control, avoiding the parameter's tuning process at the cost of training the policy in simulation.

6.5 Vocal Human-Robot Interface

Deep Learning models for vocal assistants requires an extremely high amount of data to be trained [167]. Moreover, state-of-the-art models in Natural Language Processing (NLP) [168, 169] provide great performances at the cost of a much higher computational cost, which forbids their usage on embedded devices with constrained hardware resources. Commercial solutions such as Siri, Alexa, or Google Home exploit a cascade activation pipeline of multiple models that transfer the computation from the physical device, when triggered, to the cloud servers to run their NLP algorithms. Indeed, the development of a full pipeline of algorithms for fast-interference low-cost vocal assistance in robots is rare to be found in the research literature. The proposed *PIC4Speech* vocal assistant has the aim of providing a low-cost, efficient solution to be executed on board the robotic platforms without the need for expensive hardware and, above all, without relying on a stable internet connection. The exposure of private data to the internet can create controversial privacy issues. *PIC4Speech* combines state-of-the-art Deep Neural Networks (DNN) for speech-to-text translation and a simple rule-based model for Natural Language Processing (NLP) to minimize the computational cost of the pipeline and preserving a flexible interaction. Similarly to commercial products it exploits a cascade of models that are progressively activated when the previous one is enabled. In Fig. 6.8, an overview of the overall architecture of the PIC4Speech vocal assistant is represented, showing the subsequent activation of each block. Although PIC4Speech is designed as an offline vocal assistant, its usage in this primitive version is mainly devoted to allowing the user to give commands to the robot vocally and not to hold a complete conversation. In particular, the system aims at matching a vocal instruction expressed by the user to the corresponding required task to successively start the correct control process.

The PIC4Speech operative chain can be summarized as follows:

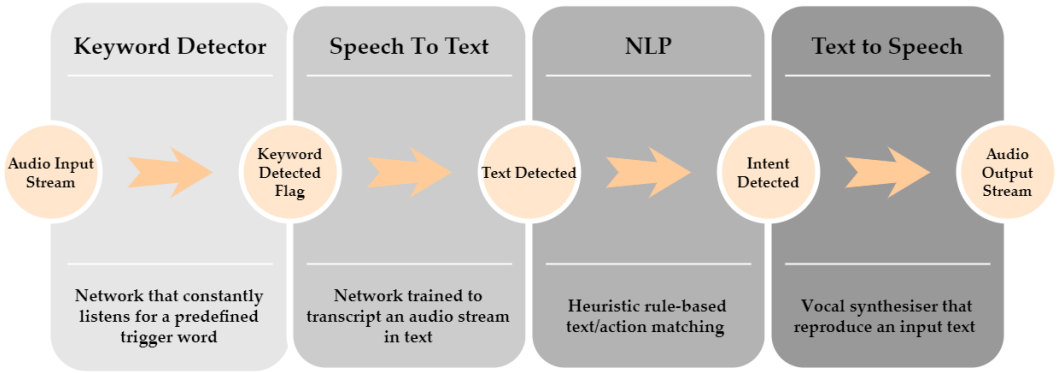


Fig. 6.8 Overview of PIC4Speech vocal assistant architecture. The scheme describes the successive cascade activation of the different components of the vocal assistant pipeline.

1. The first component is the *keyword detector* [170], which constantly monitors the input audio stream in search of the specific triggering command. In this specific case, that word is the name of the robot: *"Marvin"*.
2. Once the trigger word is detected, a second model performs a *speech-to-text* operation. We exploited the Vosk offline speech recognition API [171] for this block which gives the flexibility to switch language and has ample community support. It continuously analyzes the input audio stream until the volume is below a certain threshold and performs the transcription.
3. The transcribed text is subsequently passed to a *natural language processing (NLP)* algorithm that matches the input with a certain number of predefined intents. The recognized robot action is therefore sent to the robot control framework.
4. The response of the vocal assistant is also given to the user with a *text-to-speech* process. Each OS comes with a default vocal synthesizer that can directly access the speakers.

More in detail, the keyword detection is performed with a DNN based on a Vision Transformer that constantly listens to the audio stream, looking for the target command. First, the mel-scale spectrogram is extracted from each sample of the input stream. These features are treated as visual information and, therefore, they are processed with a Vision Transformer [67], a state-of-the-art model for image classification. We re-trained the keyword detector from scratch on the Speech Commands dataset [172], constituted by 1-second-long audio samples from 36 classes:

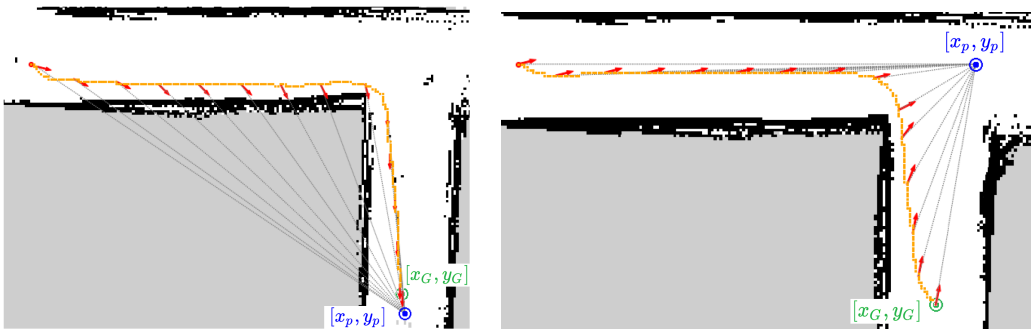
35 standard keywords plus a silence/noise class. The re-train model achieved a test accuracy of 97% over the different classes on the 11,005 test samples of the Speech Commands dataset. Specifically, for the current target class 'Marvin', we get the results reported in Table 6.1, evaluating the performance with standard classification metrics: $Precision = TP/(TP+FP)$, $Recall = TP/(TP+FN)$ and $F1score = 2 \cdot (recall \cdot precision)/(recall + precision)$. Thanks to the multi-class approach, the keyword can be changed at run-time. Being constantly active, it is of primary importance that this network consumes less energy as possible, but at the same time, it is capable of maintaining a good compromise between false positive and false negative detection. At the same time, the network should deal with different sound environments and noise levels. Further improvements to the keyword detector can be reached by augmenting the training set with newly generated samples to increase the robustness of the model in crowded, noisy environments. Here, a simple rule-based matching mechanism is used for the NLP stage. Although it represents a simple approach, a good level of flexibility is guaranteed by the actual solution as the user can introduce new actions for the robotic platform associated with several indicative sentences. A more advanced version may be developed with the aim of semantically matching the encoded query text and the robot actions, using a universal sentence encoder [173, 174].

Table 6.1 Classification results of the target keyword 'Marvin' on the 11,005 test samples of the Speech Commands dataset. Standard classification metrics are used for the evaluation: $Precision = TP/(TP+FP)$, $Recall = TP/(TP+FN)$ and $F1score = 2 \cdot (Recall \cdot Precision)/(Recall + Precision)$.

Keyword Detector Classification Metrics	**Results**
True Positives (TP)	189
True Negatives (TN)	10,806
False Positives (FP)	4
False Negatives (TN)	6
F1 Score	0.9742
Precision	0.9793
Recall	0.9692

Different from commercial vocal assistants, which require a stable internet connection, PIC4Speech works completely offline, running uniquely on the hardware resources of the platform. This competitive advantage derived from the choice of lightweight models in the algorithms pipeline prevents Marvin from exposing the visual and audio data of the domestic environment to internet-derived risks.

6.6 Navigation Experiments and Results



(a) *The person is close to the navigation goal.* (b) *The person is at the corner of the hallway.*

Fig. 6.9 Omnidirectional person-centered navigation results in two scenarios with the person in different positions: in (a) the person is close to the navigation goal, in (b) the person is at the corner of the hallway. Red arrows indicate position and orientation of the rover at different time instants, the blue point is the person's position, while the orange spline represents the path crossed by the rover.

Two different kinds of experiments are conducted to validate the navigation capabilities of Marvin:

1. the first experimental stage aims at demonstrating the efficiency of the person-centered navigation task for monitoring purposes, where the rover has to navigate from a point A to a target point B of coordinate (x_G, y_G), maintaining its focus on the subject located in (x_P, y_P);
2. the second series of experiments tests the person following task, where (x_G, y_G) and (x_P, y_P) coincide and represent the dynamic goal obtained from the visual perception pipeline, which identifies and tracks the person of interest.

For each tested scenario, tests are performed with Marvin in two different configurations. In the first configuration, the rover adopts our decouple navigation system: it plans collision-free trajectories fully exploiting its omnidirectional kinematics,

combining both the two linear velocities $[v_x, v_y]$. The angular yaw velocity ω is controlled by the person tracking module to always maintain visual contact with the followed person. In the second configuration, the rover behaves like a differential platform. This means it can only exploit velocity v_x, while control of velocity v_y is denied, and the angular yaw velocity ω is solely dedicated to navigation purposes. This procedure allows the comparison of performances between our solution and a generic differential platform in tracking the user.

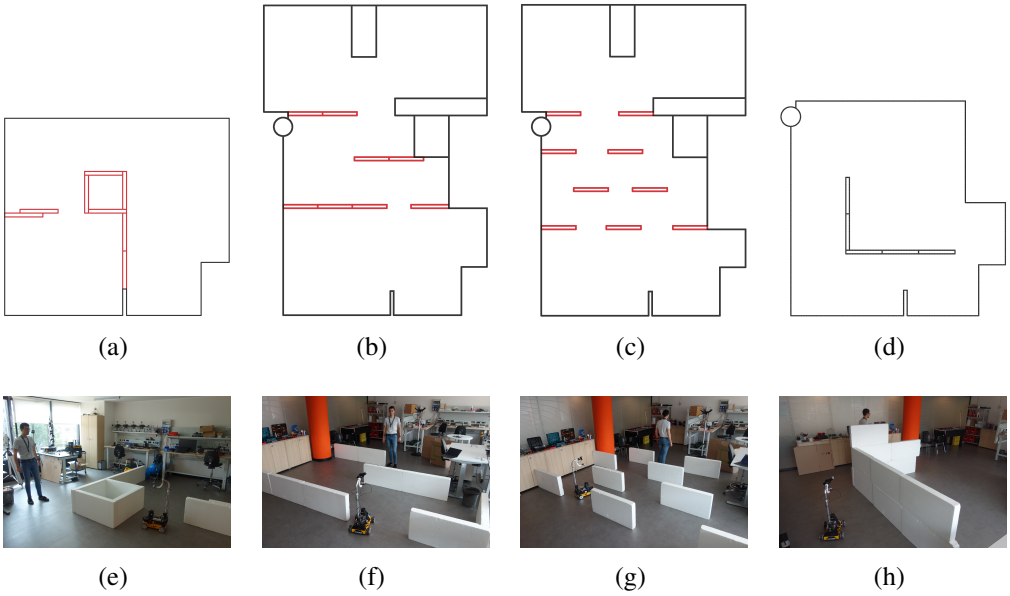


Fig. 6.10 Qualitative visualization of the four scenarios set up for the person following test. In the upper row, a schematic representation is shown, where red objects represent low-height obstacles over which the robot's camera can see. In the lower row, the real testing area with the robot is shown.

6.6.1 Person-centered navigation

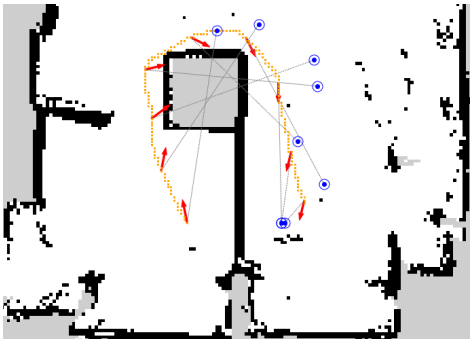
Tests are performed in two different scenarios composed of a 90° hallway with low walls, which represent any potential obstacle present in a realistic domestic scene. The rover camera can see over the walls, but the platform is forced to avoid them in order to reach its goal. The starting point and the destination (x_G, y_G) are the same in the two cases. What changes is the position of the person (x_P, y_P): near the destination point in the first scenario (Fig. 6.9a), and in the corner of the hallway in the second (Fig. 6.9b). In these preliminary trials, the person maintains their position

Table 6.2 Results obtained from the person-centered navigation test are expressed in terms of mean angular difference $\Delta\theta$, its standard deviation, root mean square error (RMSE), and mean absolute error (MAE) considering $\Delta\theta = 0$ as the optimal value. The person is located close to the destination point in the first scenario (Fig. 6.9a) and in the corner of the hallway in the second (Fig. 6.9b). Contrary to the differential configuration, omnidirectional motion drastically reduce the maximum error $\Delta\theta$ between the orientation of the rover and the person during the navigation.

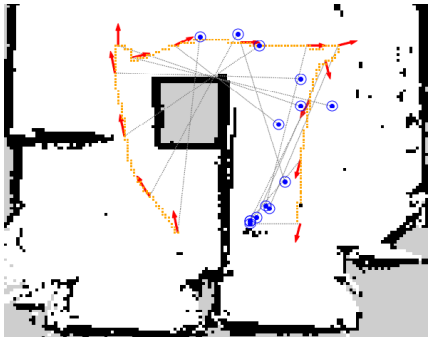
Scenarios	$\Delta\theta$ **Error**	**Mean**	**Std.Dev.**	**RMSE**	**MAE**
1	Omnidir.	-2.88	4.63	5.47	4.32
	Differential	-32.75	28.71	43.55	33.94
	Improvement	**91.21%**	**83.87%**	**87.44%**	**87.27%**
2	Omnidir.	-2.23	3.98	4.58	2.51
	Differential	-75.08	79.88	109.62	75.08
	Improvement	**97.03%**	**95.02%**	**95.82%**	**96.66%**

during the whole extent of the test. The rover odometry data are acquired with a frequency of 5Hz.

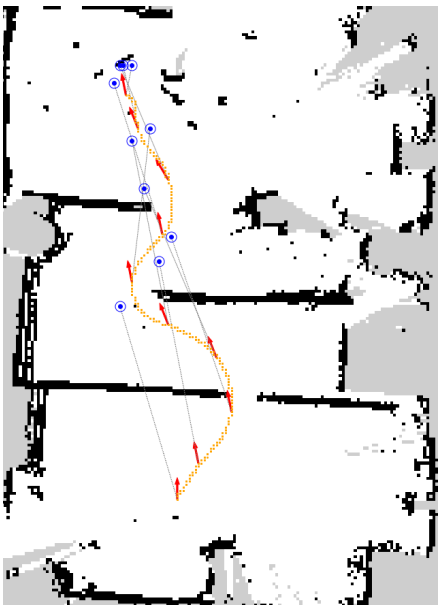
Seven tests are performed for each scenario and both configurations, omnidirectional and differential. The error term is represented by the angular difference $\Delta\theta$ between the orientation of the rover and the orientation of the vector connecting the rover's center of rotation with the person's position. The horizontal FOV of the RealSense D435i (RGB stream) is equal to 69°. The angular difference $\Delta\theta$ should never be higher than half this angle, approximately 34.5°, to constantly keep track of the person's position.



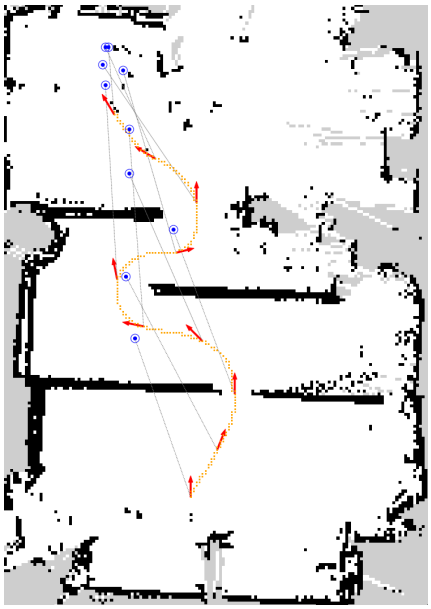
(a) *Scenario 1 - Omnidirectional configuration*



(b) *Scenario 1 - Differential configuration*



(c) *Scenario 2 - Omnidirectional configuration*

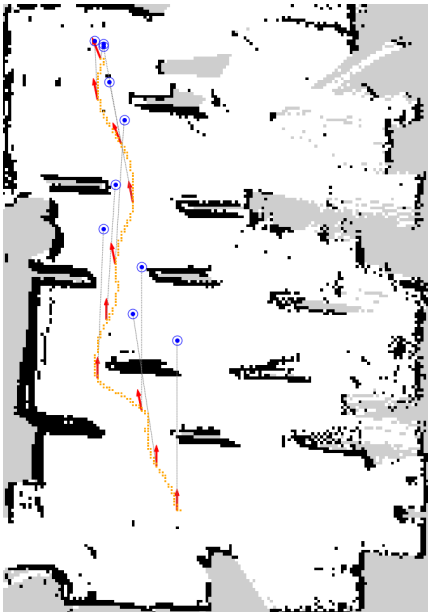


(d) *Scenario 2 - Differential configuration*

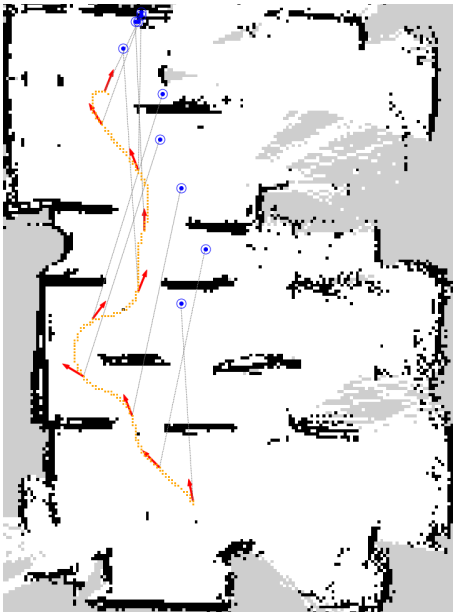
Fig. 6.11 Person following results in the first two scenarios: scenario 1 is composed of a wide U-shaped path, while scenario 2 presents narrow passages through obstacles. Red arrows indicate position and orientation of the rover associated with the person's position (blue point) at the same instant. The orange spline represents the path crossed by the rover.

The metrics considered for each test are the average angular difference $\Delta\theta$ with its standard deviation, the root mean square error (RMSE), and the mean absolute error (MAE) maintained along the whole path, considering $\Delta\theta = 0$ as the optimal value. In Table 6.2 are reported, for each scenario and each metric, the average value computed over all the different tests, and the percentage of improvement introduced

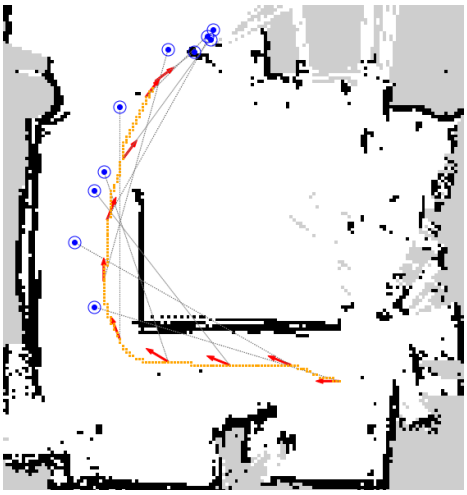
by the proposed method. As seen from the results and Fig. 6.9, the omnidirectional system is able to efficiently navigate towards the goal, constantly maintaining its orientation towards the person. The $\Delta\theta$ angular error is kept at extremely low average values equal to -2.88 and -2.23 respectively in the two scenarios. Furthermore, the maximum recorded value of $\Delta\theta$ does not exceed 17°, which is well below the limit of 34° imposed by the camera's FOV. This means the system can keep tracking the person for the whole extent of the navigation. Moreover, from data collected during the experimentation, the perception and tracking system was able to correctly recognize and localize the followed person within the environment on average 29 times per second. On the other hand, velocity commands are provided with frequencies over 15 fps at any time. For comparison, we also added the results obtained with the differential drive configuration. Its limitation is particularly evident in the second scenario, where the person and the goal have two completely different positions.



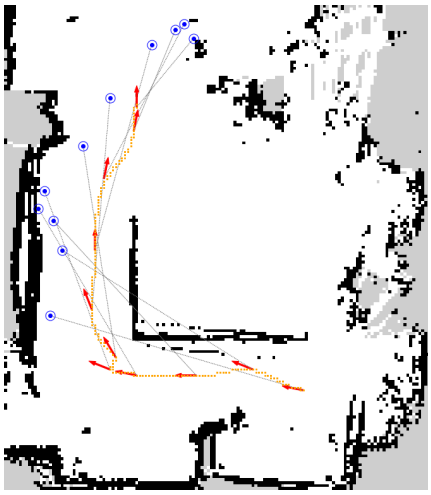
(a) *Scenario 3 - Omnidirectional configuration*



(b) *Scenario 3 - Differential configuration*



(c) *Scenario 4 - Omnidirectional configuration*



(d) *Scenario 4 - Differential configuration*

Fig. 6.12 Person following results in the third and fourth scenario: scenario 3 presents a high number of obstacles and possible paths, while scenario 4 is composed of a high 90° wall to be circumnavigated. Red arrows indicate position and orientation of the rover associated with the person's position (blue point) at the same instant. The orange spline represents the path crossed by the rover.

6.6.2 Person following

Table 6.3 Results obtained from the person following test in four different scenarios are expressed in terms of mean angular difference $\Delta\theta$, its standard deviation, root mean square error (RMSE), and mean absolute error (MAE) considering $\Delta\theta = 0$ as the optimal value. Our omnidirectional planning and control system clearly demonstrates a performance gap in keeping the tracking of the person while following its motion: the $\Delta\theta$ error is drastically reduced in comparison with a differential drive navigation.

Scenario	$\Delta\theta$ **Error**	**Mean**	**Std.Dev.**	**RMSE**	**MAE**
A	Omnidir.	2.99	10.54	11.00	8.98
	Differential	16.00	63.41	68.31	57.20
	Improvement	**81.31%**	**83.38%**	**83.90%**	**84.30%**
B	Omnidir.	−4.09	8.75	9.93	8.19
	Differential	−15.67	53.99	58.48	50.11
	Improvement	**73.90%**	**83.79%**	**83.02%**	**83.66%**
C	Omnidir.	0.31	8.28	8.81	6.93
	Differential	12.34	42.19	45.05	37.38
	Improvement	**97.49%**	**80.37%**	**80.44%**	**81.46%**
D	Omnidir.	4.46	11.84	12.84	10.10
	Differential	27.66	20.95	35.07	29.19
	Improvement	**83.88%**	**43.48%**	**63.39%**	**65.40%**

For the person following task, tests are performed in four different scenarios. The geometric configuration can be seen in Fig. 6.10. Similar to the previous test stage, obstacles are constituted by low walls, except for the fourth, where they are full-height walls. Contrary to the previous case, the person to be followed moves for the whole extent of the test. For this reason, a ground truth data collection system is used, localizing the person and the rover with ultra-wideband tag signals. Four ultra-wideband anchors have been placed at the corner of the rectangular testing area. The rover's orientation is aligned with the one used by the ultra-wideband system. This allows us to correctly compute the angular difference $\Delta\theta$ between rover and person at any time instant. To our knowledge, this experimental setting is the first attempt in the literature to quantitatively measure a person's quality following system performance, going beyond the typical qualitative evaluation.

As already done for the first test, seven validation runs are performed for each rover configuration in every scenario. The same error term $\Delta\theta$ and metrics discussed in the previous section are used to evaluate the person following performances. Results can be consulted in Table 6.3. Furthermore, in Fig. 6.11 and Fig. 6.12, for each

scenario and each configuration, a visualization of the performed test is reported. The gridmaps reported in the figure are directly obtained from the rover during the navigation, while rover and person poses are obtained from the ultra-wideband system. As can be seen, our methodology proves to robustly track the followed person more effectively than a traditional differential drive navigation in all the considered scenarios. In the omnidirectional configuration (Fig. 6.11a,6.11c,6.12a,6.12c) the rover manages to always maintain the user within the camera's view, contrary to the differential drive case, where the visual contact is instead lost several times. This generally leads to higher performance in following the user, fully satisfying the person monitoring requirement. The obtained values of $\Delta\theta$ clearly show the performance gap in all scenarios, demonstrating the successful behavior in monitoring the person provided by our solution. Also in the fourth scenario (Fig. 6.12c,6.12d), where after the curve the wall obstructs the rover's view of the user, it appears clear that the ability to remain facing the human dynamic goal allows for a more accurate re-acquisition of tracking as soon as the obstacle is passed. In this last scenario, the differential drive system registers the highest orientation error, with a substantial $\Delta\theta$ average gap from our solution.

6.7 Experimental Demo

Finally, a qualitative demonstration has been conducted at Officine Edison, Milan, during an institutional presentation specifically organized to test and show Marvin's overall capabilities. The demonstration took place in an area called Domus (Fig. 6.13), which simulates a real domestic environment made up of a kitchen, bedroom, living room, and bathroom. Like a normal house, the Domus features different obstacles of various heights and dimensions, and rooms are separated by regular size doors.

In the setup phase, Marvin was guided in each of the different rooms and their relative positions were saved with respect to the starting point, where a docking station for recharging could eventually be placed. Moreover, a telephonic number was memorized for the emergency call task. In the demonstration, all the functionalities of the robot were tested, and a qualitative analysis was conducted. From the starting point (Fig. 6.13a), the rover was asked to autonomously reach the bedroom waypoint, passing through the double-leaf door. Here, the user monitor function was demonstrated, showing how Marvin was able to correctly classify the pose

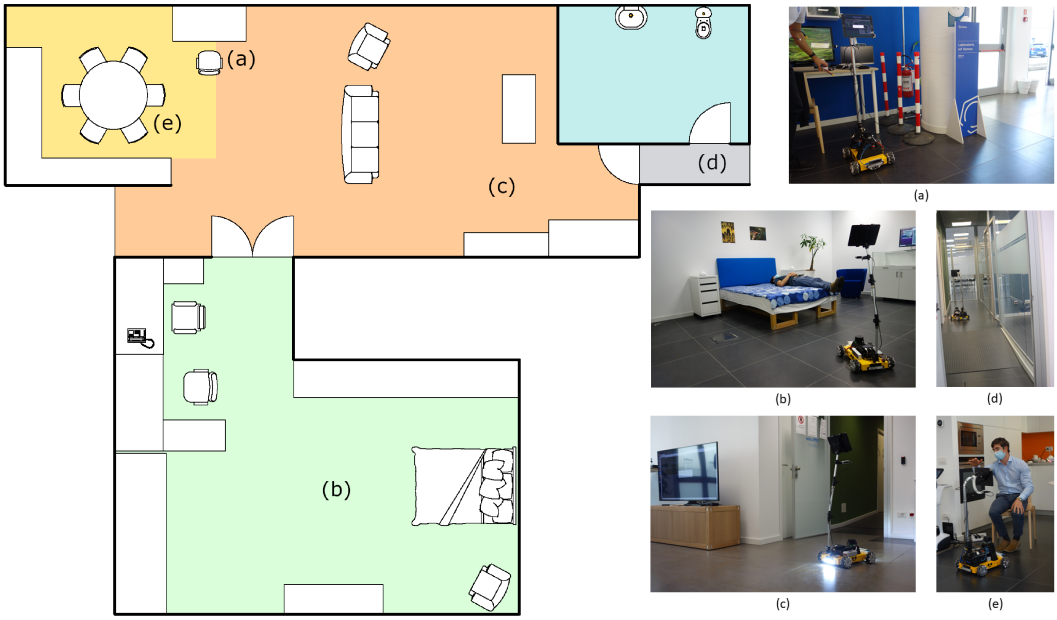


Fig. 6.13 Simplified map of the Domus area at Officine Edison, Milan, with the four rooms, kitchen, living room, bedroom, and bathroom, respectively in yellow, orange, green, and blue. Letters on the map indicate the various goals saved in the environment, associated with the corresponding image: (**a**) starting point, (**b**) bedroom keypoint, with user's pose recognition, (**c**) living room keypoint, with lights turned on, ready for night assistant task (**d**) bathroom keypoint, (**e**) kitchen keypoint, with demonstration of the positioning device capabilities.

of the visualized user, standing, sitting on an armchair, or laying in the bed (Fig. 6.13b). In addition, it was also demonstrated how, after a request from the user, the system was capable of connecting with the pre-configured telephone to call the emergency number. Then, the rover was asked to follow the user from the bedroom to the living room (Fig. 6.13c). Later, the user activated the night assistance task by asking the rover to accompany him to the bathroom, causing the robot to turn on the on board lights (Fig. 6.13d). Finally, Marvin was asked to reach the kitchen and to adjust the inclination and height of the positioning device to adapt to the user, sitting on the chair, so that the mounted tablet could be more easily accessed and operated (Fig. 6.13e). Marvin successfully completed the demo addressing all the service function required for domestic assistance. Thanks to its mobility and the elevated position of the camera, mounted on the positioning device, the rover managed to efficiently track and follow the person, albeit the several obstacles with diverse heights.

Part III

Autonomous Navigation for Precision Agriculture

Chapter 7

A Deep Learning Pipeline for Autonomous Navigation in Row-based Crops

Agriculture 3.0 and 4.0 paradigms recently guided technological innovation in precision farming to focus the research activity on four essential requirements: increasing productivity, allocating resources reasonably, adapting to climate change, and avoiding food waste [175]. One fundamental step for introducing an efficient and reliable automation in the agriculture processes is the development of an autonomous navigation pipeline for vehicles and robots. This is the first requirement to successfully design a platform that takes care of several tasks such as harvesting [176], spraying [177, 178], vegetative assessment and yield estimation [179, 180], and many others [181, 182]. In this part of the thesis, we focus on the development of a complete autonomous navigation system for an Unmanned Ground Vehicle (UGV) to be used in row-based crops such as vineyards and orchards. Row-based crops account for about 75% of all planted acres of agriculture in the United States [183]. Nowadays, autonomous navigation in agricultural contexts is tackled with the usage of expensive high-precision GNSS sensors, such as GPS receivers with RTK corrections [184, 185], usually in combination with laser sensors [186, 187]. However, the presence of thick canopies and lush vegetation decreases the reliability of GNSS sensors [188, 189], especially during spring and summer. This condition strengthens the need for alternatives to reduce the cost of the system without affecting its robustness.

Visual odometry [190–192] and computer vision methods [193] have been proposed as different valid options to tackle navigation inside fields. However, these methods generalize poorly in the case of long outdoor paths with repetitive visual patterns.

Deep Learning (DL) methods are spreading as powerful solutions to tackle many different precision agriculture tasks such as fruits detection [194, 195], counting [196], land crop classification [24, 197]. Recently, DL solutions have been proposed to solve the autonomous navigation problem, overcoming the limits of localization in row crops scenarios with methods of orientation classification [198], plant segmentation [199, 20].

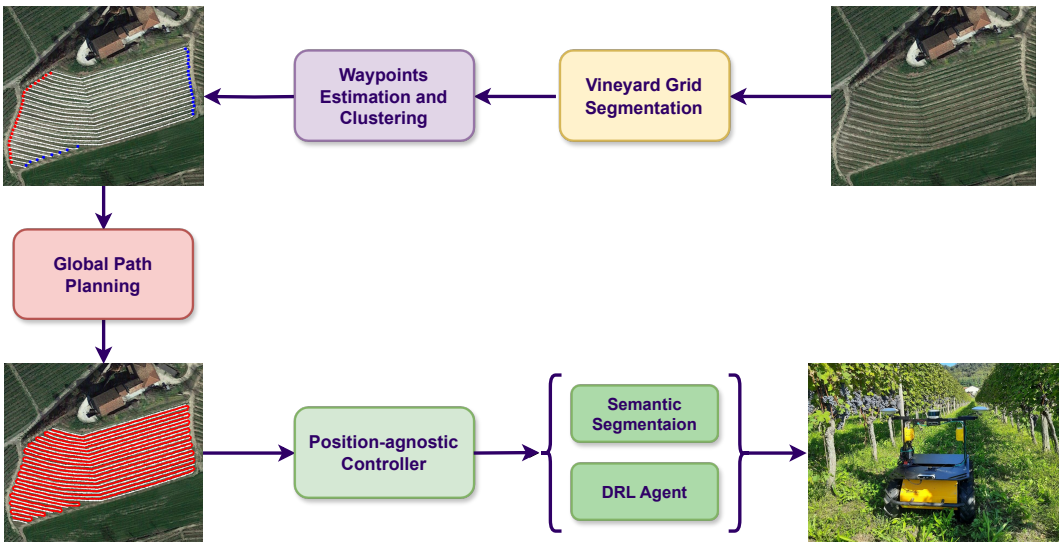


Fig. 7.1 A representation of the overall pipeline. First, an occupancy grid of the crop is generated to estimate and cluster the waypoints at start/end of vineyards row. Thus, a global path can be planned. Finally, a visual-based local controller policy computes the velocity commands to guide the robot inside each row of the field.

The Navigation Pipeline

A DL-based pipeline is proposed here, composed of multiple modules to tackle the full navigation problem in row-based crops, as illustrated in Fig. 7.1. The proposed pipeline is based on the intuitive idea to decouple the navigation in two stages: inside and outside the plant rows. This choice is driven by the fact that robot localization is usually hindered by the environment inside the rows, while standard methods can be used to navigate outside. Firstly, an aerial image of the field is segmented to generate a binary grid representation of the crop rows. Then, the grid is used to estimate

and cluster the waypoints at the starting/ending point of each row. The clustered waypoints can be exploited as reference positions to trigger the switch from the intra-row navigation logic to the inter-rows one. Moreover, a complete global path can be obtained through the waypoints with a standard planning algorithm such as A* [200]. In this chapter, we focus on the development of different visual-based controller that allow to guide the robot along the intra-row segments of the row-based field and avoid a costly sensorization of the robotic platform for localization purposes. In Chapter 8 the global waypoints generation and clustering process will be explained. The navigation of the robot outside the rows, from one row's ending waypoint to the next starting one can be tackled with a classic local planner relying on odometry or GNSS positioning signal, as demonstrated in [17]. GNSS localization can still be hindered by adverse weather conditions, however, the path to be covered outside the rows is usually short and does not present the obstruction of plants canopies. Future work, currently under development, will investigate alternative methods to control the robot without localization also outside the rows.

Position-agnostic visual control methods

The competitive advantage of GPS-free visual approaches has been partially investigated in the last years of service robotics research. Different computer vision and machine learning techniques have been applied so far to improve the performance of robot guidance. A first vision-based approach was proposed in [201] using mean-shift clustering and the Hough transform to segment RGB images and generate the optimal central path. Later, [202] achieved promising results using multispectral images and simply thresholding and filtering on the green channel. Recently, DL approaches have been successfully applied to the task. [203, 204] proposed a classification-based approach in which a model predicts the discrete action to perform. In contrast, [198, 199] proposed a proportional controller to align the robot to the center of the row using heatmaps of the scene first and segmented images in the latest version. In Section 7.1 a family of enhanced semantic segmentation based controllers is presented, while in Section 7.2 a Deep Reinforcement Learning paradigm is adopted.

7.1 Semantic Segmentation-based control

Although previously proposed local controllers proved effective in their testing scenarios, they have only been applied in crops where a full view of the sky favors both GPS receivers [205], and vision-based algorithms to distinguish plants canopies form the background [191]. Moreover, the increasing necessity of open access datasets to train DL models guided researchers to build wide and reliable synthetic datasets for crop semantic segmentation [18, 25].

7.1.1 Methodology

To navigate high-vegetation orchards and arboriculture fields, this study provides a real-time control algorithm with two variations, which enhances the method described in [199]. The proposed method completely avoids the employ of GPS localization, which can be less accurate due to signal reflection and mitigation due to high and thick vegetation. Therefore, our algorithms consist of a straightforward operating principle, which exclusively employs RGB-D data and processes it to obtain effective position-agnostic navigation. It can be summarized in the following four steps:

1. Semantic segmentation of the RGB frame, with the purpose of identifying the relevant plants in the camera's field of view.
2. Addition of the depth data to the segmented frame to enhance the spatial understanding of the surrounding vegetation of the robot.
3. Searching for the direction towards the end of the vegetation row, given the previous information.
4. Generation of the velocity commands for the robot to follow the row.

However, the two suggested approaches only vary in steps 2 and 3, where they utilize depth frame data and generate the robot's desired direction. Conversely, the segmentation technique 1 and the command generation 4 are executed in a similar manner. A visual depiction of the proposed pipeline is illustrated in Fig. 5.1.

The first step of the proposed algorithm, at each time instant t consists in acquiring an RGB frame $\mathbf{X}^t_{rgb}$ and a depth frame $\mathbf{X}^t_d$, where $\mathbf{X}^t_{rgb} \in \mathbb{R}^{h\times w\times c}$ and $\mathbf{X}^t_d \in \mathbb{R}^{h\times w}$. In both cases, h represents the frame height, w represents the frame width, and c

is the number of channels. The RGB data received is subsequently inputted into a segmentation neural network model H_{seg}, yielding a binary segmentation mask that conveys the semantic information of the input frame.

$$\hat{\mathbf{X}}^t_{seg} = H\left(\mathbf{X}^t_{rgb}\right) \tag{7.1}$$

where $\hat{\mathbf{X}}^t_{seg}$ is the obtained segmentation mask.

Furthermore, the segmentation masks from the previous N time instances, ranging from $t-N$ to t, are combined to enhance the robustness of the information.

$$\hat{\mathbf{X}}^t_{CumSeg} = \bigcup_{j=t-N}^{t} \hat{\mathbf{X}}^j_{seg} \tag{7.2}$$

where, $\hat{\mathbf{X}}^t_{CumSeg}$ denotes the cumulative segmentation mask, and the symbol $\bigcup$ signifies the logical bitwise *OR* operation applied to the last N binary frames.

Moreover, the depth map $\mathbf{X}^t_d$ is employed to assess the segmented regions between the camera position and a specified depth threshold d_{th}. This process helps eliminate irrelevant information originating from distant vegetation, which has no bearing on controlling the robot's movement.

$$\hat{\mathbf{X}}^t_{segDepth\,\substack{i=0,\dots,h\\ j=0,\dots,w}}(i,j) = \begin{cases} 0, \text{ if } \hat{\mathbf{X}}^t_{CumSeg(i,j)} \cdot \hat{\mathbf{X}}^t_{d(i,j)} > d_{th} \\ 1, \text{ if } \hat{\mathbf{X}}^t_{CumSeg(i,j)} \cdot \hat{\mathbf{X}}^t_{d(i,j)} \leq d_{th} \end{cases} \tag{7.3}$$

where, $\hat{\mathbf{X}}_{segDepth}$ represents the resultant intersection of the cumulative segmentation frame and the depth map, restricted to a distance threshold of d_{th}.

From this point forward, the proposed algorithm diverges into two variants, namely, *SegMin* and *SegMinD*, as elaborated in the following sections 7.1.1 and 7.1.1, respectively.

SegMin

The initial variant refines the methodology introduced in [199]. Following the segmentation mask processing, a column-wise summation is executed, generating a histogram $\mathbf{h} \in \mathbb{R}^w$ that characterizes the vegetation distribution along each column

as in the following formula:

$$\mathbf{h}_j = \sum_{i=1}^{h} \hat{\mathbf{X}}_{segDepth(i,j)} \tag{7.4}$$

where $i = 0, \ldots, w$ is the index along the vertical direction of each frame column. Subsequently, a moving average is applied to this histogram using a window of size n to enhance robustness by smoothing values and mitigating punctual noise from previous passes. In an ideal scenario, the minimum value x_h in this histogram corresponds to regions with minimal vegetation, effectively pinpointing the central path within the crop row. If multiple global minima are identified, indicating areas with no detected vegetation, the mean of these points is calculated and considered as the global minimum. This approach ensures a more reliable identification of the continuation of the row, accommodating variations in the vegetation distribution.

SegMinD

The second proposed methodology presents a variation of the earlier algorithm tailored specifically for wide rows featuring tall and dense canopies. In such scenarios, the initial algorithm might encounter challenges in determining a clear global minimum, as the consistent presence of vegetation above the robot complicates the interpretation. This variant addresses this issue by incorporating a multiplication operation between the previously processed segmentation mask and the normalized inverted depth data.

$$\hat{\mathbf{X}}^t_{depthInv} = \hat{\mathbf{X}}^t_{segDepth} \bigcap \left(1 - \frac{\mathbf{X}^t_d}{d_{th}}\right) \tag{7.5}$$

where, $\hat{\mathbf{X}}^t_{depthInv}$ is the outcome of an element-wise multiplication, denoted by $\bigcap$, involving the binary mask $\hat{\mathbf{X}}^t_{segDepth}$ and the depth frame $\hat{\mathbf{X}}^t_d$ that has been normalized over the depth threshold d_{th}. Similar to the previous scenario, a column-wise summation is executed to derive the array $\mathbf{h}$, followed by a smoothing process using a moving average. This introduced modification serves a crucial purpose by allowing elements closer to the robot to exert a more significant influence, thereby enhancing the algorithm's ability to discern the direction of the row. The different sum histograms obtained with SegMin and SegMinD are directly compared in Fig.

7.2, showing the sharper trend and the global minimum isolation obtained, including the depth values.

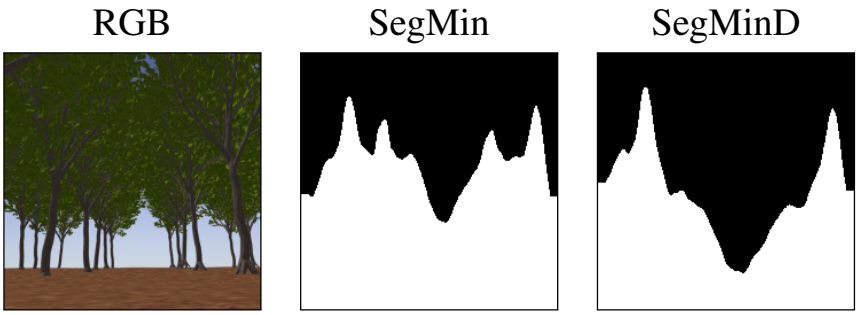


Fig. 7.2 Contrasting the histograms produced by the two distinct algorithms, considering the RGB frame on the right, reveals that SegMinD provides a more defined and less ambiguous global minimum point.

Segmentation Network

A prior study on crop segmentation in real-world conditions provided the neural network design that was chosen [199]. Fig. 7.3 illustrates its entire architecture and its primary benefit is its ability to leverage rich contextual information from the image at a lower computing cost. A MobileNetV3 backbone makes up the network's initial stage, which is designed to efficiently extract the visual features from the input image [206]. With squeeze-and-excitation attention sub-modules [207], it is comprised of a series of inverted residual blocks [208]. They increase the amount of channel features while gradually decreasing the input image's spatial dimensions. It is succeeded by a Lite R-ASPP (LR-ASPP) module [209], an enhanced and condensed variant of the Atrous Spatial Pyramid Pooling module (R-ASPP) that upscales the extracted features via two parallel branches. The first lower the spatial dimension by $1/16$ by applying a Squeeze-and-Excite sub-module to the final layer of the backbone. To modify the number of channels C to the output segmentation map, a channel attention weight matrix is produced, multiplied by the unpooled features, and then upsampled and fed through a convolutional layer. The second branch takes characteristics from an earlier stage of the backbone, which reduces the spatial dimension by $1/8$, and adds them to the output of the upsampling step, mixing lower-level and higher-level patterns in the data. The network's input has a dimensionality equal to $W \times H \times 3$, while the segmented output is equal to $W \times H$.

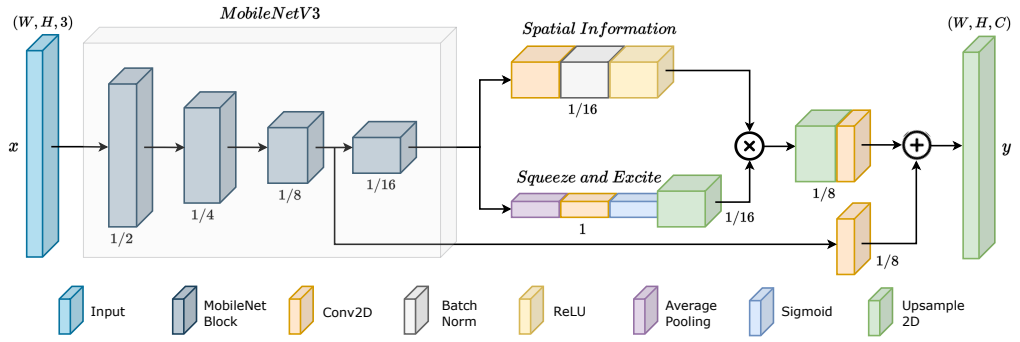


Fig. 7.3 The Deep Neural Network utilized in this study features a backbone of MobileNetV3 and an LR-ASPP head, as detailed in [206]. The spatial scaling factor of the features in comparison to the input size is provided beneath each block.

Furthermore, the output values of the neural network are scaled between 0 and 1 using a sigmoid function, as this work primarily focuses on the semantic segmentation of plant rows. The usual cross-entropy loss between the ground-truth label y and the anticipated segmentation mask is used to train the DNN:

$$L_{\text{CE}}(y, \hat{y}) = -\sum_{i=1}^{N} y_i \cdot log(\hat{y}_i) \tag{7.6}$$

which for binary segmentation becomes a simple binary cross-entropy loss.

During both the validation and testing phases, the DNN performance is evaluated through an intersection over unit (IoU) metrics:

$$mIoU(\theta) = \frac{1}{N}\sum_{i=0}^{N}\left(1 - \frac{\hat{X}^i_{seg} \cap X^i_{seg}}{\hat{X}^i_{seg} \cup X^i_{seg}}\right) \tag{7.7}$$

where X^i_{seg} is the ground truth mask, $\hat{X}^i_{seg}$ is a predicted segmentation mask, and θ is the vector representing the network parameters. Since there are only plants as the target class of interest, N in the IoU computation always equals 1. The model is trained on the AgriSeg synthetic dataset [18][1]. Further details on the training strategy and hyperparameters are provided in Section 7.1.2.

[1] https://pic4ser.polito.it/AgriSeg

Robot heading control

The goal of the controller pipeline is to maintain the mobile platform at the center of the row, which, in this study, is equated to aligning the row center with the middle of the camera frame. Consequently, following the definition in the preceding step, the minimum of the histogram should be positioned at the center of the frame width. The distance d from the frame center to the minimum is defined as:

$$d = x_h - \frac{w}{2} \tag{7.8}$$

The generation of linear and angular velocities is accomplished using custom functions, mirroring the approach employed in [17].

$$v_x = v_{x,max} \left[1 - \frac{d^2}{\left(\frac{w}{2}\right)^2} \right] \tag{7.9}$$

$$\omega_z = -k_{\omega_z} \cdot \omega_{z,max} \cdot \frac{d^2}{w^2} \tag{7.10}$$

where, $v_{x,max}$ and $\omega_{z,max}$ represent the maximum attainable linear and angular velocities, and k_{ω_z} serves as the angular gain controlling the response speed. To mitigate abrupt changes in the robot's motion, the ultimate velocity commands $\bar{v}x$ and $\bar{\omega}z$ undergo smoothing using an Exponential Moving Average (EMA), expressed as:

$$\begin{bmatrix} \bar{v}_x^t \\ \bar{\omega}_z^t \end{bmatrix} = (1-\lambda) \begin{bmatrix} \bar{v}_x^{t-1} \\ \bar{\omega}_z^{t-1} \end{bmatrix} + \lambda \begin{bmatrix} v_x^t \\ \omega_z^t \end{bmatrix} \tag{7.11}$$

where, t represents the time step, and λ stands for a selected weight.

7.1.2 Experiments and Results

Segmentation Network Training and Evaluation

We train the crop segmentation model using a subset of the AgriSeg synthetic segmentation dataset [18, 25]. In particular, for the pear trees and apple trees, we train on generic tree datasets in addition to pear and apples; for vineyards, we train on vineyard and pergola vineyards (note that the testing environments are different from

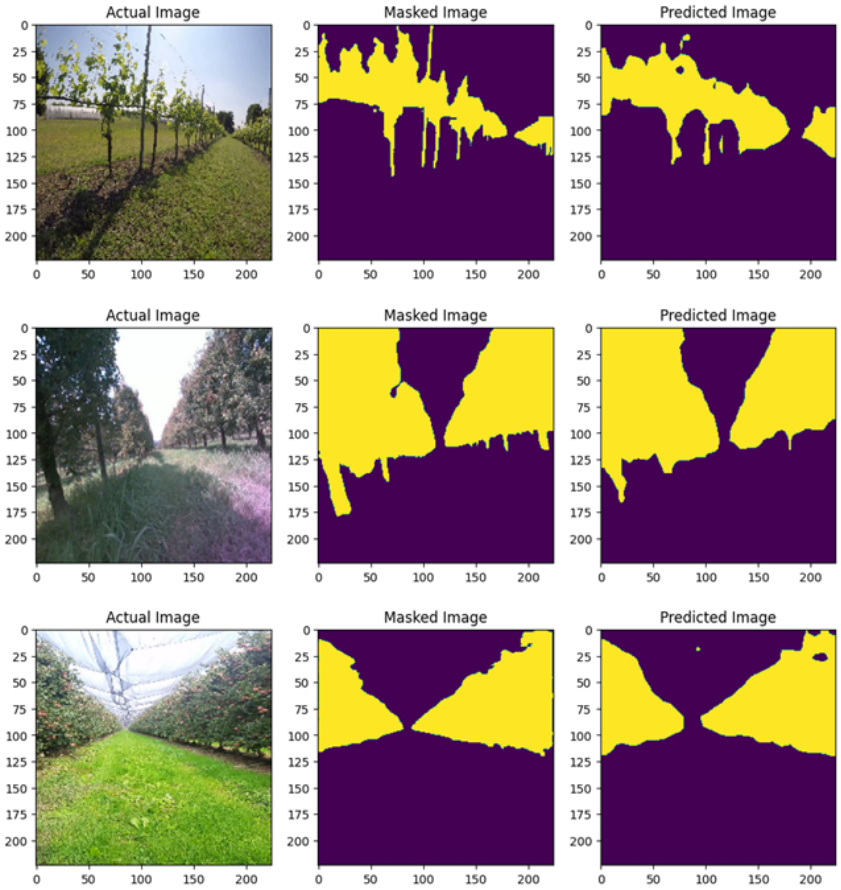


Fig. 7.4 Test of semantic segmentation DNN on real-world test samples from vineyard (top), pear trees (middle) and apple trees (bottom) fields. For each crop, RGB input image (left), ground truth mask (center) and the predicted mask (right) are reported.

the ones from which the training samples are generated). Only 100 miscellaneous real images of different crop types are added to the training dataset in all the cases. Thanks to the high-quality rendering of the AgriSeg dataset, this small amount of real images is sufficient to reach generally good performance in real-world conditions. A more in depth analysis of generalization properties of the semantic segmentation network is discussed in Chapter 10 In both cases, the model is trained for 30 epochs with Adam optimizer and learning rate 3×10^{-4}. We apply data augmentation by randomly applying cropping, flipping, greyscaling, and random jitter to the images. Our experimentation code is developed using TensorFlow as the deep learning framework. We train models starting from ImageNet pretrained weights, so the input size is fixed to (224×224). All the training runs are performed on a single Nvidia RTX 3090 graphic card.

Table 7.1 Semantic Segmentation results on real images in different crop fields. For each crop, a model has been trained on synthetic data, only using 100 additional real images containing miscellaneous crops different from the test set.

Model	Real Test IoU	Train Data	Real Test Data
Vineyard	0.6950	13840	500
Apples	0.8398	15280	210
Pear	0.8778	7980	140

Table 7.1 reports the results obtained testing the trained segmentation DNN on real images in terms of Intersection over Union (IoU). Fig. 7.4 also shows some qualitative results on sample images collected on the field during the test campaign.

Simulation Environment

The proposed control algorithm underwent testing using the Gazebo simulation software[2]. Gazebo was chosen due to its compatibility with ROS 2 and its ability to integrate plugins simulating sensors, including cameras. A Clearpath Jackal model was employed to evaluate the algorithm's performance. The URDF file from Clearpath Robotics, containing comprehensive information about the robot's mechanical structure and joints, was utilized. In the simulation, an Intel Realsense D435i plugin was employed, placed 20 cm in front of the robot's center, and tilted upward by 15°: this configuration enhanced the camera's visibility of the upper branches of trees. The assessment of the navigation algorithm took place in four customized simulation environments, each designed to mirror distinct agricultural scenarios. These environments included a conventional vineyard, a pergola vineyard characterized by elevated vine poles and shoots above the rows, a pear field populated with small-sized trees, and a high-tree field where the canopies interweave above the rows. Each simulated field features varied terrains, replicating the irregularities found in real-world landscapes. Comprehensive measurements for each simulation world can be found in Table 7.2. In the experimental phase of this study, we adopted frame dimensions of $(h,w) = (224,224)$, matching the input and output sizes of the neural network model, with a channel count of $c = 3$. The maximum linear velocity was set to $v_{x,max} = 0.5m/s$, and the maximum angular velocity was capped at $\omega_{z,max} = 1rad/s$. The angular velocity gain, denoted as $\omega_{z,gain}$, was fixed at 0.01,

[2] https://gazebosim.org

and the Exponential Moving Average (EMA) buffer size was set to 3. Additionally, the depth threshold was adjusted based on the specific characteristics of different crops. Specifically, it has been empirically set at 5 m for vineyards, raised to 8 m for pear trees and pergola vineyards, and further increased to 10 m for tall trees, taking into account the average distance from the rows in various fields.

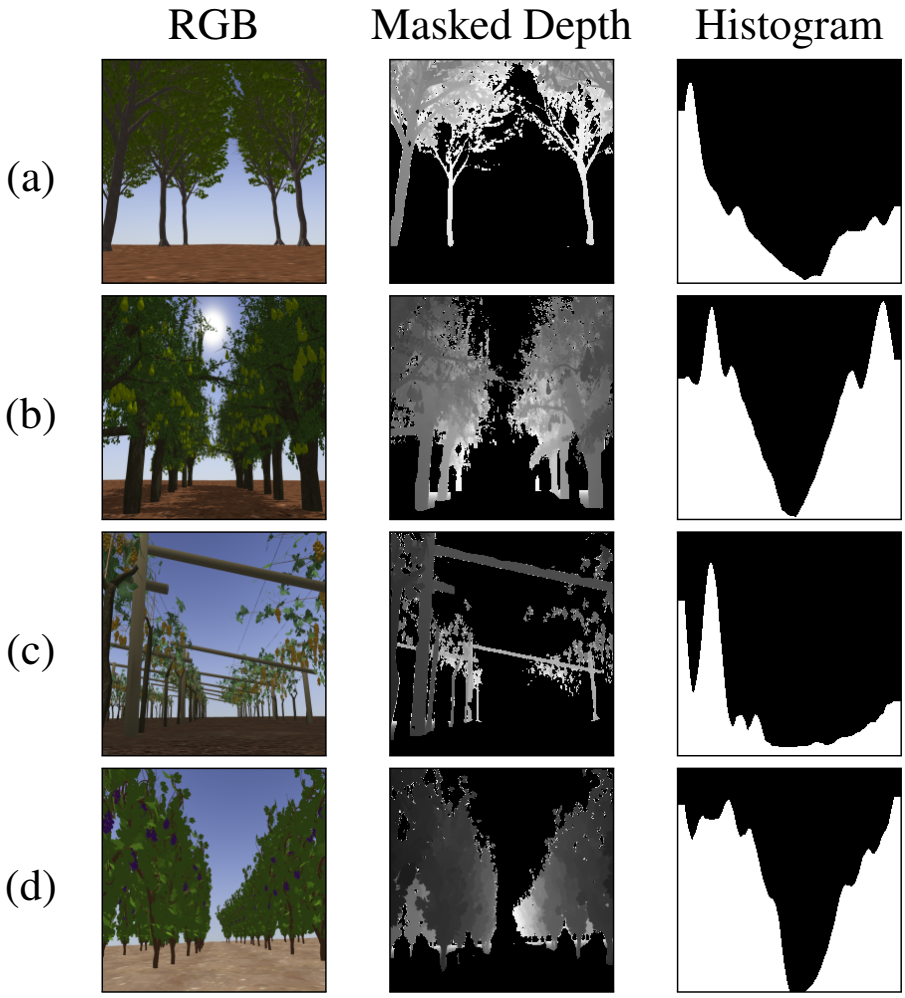
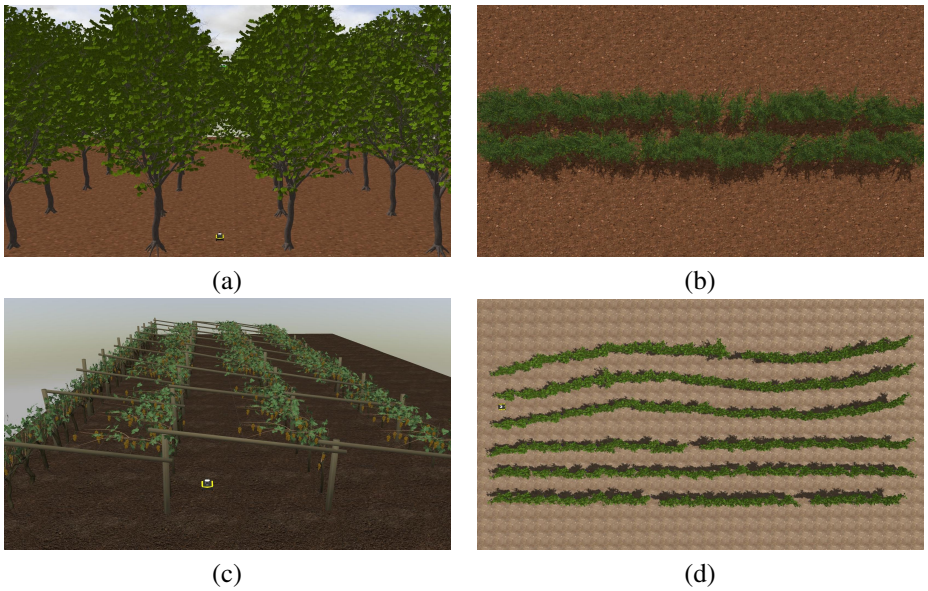


Fig. 7.5 Sample outputs of the proposed SegMinD algorithm for High Trees (a), Pear Trees (b), Pergola Vineyard (c), and Vineyard (d). Predicted segmentation masks are refined cutting values exceeding a depth threshold. The sum over mask columns provides the histograms used to identify the center of the row as its global minimum.

Table 7.2 Dimensions of various simulated crops indicate the average values for the distance between rows, the spacing between plants within a row, and the heights of the plants.

Gazebo worlds	Rows distance [m]	Plant distance [m]	Height [m]
Common vineyard	1.8	1.3	2.0
Pergola vineyard	6.0	1.5	2.9
Pear field	2.0	1.0	2.9
High trees field	7.0	5.0	12.5



(a) (b) (c) (d)

Fig. 7.6 Gazebo simulated environments were employed to assess the SegMin approach in various crop rows of significance, including wide rows with high trees (a), a slender row of pear trees (b), an asymmetric pergola vineyard with irregular rows (c), and both straight and curved vineyard rows (d). For the latter scenario, the evaluations were conducted in both the second row from the top and the second row from the bottom.

Navigation Results in Simulation

The comprehensive evaluation of the SegMin navigation pipeline and its variant, SegMinD, took place in realistic crop fields within a simulation environment, employing pertinent metrics for visual-based control without the need for precise robot localization, aligning with methodologies from prior studies [199, 19]. The camera frames were published at a frequency of 30 Hz, with inference conducted at 20 Hz and velocity commands from controllers published at 5 Hz. The evaluation utilized the testing package from the open-source PIC4rl-gym[3] in Gazebo [27]. The chosen

[3] https://github.com/PIC4SeR/PIC4rl_gym

Table 7.3 Navigation outcomes across diverse test fields were assessed using the SegMin, SegMinD, and the SegZeros segmentation-based algorithms. The evaluation employed metrics to gauge the efficacy of navigation, including clearance time, and assessed precision through Mean Absolute Error (MAE) and Mean Squared Error (MSE) by comparing the obtained path with the ground truth. Additionally, kinematic information about the robot's navigation was captured through the cumulative heading average $\gamma[rad]$, mean linear velocity $v_{avg}[m/s]$, and the standard deviation of angular velocity $\omega_{stddev}[rad/s]$. Notably, SegZeros proved impractical for scenarios involving tall trees, pear trees, and pergola vineyards.

Test Field	Method	Clearance [s]	MAE [m]	MSE [m]	Cum. γ_{avg} [rad]	v_{avg} [m/s]	ω_{stddev} [rad/s]
High Trees	SegMin	**40.41 ± 0.12**	0.27 ± 0.01	0.08 ± 0.00	0.08 ± 0.00	0.49 ± 0.00	0.05 ± 0.00
	SegMinD	40.44 ± 0.51	**0.17 ± 0.01**	**0.04 ± 0.00**	0.05 ± 0.00	0.48 ± 0.01	0.06 ± 0.02
Pear Trees	SegMin	**42.06 ± 1.23**	0.03 ± 0.01	**0.00 ± 0.00**	0.01 ± 0.00	0.48 ± 0.00	0.11 ± 0.05
	SegMinD	42.26 ± 1.91	**0.03 ± 0.02**	**0.00 ± 0.00**	0.02 ± 0.00	0.48 ± 0.01	0.03 ± 0.00
Pergola Vine.	SegMin	**40.86 ± 0.39**	**0.08 ± 0.01**	**0.01 ± 0.00**	0.03 ± 0.02	0.48 ± 0.00	0.17 ± 0.02
	SegMinD	41.14 ± 0.33	0.10 ± 0.05	0.01 ± 0.01	0.03 ± 0.01	0.48 ± 0.00	0.20 ± 0.03
Straight Vine.	SegMin	**50.51 ± 0.31**	**0.11 ± 0.00**	**0.01 ± 0.00**	0.03 ± 0.00	0.49 ± 0.00	0.08 ± 0.01
	SegMinD	50.63 ± 0.28	0.11 ± 0.01	0.02 ± 0.00	0.03 ± 0.01	0.49 ± 0.00	0.09 ± 0.01
	SegZeros	53.69 ± 1.03	0.14 ± 0.03	0.02 ± 0.01	0.03 ± 0.0	0.46 ± 0.01	0.09 ± 0.01
Curved Vine.	SegMin	53.32 ± 0.25	0.12 ± 0.01	0.02 ± 0.00	0.04 ± 0.01	0.49 ± 0.00	0.09 ± 0.02
	SegMinD	**51.44 ± 1.03**	**0.09 ± 0.01**	**0.01 ± 0.00**	0.01 ± 0.00	0.48 ± 0.01	0.06 ± 0.01
	SegZeros	71.05 ± 27.13	0.11 ± 0.04	0.02 ± 0.01	0.05 ± 0.01	0.40 ± 0.13	0.11 ± 0.04

metrics aimed to assess the navigation effectiveness, measured by clearance time and precision, involving a quantitative comparison of obtained trajectories with a ground truth trajectory using Mean Absolute Error (MAE) and Mean Squared Error (MSE). Ground truth trajectories were computed by averaging interpolated poses of plants within rows. In the case of an asymmetric pergola vineyard, a row referred to the portion without vegetation on top, as depicted in Fig. 7.6 (c). The algorithms' response to terrain irregularities and row geometries was also studied, encompassing significant kinematic information about the robot. The evaluation considered the cumulative heading average $\gamma[rad]$ along the path, mean linear velocity $v_{avg}[m/s]$, and standard deviation of angular velocity $\omega_{stddev}[rad/s]$. These metrics provided insights into how well the algorithms maintained the robot's correct orientation, with the mean value of ω consistently approaching zero due to successive orientation corrections.

The complete set of results is outlined in Table 7.3. Each metric is accompanied by both the average value and standard deviation, reflecting the repetition of experiments in three runs on a 20 m long track within each crop row. The proposed method effectively addresses the challenge of guiding the robot through rows of trees with dense canopies, such as high trees and pears, even in the absence of a localization

system. It also demonstrates proficiency in unique scenarios, like navigating through pergola vineyards. The presence of plant branches and wooden supports poses a challenge for existing segmentation-based solutions. These solutions, built on the assumption of identifying a clear passage by focusing solely on zeros in the binary segmentation mask [199], encounter limitations in our tested scenarios. In our result comparisons, we term this prior method as SegZeros, utilizing the same segmentation neural network for assessment.

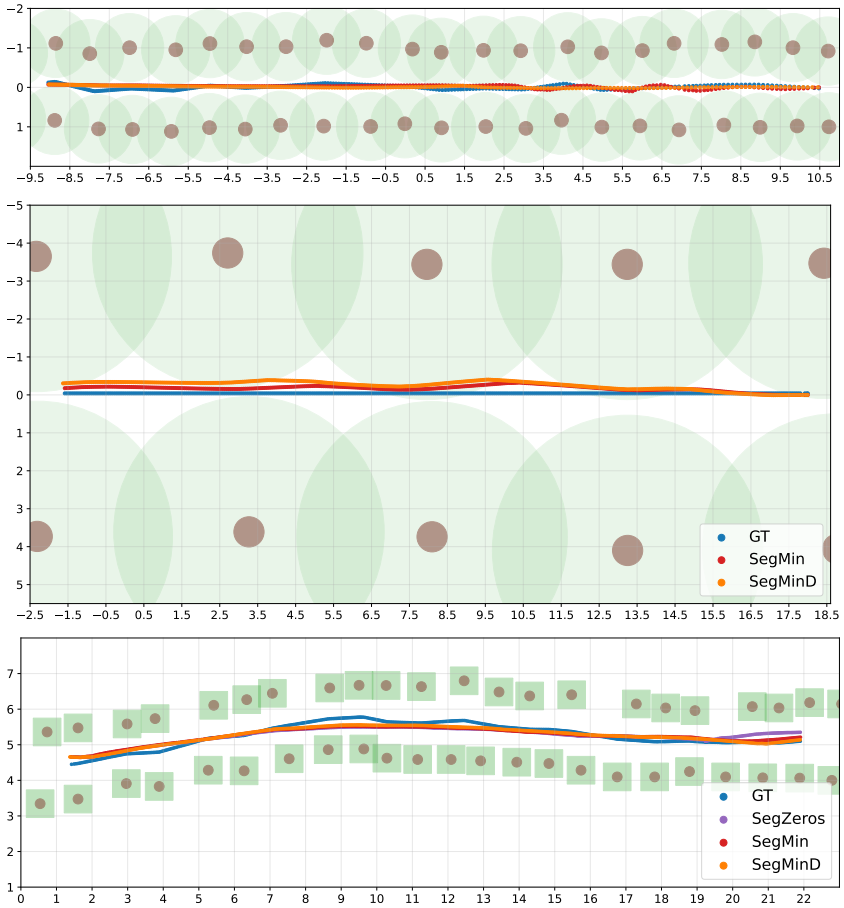


Fig. 7.7 Trajectories comparison between our proposed algorithms (SegMin and SegMinD) and the ground truth central path (GT): Pears (top), High Trees (center), Curved Vineyard (bottom). In the last graph, the trajectory generated with the SegZeros algorithm is also reported for comparison.

The SegMin methodology, based on histogram minimum search, proves to be a resilient solution for guiding the robot through tree rows. The incorporation of depth inverse values as a weighting function in SegMinD enhances the algorithm's

precision, particularly in navigating through challenging scenarios like wide rows (high trees) and curved rows (curved vineyard). Furthermore, these innovative methods exhibit competitive performance even in standard crop rows, where a clear passage to the end of the row is discernible in the mask without canopy interference. Compared to the previous segmentation-based baseline method, the histogram minimum approach significantly reduces navigation time and enhances trajectory precision in both straight and curved vineyard rows. On the other hand, the search for plant-free zero clusters in the map proves to be less robust and efficient, leading to undesired stops and an overall slower and more oscillating behavior during navigation. Additionally, the standard deviation of angular velocity aligns with the results obtained, being smaller in cases where the trajectory is more accurate, while the cumulative heading exhibits larger values when the algorithms demonstrate increased reactivity. The trajectories generated by SegMin, SegMinD, and SegZeros algorithms are visually depicted in Fig. 7.7 within representative scenarios. These scenarios include a cluttered, narrow row featuring small pear trees, a wide row with high trees, and curved vineyards where the state-of-the-art SegZeros method is applied.

Navigation test on the field

The overall navigation pipeline of SegMin and its variant SegMinD are tested in real crop fields, evaluating the results with relevant metrics for visual-based control without precise localization of the robot, as done in previous works [199, 19]. The robotic platform employed to perform the tests is a Clearpath Husky UGV equipped with a LiDAR Velodyne Puck VLP-16, an RGBD camera Realsense D455, an AHRS Microstrain 3DM-GX5 and a Mini-ITX computer with an Intel Core i7 processor and 16 GB of memory. The camera frames were captured at a rate of 30 Hz, inference was performed at 20 Hz, and velocity commands were published at 5 Hz. In this section's experiments, ground truth trajectory was unavailable due to the complexity and demanding nature of measurement, which requires sophisticated instruments for sufficient accuracy. Instead, the lateral displacement of the rover within the row was determined using point clouds from the LiDAR. Points were clustered to separate the two rows, then fitted by a straight line, followed by computing the shortest distance from the plants to the origin, i.e., the center of the robot, where the sensor is mounted, for both lanes. The AHRS measured the rover's heading and compared it with the

average heading of the row obtained from satellite images, considering the tested rows are straight.

Table 7.4 Navigation results of the real-world testing of the algorithms. SegMinD has been tested on apple and pear orchards. A comparison has been performed between the algorithms SegZeros, SegMin and SegMinD in a vineyard row choosing the same parameters: depth threshold set to 8.0 m and pixel-wise confidence to 0.7.

	Algorithm	MAE [m]	RMSE [m]	Cum. γ_{avg} [rad]	ω STD [rad/s]
Apple Trees	SegMin	0.16708	0.18841	-0.03278	0.04112
	SegMinD	0.07208	0.09110	0.10709	0.06881
Pear Trees	SegMin	0.46540	0.47309	0.03066	0.12924
	SegMinD	0.28435	0.29792	0.03004	0.13074
Vineyard	SegZeros	0.16669	0.17011	-0.01282	0.02675
	SegMin	0.23045	0.24193	-0.03372	0.11451
	SegMinD	0.16007	0.17093	-0.11478	0.10127

The performances of the proposed control have been evaluated on two different plant orchards, i.e., apples and pears, and a vineyard. The performance metrics are reported in Table 7.4. The trajectories of the best test for each crop field and the comparison between the proposed algorithms, SegMin, SegMinD, and SegZeros, are represented in Fig. 7.8. Overall, the novel control laws can effectively solve the problem of guiding the robot through tree rows with thick canopies (high trees and pears) without a localization system in a real-world scenario.

Moreover, the algorithms SegMin and SegMinD demonstrate the ability to generalize to the common case without obstruction by canopies. As shown by the comparison performed in the vineyard, they obtained results in line with the existing SegZeros. The algorithms SegMin and SegMinD show their effectiveness in maintaining the robot on the desired central line, even recovering from strong disturbances. As can be noticed by the trajectories in pear and apple trees (top and central plots in Fig. 7.8), sudden drifts of the robot are caused by fruits, branches, stones, and disparate irregularity of the terrain. Those small obstacles cannot be precisely sensed and tackled with classic obstacle avoidance algorithms; hence, the resilience of the control algorithms to these external factors is crucial to keep the robot on track. Differently, in vineyard rows, grass and cleaner terrain induce smoother overall trajectories.

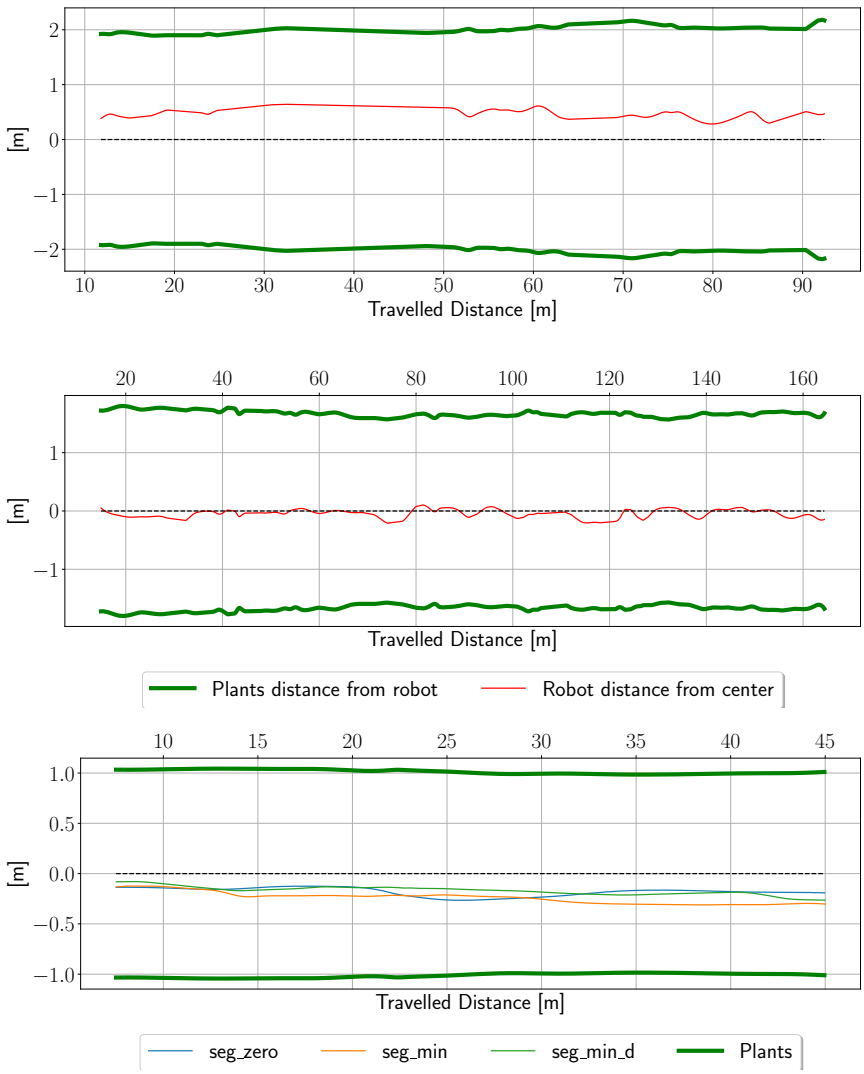


Fig. 7.8 Trajectory results of relevant tests performed on the field. In order from top to bottom: navigation in pear tree rows using SegMinD, navigation in apple tree rows using SegMinD, and trajectory comparison of all three algorithms in a vineyard row. Sudden drifts in orchards traversal are caused by fruits, small obstacles, and irregularity in the terrain.

Finally, an ablation study is carried out on a vineyard row to assess the impact of key parameters on the proposed control strategy within the novel SegMinD algorithm. Specifically, the study explores the effects of the depth image max distance and pixel-wise confidence threshold of the predicted segmentation mask. The findings, detailed in Table 7.5, indicate that a confidence level greater than 0.5 is required for achieving robust behavior, filtering mask portions with uncertain prediction. Indeed, results with a confidence level of 0.3 exhibit a high standard deviation in the angular

Table 7.5 Ablation study on SegMinD algorithm performance: relevant parameters of the segmentation control system are explored for a better understanding of their impact on the overall result. Three values of depth threshold and prediction confidence are selected for the ablation.

Depth threshold	MAE	RMSE	Cum. γ_{avg} [rad]	ω STD [rad/s]
Confidence 0.3				
5.0	0.35234	0.36031	-0.27982	0.99039
8.0	0.22854	0.23814	-0.01200	0.25087
11.0	0.35295	0.36281	0.06704	0.38422
Confidence 0.5				
5.0	0.22456	0.23915	0.02729	0.42077
8.0	0.15794	0.17106	-0.19994	0.40949
11.0	0.45508	0.45754	0.00374	0.10303
Confidence 0.7				
5.0	0.38653	0.39433	0.02949	0.46574
8.0	**0.11928**	**0.15057**	**-0.04034**	**0.12565**
11.0	0.39656	0.40279	-0.01696	0.35989

velocity command. Regarding the depth image maximum distance, three values are tested: 5, 8, and 11 m. A low value of 5 m produces sub-optimal results compared to an intermediate value of 8 m. In this scenario, the noise in the segmentation has a more pronounced effect as the long-view geometry of the row is not considered in the computation of the histogram, including only close plants. Conversely, a high value for the depth threshold leads to inferior results due to the insufficient precision of the depth camera, resulting in artifacts that can compromise overall performance. In conclusion, the optimal outcome is achieved with a high confidence in the prediction and an intermediate depth threshold of 8 m.

7.2 Position-agnostic controller with Deep Reinforcement Learning

Classical navigation algorithms execute perception, planning and control as separate concatenated stages, increasing the overall risk of error in each sub-module. Differently, policy learning methods can directly map raw input data to actions, drastically simplifying the whole navigation algorithmic pipeline. Model-free Deep Reinforcement Learning (DRL) optimizes a parametric policy without accessing the

dynamic model of the environment, allowing to train an agent to navigate in unseen environments. DRL has recently emerged as a powerful approach for autonomous vehicles [210] and mobile robot navigation [93].

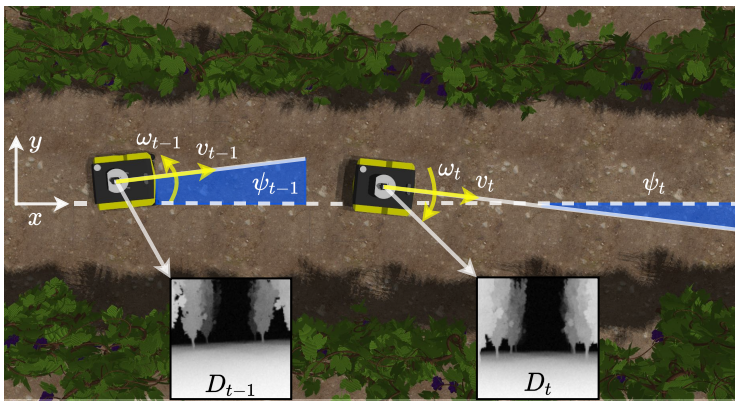


Fig. 7.9 At each time instant t, the proposed agent receives as inputs a raw noisy depth image D_t, the previous velocity commands $[v_{t-1}, \omega_{t-1}]$ and the yaw ψ_t, to generate the new commands $[v_t, \omega_t]$. Since no explicit localization is required, the agent performs a positioning-independent navigation in the vineyard row.

7.2.1 Task Formulation

Similarly to the problem described in Chapter 4, the navigation problem is modeled in a reinforcement learning framework. Therefore, the problem is formulated as a Markov Decision Process (MDP) described by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$ [211]. An agent starts its interaction with the environment in an initial state s_0, drawn from a pre-fixed distribution $p(s_0)$ and then cyclically select an action $\mathbf{a_t} \in \mathcal{A}$ from a generic state $\mathbf{s_t} \in \mathcal{S}$ to move into a new state $\mathbf{s_{t+1}}$ with the transition probability $\mathcal{P}(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t})$, receiving a reward $r_t = R(\mathbf{s_t}, \mathbf{a_t})$. A parametric policy π_θ representing the agent behaviour is optimized during the training. In the context of autonomous navigation, we model the MDP with an episodic structure with maximum time steps T, hence the agent is trained to maximize the cumulative expected reward $\mathbb{E}_{\tau\sim\pi} \sum_{t=0}^{T} \gamma^t r_t$ over each episode, where $\gamma \in [0,1)$ is the discount factor. More in detail, we use a stochastic agent policy in an entropy-regularized reinforcement learning setting, in which the optimal policy π^*_θ with parameters θ is obtained maximizing a modified discounted term:

$$\pi^*_\theta = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi} \sum_{t=0}^{T} \gamma^t [r_t + \alpha \mathcal{H}(\pi(\cdot|s_t))] \tag{7.12}$$

Where $\mathcal{H}(\pi(\cdot|s_t))$ is the entropy term which increases robustness to noise through exploration, and α is the temperature parameter which regulates the trade-off between reward optimization and policy stochasticity.

Reward

The potential outcome of the vineyard navigation task can be easily summarized in a binary output: the robot successfully arrives at the end of the row or it does not. However, this sparse reward feedback can be assigned to the agent only in the case of a completed successful episode, which is an improbable event all over the initial learning phases. According to this, reward shaping is the typical process which leads researchers to analytically specify the desired behaviour to the agent thanks to a dense reward signal assigned at each time step. Moreover, this approach allows to precisely express secondary desired behaviours. In this application scenario, we identify three key features of an ideal optimal policy: a complete collision-free travel in the vineyard row, a centered trajectory and a proper orientation. Nonetheless, the reward can be computed during the training in simulation exploiting positioning data that is not needed at test time by the agent. To this end, we first define a reward contribution r_h to keep the robot oriented towards the end of the row:

$$r_h = \left(1 - 2\sqrt{\left|\frac{\phi}{\pi}\right|}\right) \tag{7.13}$$

where ϕ is the heading angle of the robot, namely the angle between its linear velocity and the end of the row. We consider r_h as a fundamental feedback to let the agent understand how to counteract the sudden angular deviations imposed by the irregular terrain, which is realistically generated in our simulated world. Then, in order to obtain a central trajectory, we consider the possibility of directly scoring the distance of the robot from the center of the row. However, this approach requires to compare the robot pose with the mean line of each specific row at each step, and nonetheless it results in a slow and inefficient policy when combined with the heading reward r_h. For this reason, we prefer a distance-based reward to strongly encourage the agent to reach the end of the row:

$$r_d = d_{t-1} - d_t \tag{7.14}$$

where d_{t-1} and d_t are euclidean distances between the robot and the end of the row (EoR) at successive time steps, as shown in Fig. 7.10. Robot pose information is

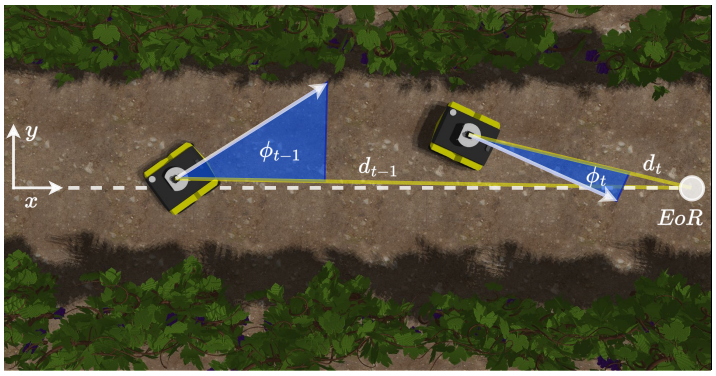


Fig. 7.10 The reward at each time step t is computed as a function of the distances from the end of the row (EoR) d_t and d_{t-1}, and the angle ϕ_t between the robot orientation and the shortest path to EoR. This information is available while training the agent although it does not constitute its input.

uniquely used for reward computing while training, and is not included as agent input, as better specified in the following subsection 7.2.1. We finally include a sparse reward contribution for end-of-episode states, assigning $r_s = 1000$ for the successful completion of the task, $r_c = -500$ if collision occurs, and $r_\psi = -500$ if the robot overcomes a $\pm 85^\circ$ yaw limit. Stopping the episode when the robot exits the vineyard row or reverses its motion direction is fundamental to keep collecting meaningful sample transitions for the task.

The final reward signal results to be as follows:

$$r = a \cdot r_h + b \cdot r_d + \begin{cases} r_s \ \ if\ Success \\ r_c \ \ if\ Collision \\ r_\psi \ \ if\ Reverse \end{cases} \tag{7.15}$$

Where $a = 0.6$ and $b = 35$ are numerical coefficients to efficiently integrate the diverse reward contributions in the final signal.

Policy Network

We define the parametrized agent policy with a deep neural network. We train the agent with the Soft Actor-Critic (SAC) algorithm presented in [76], which allows for a continuous action space. In particular, we instantiate a stochastic Gaussian policy for the actor and two Q-networks for the critics.

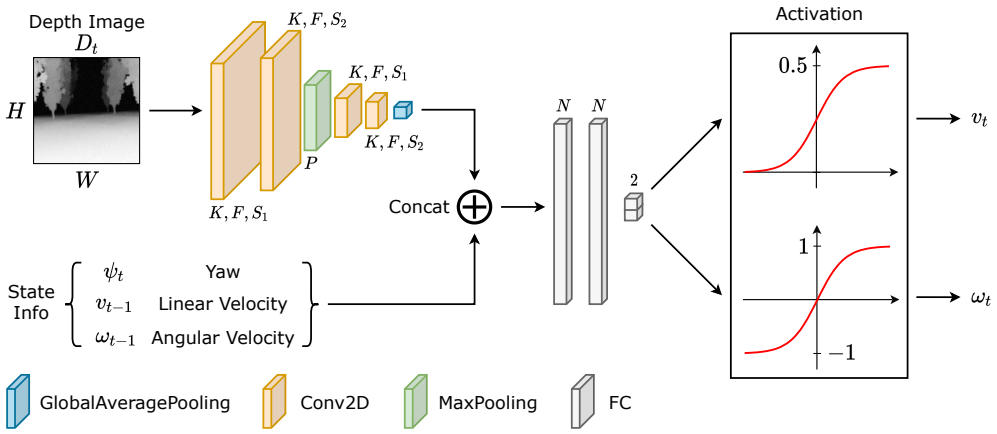


Fig. 7.11 Architecture of the actor policy network. A convolutional backbone extracts features from the depth image D_t. The features are then concatenated to the input vector $[v_{t-1}, \omega_{t-1}, \psi_t]$ and two fully connected layers output the action vector $[v_t, \omega_t]$ with specific activation functions.

Input features We select the input features of the policy network only considering the odometric and perception data available during the vineyard navigation task. To this end, we identify a set of input features which enables a localization-independent navigation, and an affordable perception system such as a simple depth camera. Several iterations lead us to the choice of three key elements as agent inputs.

1) The previous set of velocity commands $[v_{t-1}, \omega_{t-1}]$ to provide information about its current motion and temporal continuity to the agent and avoid strongly oscillating or disentangled commands.

2) The yaw of the robot ψ_t measured at the current time instant t, which is always available thanks to an IMU sensor on the robot. The simple yaw angle does not represent the required optimal heading angle ϕ_t for the task. Indeed, such angle depends on the robot position, which is not available at test time in GPS-denied conditions. However, the yaw ψ_t provides awareness about the actual orientation of the robot and helps in generating smooth collision-free velocity commands.

3) A raw noisy depth image of size (112×112). Each pixel of the image contains a distance in the range of $[0.0, 5.0]m$ and it is the only perception data of the environment the agent exploits to guide the robot through the vineyard. Processing the image, the agent acquires the necessary knowledge on obstacles and it can also visually infer its current orientation with respect to the end of the visible row. A reduced size of the image allows to obtain a compact latent representation of only

32 features. The choice of depth images is also motivated by the aim of reducing the simulation-to-reality gap. Indeed, depth images are only marginally affected by visual features. On the other hand, real depth images may present peculiar noisy behaviours, so we add to each raw image two different noises: a first uniform random noise with values in $[-0.5, 0.5]m$, and a second random noise proportional to the depth values in the image, also in the range $[-0.5, 0.5]m$. The second noise aims at disturbing more heavily the higher-distance points. Overall, each pixel presents at most a perturbation of $\pm 1m$.

Output actions The policy network predicts an action $a_t = [v_t, \omega_t]$ at each time step, which directly represents the required linear and angular velocity command to control the robot. This choice is mainly dictated by the differential-drive steering model of the robotic platforms we use, and it also allows to easily integrate the system with ROS standard command messages.

Network architecture The architecture of the policy network presents a double input structure, represented in Fig. 7.11. A convolutional feature extractor is designed to efficiently map the depth image in a compact latent representation, inspired by the work proposed for visual SAC in [212]. However, we avoid the adoption of an encoder-decoder structure and an additional reconstruction loss. Our solution allows the agent to easily learn how to extract relevant feature for the task autonomously. Firstly, the feature extractor takes the depth image D_t $(H \times W)$ as input. It consists of two convolutional layers (kernel size $K = 3$, $F = 32$ filters, strides $S_1 = 2$ and $S_2 = 1$ respectively) with ReLU activations followed by a max pooling layer (pool size $P = 2$), two more convolutional layers (with the same structure as the previous ones), and a global average pooling layer. The features are then concatenated to the position-agnostic robot state input vector, which includes the measured yaw ψ_t and the previous action command (linear velocity v_{t-1} and angular velocity ω_{t-1}). The resulting vector is processed by two fully connected layers with $N = 256$ neurons and ReLu activation. Finally, an additional output layer predicts the action vector $[v_t, \omega_t]$, using *tanh* as activation function. The linear velocity v_t is further squashed to $(0, 0.5)$ to match the velocity profile of the robot. As we use a stochastic Gaussian policy, the network actually predicts the mean and the standard deviation of each action distribution, which are used to sample a value from the derived distribution while training. Instead, the mean value is directly used as output at test time. The critic network structure is identical to the actor one, except it takes

also the predicted action vector in input and outputs an estimate of the Q value and it is trained according to the SAC algorithm.

Random Initialization Strategy A critical condition of the vineyard environment is its constrained geometry. As a consequence, the agent usually collides in few steps in the early stage of the DRL training, exploring a drastically reduced number of states of the environment. This behaviour negatively affects the generalization properties of the agent and more generally the probability of a stable convergence of the training algorithm. For this reason, we identify a set of counteractions to effectively train the policy: 1) The vineyard training stage comprises different vineyard rows where the robot travels during the same training simulation to encourage better generalization derived from a higher number of visited states. 2) The robot is initialized in a new random pose in the vineyard every 10 episodes, varying its $[x_0, y_0]$ position coordinates and its initial yaw ψ_0, enabling also the agent to travel the rows in both direction. Consequently, the agent is able to visit the final sections of rows from the beginning of the simulation, significantly speeding up the convergence of the training with better generalization results. 3) An initial exploration phase is combined with an additional ε-greedy policy which samples random values in the action spaces with a probability that is exponentially reduced with the increase of episodes to maintain a proper level of exploration during the whole training.

7.2.2 DRL agent experiments

In this section we present the experimental session followed to validate the proposed methodology in a simulated vineyard, with the aim of answering to the following questions:

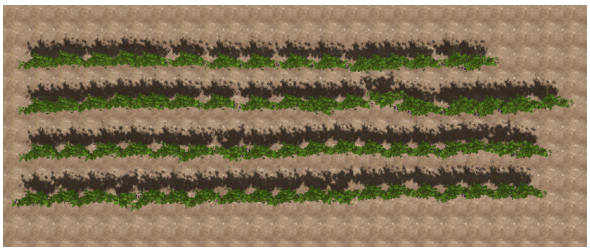
1) Is the agent able to successfully accomplish the task of vineyard navigation in an unseen environment?

2) How well can the agent guide the robot through the vineyard? In particular, our main objective is to evaluate the quality of the produced trajectory in terms of number of collisions, centrality, and average velocity commands.

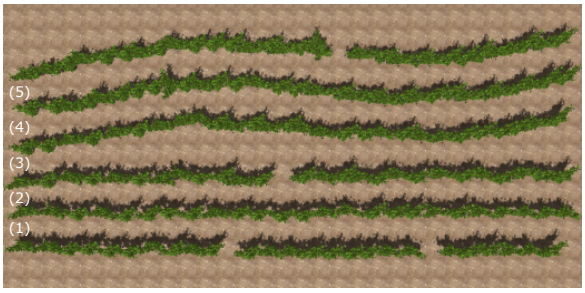
3) How well can the agent generalize to new testing conditions such as a different robotic platform or an increased level of noise in the depth image?

To this end, we firstly report the performance of our navigation agent in the testing environment evaluating the quality of the trajectory and the overall behaviour of the

robot on several repeated tests. Moreover, we test the generalization and robustness properties of our trained policy, varying the robotic platform and increasing the noise in the depth images. Finally, we provide an analysis of the real-time performance on different computational systems.



(a) Training Env



(b) Testing Env

Fig. 7.12 Training (**a**) and testing (**b**) simulation environments. Intentional gaps and curves are introduced in (**b**) to provide a challenging environment for the agent. Testing rows are numbered as referenced in the results section.

Vineyard navigation test in simulation

The agent have been validated on the five vineyard rows of the testing environment of Fig. 7.12: two straight rows, two curved rows and a hybrid row. We perform a total amount of ten tests for each row, half in the forward direction (F) and half in the reverse direction (R). We repeat tests multiple times to better validate our approach and obtain more consistent results. The main approach to validate autonomous navigation algorithms is to compare the trajectory followed by the platform with a ground truth path. We compute the median line for each vineyard row in the testing environment, performing a mean operation between two adjacent rows, then we fit the obtained median points with a fifth-order polynomial in order to achieve an accurate ground truth line. Afterwards, we compare the robot trajectory with the computed ground truth lines in order to estimate its Mean Absolute Error (MAE)

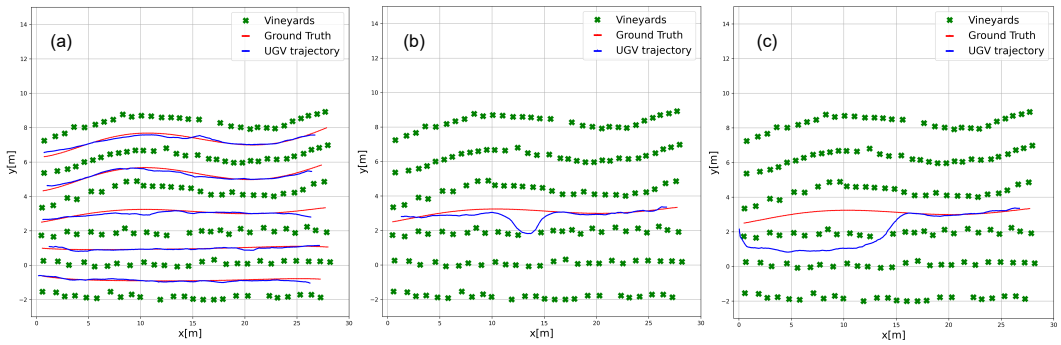


Fig. 7.13 Results achieved by the agent in the testing environment. (**a**) resumes the performance on all the rows, where the agent follows a trajectory very close to the optimal one. In (**b**), the agent recovers from a critical situation given by gaps in the row. In (**c**), the agent fails and switches to the adjacent row but still reaches the end of the vineyard without collisions.

and Root Mean Square Error (RMSE). Moreover, for each test we report the number of trials in which the UGV is able to successfully reach the end of the row.

Table 7.6 presents the complete quantitative evaluation of the experiment. Despite the difficulty of the test environment, the learned policy demonstrates the ability of correctly guiding the robot through different rows. As expected, the agent performs slightly better on the straight test rows (1 and 2) compared to the curved ones (4 and 5), since the conditions are closer to the training scenario. That is proved by the lower MAE and RMSE values. Failures mostly occur in the hybrid test row 3R (2 successes out of 5) and in the curved row 5F (3 successes out of 5), due to the presence of a wide gap between the plants. However, the agent is able to correctly cope with gaps in the other tests. Fig. 7.13 presents some examples of trajectories obtained during the test phase, particularly relevant to the gap handling. During the tests shown in Fig. 7.13a, the agent is able to follow a path very close to the optimal one. In Fig. 7.13b, the gap causes a suboptimal behavior, but the agent is still able to recover the correct path. On the other hand, in Fig. 7.13c, an example of a failure is presented. It is worth citing that, despite following the wrong path, the agent is still able to avoid collisions. The overall results report an MAE of 0.114m, an RMSE of 0.136m and a success rate of 45/50 total runs, which is a considerably positive result.

Generalization, Robustness and Real-Time Performance

We conduct further experiments to evaluate the generalization and robustness properties of the trained policy varying two separate impacting conditions: the rover

Table 7.6 Performance of the agent in the test environment (forward F and reverse R). We include both MAE and RMSE metrics (lower is better), the actions v and ω (mean and standard deviation) and the success rate (higher is better). The last row reports the overall mean results and metrics.

Test Row	Row Shape	MAE [m]	RMSE [m]	v [m/s]	ω [rad/s]	Success
1F	Straight	0.076	0.089	0.493 ± 0.020	-0.027 ± 0.538	5/5
1R	Straight	0.131	0.141	0.478 ± 0.053	-0.030 ± 0.623	5/5
2F	Straight	0.068	0.076	0.488 ± 0.037	-0.028 ± 0.642	5/5
2R	Straight	0.087	0.105	0.489 ± 0.037	-0.001 ± 0.610	5/5
3F	Hybrid	0.120	0.154	0.497 ± 0.003	-0.012 ± 0.353	5/5
3R	Hybrid	0.100	0.130	0.494 ± 0.017	-0.036 ± 0.610	2/5
4F	Curved	0.164	0.191	0.468 ± 0.079	-0.008 ± 0.687	5/5
4R	Curved	0.081	0.097	0.484 ± 0.059	-0.014 ± 0.493	5/5
5F	Curved	0.112	0.145	0.486 ± 0.057	-0.005 ± 0.571	3/5
5R	Curved	0.201	0.233	0.445 ± 0.122	-0.042 ± 0.656	5/5
Overall	-	**0.114**	**0.136**	**0.482 ± 0.048**	**-0.020 ± 0.578**	**45/50**

Table 7.7 Comparison of agent generalization capabilities on a different robotic platform. We test the rovers on both straight (1F) and curved (4F) rows, reporting the average results among 3 runs.

Row	Rover	Success	T_{avg} [s]	MAE [m]	RMSE [m]
Straight	Jackal	3/3	78	0.079	0.088
	Husky	3/3	69	0.081	0.098
Curved	Jackal	3/3	78	0.085	0.101
	Husky	2/3	88	0.112	0.138

platform and the amount of noise in the depth image. These additional tests are performed on two different rows of the testing environments: one straight (Row 1F) and one curved (Row 4F). We repeat each test three times for a consistent comparison in both the configurations. Moreover, we analyze the inference timing of the proposed agent with different computational systems in order to report the ability of the actual platforms to run the policy on-board in real-time.

Different Robotic Platform We choose the Husky UGV[4] to test our policy with a different platform. The Husky URDF model is also provided by Clearpath Robotics. Husky has an overall size Length $\times$ Width $\times$ Height equal to $990 \times 670 \times 390$ mm, compared to the smaller Jackal size of $508 \times 430 \times 250$ mm. The larger footprint

[4]https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/

Table 7.8 Success rate of the agent tested with increasing noise factor in both straight (1F) and curved (4F) vineyard rows with three trials for each test run.

Noise Factor	2	4	6	8	10
Success (Straight Row)	3/3	3/3	3/3	2/3	0/3
Success (Curved Row)	3/3	3/3	3/3	3/3	0/3

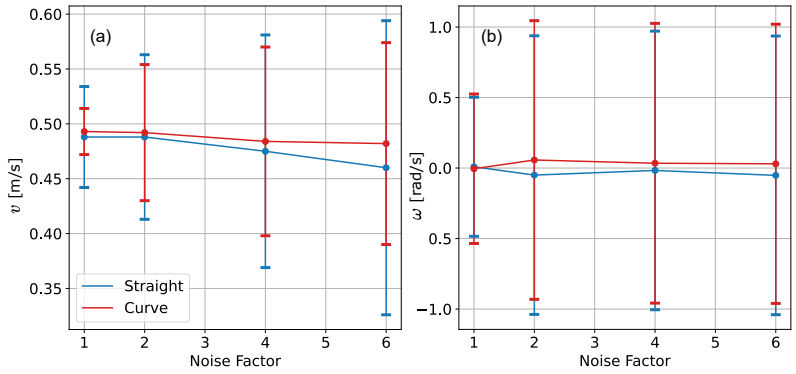


Fig. 7.14 Model robustness to noise. We report the mean and standard deviation for v **(a)** and ω **(b)**, highlighting how the predictions of the policy network tend to oscillate more when the noise level increases. However, the agent is able to reach the end of the row in almost all the cases even with high levels of noise.

and the increased height of the camera constitute sensible variations for the agent, which needs to select actions accordingly to manage collision avoidance. Overall, the results presented in Table 7.7 can be considered successful: the agent is able to guide the new UGV to the end of the vineyard row with a success rate of $3/3$ in the straight case and $2/3$ in the curved case. Despite the slight degradation in the MAE and RMSE of the trajectory and a longer time required in the curved row, the agent demonstrates to generalize well with different platforms, taking into account the considerable differences between the two robotic platforms.

Depth Image Noise Real-world depth images often present unpredictable noisy behaviours. Aware of that, we investigate the ability of our solution to resist to heavier noise in the observations, by gradually applying a multiplicative factor to the noise in the images. The trained policy demonstrates to be robust to noise by successfully reaching the end of the row until using a maximum noise factor of 10. We consider noise multiplication factors of $2, 4, 6, 8, 10$, obtaining the success rate presented in Table 7.8. Moreover, we show the effect of the increased noise in the

velocity commands generated by the agent in Fig. 7.14. The mean value of the linear velocity commands is reduced proportionally to the increase of the perturbation in the depth image, demonstrating how the agent prioritizes a safe traveling with respect to speed. On the other hand, the standard deviation in both velocities result to be much higher, suggesting uncertainty and oscillation in the robot behaviour. Nonetheless, considering the degradation level used in the depth image, the obtained performances are strongly encouraging in terms of noise tolerance and robustness.

Table 7.9 Real-time inference performance of the actor policy network on different hardware configurations. Each test is reported with a statistic on 100 independent trials.

Platform	CPU	Model	T [ms]
Jackal	Intel i3-4330TE@2.40GHz	TF	6.20 ± 1.50
		TFLite	1.96 ± 0.11
Husky	Intel i7-6700TE@2.40GHz	TF	4.42 ± 1.35
		TFLite	1.24 ± 0.21
NUC	Intel i5-1145G7@2.60GHz	TF	2.80 ± 0.62
		TFLite	0.63 ± 0.39

Real-Time Performance As additional analysis, we test inference timings of the actor policy network on three different computing hardware to investigate its performance on real platforms. In particular, we test the computers mounted on Jackal and Husky UGVs, together with an Intel NUC mini PC, an emerging solution as on-board computing hardware for robotics platforms. For our test, we apply network optimizations with the TensorFlow Lite[5] library, obtaining the float32 *.tflite* converted model. Results presented in Tab. 7.9 show how all the considered hardware configurations can easily reach real-time performance and that TFLite conversion heavily decreases inference timings.

[5]https://www.tensorflow.org/lite

Chapter 8

Waypoint Generation in Row-based Crops with Deep Learning and Contrastive Clustering

A reliable autonomous navigation system is a fundamental problem to enable automatic operation in agriculture. Row-based crops fields such as vineyards and orchards presents constrained geometries, that often hinder the usage of common navigation solution. As discusses in Chapter 7, many local planners have been proposed combining Deep Learning (DL) with computer vision [204, 21, 19] or other sensor processing methods [213–215]. However, local planners provide a solution for intra-row navigation only, and therefore a global path generator is always needed. In a complex scenario such as a row-based environment, where traversing each row is the practical navigation goal, the problem of developing an efficient global path planner has been quite neglected by the research community. Existing solutions usually tackle the problem by clustering visual data obtained from satellites or UAVs. For example, in [216] authors use a classical clustering method to identify vineyard rows from a 3D model of the terrain reconstructed from UAV data and then compute the path accordingly. However, as pointed out in [217], the extraction of relevant information about rows geometry from images can be a complex task, in addition to being extremely computationally expensive. This limitation also holds considering other approaches besides clustering. For instance, in [218] authors adopted 3D point cloud aerial photogrammetry to detect the structure of vineyards.

Recently, the DeepWay method [219] has been proposed to efficiently combine DL and clustering for the generation of start and end row waypoints given an occupancy grid of the vineyard. Moreover, novel contributions adopted the same paradigm and training procedure to extend the coverage to arbitrary unstructured environments [220]. Despite being an important baseline for row-based path generation, DeepWay leaves substantial space for improvement. In particular, it applies DBSCAN clustering [221], followed by a complex heuristic geometrical post-processing heavily based on angle estimation, to discriminate start waypoints from end waypoints. However, this method performs poorly in a wide range of real-world situations, including curved crops and rows of different lengths.

In this chapter, we complete the description of the Deep Learning pipeline for row-based crops navigation introduced in Chapter 7 with a novel solution for waypoint generation that combines DL with a contrastive clustering approach [22]. To this end, we conceive a new DNN architecture to simultaneously predict the position of the navigation waypoints for each row and cluster them in a single forward step. Hence, we train our model with an additional contrastive loss on a synthetic dataset of top-view vineyard maps and test it on manually-labeled real satellite images. The experimentation conducted demonstrates that the proposed solution successfully predicts precise waypoints also in real-world crop maps. We also consider complex conditions such as curved rows, differently from previous solutions based on classical clustering algorithms.

8.1 Methodology

Due to its intrinsic nature, every row-based crop is characterized by a set of lines or curves that identify two regions comprising the starting and ending points of each row, respectively. In this scenario, a robotic path should cover the whole field, and it can be divided into intra-row segments, that connect the starting region to the ending region, and inter-row segments, that connect two starting or two ending points. Given an optimal estimation of these starting and ending waypoints, it is possible to plan a full-coverage path in the row-based environment simply by alternating intra-row and inter-row segments. Therefore, the planning process heavily relies on two main steps: waypoint estimation, which identifies candidates for the points of interest, and waypoint clustering, which assigns each estimated point to one of the two regions.

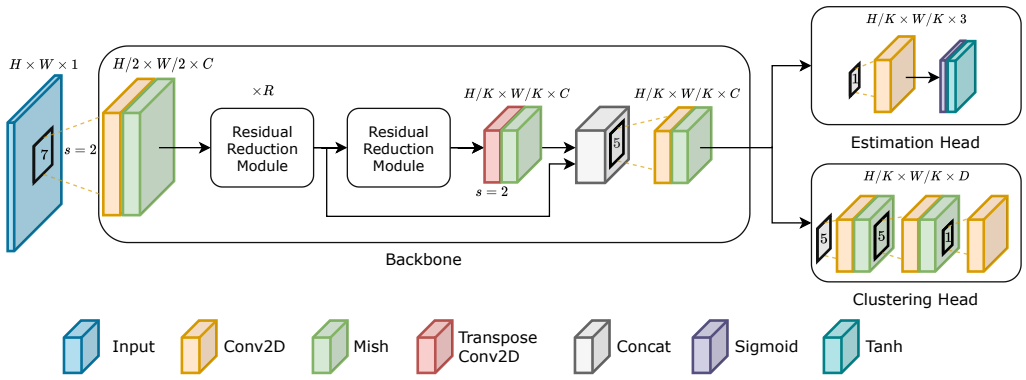


Fig. 8.1 Architecture of the backbone and the two regression heads. The number of residual reduction modules in the main block R determines the backbone compression factor $K = 2^{R+1}$.

Following the same approach presented in [219], we frame the waypoint generation process as a regression problem, in which we estimate the coordinates of the points with a deep neural network, starting from a top-view map of the environment. The map consists of a 1-bit single-channel occupancy grid that identifies with 1 the plant rows and with 0 the free terrain. Therefore, this kind of estimation process can be easily applied to geo-referenced segmented masks of the target fields obtained from satellites or UAV imagery. The waypoints and the planned path can then be converted from the image reference system to a Global Navigation Satellite System (GNSS) reference frame to be used in real-world navigation. In addition to waypoint detection, differently from classical unsupervised methodologies for point clustering, we propose a supervised approach based on a contrastive loss to perform point assignment. Therefore, the proposed model simultaneously performs both estimation and clustering with a single forward pass, without the need for complex post-processing operations based on heuristic geometrical-based rules.

8.1.1 Backbone Design

We implement the model as a convolutional neural network characterized by a feature extraction backbone, followed by two specialized heads. A head is responsible for the estimation task, while the other deals with clustering.

The backbone is designed following the same architecture used in [219]. The basic block of the network is the residual module, characterized by a stack of a 2D convolution and spatial and channel attention [63]. Each residual block is

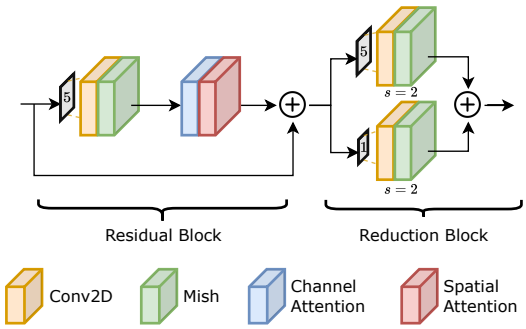


Fig. 8.2 Residual reduction module architecture. The channel and spatial attentions are implemented as in [63].

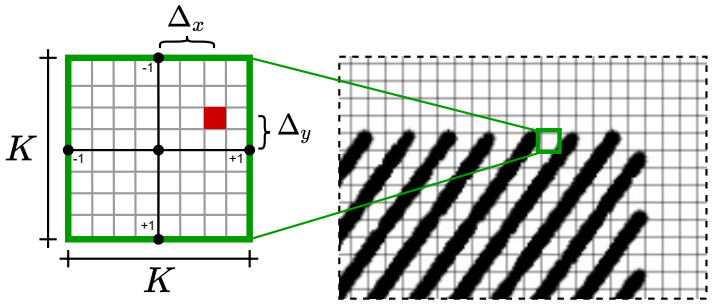


Fig. 8.3 The input occupancy grid is subdivided into a grid of $K \times K$ cells. For each cell, the waypoint estimation head outputs the probability p of a waypoint presence, as well as the relative horizontal and vertical displacements with respect to the cell center $\mathbf{\Delta} = (\Delta x, \Delta y)$.

followed by a reduction module characterized by convolutions with stride 2 that progressively halve the spatial dimensions. The backbone is a stack of R residual reduction modules, made by combining a residual module and a reduction module. The final part of the network is made by an additional downsampling block, followed by a transposed convolution upsampling stage, all arranged in a residual fashion. This combination of compression and expansion has been proved very effective for different computer vision tasks such as segmentation [222] and representation learning [223]. Overall, the model performs a dimensionality compression of a factor of 2^{R+1}, where R is the number of residual reduction modules in the main block. The complete backbone structure is detailed in Fig. 8.1 and Fig. 8.2.

8.1.2 Waypoint Estimation

The waypoint estimation is framed as a regression problem, similarly to object detection approaches in computer vision [162]. In particular, given an input occupancy grid map X with dimensions $H \times W$, we subdivide it into a grid of $K \times K$ cells. Each cell is responsible for predicting the probability p that a waypoint falls in that region, as well as its relative horizontal and vertical displacements with respect to the cell center $\boldsymbol{\Delta} = (\Delta x, \Delta y)$. The displacements are defined in the range $[-1, +1]$ and represent a shift relative to half of the cell dimension, with -1 identifying the left/top borders and $+1$ the right/bottom ones. An example of prediction with its correspondent displacements is shown in Fig. 8.3. Given a prediction $\hat{\boldsymbol{p}}_{out} = (\hat{x}_{out}, \hat{y}_{out})$ in the output reference frame, the waypoint coordinates in the input reference frame $\hat{\boldsymbol{p}}_{in}$ can be reconstructed with the following equation:

$$\hat{\boldsymbol{p}}_{in} = \hat{\boldsymbol{p}}_{out}\, K + \frac{K}{2} + \boldsymbol{\Delta}\frac{K}{2} \tag{8.1}$$

The waypoint estimation head maps the high-level features extracted with the backbone to the output space with a 1x1 convolution. The backbone compression factor 2^{R+1} corresponds to the grid dimension K. Therefore, the output tensor of the estimation branch has a dimension of $H/K \times W/K \times 3$. We apply a sigmoid activation to the probability output and a tanh activation to the displacement outputs. We optimize the network for the waypoint estimation task with a weighted mean squared error loss. For each output cell $\boldsymbol{u}_{i,j}$, the estimation loss is therefore computed as:

$$l_{i,j}^{\text{est}} = \mathbb{1}_{i,j}^{\text{wp}} \lambda \|\boldsymbol{u}_{i,j} - \hat{\boldsymbol{u}}_{i,j}\|_2 + (1 - \mathbb{1}_{i,j}^{\text{wp}})(1-\lambda)\|\boldsymbol{u}_{i,j} - \hat{\boldsymbol{u}}_{i,j}\|_2 \tag{8.2}$$

where $\mathbb{1}_{i,j}^{\text{wp}} \in \{0,1\}$ is an indicator Boolean function evaluating 1 if a waypoint is present in that cell, and λ is the relative constant that weights differently positive and negative cells.

At inference time, we get the list of predicted waypoints by considering all the cells with probability p over a certain threshold t_p. As in standard object detection methodologies, we also apply a suppression algorithm to decrease the number of redundant predictions that typically occur when multiple adjacent cells detect the same waypoint. The algorithm identifies all the groups of predictions with Euclidean distance within a certain threshold t_{sup} in the input reference frame. For each group,

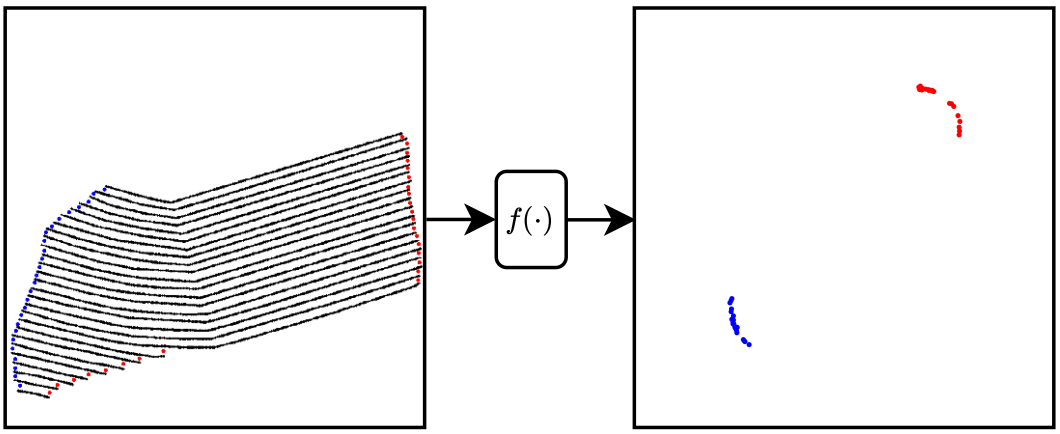


Fig. 8.4 In the latent space mapped by $f(\cdot)$, points of the same cluster appear closer together with respect to points of the other cluster. The mapping function $f(\cdot)$ is implemented with the backbone and the clustering head together. In this example, the latent space has a dimensionality $D = 2$.

the point with highest confidence p is selected, while the remaining predictions are discarded.

8.1.3 Contrastive Clustering

Once the waypoints are detected, they should be assigned to starting or ending regions. This task can be seen as a simple binary classification, in which the labels represent the two clusters. However, in this scenario the actual assigned label is not relevant, as the only fundamental aspect is whether points of the same group are assigned the same label. The aim is to discriminate the points of the two regions without caring about which of them is classified as starting or ending. Indeed, an optimal path can be successfully planned regardless of the choice of the starting cluster. This invariance cannot be guaranteed by supervised classification.

For this reason, we model the clustering problem as a supervised representation learning process. Given the two sets of points $A = \{\boldsymbol{p} \mid \boldsymbol{p} \in \text{first cluster}\}$ and $B = \{\boldsymbol{p} \mid \boldsymbol{p} \in \text{second cluster}\}$, we want to find a non-linear mapping $f(\cdot)$ such that

$$d\big(f(\boldsymbol{p_i}), f(\boldsymbol{p_j})\big) \ll d\big(f(\boldsymbol{p_i}), f(\boldsymbol{p_k})\big) \quad \text{for} \quad \boldsymbol{p_i}, \boldsymbol{p_j} \in A \,,\, \boldsymbol{p_k} \in B \tag{8.3}$$

and vice versa, where d is a distance measure. In the latent space mapped by $f(\cdot)$, points of different clusters are well-separated according to distance d. This means

that a simple clustering method such as K-means [224] can successfully discriminate the two groups in the latent space, as shown in Fig. 8.4. Inspired by the contrastive framework used for unsupervised learning in [225], we select as distance metric d the inverse of the cosine similarity:

$$\text{sim}(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u}^\top \boldsymbol{v}}{\|\boldsymbol{u}\|_2 \, \|\boldsymbol{v}\|_2} \tag{8.4}$$

For each image, we consider the N ground-truth waypoints as independent samples. Given a point $\boldsymbol{p_i}$, we consider as positive examples all the other $N/2-1$ points in the same cluster, and as negative examples the $N/2$ points of the other cluster. Therefore, we define the clustering loss contribution for the sample i as:

$$\begin{aligned} l_i^{\text{clus}} = \frac{1}{N-1} \sum_{\substack{j=1 \\ j \neq i}}^{N} \Bigg[& \mathbb{1}_{\substack{\boldsymbol{p_i}, \boldsymbol{p_j} \in A \\ \vee \, \boldsymbol{p_i}, \boldsymbol{p_j} \in B}} \log\Big(\text{sig}\left(\text{sim}\left(f(\boldsymbol{p_i}), f(\boldsymbol{p_j})\right)\right)\Big) + \\ & + \Big(1 - \mathbb{1}_{\substack{\boldsymbol{p_i}, \boldsymbol{p_j} \in A \\ \vee \, \boldsymbol{p_i}, \boldsymbol{p_j} \in B}}\Big) \log\Big(1 - \text{sig}\left(\text{sim}\left(f(\boldsymbol{p_i}), f(\boldsymbol{p_j})\right)\right)\Big) \Bigg] \end{aligned} \tag{8.5}$$

where $\mathbb{1}_{\substack{\boldsymbol{p_i}, \boldsymbol{p_j} \in A \\ \vee \, \boldsymbol{p_i}, \boldsymbol{p_j} \in B}} \in \{0, 1\}$ is an indicator function evaluating 1 if $\boldsymbol{p_i}$ and $\boldsymbol{p_j}$ are in the same cluster and 0 otherwise, while 'sig' represents the sigmoid function. Basically, this loss computes the binary cross-entropy of the cosine similarity in the latent space mapped by $f(\cdot)$ for the pair $(\boldsymbol{p_i}, \boldsymbol{p_j})$. $f(\cdot)$ is optimized to push the cosine similarity towards the maximum +1 if the points are in the same cluster and towards the minimum -1 otherwise. The final loss is computed over all the pairs (i, j) as well as (j, i) for each input image. This loss can be seen as a variation of the one used in [226, 227, 225], but instead of N groups with 2 elements each, optimized with categorical cross-entropy and softmax, we have 2 groups with $N/2$ elements each, optimized with binary cross-entropy and sigmoid.

The mapping $f(\cdot)$ is modeled by the clustering head in the output space reference system. The head is composed of two convolutional layers with Mish activation and one final 1x1 convolution with linear activation. The output tensor of the clustering branch has a dimension of $H/K \times W/K \times D$, where D is the latent space dimensionality.

Fig. 8.5 Examples of curved occupancy grids: synthetic (a) and real-world from Google Maps satellite database without (b) and with (c) manual annotation. Red and blue points are the ground-truth waypoints divided in the two clusters.

At inference time, for each waypoint detected in the estimation phase, we select the correspondent feature from the clustering head output. We can predict the clustering assignment by fitting a K-means predictor with two centroids on the selected features. Since we use the cosine similarity in the loss computation, we are optimizing the clustering in the normalized latent space. For this reason, the features should be divided by their Euclidean norm before clustering. This normalization decreases by one the latent space dimensionality, and therefore the minimum number of dimensions D for the clustering head is 2.

8.2 Experimental Setting

In this section, we present all the details of our experimentation. We describe the datasets used for network training and testing as well as the main hyperparameters adopted during the training phase.

8.2.1 Dataset Description

Considering the lack of open datasets of row crops bird-eye maps and the time required to manually annotate a large set of real images, we define a method to build realistic synthetic occupancy grids to train the model. We modify the method presented in [219] to extend it to both straight and curved occupancy grids. The generation process can be summarized as follows:

1. sample a uniformly random number of rows $n \in [10, 50]$ and angle $\alpha \in [-\pi/2, \pi/2]$;

2. generate row centers with a random inter-row distance, along the line perpendicular to α and passing through the image center;
3. generate random field borders and find starting and ending points for each row with orientation α;
4. to create curved maps, add a random displacement to the row centers and compute a quadratic Bézier curve with the starting, ending and center points as control points; this ensures that the curves are continuous and smooth;
5. generate the occupancy grid by drawing circles with random radius $r \in [1,2]$ pixels to model irregularities in the row width
6. create random holes in the rows to emulate segmentation errors or missing plants;
7. compute the $N = 2n$ ground-truth waypoints as the mean points of the lines connecting the ending points of the rows with the adjacent ones.

To further increase variability, we randomly add displacement noise every time we sample a point coordinate during the generation process. We select $H = W = 800$ pixels as input dimension for all the generated images. To investigate the effect of including synthetic curved images in the training set, we randomly generate two independent datasets, one with straight rows only, the other with both straight and curved rows. Overall, each dataset contains 3000 images for training, 300 for validation, and 1000 for testing. In addition to the synthetic data, we manually annotate real row-based images of vineyards and orchards from Google Maps (100 straight and 50 curved). These satellite images are fundamental to test the ability of the network to generalize to real-world scenarios and to prove the effectiveness of the synthetic generation process. Fig. 8.5 shows examples of both synthetic and manually-annotated images.

8.2.2 Network Training

To select the best hyperparameters, we perform a random search over a set of reasonable values. For all the convolutional layers, we set a kernel size of 5 and channel dimension $C = 16$. For the main block of the backbone, we set the number of residual reduction modules $R = 2$. Therefore, the backbone compression factor and output cell dimension is $K = 2^{(R+1)} = 8$. We set the clustering space dimensionality

Table 8.1 Performance of waypoint estimation on both straight and curved test datasets. We first test the model on our synthetic datasets (Straight Synth, Curved Synth) and then validate the results on manually annotated occupancy grids obtained from real satellite images (Straight Real, Curved Real). For each test set, we compare the results of the model trained on straight rows with those obtained training on curved rows. We report the mean and standard deviation for the Average Precision AP_r, where r is the maximum accepted distance in pixels between predicted and ground-truth waypoints.

Test	**Train**	AP_2	AP_3	AP_4	AP_6	AP_8
Straight Synth	Straight	0.6404 ± 0.0171	0.9284 ± 0.0088	0.9856 ± 0.0021	0.9991 ± 0.0001	0.9993 ± 0.0001
	Curved	0.5751 ± 0.0241	0.8921 ± 0.0107	0.9743 ± 0.0022	0.9979 ± 0.0001	0.9984 ± 0.0001
Straight Real	Straight	0.5191 ± 0.0288	0.8155 ± 0.0109	0.9116 ± 0.0032	0.9482 ± 0.0017	0.9507 ± 0.0024
	Curved	0.4597 ± 0.0166	0.7634 ± 0.0076	0.8788 ± 0.0089	0.9391 ± 0.0052	0.9433 ± 0.0049
Curved Synth	Straight	0.5143 ± 0.0193	0.8224 ± 0.0236	0.9232 ± 0.0166	0.9726 ± 0.0078	0.9768 ± 0.0065
	Curved	0.5664 ± 0.0226	0.876 ± 0.0066	0.9632 ± 0.0009	0.9937 ± 0.0006	0.9949 ± 0.0006
Curved Real	Straight	0.4685 ± 0.0906	0.7110 ± 0.0625	0.8125 ± 0.0625	0.8802 ± 0.0374	0.8891 ± 0.0355
	Curved	0.5327 ± 0.0269	0.8010 ± 0.0095	0.8881 ± 0.0094	0.9333 ± 0.0026	0.9374 ± 0.0033

to $D = 3$. Thus, the output tensors have both a dimension of $100 \times 100 \times 3$. The resulting network is a lightweight model with less than 73,000 parameters. We select Adam [50] as optimizer with a constant learning rate of $\eta = 3e{-}4$ and batch size of 16. Experimentally, we find more effective to first train the estimation head and the backbone together with the loss of Eq. 8.2. We set the loss weight to $\lambda = 0.7$ to compensate for the high imbalance in the number of positive and negative cells and stabilize the training. We then freeze the backbone weights and train the clustering head only with the loss of Eq. 8.5. To highlight the challenge posed by curved scenarios, we independently train the model on both the straight and curved training sets. We train each model for a total of 200 epochs on an Nvidia 2080 Ti GPU using the TensorFlow 2 framework. To obtain significant statistics, we run each training session three times, so that the results can be described in terms of mean and standard deviation.

8.3 Results

In this section, we report and comment the main results regarding both waypoint detection and clustering. Visual examples are included as well, to give a qualitative idea of the performance of our model. We extensively test our approach on both straight and curved rows, including a final evaluation on real satellite data. All the related code is open source and available online[1].

[1] www.github.com/fsalv/ClusterWay

8.3.1 Waypoint Estimation

As regards waypoint estimation, we use Average Precision (AP_r) as principal metric, considering different values of the range threshold r, such that a waypoint is considered correctly detected if its Euclidean position error in pixels is smaller than r. In this way, we can highlight the precision of the model at different levels of proximity. The AP is commonly used for evaluating object detection tasks [228, 229] and is computed as the area-under-the-curve of the precision-recall plot obtained varying the confidence threshold t_p. The waypoint estimation results are reported in Table 8.1, where each value is detailed with its mean and standard deviation. All the tests are performed setting a waypoint suppression threshold equal to the minimum inter-row distance of the synthetic datasets, $t_{sup} = 8$ pixels.

The first important result is the model trained on curved crops being able to reach an AP_8 of about 94% on all four test scenarios. This achievement confirms the effectiveness of our model far beyond the synthetic training scenario, as real satellite data does not seem to create substantial performance drops (5.7% at worst). Looking at lower values of r, the synthetic-to-real gap rises to 11.5%, showing how the model is able to estimate synthetic waypoints with higher precision. The model trained on straight crops achieves excellent performance on its corresponding test set and even on real satellite data, but generalizes poorly on curved rows: the precision drop reaches 11% on AP_8 and even 22% considering AP_3. On the contrary, the model trained on curved crops scales very well on straight scenarios. This outcome confirms the importance of training on curved crops to obtain robust models able to cope with challenging situations.

8.3.2 Waypoint Clustering

As regards waypoint clustering, we adopt two separate metrics. The first is an adjusted binary accuracy, assigning a score of 0 to the worst outcome (all the points in the same cluster, meaning 50% of the points correctly clustered) and 1 to perfect clustering. However, the number of waypoints in a crop is variable and accuracy alone does not give an insight of the distribution of errors among different samples. For example, crops with a small number of waypoints tend to be easier to cluster than dense ones. Considering the fact that full-coverage path planning is possible only if every waypoint is correctly clustered, we add a clustering error metric computing the average number of wrongly labeled points per image. The results are detailed in

Table 8.2 Performance of waypoint clustering on both straight and curved datasets, comparing our approach with K-means and the DBSCAN pipeline proposed by [219]. We first test models on our synthetic datasets (Straight Synth, Curved Synth) and then validate the results on real occupancy grids obtained from satellite images (Straight Real, Curved Real). For each test, we compare the results of models trained on straight rows with those obtained training on curved rows. We report the mean adjusted accuracy and clustering error with their standard deviations.

Test	Method	Train	Adjusted Accuracy	Clustering Error
Straight Synth	K-means	Straight	**1.0000 ± 0**	**0 ± 0**
		Curved	0.9913 ± 0.0076	0.6667 ± 0.5774
	DBSCAN	Straight	**1.0000 ± 0**	**0 ± 0**
		Curved	0.9724 ± 0.0240	2.0000 ± 1.7321
	Ours	Straight	0.9994 ± 0.0003	0.0187 ± 0.0114
		Curved	0.9985 ± 0.0006	0.0527 ± 0.0219
Straight Real	K-means	Straight	0.4243 ± 0.1037	26.3333 ± 7.0238
		Curved	0.4635 ± 0.0873	26.0000 ± 5.1962
	DBSCAN	Straight	0.9532 ± 0.0429	2.3333 ± 2.0817
		Curved	0.9585 ± 0.0026	2.0000 ± 0
	Ours	Straight	0.9707 ± 0.0135	1.0400 ± 0.5197
		Curved	**0.9716 ± 0.0123**	**0.7700 ± 0.3012**
Curved Synth	K-means	Straight	0.9714 ± 0.0336	1.0000 ± 1.0000
		Curved	0.9885 ± 0.0199	0.3333 ± 0.5774
	DBSCAN	Straight	0.9563 ± 0.0757	1.3333 ± 2.3094
		Curved	0.8898 ± 0.0337	3.0000 ± 1.0000
	Ours	Straight	0.9823 ± 0.0138	0.3414 ± 0.3278
		Curved	**0.9992 ± 0.0006**	**0.0127 ± 0.0038**
Curved Real	K-means	Straight	0.2443 ± 0.0984	73.3333 ± 29.2632
		Curved	0.2721 ± 0.1493	70.0000 ± 19.5192
	DBSCAN	Straight	0.7247 ± 0.2734	27.0000 ± 25.5343
		Curved	0.5181 ± 0.1061	45.3333 ± 6.6583
	Ours	Straight	0.8571 ± 0.0924	3.4667 ± 2.4437
		Curved	**0.9344 ± 0.0116**	**1.1933 ± 0.1858**

Table 8.2. To have a baseline, we compare our approach with the K-means algorithm directly applied in the image reference system and the DBSCAN clustering with geometrical assignment approach proposed by [219]. All the clustering tests are performed setting the confidence threshold to $t_p = 0.4$ and the waypoint suppression threshold to $t_{sup} = 8$ pixels. As for the previous results, each value is reported with its mean and standard deviation.

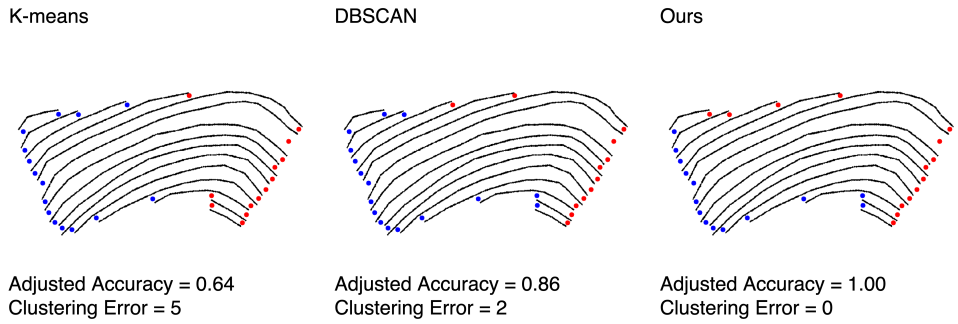


Fig. 8.6 Examples of clustering on a real-world curved sample: K-means and DBSCAN pipeline [219] are not able to correctly cluster the predicted waypoints; on the other hand, the proposed method correctly assigns the points.

Our methodology achieves remarkable results, outperforming or at least matching existing solutions in all the testing scenarios. In particular, both the training strategies (based on straight and curved crops) approach perfect clustering on the synthetic straight dataset and generalize well to real crops. On the contrary, K-means, which perfectly works for the well-separated synthetic samples, loses more than half of its adjusted accuracy and presents a very high clustering error when switching to real test rows, mainly due to the irregular shapes typical of real-world vineyards. The DBSCAN pipeline, instead, is able to generalize to straight satellite crops, since the methodology was specifically designed to cope with real-world straight rows.

As regards curved test sets, K-means clustering is totally unable to generalize to the real dataset. At the same time, also the DBSCAN pipeline results drop significantly when switching to real samples, due to its heavy dependence on angle estimation. Our model, trained on straight rows, obtains 0.98 adjusted accuracy and 0.34 clustering error on synthetic data, outperforming both the baselines. However, it struggles to generalize to real crops, reaching an adjusted accuracy of 0.86. On the other hand, the model trained on curved data outperforms the baselines in synthetic and real data, where it achieves an adjusted accuracy of 0.93. This result can be considered extremely positive, taking into account the strong challenges present in satellite data. In particular, a clustering error of 1.19 is remarkably smaller than those obtained by K-means and DBSCAN. In conclusion, these results confirm how the proposed methodology, combined with a well-devised generation process of curved synthetic samples, allows path planning even in challenging scenarios.

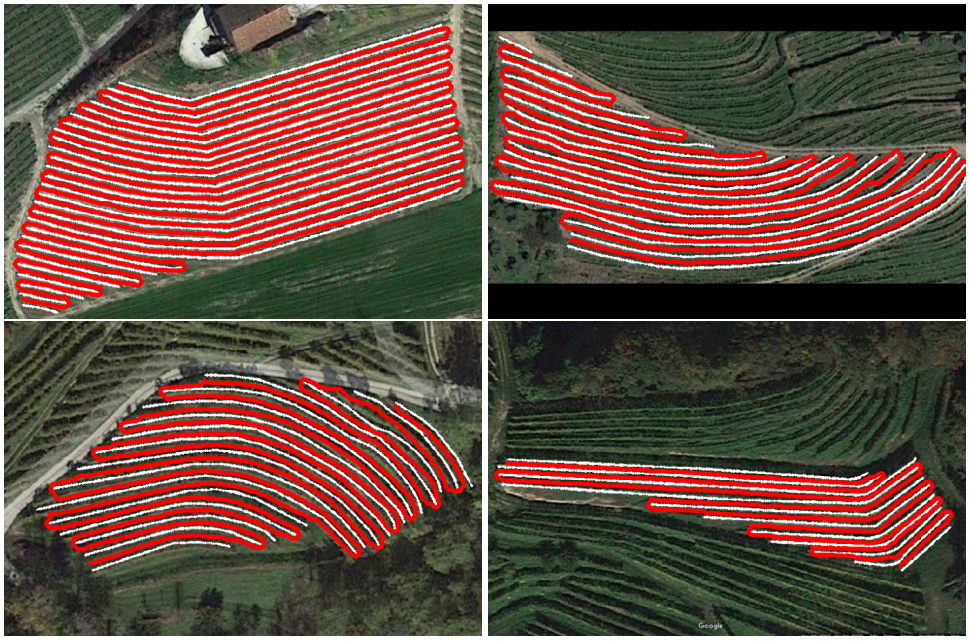


Fig. 8.7 Examples of full-coverage path planning in real-world curved vineyards taken from Google Maps satellite database.

8.3.3 Qualitative Results

To give further insight into the performance of the proposed methodology, we present some qualitative examples on real-world curved samples. Fig. 8.6 shows a comparison between the three clustering methodologies. K-means and the DBSCAN pipeline are clearly unable to correctly assign points in challenging scenarios. Finally, Fig. 8.7 shows some examples of full-coverage path planning. The planning is performed by selecting the points in an A-B-B-A fashion and using the planner proposed by [230]. With geo-referenced maps, the planned path can be converted from the image reference system to a Global Navigation Satellite System (GNSS) reference frame to be used in real-world navigation. All the tests are performed with the model trained on the curved dataset and setting the confidence threshold to $t_p = 0.4$ and the waypoint suppression threshold to $t_{sup} = 8$ pixels.

Part IV

Generalization and Optimization of Deep Learning Models

Chapter 9

Back-to-Bones: a Domain Generalization Benchmark for Backbones

Machine Learning algorithms without generalization properties would work only in situations identical to the ones previously experienced [231]. Deep neural networks (DNNs) are powerful models capable of extracting subtle regularities from training data. Nevertheless, they often fail to generalize to out-of-training data. Indeed, disparate independent studies report how neural networks could easily fail without effective generalization capabilities, hindering the introduction of novel high-tech systems in real-world [232, 233]. In Chapters7 and 10 the segmentation task on the multi-crop dataset AgriSeg and in real robotics settings demonstrate the challenges of training fully reliable DNNs. Indeed, in autonomous driving different environments and circumstances not encountered during the training phase can be faced. They can be caused by light, weather, background, synthetic textures, and they represent what is commonly called a domain gap in the data. Domain Generalization (DG) aims at training models that generalize to out-of-distribution (OOD) data. The access to a set of source datasets provides a predictor with the ability to extract and learn general invariant patterns, which are, hypothetically, also recognizable in the target domain dataset [234, 235]. As an extension of supervised learning, this approach aims to minimize empirical risk at training time to extrapolate an overall probability distribution from source datasets that enables accurate classification of OOD data.

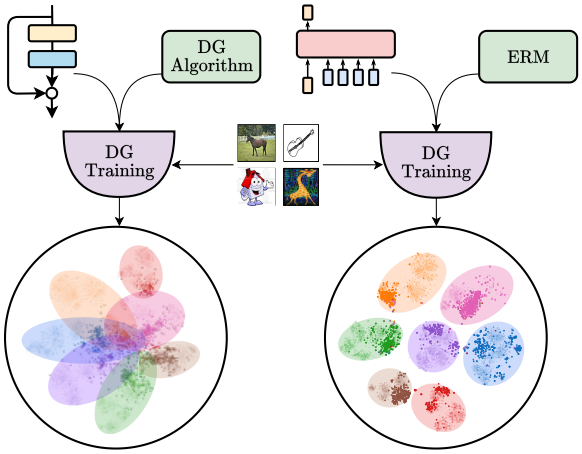


Fig. 9.1 Our experimentation proves the importance of backbones in Domain Generalization. We find that novel architectures, such as transformed-based models, lead to a better representation of data, outperforming outdated backbones, such as ResNets, and leaving marginal room for feature mapping improvement using DG algorithms.

In the last decade, aware of the tremendous impact of generalization on computer vision applications, the DG research community has tackled the problem with algorithms that aim to find invariant features that hold with novel domains. Among the constellation of proposed approaches, we identify the principal broad strategies adopted for domain generalization in augmenting the source domain [236, 237], aligning domain distributions [238–242], meta-learning [243–245], self-supervised learning [246–248], and regularization strategies [249–253]. Although methodologies have given meaningful insights about the nature of DG over the years, only recent research contributions have proposed a rigorous testing benchmark to evaluate and compare the advantages provided by DG algorithms fairly. With DomainBed [254], the results obtained by the most relevant solutions have been critically analyzed over DG datasets, unmasking the marginal positive or negative improvement obtained in most cases compared to naive empirical risk minimization (ERM). Nevertheless, the study has been carried out uniquely with ResNet50 [59] as a feature extractor. Thus, new DG algorithms are still proposed overlooking a fundamental aspect of practical Deep Learning applications: the importance of the backbone.

In this study, we claim that the domain gaps existing in realistic scenarios should be tackled starting from accurately selecting the model architecture, which is undeniably central in most Deep Learning applications (Fig. 9.1). In particular, we conduct extensive experimentation on the principal DG datasets and assess a wide variety of

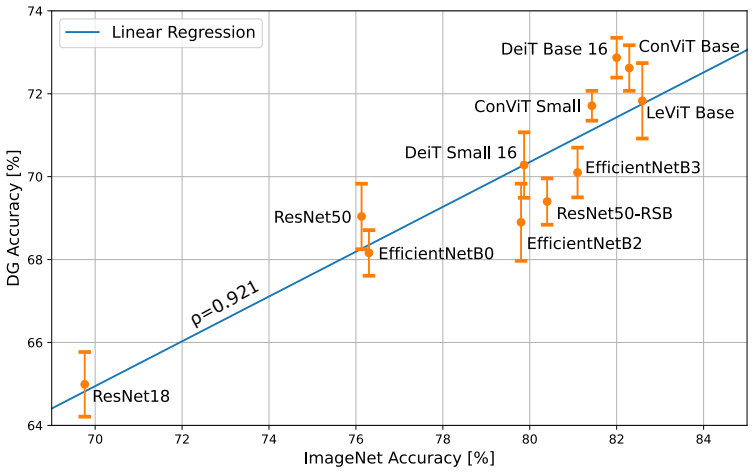


Fig. 9.2 DG accuracy achieved by tested backbones compared with their performance on ImageNet, with error bars. Regardless of different architectures and priors, we find a strong linear correlation between the two metrics ($\rho = 0.921$). In 9.2.1, we also compare DG accuracy with the number of parameters, finding a much weaker correlation.

backbone architectures, from novel vision transformers to standard convolutional models. Our results demonstrate an evident linear correlation between large-scale single-domain classification accuracy and domain generalization performance (Fig. 9.2). Moreover, we achieve state-of-the-art results in DG with naive ERM and simple data augmentation, remarking that, under fair testing conditions, the most promising algorithms presented so far give no substantial advantage. We reinforce the experimentation with a visual analysis of the feature extractors. As an outcome of this work, we release BACK-TO-BONES[1], a testbed for the Deep Learning community to evaluate and compare the domain generalization performance of newly proposed backbones.

9.1 Problem Framework

In this section, we first define necessary notations and concepts to frame the problem of domain generalization and empirical risk minimization. Secondly, we introduce a formal definition of a backbone and its constituents.

Problem Definition Given the input random variable X with values $x \in \mathcal{X}$, and the target random variable Y with values $y \in \mathcal{Y}$, the definition of *domain* is associated with the joint probability distribution P_{XY}, or $P(X,Y)$, over $\mathcal{X}$x$\mathcal{Y}$. Supervised learn-

[1] https://github.com/PIC4SeR/Back-to-Bones

ing aims to train a classifier $f : \mathcal{X} \to \mathcal{Y}$ exploiting N available labeled examples of a dataset $D = (x_i, y_i)_{i=1}^{N}$ that are identically and independently distributed and sampled according to P_{XY}. The goal of the training process is to minimize the *empirical risk* associated with a loss function $l : \mathcal{Y} \times \mathcal{Y} \to [0, +\infty)$,

$$R_{\text{emp}}(f) = \frac{1}{N} \sum_{i=1}^{N} l(f(x_i), y_i) \tag{9.1}$$

by learning the classifier f. The dataset D is the only available source of knowledge to learn P_{XY}. We refer to this basic learning method as empirical risk minimization [255].

In domain generalization, a set of different K source domains $\mathcal{S} = (S_k)_{k=1}^{K}$ is used to learn a classifier f that aims at generalizing well on an unknown target domain T. Each source domain is associated with its joint probability distribution P_{XY}^{k}, whereas $P_{XY}^{\mathcal{S}}$ indicates the overall source distribution learned by the classifier [256]. Indeed, DG aims to enable the classifier to predict well on out-of-distribution data, namely on the target domain distribution P_{XY}^{T}, by learning an overall domain invariant distribution from the source domains seen during training.

Backbone Definition We define a backbone $\mathcal{B} = f(\mathcal{A}, \mathcal{T}_B, \mathcal{D})$ as a function of three elements: the model architecture $\mathcal{A}$, the training procedure $\mathcal{T}_B$ (including optimization, regularization, and data augmentation), and the training data $\mathcal{D}$. Consequently, all three factors introduce a certain degree of variability to the domain generalization accuracy:

$$\text{DG}_{\text{accuracy}}\ (\mathcal{S}, T) = g(\mathcal{B}, \mathcal{T}_{DG}, \mathcal{N}_{exp})$$

where $\mathcal{T}_{DG}$ is the adopted DG training procedure and $\mathcal{N}_{exp}$ is the experimentation noise. $\mathcal{T}_{DG}$ usually includes a dedicated algorithm to cope with domain shifts. $\mathcal{N}_{exp}$ comprehends a systematic error due to the adopted model selection strategy and a random component caused by the stochasticity in the training process.

9.2 Back-to-Bones

We set up our experimental benchmark to run a detailed analysis of the role of feature extractors in domain generalization. Besides choosing architectures, datasets, and DG algorithms to evaluate, particular attention is given to model selection strategy

and statistical interpretation to obtain a fair and accurate benchmark. In the following subsections, we provide details on our experimental setup.

Table 9.1 Baselines comparison of different backbones for DG. We report the average accuracy over three runs and the associated standard deviation for each model. We include the results achieved by DOMAINBED with ResNet50 for reference. The models marked with * are pretrained on Imagenet21K instead of ImageNet1K. The rightmost column indicates the accuracy of the networks on ImageNet1K. In [23] we included the detailed results on single domains.

Backbone	PACS	VLCS	Office-Home	Terra Incognita	Average	ImageNet	Parameters
ResNet18	80.51 ± 0.29	74.64 ± 0.61	63.87 ± 0.36	40.93 ± 1.85	64.99 ± 0.78	69.76	11.69M
ResNet50 [254]	85.50 ± 0.20	77.50 ± 0.40	66.50 ± 0.30	46.10 ± 1.80	68.90 ± 0.68	76.13	25.56M
ResNet50	83.85 ± 0.77	76.21 ± 1.20	68.79 ± 0.21	47.32 ± 0.97	69.04 ± 0.79	76.13	25.56M
ResNet50 A1	84.52 ± 0.68	78.37 ± 0.56	72.47 ± 0.13	42.23 ± 0.87	69.40 ± 0.56	80.40	25.56M
EfficientNetB0	85.46 ± 0.65	75.16 ± 0.34	67.27 ± 0.27	44.76 ± 0.94	68.16 ± 0.55	76.30	5.29M
EfficientNetB2	87.02 ± 1.37	75.44 ± 0.20	69.35 ± 0.24	43.80 ± 1.90	68.90 ± 0.93	79.80	9.11M
EfficientNetB3	86.71 ± 0.30	78.14 ± 0.18	69.84 ± 0.08	45.70 ± 1.84	70.10 ± 0.60	81.10	12.23M
DeiT Small 16	86.22 ± 1.33	79.47 ± 0.41	72.03 ± 0.33	43.40 ± 1.08	70.28 ± 0.79	79.87	22.05M
DeiT Base 16	**88.10 ± 0.48**	79.80 ± 0.32	76.35 ± 0.36	47.22 ± 0.75	72.87 ± 0.48	82.00	86.57M
ConViT Small	87.10 ± 0.33	80.00 ± 0.34	73.90 ± 0.17	45.83 ± 0.61	71.71 ± 0.36	81.43	27.78M
ConViT Base	87.27 ± 0.40	**80.31 ± 0.67**	76.51 ± 0.25	46.37 ± 0.89	72.62 ± 0.55	82.29	86.54M
LeViT Base	87.55 ± 1.50	78.91 ± 0.50	75.16 ± 0.13	45.68 ± 1.50	71.83 ± 0.91	82.59	39.13M
ViT Small 16*	83.59 ± 0.43	79.96 ± 0.60	77.25 ± 0.33	44.12 ± 1.07	71.23 ± 0.61	81.40	22.05M
ViT Base 32*	84.00 ± 1.17	78.46 ± 0.64	76.84 ± 0.17	36.71 ± 2.07	69.00 ± 1.01	80.72	88.22M
ViT Base 16*	**88.48 ± 1.22**	**80.05 ± 0.15**	**81.47 ± 0.21**	**49.77 ± 1.28**	**74.94 ± 0.72**	84.53	86.57M

Backbones To be consistent with previous works, we include ResNet18 and ResNet50 [59] in the benchmark and compare them with some of the most successful architectures proposed in recent image classification research. We also consider the latest ResNet50 A1 [257], trained using the most recent practices in optimization and data augmentation and reaching a remarkable 80.4% top-1 accuracy on Imagenet1K. We include different sizes for each network to glimpse the effects of model dimension on DG accuracy. EfficientNet [258] demonstrated that systematical model scaling and dimension balancing yield remarkable results with fewer parameters. For this reason, we select three network versions, namely B0, B2, and B3. Finally, transformers [40] recently revolutionized Deep Learning by proving the effectiveness of self-attention for feature extraction; hence, four transformer-based architectures are included in the comparison. In particular, we choose DeiT (Small and Base) [259], ConViT [260] (both in its Small and Base configurations), and LeViT Base [261]. To provide further insights on the effect of additional pretraining data besides standard ImageNet [36], we also include Vision Transformer (ViT) [67] trained on ImageNet21K in its Small and Base versions. Regarding ViT Base, a configuration with a 32x32 patch size has been added to the standard 16x16 format to test the impact of patch size on

DG. Further information on architectural details can be found in the cited papers. We report the number of parameters for each model in the last column of Table 9.1.

Datasets Among the various datasets created explicitly for DG in the last years, we use four of the most widely adopted ones for our primary experimentation. VLCS [262] considers four previous classification datasets as domains, while PACS [263] and Office-Home [264] focus more on style shifts (e.g. from photos to cartoons, sketches, and paintings). Terra Incognita [265] comprehends several animal photos taken with camera traps placed in different locations by day and night. To those, we add DomainNet [266], a bigger and more recent dataset that contains six domains divided by style and 345 classes. We use it to further stress the generalization capability of the best-performing backbones in the presence of more transfer learning data and fewer samples per class. We omit Rotated MNIST [267] and Colored MNIST [268] since we consider them too distant from any practical application. Moreover, from our perspective, simple rotation and colorization do not constitute actual domain shifts.

DG Algorithms We choose some of the most promising DG algorithms in recent research, particularly considering their performance on DOMAINBED [254]. Moreover, we select them to explore different approaches to the DG problem. CORAL [239] and MMD [241], indeed, focus on aligning the extracted features through second-order statistics (covariance). On the other hand, Mixup [269] works directly on input images, interpolating samples from different domains and considering the loss coming from both precursors. RSC [250], instead, introduces a heuristic that discards dominant features in the label determination, stimulating the model to rely on weaker data correlations. CausIRL [270] (used in combination with MMD or CORAL) builds from a causal analysis of generalization enforcing soft domain invariance to interventions on the source domain. CAD [271] introduces a contrastive adversarial domain bottleneck to guarantee convergence to target domains that preserve the Bayes predictor. ADDG [272] exploits a double mechanism (Intra-model and Inter-model) to diversify attention between features and suppress domain-related attention.

Data Augmentation Many research works prove that data augmentation plays a fundamental role in DG, as it can partially compensate for certain domain shifts [237]. That is particularly true in the presence of style changes, as popular data augmentation strategies involve the alteration of saturation, hue, and contrast. Since

the effect of data augmentation on DG has already been investigated, in this work, we use a standard setup to keep the focus on backbones. The de-facto standard augmentation strategy for DG, which we use in our benchmark, includes random cropping keeping at least 80% of the original image, horizontal flipping with 50% probability, image grayscaling with 10% chance, and random changes in color brightness, contrast, saturation, and hue, with a maximum of 40%. Since all the models are pretrained on ImageNet1K or ImageNet21K, input images are further normalized according to the mean and standard deviation of that datasets.

Model Selection To assess the DG capability of the considered pretrained networks, we fine-tune each of them on a set of K source domains $\mathcal{S}$ and test them on a target domain T. As pointed out by [254], "a domain generalization algorithm should be responsible for specifying a model selection method" and avoid improper comparisons between results obtained adopting different selection methods. In total agreement with their recommendations, we use the *training-domain validation set* strategy, which picks the model maximizing the accuracy on a validation split of the training set (in our case 10%, uniform across domains) at the end of each epoch. This selection method assumes that the average distribution of source domains is similar to that of the target domain on which the best model is tested.

Hyperparameter Search We conduct a random search for each backbone and dataset to determine the optimal training hyperparameters for the baselines. We define a range of values for continuous arguments and a set of choices for discrete ones, running approximately 32 iterations for each search and selecting the best combination via the previously defined model selection strategy. The learning rate is bounded in the range $[10^{-6}, 10^{-2}]$, choosing its scheduler among step (90% reduction after 80% of the epochs), exponential (with a decay in the range $[0.9, 1)$), and cosine annealing. The batch size and the number of training epochs are the same for all the experimentation, fixing their values at 32 and 30, respectively. Finally, we use cross-entropy loss and select the optimizer among SGD (with a momentum of 0.9) and Adam, keeping the weight decay to $5 \cdot 10^{-4}$.

Experimental Framework Our benchmarks are developed in Python 3 using the Deep Learning framework PyTorch. As the experimentation applies transfer learning to pretrained models, we use existing implementations of the considered backbones. Only the classification head is changed, adapting the network to the different number of classes. In particular, standard ResNets are taken from the PyTorch library *torchvi-*

Table 9.2 Baseline comparison of a selection of the best backbones on DomainNet (*Clipart*, *Infograph*, *Painting*, *Quickdraw*, *Real*, and *Sketch* domains). We include the results achieved by DOMAINBED with ResNet50 for reference. The model marked with * is pretrained on Imagenet21K instead of ImageNet1K.

Backbone	C	I	P	Q	R	S	Avg
ResNet50 [254]	58.1	18.8	46.7	12.2	59.6	49.8	40.9
DeiT Base 16	69.1	25.0	55.8	17.1	69.3	57.0	48.9
ConViT Base	69.5	24.3	55.7	**17.7**	69.3	57.0	48.9
ViT Base 16*	**74.9**	**28.9**	**60.8**	17.5	**77.3**	**61.8**	**53.5**

sion[2], EfficientNets from *EfficientNet-PyTorch*[3], transformers and ResNet50 A1 from *timm*[4]. The implementations of DG algorithms are taken from DOMAINBED[5] and adapted to work with the architectures under test.

We repeat each training three times with different and randomly generated seeds to give more statistical information about accuracy results. In this way, both hyperparameter search and benchmarks cannot take advantage of the repeatability of trials, as data splitting, augmentation, and weight initialization change from one iteration to the next. Therefore, each of the results of our benchmark is reported as the mean over three repetitions, along with its standard deviation.

9.2.1 Baseline Benchmark

The first analysis of our work consists of a precise and fair benchmark of the DG capabilities of recent Deep Learning architectures for image classification, trying to determine what solutions work best and, possibly, why. Every pretrained backbone, after a hyperparameter search, is trained following the standard DG *leave-one-domain-out* procedure using the previously described model selection strategy. Our benchmark results are reported in Table 9.1 as the mean and standard deviation over three iterations.

Firstly, our benchmark highlights a strong correlation between DG accuracy and ImageNet performance. As depicted in Fig. 9.2, we find a direct proportionality between the two metrics (excluding the ViT models due to their different pretraining).

[2] pytorch.org/vision/stable/models
[3] github.com/lukemelas/EfficientNet-PyTorch
[4] github.com/rwightman/pytorch-image-models
[5] github.com/facebookresearch/DomainBed

We apply linear least-square regression and obtain a Pearson correlation coefficient $\rho = 0.921$. Indeed, a quick look at the results is sufficient to notice how newer and more performing backbones tend to achieve a higher DG accuracy on nearly all the datasets. That is primarily true for different sizes of the same architecture. ResNet50 reaches better results than ResNet18 for all the datasets, and the same happens for EfficientNet, ConViT, and ViT variants. For ResNet50, we also compare our results with those obtained by DOMAINBED and find comparable values. ResNet50 A1 benefits from its stronger pretraining, largely improving the accuracy obtained by the standard model on VLCS and Office-Home. However, Terra Incognita seems to penalize the network with its peculiar light conditions, resulting in a slight overall enhancement. Regarding different architectures, EfficientNetB2 performs very similarly to ResNet50 while the B3 version gains an additional 1% on them. Transformer-based models bring further improvements by exploiting their self-attention-based feature extraction, even in the case of DeiT Small and ConViT Small. In particular, they strongly outperform EfficientNet on OfficeHome by over 4%, while Terra Incognita is the only dataset without any significant progress. That is probably due to the peculiarity of the domains, comprehending many night shots that can be challenging even for humans and rewarding less effective ImageNet pretraining. Among other transformers, DeiT Base 16 and ConViT Base prove to be the best, the latter being slightly more performing. Finally, the three ViT models show that pretraining on a more significant amount of data improves generalization. However, only ViT Base 16 registers a considerable step forward, suggesting that the abundance of data is fully exploited only by larger models. Nonetheless, ConViT Small performs similarly to the same-sized ViT Small 16, while larger patches demonstrate to degrade the accuracy of ViT Base 32. In conclusion, our results show how better DG comes from the union of a good feature extractor architecture and an optimal pretraining, as none of the two is sufficient alone.

As an additional comparison, we plot the achieved DG accuracy compared to the number of parameters of the backbones (Fig. 9.3). Contrary to the graph of Fig. 9.2, in this case, the correlation between model dimension and generalization is much less marked, with a Pearson correlation coefficient (ρ) of 0.740. This confirms the central role of model architecture in DG tasks and our idea of backbone as the union of architecture, training procedure, and data.

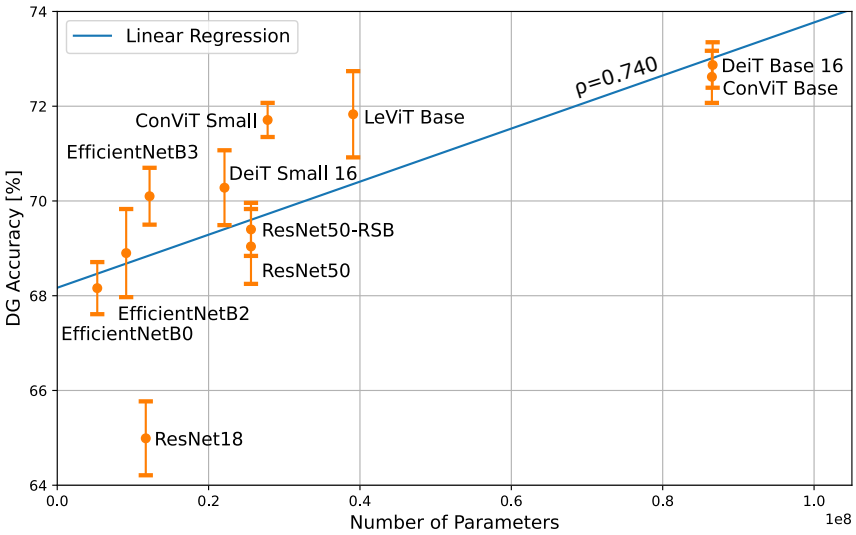


Fig. 9.3 DG accuracy achieved by tested backbones compared with their number of parameters, with error bars. We find a much weaker correlation between the two metrics ($\rho = 0.740$) than the one reported in Fig. 9.2.

Finally, we conduct an additional benchmark on the DomainNet dataset. Although representing a significant challenge for large-scale generalization, we choose to include DomainNet only in this second stage of the study due to its demanding computational nature and strong class unbalancing. Indeed, our main intention is to promote a practical and accessible benchmark that aims to become a widespread reference for DG. We select only the best three models from the previous tests for this one (DeiT Base 16, ConViT Base, and ViT Base 16). In Table 9.2, we report the results achieved on each test domain, including those obtained by DomainBed on ResNet50 for reference. It is well evident that the feature extraction capabilities of modern backbones bring substantial improvement in all the domains, with an average increase in DG accuracy up to 12.6%. Moreover, ViT further enhances the results by exploiting its stronger pretraining.

9.2.2 Model Introspection

After assessing the DG performance of different backbones, we propose a series of insights on how different architectures leverage training data to create their inner representation. First, we investigate the benefits of ImageNet pretraining for DG with a k-NN classifier, comparing ResNet50 and the best models from our benchmark. Then, we apply t-SNE [273] on the same extracted features to visualize how close

same-class and same-domain samples are and the effect of fine-tuning on DG datasets. Finally, we inspect the attention maps of one of the transformer-based models to have a qualitative insight on the region of the images it focuses on.

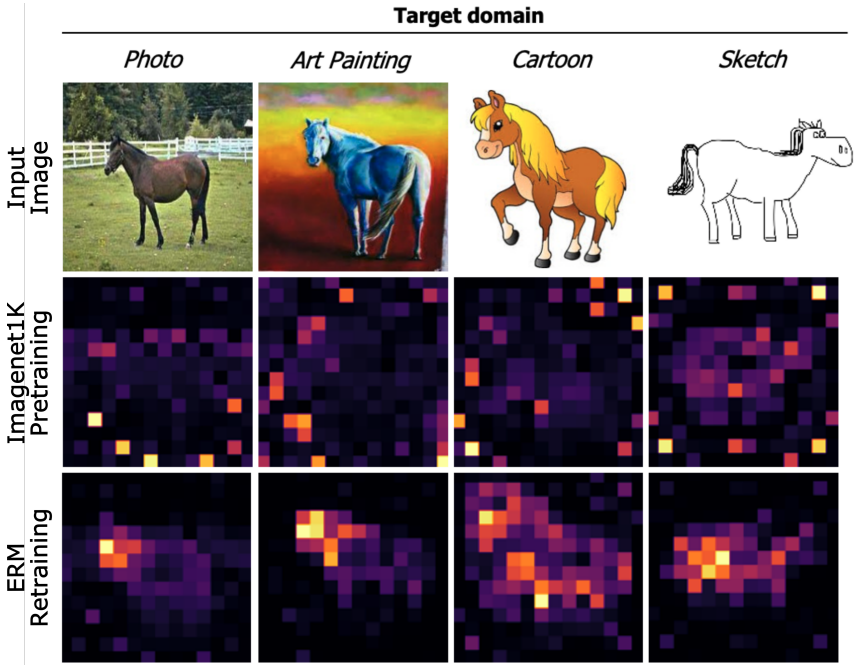


Fig. 9.4 DeiT Base attention maps when using the `[CLS]` token as a query for the different heads in the last layer. We select the same head for all examples. ERM encourages the backbone to focus on domain-invariant features, highly mitigating pretraining noise.

K-NN Evaluation Firstly, we take ResNet50 and the best-performing models from our benchmark and evaluate their ability to tackle DG without fine-tuning. To do so, we use ImageNet weights to extract features from training domains and a k-NN (with $k = 5$) to fit that data. Then, we use test-domain images for the evaluation. To have a fair comparison with our benchmark, we use the same amount of training data, leaving out 10% of samples from source domains. The results in Table 9.3 show an overall difference of about 5% between ResNet50 and transformer-based models pretrained on ImageNet1k. This outcome is consistent with the generalization boost achieved in the standard DG framework (Table 9.1), although k-NN results tend to oscillate among different datasets. On the same trend, ViT Base 16 gains an additional 10% average accuracy, thanks to its pretraining on the larger ImageNet21K dataset. This outcome suggests that learning a wider overall source distribution $P_{XY}^{\mathcal{S}}$ is always needed to tackle a substantial domain gap effectively. That pretraining alone does not guarantee the ability to extract domain-invariant features.

Table 9.3 Comparison of different feature extractors without fine-tuning, using a k-NN classifier ($k = 5$). The model marked with * is pretrained on Imagenet21K instead of ImageNet1K.

Backbone	PACS	VLCS	Office-Home	TerraInc.	Avg
ResNet50	56.04	69.57	56.26	14.75	49.16
DeiT Base 16	56.27	65.50	65.57	27.06	53.60
ConViT Base	56.83	64.50	66.63	**27.96**	53.98
ViT Base 16*	**75.14**	**75.14**	**82.72**	25.64	**64.66**

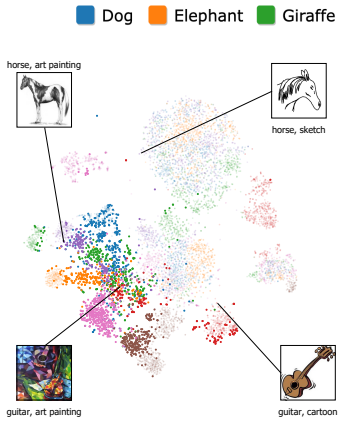


Fig. 9.5 ResNet50 (ImageNet1K)
$S = 0.1430$

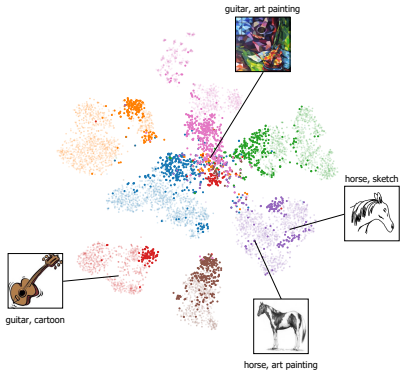


Fig. 9.6 ResNet50 ($\mathcal{S}$: $[P, C, S] \rightarrow T$: A)
$S = 0.3107$

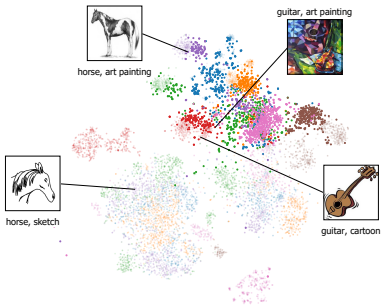


Fig. 9.7 ConViT Base (ImageNet1K)
$S = 0.1558$

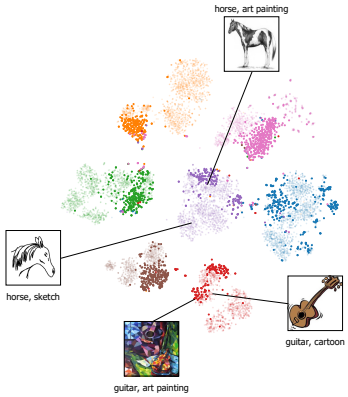


Fig. 9.8 ConViT Base ($\mathcal{S}$: $[P, C, S]$ $\rightarrow T$: A)
$S = 0.5688$

Fig. 9.9 Backbone features visualization with t-SNE on PACS (Photo (P), Art Painting (A), Cartoon (C) and Sketch (S) domains). Target domain samples are highlighted. Some image examples from different domains and classes are visualized for better interpretability. After the fine-tuning, the ConViT Base architecture achieves a better class separation than ResNet50, clustering together same-class samples of different domains.

Feature Mapping Visualization To further enlighten the role of backbones in extracting meaningful and invariant features to deal with DG, we can visualize the distributions in the feature space by projecting them in a two-dimensional space using t-SNE. Fig. 9.9 shows t-SNE visualization for ResNet50 and ConViT Base, pretrained on ImageNet1K and fine-tuned on PACS, targeting the *Art Painting* domain. For each model, we remove the classification head and extract the features for the whole dataset. The more clustered the same class features appear in the t-SNE, the more separable from other classes they are in the original space. We also include the silhouette score (S) as a quantitative metric of the separation of classes below each plot. Fig. 9.5 shows how ResNet50 pretrained on ImageNet tends to map together same-domain samples and not same-class ones, being therefore unsuitable for DG without fine-tuning. After the fine-tuning process (Fig. 9.6), the model achieves a better separation of source domain classes. However, many target domain samples are still mapped in the same space, far from the same-class source clusters (e.g. the *Art Painting* guitar example). Similarly to ResNet50, without fine-tuning, domains dominate the features space distribution of ConViT (Fig. 9.7), causing several clusters of the same class but different domains to emerge in different locations (e.g. horse samples). However, some same-class samples of more similar domains, such as the guitars of *Cartoon* and *Art Painting*, are effectively clustered together. The fine-tuning process (Fig. 9.8) distinctly pushes together same-class clusters, resulting in good generalization over the target domain. This analysis suggests that the ConViT backbone is more suited for DG than ResNet50 since it tends to give more similar representations to same-class samples from different domains. Additional feature mapping visualizations have been reported in the appendix section of the paper [23].

Self-attention Visualization In literature, DG algorithms are often presented with a qualitative analysis, highlighting the regions the network focuses on using interpretation methods such as GradCAM [274]. Indeed, heat maps are brought as evidence of their capability to push attention toward more localized and domain-invariant features. Nevertheless, this section shows that competitive backbones with naive ERM can perfectly localize class-discriminative regions. In particular, Fig. 9.4 shows the attention maps extracted using the `[CLS]` token as a query for the different heads in the last layer of the DeiT Base architecture. We provide four random examples for different target domains of PACS showing the same attention head map before and after DG fine-tuning. It is remarkable how naive ERM is able to redirect attention

towards more invariant features. Additional attention visualizations are reported in the original paper [23].

Table 9.4 Comparison between ERM and three promising DG algorithms on the best-performing backbones of our benchmark. We report the average accuracy over three runs and the associated standard deviation for each model. We highlight in bold the best result for each dataset, including ERM, when its accuracy is in the same confidence interval. We include the results achieved by DOMAINBED with ResNet50 for reference. The model marked with * is pretrained on Imagenet21K instead of ImageNet1K. We report in detail the results obtained for all the domains in [23].

Backbone	Algorithm	PACS	VLCS	Office-Home	Terra Incognita	Overall
ResNet50 [254]	ERM [255]	85.50 ± 0.20	77.50 ± 0.40	66.50 ± 0.30	46.10 ± 1.80	68.90 ± 0.68
	RSC [250]	85.20 ± 0.90	77.10 ± 0.50	65.50 ± 0.90	46.60 ± 1.00	68.60 ± 0.83
	Mixup [269]	84.60 ± 0.60	77.40 ± 0.60	68.10 ± 0.30	**47.90 ± 0.80**	69.50 ± 0.58
	CORAL [239]	**86.20 ± 0.30**	**78.80 ± 0.60**	**68.70 ± 0.30**	47.60 ± 1.00	**70.33 ± 0.55**
	MMD [241]	84.60 ± 0.50	77.50 ± 0.90	66.30 ± 0.10	42.20 ± 1.60	67.65 ± 0.78
	CausIRL CORAL [270]	85.80 ± 0.10	77.50 ± 0.60	68.60 ± 0.30	47.30 ± 0.80	69.80 ± 0.45
	CausIRL MMD [270]	84.00 ± 0.80	77.60 ± 0.40	65.70 ± 0.60	46.30 ± 0.90	68.40 ± 0.68
	CAD [271]	85.20 ± 0.90	78.00 ± 0.50	67.40 ± 0.20	47.30 ± 2.20	69.48 ± 0.95
	ADDG [272]	89.2	-	72.5	-	-
DeiT Base 16	ERM [255]	**88.10 ± 0.48**	**79.80 ± 0.32**	76.35 ± 0.36	47.22 ± 0.75	**72.87 ± 0.48**
	RSC [250]	85.37 ± 1.30	77.27 ± 0.51	76.47 ± 0.28	45.41 ± 1.50	70.97 ± 0.90
	Mixup [269]	85.67 ± 0.61	78.25 ± 0.60	75.96 ± 0.11	46.63 ± 0.49	71.32 ± 0.48
	CORAL [239]	85.13 ± 0.82	78.34 ± 0.86	76.48 ± 0.14	46.33 ± 1.83	71.38 ± 0.93
	MMD [241]	87.22 ± 0.28	78.71 ± 0.22	**77.03 ± 0.10**	**49.35 ± 1.42**	**73.08 ± 0.50**
	CausIRL CORAL [270]	83.86 ± 0.75	77.80 ± 0.40	76.12 ± 0.04	46.73 ± 0.81	71.13 ± 0.50
	CausIRL MMD [270]	85.46 ± 0.68	77.27 ± 0.42	76.53 ± 0.42	45.77 ± 1.66	71.26 ± 0.79
	CAD [271]	**87.74 ± 0.62**	79.28 ± 0.36	76.61 ± 0.15	47.46 ± 0.64	**72.77 ± 0.44**
	ADDG [272]	75.30 ± 0.34	78.28 ± 0.77	**77.58 ± 0.30**	29.14 ± 2.24	65.07 ± 0.91
ConViT Base	ERM [255]	**87.27 ± 0.40**	**80.31 ± 0.67**	76.51 ± 0.25	**46.37 ± 0.89**	**72.62 ± 0.55**
	RSC [250]	85.73 ± 0.81	79.05 ± 0.61	76.77 ± 0.26	44.94 ± 1.47	71.62 ± 0.79
	Mixup [269]	86.00 ± 0.45	80.00 ± 0.76	76.48 ± 0.16	43.95 ± 0.18	71.61 ± 0.39
	CORAL [239]	86.24 ± 0.24	79.62 ± 0.38	75.33 ± 0.22	44.41 ± 1.33	71.40 ± 0.54
	MMD [241]	86.84 ± 0.63	**80.72 ± 0.55**	**77.94 ± 0.31**	**46.78 ± 1.22**	**73.07 ± 0.68**
	CausIRL CORAL [270]	84.71 ± 0.31	79.14 ± 0.69	77.05 ± 0.16	45.63 ± 2.03	71.63 ± 0.80
	CausIRL MMD [270]	86.59 ± 0.96	**80.30 ± 0.56**	**77.92 ± 0.35**	**46.85 ± 0.59**	**72.92 ± 0.61**
	CAD [271]	**87.42 ± 0.66**	79.99 ± 0.41	**77.71 ± 0.09**	**46.77 ± 3.31**	**72.97 ± 1.12**
	ADDG [272]	86.34 ± 0.76	79.79 ± 0.30	76.29 ± 0.33	43.97 ± 1.75	71.60 ± 0.78
ViT Base 16*	ERM [255]	**88.48 ± 1.22**	80.05 ± 0.15	81.47 ± 0.21	49.77 ± 1.28	**74.94 ± 0.72**
	RSC [250]	86.58 ± 2.14	79.59 ± 0.63	78.74 ± 0.64	40.79 ± 1.41	71.42 ± 1.20
	Mixup [269]	**88.62 ± 0.54**	**80.77 ± 1.28**	**82.93 ± 0.07**	48.59 ± 0.92	**75.23 ± 0.70**
	CORAL [239]	84.60 ± 1.31	**80.89 ± 0.49**	80.92 ± 0.25	**50.58 ± 0.26**	74.25 ± 0.58
	MMD [241]	87.99 ± 0.08	79.54 ± 0.37	81.71 ± 0.28	49.40 ± 2.45	**74.66 ± 0.79**
	CausIRL CORAL [270]	**88.26 ± 1.09**	80.10 ± 0.91	81.73 ± 0.13	47.29 ± 2.64	74.35 ± 1.19
	CausIRL MMD [270]	86.57 ± 1.13	79.48 ± 1.12	81.62 ± 0.22	49.52 ± 0.58	74.30 ± 0.76
	CAD [271]	87.44 ± 0.53	78.79 ± 2.43	79.80 ± 0.36	39.45 ± 4.15	71.37 ± 1.87
	ADDG [272]	75.33 ± 0.54	77.77 ± 0.32	77.72 ± 0.09	25.60 ± 0.64	64.11 ± 0.40

9.2.3 Domain Generalization Algorithms

Domain generalization research mainly focuses on studying non-trivial algorithms to reduce the effect of domain shifts on classification accuracy. However, these algorithms are uniquely proposed in combination with outdated backbones such as ResNet50, ResNet18, or even AlexNet. According to the results in Table 9.1, recent backbones can provide significant improvements compared to ResNet50 with simple ERM. At this point, it is worth determining whether the application of DG algorithms brings a further boost in generalization to our baselines. To do so, we combine some of the most promising and recent algorithms available on DOMAINBED with three of our best baselines. We evaluate the methods introduced at the beginning of this Section (MMD, CORAL, Mixup, RSC, CAD, CausIRL CORAL, CausIRL MMD, and ADDG) using ViT Base 16, DeiT Base 16, and ConViT Base as backbones and repeating each training three times. Table 9.4 reports the obtained results, composed of average accuracy and associated standard deviation. Results obtained with ResNet50 are also reported directly from DOMAINBED for the same group of datasets as a reference. The only exception is the most recent ADDG, which the authors have not tested on VLCS and Terra Incognita and does not report standard errors. As highlighted by the values in bold, the overall performance of ERM is equal to or better than other DG algorithms for all the considered datasets and backbones. Indeed, even where another methodology slightly outperforms ERM, the accuracy results mostly fall in the same confidence interval and hence differ very little statistically. We can then conclude from our experimentation that DG algorithms improve generalization properties marginally or even negatively for transformer-based backbones. This outcome extends the recent findings of DOMAINBED to other baselines and strongly reinforces the belief that choosing an effective backbone is the first step towards filling domain gaps. A more detailed presentation and discussion of the results obtained, including the results on each single domains, can be found in the original paper [23].

Chapter 10

Crop Segmentation with Knowledge Distillation: Domain Generalization on the AgriSeg dataset

Among all the DL solutions developed for precision agriculture [275], semantic segmentation is one of the most adopted perception techniques [276], being used to identify objects on different scales: detailed leaf disease [277, 278], single fruits or branches [279, 280], crop rows [199], and entire fields [281, 282]. However, as shown in the robotic application in Chapter 7, operating autonomously in agricultural environments may present peculiar generalization challenges due to weather or lighting conditions, terrain, and plant shapes and colors. According to this, DL models easily fail in realistic applications without effective generalization ability, leading autonomous systems to failure [232, 233]. Moreover, the scarcity of task-specific labeled data has recently favored the practice of synthetic data generation, leading to an additional Simulation-to-Reality (Sim2Real) gap problem [283].

For this reason, robustness in realistic scenarios needs to be investigated and enhanced with a Domain Generalization (DG) approach. DG is a set of representation learning techniques that aims to train DL models capable of generalizing to unseen domains, i.e., out-of-distribution (OOD) data. In Chapter 9 the DG problem has been formulated and framed in the landscape of existing methodologies, underlining the importance of the backbones and proposing a rigorous benchmark [23], together with [254]. However, these studies are limited to the image classification task and

DG methods are often evaluated on small artificial datasets [256]. The application of generalization methods to realistic tasks is still limited to a few attempts [284, 285, 24].

In the meantime, segmentation across multiple scenarios has been studied through the design of massive foundation models [286] and specific DG methods. As we aim to push the limits of generalization for small and efficient architectures, we focus on the latter approach. In particular, [287] proposed an Instance Batch Normalization (IBN) block for residual modules to avoid networks' bias toward low-level domain-specific features like color, contrast, and texture. [288], on the same line, proposed a permuted Adaptive Instance Normalization (PAdaIN) block, which works at both low-level and high-level features, randomly swapping second-order statistics between source domains and hence regularizing the network towards invariant features. [284] proposed RobustNet, a model incorporating an Instance Selective Whitening (ISW) loss disentangling and removing the domain-specific style in feature covariance. [285] proposed to extract domain-generalized features by leveraging a variety of contents and styles using a wild dataset. Differently, [289] has been the first attempt to apply knowledge distillation in the DG framework for classification tasks proposing a gradient filtering approach. [290], instead, proposed Cross-domain Ensemble Distillation (XDED) to extract the knowledge from domain-specific teachers and obtain a general student. However, this setup was only applied to classification, while the authors used a different approach for segmentation based on a single training domain. This was probably because largely-used segmentation datasets do not allow benchmarking on multiple domains.

This work aims to effectively exploit knowledge distillation to enhance DG and propose a novel multi-domain benchmark for crop segmentation. For this research, we adopt the lightweight (LR-ASPP) network architecture described in Chapter 7 to perform the semantic segmentation task at a reasonable inference time on low-resources hardware. The proposed method distills knowledge from an ensemble of models individually trained on source domains to a student model that can adapt to unseen target domains as depicted in Fig. 10.1. Furthermore, we investigate the effect of feature whitening to reduce domain-specific bias and improve the ability of the model to focus on domain-independent features. To properly validate the proposed method, the synthetic multi-domain dataset for crop segmentation AgriSeg is presented, containing 10 crop types and covering different terrain styles, weather

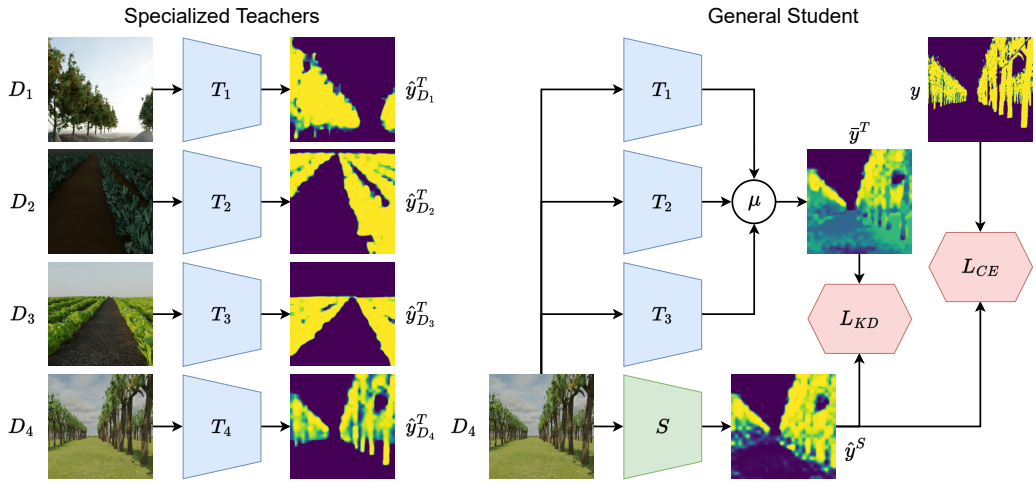


Fig. 10.1 Schematic representation of the proposed distillation methodology for crop segmentation. Ensembled specialized teachers allow the student to obtain a distillation mask ($\bar{y}^T$) that is much more informative than the label (y).

conditions, and light scenarios for more than 50,000 samples. Quantitative and qualitative experiments have been conducted to demonstrate the effectiveness of the method compared to other state-of-the-art methodologies. The code used for the experiments[1] and the AgriSeg dataset[2] are publicly available.

10.1 Methodology

A formal definition of the Domain Generalization problem has been given in the previous Chapter 9. Here, we decline the same problem in the semantic segmentation task adopting the backbone architecture describe in Chapter 7. Therefore, we provide a brief explanation of the knowledge distillation and ensemble distillation methodologies used to composed our overall solution for the AgriSeg application.

10.1.1 Knowledge Distillation

Knowledge distillation aims at transferring the knowledge learned by a *teacher* model to a smaller or less expert *student* model. It has first been proposed in [291], received greater attention after [292], and represents today one of the most promising techniques for model compression and regularization. In its original formulation

[1] https://github.com/PIC4SeR/AgriSeg
[2] https://pic4ser.polito.it/agriseg/

based on classification, knowledge distillation consists in applying an auxiliary loss to the output logits of the student $z_S(x) \in \mathbb{R}^C$, where C is the number of classes. The posterior predictive distribution of x can be formulated as:

$$P(y|x;\theta,\tau) = \frac{exp(z_y(x)/\tau)}{\sum_{i=1}^{C} exp(z_i(x)/\tau)} \tag{10.1}$$

where y is the label, θ is the set of parameters of the model, and τ is the temperature scaling parameter. To match the distributions of student and teacher, knowledge distillation minimizes the Kullback-Leibler Divergence between the two:

$$L_{\mathrm{KD}}(X;\theta,\tau) = \sum_{x_i \in X} \sum_{c=1}^{C} D_{\mathrm{KL}}(P(c|x_i;\theta_T,\tau)||P(c|x_i;\theta_S,\tau)) \tag{10.2}$$

where X is a batch of input samples and θ_T and θ_S are the parameters of teacher and student, respectively. In this work, we apply a novel knowledge distillation technique for semantic segmentation to improve the ability of models to generalize across domains.

10.1.2 Ensemble Distillation

We propose a simple yet effective training procedure based on model ensemble and knowledge distillation to encourage the model to learn domain-invariant features. We draw inspiration from the Cross-Domain Ensemble Distillation (XDED) methodology proposed for image classification in [290], which leverages the separate pretraining of a teacher for each source domain and distills the ensembled logits predicted by them. We aim to apply the same intuition to semantic segmentation, taking into account the differences between the two tasks and improving the methodology accordingly. As a remark, the authors of XDED also proposed a semantic segmentation method in the same paper, but radically different from the original XDED. The choice was probably because the adopted benchmark (GTA V $\rightarrow$ Cityscapes) provided only one source domain, and a proper cross-domain ensemble was impossible. In particular, they instead average all the output logits in a training batch that correspond to the same ground-truth label. We compare with XDED in 10.3.

In our proposed method, we improve on the work of [290] by fully adapting XDED to semantic segmentation. In particular, we train a teacher for each source domain

and ensemble them to create the distillation knowledge:

$$\bar{y}^T(x) = \frac{1}{D}\sum_{d=1}^{D} \hat{y}_d^T(x) \tag{10.3}$$

where $\hat{y_d^T}$ is the predicted logits tensor for the source domain d, $\bar{y}^T$ is the ensembled teacher logits tensor, and D is the number of source domains. The motivation behind this choice is that by averaging the predictions of different specialized models, the resulting map is much more informative than the ground-truth label. As depicted in figure 10.3, the teacher's segmentation is less confident and often assigns non-zero probabilities to disturbing elements such as grass and background vegetation. This spurious information guides the student towards implicitly recognizing what features are more likely to confound at test time, as the distillation loss has a relatively low weight in the optimization process. On the contrary, if the distillation mask is very confident, the student is guided toward being more confident and implicitly incorporates the information that a certain domain is easier to segment. This effect can be enhanced using a temperature factor. For this reason, we train the student in the standard ERM DG framework with an additional distillation loss based on the distance between the output logits of the student and the ensembled teacher. We leverage the recent findings by [293] and modify the distillation loss function to exploit the channel-wise information extracted from the network.

In particular, we apply the softmax operator ϕ along the flattened spatial dimension instead of the channel dimension before computing the loss:

$$\phi(\hat{y}^S) = \frac{exp(\hat{y}_i^S/\tau)}{\sum_{i=1}^{W\cdot H} exp(\hat{y}_i^S/\tau)} \tag{10.4}$$

where $\hat{y}_i^S$ is the i-th element of the flattened student logit tensor $\hat{y}^S$, $W \cdot H$ is its spatial dimension, and τ is the temperature. The same operation is applied to the teacher logits $\bar{y}^T$. The distillation loss is calculated as the Kullback-Leibler Divergence (KLD) between the teacher and student logits:

$$L_{\text{KD}}(\bar{y}^T, \hat{y}^S) = \frac{\tau^2}{C}\sum_{c=1}^{C}\sum_{i=1}^{W\cdot H} \phi(\bar{y}_{c,i}^T)\cdot log\left(\frac{\phi(\bar{y}_{c,i}^T)}{\phi(\hat{y}_{c,i}^S)}\right) \tag{10.5}$$

where C is the number of output channels and, hence, of semantic classes. For the specific case of binary segmentation, the formulation is simplified as the predicted mask consists of only one channel ($C = 1$).

In combination with the distillation loss, we optimize the standard cross-entropy loss between the student logits and the ground-truth labels y:

$$L_{\text{CE}}(y, \hat{y}^S) = -\sum_{i=1}^{C} y_i \cdot log(\hat{y}_i^S) \tag{10.6}$$

which for binary segmentation becomes a simple binary cross-entropy loss. The overall loss can be written as follows:

$$L(y, \bar{y}^T, \hat{y}^S) = L_{\text{CE}}(\bar{y}, \hat{y}^S) + \lambda L_{\text{KD}}(\bar{y}^T, \hat{y}^S) \tag{10.7}$$

where λ is a weighting parameter to balance the loss components. We provide a thorough ablation of the various component of our method in 10.3.2 to highlight the strong improvement on similar solutions.

10.2 Experimental Setting

This section describes the details of the proposed synthetic AgriSeg segmentation dataset and the procedure we followed to validate the effectiveness of our DG methodology. In 10.2.1, we review the procedure followed to generate the AgriSeg dataset, while in section 10.2.2, details on the training framework and implementation are given.

10.2.1 Dataset

To generate the synthetic crop dataset with realistic plant textures and measurements, high-quality 3D plant models have been created using Blender[3]. A wide variety of crops have been included in the dataset to validate the segmentation performance of the model trained with the proposed DG method. Depending on the plant's height, three primary macro-categories of crops have been identified. Low crops, such as *Lettuce* and *Chard*, have an average height of 20-25 cm. Medium crops, such as *Zucchini*, grow to 60 cm. Tall crops, which include vineyards and trees, can grow up

[3]https://www.blender.org/

Fig. 10.2 Detailed example of synthetic 3D crop models realized to build the AgriSeg Dataset. A generic tree (top) and lettuce (bottom) are on the left. On the right are zucchini (top) and vines (bottom).

to 2.5-4.5 m. A meaningful target performance to be achieved by the segmentation model is set to generalize to previously unseen plants inside the same macro-category, which differ mostly in the color features and slightly in the geometrical shape. Some examples of 3D plant models are shown in Fig. 10.2.

Various terrains and sky models have been used to achieve realistic background and light conditions to achieve realistic background and light conditions. The generalization properties of the segmentation network are enhanced considering the light of different moments of the day and various weather conditions. Afterward, Blender's Python scripting functionality was used to automatically separate plants from the rest of the frame and generate a dataset of RGB images and their corresponding binary segmentation mask. This work presents the AgriSeg dataset, composed of RGB images and the associate segmentation mask samples of low crops, such as chards and lettuce, medium crops like zucchini, generic vineyard, pergola vineyards, pear trees, and generic tall trees. Nonetheless, each dataset presents four sub-datasets that differ in the background and the terrain. Cloudy and sunny skies, diverse lighting, and shadow conditions are considered. Camera position and orientation have been changed to acquire diversified image samples along the whole field for each

Crop	Samples	Type	Category↓	Height [m]
Lettuce	4800	Synthetic	Low	0.22
Chard	4800	Synthetic	Low	0.25
Lavender	4800	Synthetic	Low	0.3
Zucchini	19200	Synthetic	Medium	0.6
Vineyard	4800	Synthetic	Tall	2.5
Pergola Vineyard	4800	Synthetic	Tall	3.2
Pear Tree	4800	Synthetic	Tall	2.7
Generic Tree 1	4800	Synthetic	Tall	4.5
Generic Tree 2	2785	Synthetic	Tall	4.5
Vineyard [199]	500	Real	Tall	2.5
Miscellaneous	100	Real	Any	Any

Table 10.1 Detailed properties for each domain of the AgriSeg dataset. The section on the top reports the synthetic crops datasets generated in simulation, while the section on the bottom the real-world ones.

subdataset. The details of each subdataset are listed in Table 10.1. In the last rows, we also include two real domains to validate the considered methodologies on real data. The *Real Vineyard* dataset was originally presented in [199], but the proposed labels were coarse. Hence, we re-label the samples using the *SALT* labeling tool [4] based on Segment Anything [286]. We include another real domain, Miscellaneous, containing 100 samples from disparate crop types, and label it using *SALT*. This domain aims to benchmark the segmentation performance on any crop and is used as a final test in our experiments. Overall, the AgriSeg dataset contains more than $50,000$ samples.

10.2.2 Training

In this section, we report all the relevant information regarding the experimental setting of model training and testing: data preprocessing, hyperparameter search, and implementation. We repeat each training five times with different and randomly generated seeds to give more statistical information about accuracy results. In this way, both hyperparameter search and benchmarks cannot take advantage of the repeatability of trials, as data splitting, augmentation, and weight initialization

[4] https://github.com/anuragxel/salt

change from one iteration to the next. Each of the results of our benchmark is reported as mean and standard deviation.

Data Preprocessing

We preprocess input images through the ImageNet standard normalization [36] to use pretrained weights. We apply the same data augmentation to all the experiments, following common practice in DG for semantic segmentation. It consists of random cropping with a factor in the range $[0.5, 1]$, flipping with a probability of 50%, greyscale with a probability of 10%, random brightness, and contrast with a maximum relative change of 0.4. Experiments confirm that this configuration leads to enhanced generalization on the proposed dataset.

Hyperparameters

We conduct a random search to determine the optimal training hyperparameters for the ERM DG baseline. We define a range of values for continuous arguments and a set of choices for discrete ones and select the best combination via the *training-domain validation set* strategy proposed in [254]. It consists of picking the model that maximizes the metric (in our case, Intersection-over-Union with a threshold of 0.9) on a validation split of the training set (in our case, 10%, uniform across domains) at the end of each epoch. This selection method assumes that the average distribution of source domains is similar to that of the target domain on which the best model is tested.

We choose a batch size $B = 64$ and set the number of training epochs to 50. Since our dataset tackles binary crop segmentation, the adopted task loss is binary cross-entropy, while for the distillation loss, we choose temperature $\tau = 1$ and weight $\lambda = 3$. Following the procedure proposed in [290], we combine knowledge distillation with feature whitening and apply UniStyle to the first layers of the backbone (results are reported in 10.3). We use AdamW [294] as the optimizer with a weight decay of 10^{-5}. The learning rate is scheduled with a polynomial decay between 10^{-3} and 10^{-5}. As regards the compared methodologies, we apply IBN [287] and ISW[284] to the first three blocks of the backbone, while pAdaIN [288] is applied to all the layers with a probability of 10^{-3}. The ISW loss is weighted by a factor of 10^{-2}, while XDED [290] is applied with a weight of 10^{-3}, a temperature of 2, and in combination with UniStyle feature whitening.

Implementation

Our experimentation code is developed in Python 3 using TensorFlow as the DL framework. We train models starting from ImageNet pretrained weights, so the input size is fixed to (224×224). The considered DG methodologies are taken from the available repositories where possible or reimplemented. All the training runs are performed on a single Nvidia RTX 3090 graphic card.

10.3 Results

In this section, we present the main results of the experimentation conducted to evaluate the effectiveness of the proposed methodology. First, we compare our distillation-based approach with recent and promising DG and semantic segmentation alternatives. Inspired by popular datasets for image classification, we select four domains (*Generic Tree 2*, *Chard*, *Lettuce*, and *Vineyard*) and evaluate all the methodologies by training on three domains and testing on the fourth. The domains are selected to cover different crop dimensions and visual characteristics and guarantee a challenging generalization benchmark. Then, we perform an additional evaluation by training the model on all four datasets and testing on four additional target domains (*Pear Tree*, *Zucchini*, *Real Vineyard*, and *Real Miscellaneous*). We also report the predicted masks for a qualitative comparison on some random samples. In addition, we conduct a small ablation study to investigate the effect of UniStyle feature whitening, the difference between channel-wise and spatial-wise softmax in the computation of the distillation loss, and the importance of specialized single-domain teachers.

10.3.1 DG Benchmark

We run the leave-one-out DG benchmark described in 10.2.2 and report the results with their mean and standard deviation in Table 10.2. On average, our ensemble distillation methodology is 3% better than the second-best compared solution (ISW). Moreover, it achieves the best or second-best results on each target domain, confirming that distilling from a set of specialized teachers gives insightful information to the student and makes it less biased towards domain-specific features. The results for ERM are quite balanced across domains, proving the strong validity of this method despite its simplicity. ISW achieves positive results, generalizing well on almost all crops but failing in the *Lettuce* domain. This failure could be due to the color of

Method	Generic Tree 2	Chard	Lettuce	Vineyard	Average
ERM[255]	38.38 ± 12.10	83.22 ± 5.50	33.45 ± 13.34	46.69 ± 9.69	50.44 ± 10.15
IBN[287]	26.92 ± 12.61	83.52 ± 1.97	33.14 ± 22.82	47.72 ± 2.96	47.83 ± 10.09
ISW[284]	$\mathbf{65.72 \pm 8.47}$	86.05 ± 3.87	25.72 ± 12.89	51.34 ± 2.36	57.21 ± 6.00
pAdaIN[288]	42.27 ± 12.80	79.93 ± 1.65	13.22 ± 8.30	45.73 ± 4.81	45.29 ± 6.89
XDED[290]	38.79 ± 17.26	84.35 ± 5.11	29.99 ± 14.80	47.63 ± 6.27	50.19 ± 10.86
WildNet[285]	45.76 ± 2.17	82.45 ± 0.78	22.20 ± 0.73	$\mathbf{59.78 \pm 0.48}$	52.55 ± 1.04
Ours	50.02 ± 06.80	$\mathbf{86.17 \pm 1.79}$	$\mathbf{58.01 \pm 12.74}$	53.26 ± 3.59	$\mathbf{61.86 \pm 6.23}$

Table 10.2 Comparison between the proposed methodology and other state-of-the-art DG algorithms for semantic segmentation adopting the leave-one-out DG validation procedure described in 10.2.2. We report the Intersection-over-Union (IoU) metric (in %) for each result as mean and standard deviation. Each column's best and second-best results are highlighted and underlined, respectively.

Method	Pear Tree	Zucchini	Real Vineyard	Real Misc.	Average
ERM[255]	78.37 ± 2.51	86.51 ± 1.71	42.76 ± 11.38	64.40 ± 3.10	68.01 ± 4.68
IBN[287]	73.80 ± 4.21	86.21 ± 3.23	42.23 ± 11.32	63.36 ± 9.47	66.40 ± 7.13
ISW[284]	73.49 ± 1.81	$\mathbf{87.47 \pm 0.77}$	33.80 ± 23.85	48.36 ± 7.30	60.78 ± 8.43
pAdaIN[288]	74.53 ± 2.53	81.83 ± 4.82	41.16 ± 10.23	60.32 ± 9.09	64.46 ± 6.67
XDED[290]	76.82 ± 3.02	86.34 ± 1.07	46.38 ± 10.07	57.24 ± 8.89	66.69 ± 5.76
WildNet[285]	75.31 ± 3.50	81.88 ± 2.37	31.11 ± 1.35	46.57 ± 3.09	58.72 ± 2.58
Ours	$\mathbf{80.18 \pm 2.65}$	86.25 ± 1.42	$\mathbf{52.01 \pm 4.68}$	$\mathbf{66.69 \pm 3.18}$	$\mathbf{71.28 \pm 2.98}$

Table 10.3 Comparison between the proposed methodology and other state-of-the-art DG algorithms on additional target domains. We train the models on all four domains chosen for the previous benchmark. We report the Intersection-over-Union (IoU) metric (in %) on the unseen domains as mean and standard deviation. The best and second-best results are highlighted and underlined, respectively.

lettuce since its leaves get easily confounded with the color of grass in other domains (e.g. *Vineyard*). Our method, instead, retains good performance thanks to the insights given by the ensembled teachers and is not biased by spurious color correlations. However, the variance in results is considerable for the most challenging domains for almost all the DG methodologies tested. WildNet, instead, presents quite stable average performances over the runs but reports suboptimal results. This finding suggests that DG training offers a complex challenge, and our KD methodology could be further studied and improved to provide more robust results. We will address this aspect in future works.

To further validate the generalization capability of our method, we construct a more challenging benchmark by using four unseen test domains (*Pear Tree*, *Zucchini*,

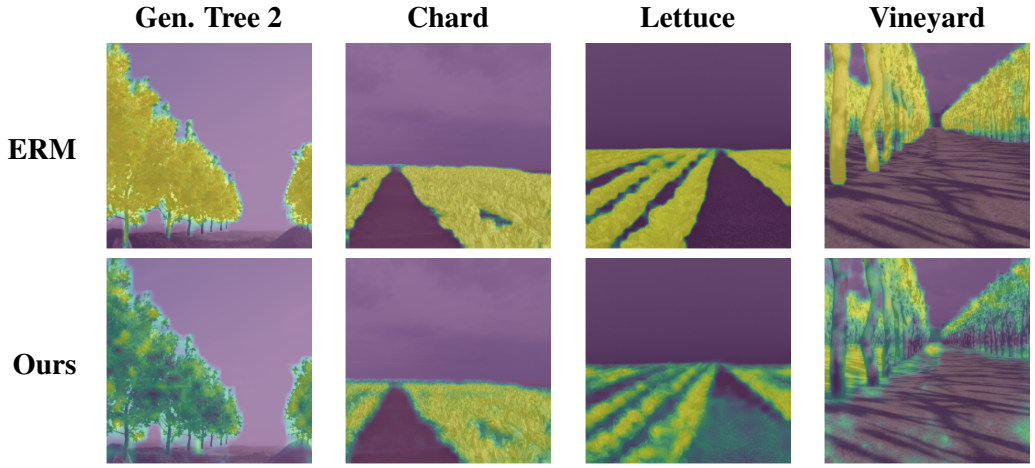


Fig. 10.3 Comparison of ERM predictions with our ensemble of specialized teachers. While for simpler domains, the predictions of the specialized teachers agree and return a high-confidence mask, for challenging ones, the teachers give an uncertain but more informative mask.

Real Vineyard, and *Real Miscellaneous*). The models are trained and validated on all four datasets used for the previous benchmark. In this way, each model has been trained on at least a domain similar in shape and size to a target domain, informing the models about the principal geometric features of different plant types. Here the domain gap resides in visual differences and correlations between plants and backgrounds. The results are reported in Table 10.3. Our method overcomes all state-of-the-art alternatives, as in the leave-one-out benchmark. The proposed solution retains the best performance on almost all the domains, except for *Zucchini*, where the difference is really small. This result enforces previous considerations on the generalization ability of knowledge distillation without any additional layers or computations at inference time. As expected, thanks to the *Generic Tree 2* source domain, all the models perform well on the *Pear Tree* domain, despite its significant difference in shape from the other crops. An interesting aspect is that ERM obtains good results on all the crops being the second-best generalizing model in this benchmark. However, its performance on the *Real Vineyard* domain is very low. While this is partially due to the dataset being very challenging, it also suggests a deeper investigation of the Sim2Real gap that will be addressed in future works. Indeed, the passage from synthetic to real crops further widens the existing domain gap between different crops and backgrounds. Another interesting insight can be found in the standard deviations, as our method obtains one of the smallest values.

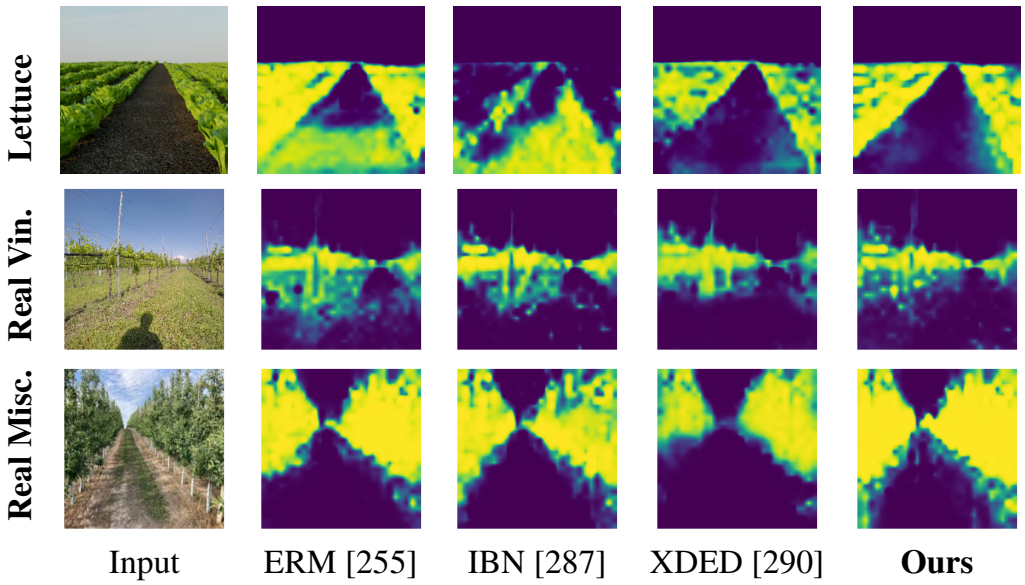


Fig. 10.4 Qualitative comparison between our distillation-based DG methodology and the most promising competitors according to our benchmark. We inspect output masks on *Lettuce*, *Real Vineyard*, and *Real Miscellaneous* domains for random samples.

This result is particularly evident for real domains, where other methods report high variance. WildNet performs very badly on real samples while obtaining satisfactory results on synthetic ones. Its small standard deviation suggests that the multiple training losses applied during training could have an over-regularizing effect on the process. On the contrary, our approach finds the best trade-off between regularization and learning.

10.3.2 Ablation Study

We conduct an ablation study to investigate the effect of different components on the generalization capability of our methodology. We also highlight the main differences between our approach and XDED [290] regarding methodological components and performance. In particular, we consider the UniStyle low-level feature whitening approach used by [290] in combination with ensemble distillation. We also analyze the effectiveness of our choice to apply the output softmax operator along the spatial dimension instead of channels following the findings of [293]. Finally, we substitute the specialized teachers with an ensemble of ERM models trained on all the source domains. The results are reported in Table 10.4, in which we included the ERM baseline as a reference. On average, the results suggest that applying only

Method	KD	UniStyle	Softmax	Gen. Tree 2	Chard	Lettuce	Vineyard	Average
ERM [255]	✗	✗	✗	38.38 ± 12.10	83.22 ± 5.50	33.45 ± 13.34	46.69 ± 9.69	50.44 ± 10.15
XDED[290]	✓	[0,1,2]	Channel	38.79 ± 17.26	84.35 ± 5.11	29.99 ± 14.80	47.63 ± 6.27	50.19 ± 10.86
Ours	✓	✗	Channel	34.61 ± 11.84	87.48 ± 2.07	21.76 ± 3.99	50.26 ± 2.77	48.53 ± 5.17
	ERM	✗	Space	43.99 ± 14.91	85.32 ± 3.47	39.11 ± 14.15	42.42 ± 10.55	52.71 ± 10.77
	✓	[0,1,2]	Space	56.32 ± 18.98	81.81 ± 3.67	43.44 ± 5.12	62.49 ± 4.50	61.01 ± 8.07
	✓	[0,1]	Space	54.78 ± 22.11	87.48 ± 2.96	47.40 ± 15.19	56.35 ± 8.72	$\underline{61.50 \pm 12.25}$
	✓	✗	Space	50.02 ± 06.80	86.17 ± 1.79	58.01 ± 12.74	53.26 ± 3.59	$\mathbf{61.86 \pm 6.23}$

Table 10.4 Ablation study highlighting the differences between our approach and [290]. We evaluate the effect of UniStyle (the numbers represent the blocks on which whitening is applied), channel-wise softmax, and ensembling methods on the proposed methodology. We report the Intersection-over-Union (IoU) metric (in %) for each result as mean and standard deviation. The best and second-best results are highlighted and underlined, respectively.

distillation is slightly better than combining it with feature whitening, especially considering that this implies additional computation at inference time. The variance of the results is also larger when feature whitening is applied, suggesting that this regularization can lead to better optimization in some cases but suboptimal solutions in others, depending on weight initialization. However, the gap is limited, and more importantly, results on single domains are not unique, as each variant seems to be more suited for some domains than others. Nonetheless, our methodology outperforms ERM and XDED by more than 10%, on average. Our study confirms that applying softmax along the spatial dimension leads to better knowledge distillation for segmentation tasks. Indeed, despite performing well in *Chard* and *Vineyard* domains, the variant with channel-wise softmax retains unsatisfactory results on *Generic Tree 2* and *Lettuce* crops. Finally, we confirm the intuition that ensembling models specialized in single domains brings more information than distilling from models trained on all the source domains together. As depicted in Fig. 10.3, the distillation masks are less confident, giving the student a better understanding of what parts of the image are more likely to confound the predictor. However, the version distilled from ERM teachers outperforms plain ERM by 2%. We further inspect the effect of the method's hyperparameters on generalization capabilities. We vary the distillation loss weight λ and the temperature T and report the results on the *Real Miscellaneous* domain in Fig. 10.5. The graphs show that our choice ($\lambda = 10^{-2}, T = 2$) is the optimal balance that ensures regularization without constraining the student. As reported in our benchmarks, this yields good generalization across various synthetic and real domains.

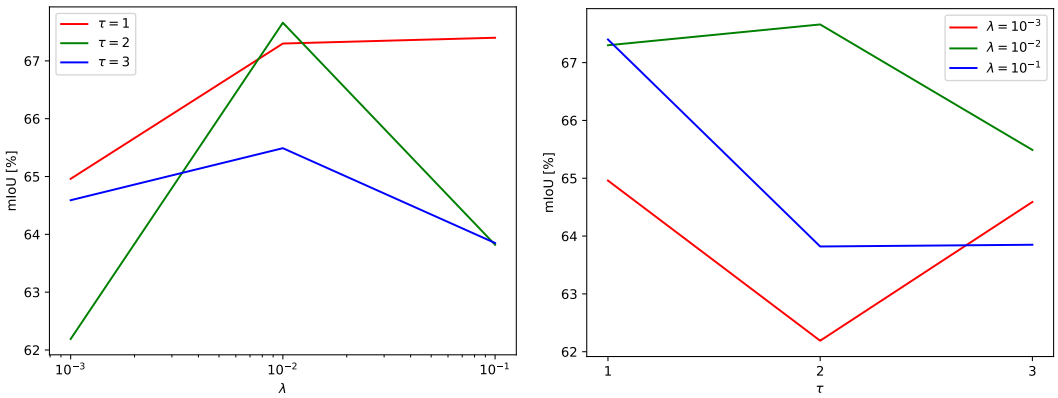


Fig. 10.5 Ablation study on the hyperparameters λ and τ. The reported IoU value is relative to the *Real Miscellaneous* domain and is averaged on three runs. We represent two views of the results for better readability.

Finally, in Fig. 10.4, we report a qualitative comparison between output masks from our method and the most promising competitors (ERM, IBN, and XDED, according to our benchmark). We inspect output masks on *Lettuce*, *Real Vineyard*, and *Real Miscellaneous* domains for random samples. Although IoU is computed with a confidence threshold of 90 %, we choose to plot the original masks to highlight unconfident predictions. The difference is most evident for the *Lettuce* domain, in which other algorithms erroneously segment the terrain (ERM, IBN) or retain low confidence on crops (XDED). The same happens for the *Real Vineyard* domain, where the predictions are generally less confident, and XDED performs similarly to our solution. On the *Real Miscellaneous* domain, XDED performs slightly worse than our solution, as the segmentation mask does not include trunks. In this scenario, IBN is more accurate and similar to our method, confirming the results of Table 10.3. In conclusion, our solution outputs satisfactory masks for all domains, performing on par or better than all other methods.

Chapter 11

Domain-Adversarial Vision Transformer for Land Crop Classification with Multi-Temporal Satellite Imagery

In this Chapter, the investigation of Deep Learning models generalization defined and discussed in Chapters 9 and 10 is declined into another application in the context of precision agriculture and remote sensing: land cover and crop classification (LC(&)CC). In the past few decades, the launch of many satellite missions has offered an extensive repository of remote sensing images. Availability of the open-source data by many Earth-observation satellites has made remote sensing easy and obtainable [295]. Open-source data sets are available free of cost from several satellite missions such as the Sentinel-2 and Landsat [296]. These satellites are equipped with multi-spectral sensors with short revisit time, and good spatial and spectral resolution, allowing researchers to test modern image analysis techniques to extract more detailed information of the target object. Overall, the new scenario has led to the opportunity for the land cover monitoring, change detection, image mosaicking, and large-scale processing using multi-temporal and multi-source images [295, 297–299]. The most essential and critical remote sensing application is land cover and crops classification (LC&CC). It facilitates labeling the cover such as forest, ocean, and agricultural land. Moreover, mapping can also be done manually

using satellite images, but the process is quite tedious, costly, and time-consuming. Finally, an exquisite global cover map is not available as yet, but there is a land cover map named Corine Land Cover (CLC) [300] which provides land cover information with 100m per pixel resolution. However, the problem with this map is that it only covers the European area and is updated once in six years.

There are several ways to perform land classification automatically. In general, the classification involves the creation of a training dataset that consists of annotated samples of the corresponding class labels, training a model using the training dataset, and evaluating the resulting predictions. The number and quality of training samples play a pivotal role in defining the performance of the trained model. From a remote sensing prospective, training sample collection requires a ground survey or visual photo-interpretation by an expert [301]. Ground surveying involves GIS expert knowledge, human resource that is not typically economical, while visual interpretation is not appropriate to be used for some applications, such as finding chlorophyll concentration [302] and classification of tree species [303]. Most of the machine learning (ML) algorithms such as random forest, support vector machines, logistic regression performs well in the context of classification of remote sensing images. However, performance of these ML algorithms are not satisfied when learning features from different sources such as active and passive sensors [304]. It was shown in [305, 306] that Convolutional Neural Networks (CNN) are better than traditional land cover classification techniques. In the land segmentation section of the deep globe challenge [307], the Deep Neural networks (DNNs) completely dominate the leaderboards. The best examples of land cover classification using DNNs are ResNet and DenseNet [308, 309]. For example, in [310], a 2D-CNN is used to obtain the spatial features of the hyperspectral imagery (HSI), analyzing the continuity of land covers in the spatial domain. Often relation among spectral bands of HSI is not linear, 2D-CNNs are normally used together with 1D-CNNs to incorporate the spectral and spatial domain of features [311]. The classification task becomes quite challenging when dealing with high-dimensional hyperspectral data with few labeled samples. Recently, generative adversarial networks (GANs) [312] have been exploited for sample generation, though it is not easy to acquire high-quality samples with authenticity [313].

The generalization problem analyzed in this study comes from the fact that there is a difference in the land covers of different locations, hence, the model trained

in one area does not perform well in the other areas. Additionally, the satellite imagery of different satellites present changes in their resolution, capture time, and other radiometric parameters. Due to these multiple changing variables, the dataset taken from a satellite covering one region and another satellite dataset covering the same or other regions leads to a domain shift between the datasets. One way to achieve a reliable outcome is possibly to train a model with a huge amount of training samples to generalize its behavior for all classes of all the regions. However, that needs an enormous labeled dataset that is time and labor-intensive. Another method to deal with the shift between the datasets is termed Domain Adaptation (DA), in which a model is trained on one dataset (source data) and predictions are made on the other dataset (target domain). DA is the restricted formulation to only two domains (source and target) of the wider Domain Generalization problem defined in 9. The distribution shift between the target and source dataset is mainly due to temporal differences in the acquisition, in the sensors, and in geographical factors such as variations of elements in the Earth's surface. The domain shift affects the performance of a model trained on a source dataset and applied on the target dataset. DA methods often rely on learning domain-invariant models that keep comparable performances on the two datasets. Existing DA techniques may be classified as supervised, unsupervised, and semisupervised [314–316]. In supervised DA methods, it is presumed that labeled data are available for both source and target domains [317]. In a semisupervised domain, the labeled data for the target domain is assumed to be small while an unsupervised method contains labeled data for the source domain only. For example in [318], a semisupervised visual DA was proposed to address classification of very high-resolution remote sensing images. To deal with the variation in features distribution between the source and target domains, multiple kernel learning DA method was employed.

Tuia et al. [301] divide the DA methodologies into four different categories: domain-invariant feature selection, adapting data distribution, adapting classifiers, and adaptive classifiers using active learning methods. In the context of classification and segmentation of remotely sensed images, in [319] an unsupervised adversarial DA method was proposed based on a boosted domain confusion network (ADA-BDC) which focuses on feature extraction to enhance the transferability of classifier which is trained by source domain images and tested on target domain images. In [320], an unsupervised DA was used using generative adversarial networks (GANs) for semantic segmentation of aerial images. A multi-source domain adaptation (MDA)

for scene classification was proposed to transfer knowledge from the multiple-source domains to the target domain in [321]. Most of the studies presented in the literature related to DA-based classification have used single date images of source and target domain. However, in [322], first approach was proposed in the context of DA for classification of multi-temporal satellite images in which Bayesian classifier-based DA was employed with only two images of Landsat-5 satellite.

This study investigates adversarial training of DNNs to bridge the domain discrepancy between distinct geographical zones. More in detail, we analyze the application of DA to challenging multi-spectral, multi-temporal data, highlighting the advantages of adopting the most recent self-attention-based models for LC(&)CC to different target zones where labeled data are not available. We choose to experiment our methodology on the BreizhCrops dataset, a large-scale time series benchmark dataset introduced in 2020 by Rußwurm et al., [323], for supervised classification of field crops from satellite data. Fig. 11.1 shows the visual representation of the crop prediction performed on a sub-region of Brittany, highlighting the benefit provided by the proposed methodology.

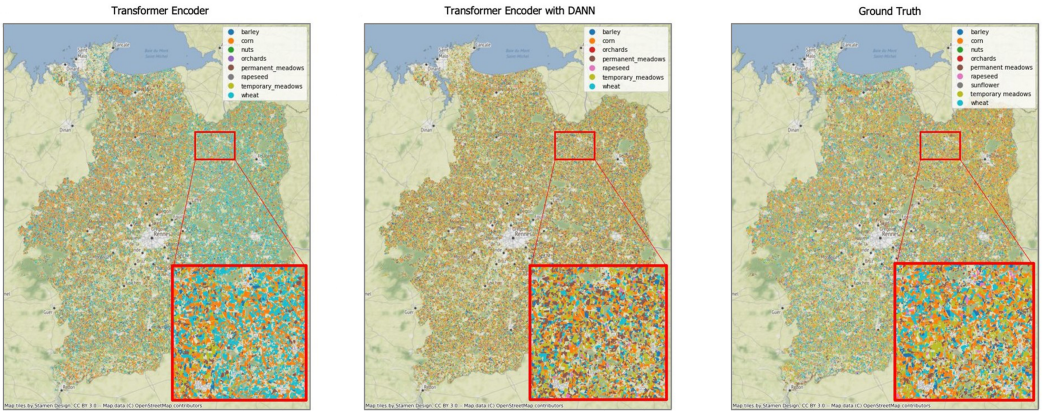


Fig. 11.1 Visual representation of land crops classification on zone 3 (Ille-et-Vilain) of the BreizhCrops dataset. For each sub-image we show the complete region and a sub-area to facilitate the visualization of the advantage obtained by the proposed methodology. In particular, on the left the crops predictions without our domain adaptation mechanism are shown, while in the center the same predictions performed adopting DANN are proposed. On the right, ground truth labeled crops can be visualized. The improvement in the classification with DANN is evident, especially in the reduction of misclassification of wheat and meadows.

11.1 Study Area and Data

To promote reproducibility of our experimentation, we rely on BreizhCrops, a large-scale time series benchmark dataset introduced in 2020 by Rußwurm et al., [323], for supervised classification of field crops from satellite data. The dataset comprises multivariate time series examples in the Region of Brittany, France, of the season 2017, from January 1 to December 31. In particular, the authors of the dataset exploited all available Sentinel 2 images from Google Earth Engine, [324], and farmer surveys collected by France National Institute of Forest and Geography Information (IGN) to collect more than 600 k samples divided into 9 classes with 45 temporal steps and 13 spectral bands. Most importantly, as shown in Fig. 11.2, acquired data are equally split into distinct regional areas. Indeed, as regulated by the Nomenclature des unites territoriales statistiques (NUTS), the overall dataset is divided into the four NUTS-3 regions Côtes-d'Armor, Finistère, Ille-et-Vilaine, and Morbihan. That, in conjunction with the challenging nature of the dataset, makes BreizhCrops an ideal benchmark to test domain adaptation for multi-spectral and multi-temporal data for LC&CC.

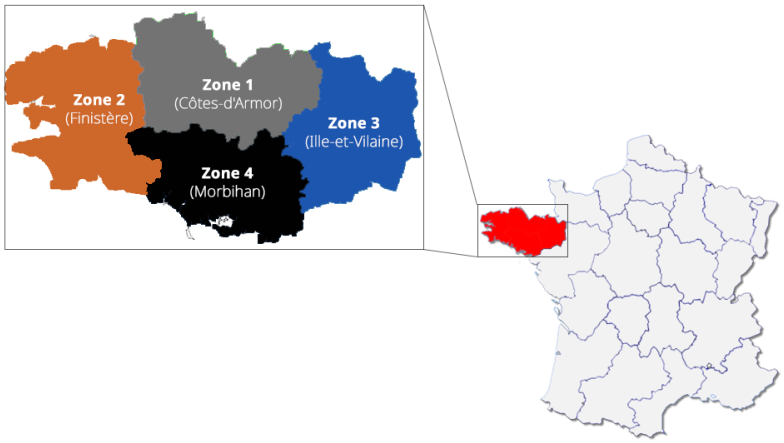


Fig. 11.2 Magnified view of the four NUTS-3 regions of Brittany, located in the northwest of France and covering 27,200 km². The strict division of the supervised BreizhCrops dataset in the four regions allows the performance of a formal and controlled analysis on domain adaptation for LC&CC with multi-spectral and multi-temporal data.

As summarized in Fig. 11.3, even if the authors of the dataset avoided broad categories, due to the nature of agricultural production, which focuses on a few dominant crop types, a class imbalance can be observed in the collected parcels. That con-

stitutes a challenge for every classifier type, but it reflects the strong imbalance in real-world crop-type-mapping datasets. On the other hand, sample classes in the different regions are balanced, making BreizhCrops a perfect bench for testing domain adaptation strategies. Finally, to disentangle the performed domain adaptation analysis from the influence of the random variation of the atmospheric conditions, we exclusively make use of L2A bottom-of-atmosphere imagery where data acquired over time and space share the same reflectance scale. Adjacent and slope effects are corrected by the MAJA processing chain [325] that employs 60-meter spectral bands to apply atmospheric rectification and detect clouds. Therefore, only ten spectral features are available for each parcel. Table 11.1 is presented as a summary of the number of samples collected for the domain adaptation experimentation divided into classes and regions. In conclusion, multi-spectral, multi-temporal pixels are individually extracted for each parcel and are constituted by 10 spectral bands and 45 temporal steps each. The class imbalanced highlighted by the number of parcels of Fig. 11.3 is reflected in the number of samples of Table 11.1 used for all experimentation.

Table 11.1 Summary of the number of samples per class divide in the four NUTS-3 regions of Brittany. Instances are derived by L2A bottom-of-atmosphere parcels to disentangle our analysis with variation of the atmospheric conditions.

	Barley	**Wheat**	**Rapeseed**	**Corn**	**Sunflower**	**Orchards**	**Nuts**	**Permanent Meadows**	**Temporary Meadows**
Zone 1	13,051	30,380	5596	44,003	1	937	10	32,641	52,013
Zone 2	10,736	15,026	2349	36,620	6	348	18	36,536	39,143
Zone 3	7154	27,202	3557	42,011	10	1217	10	32,524	52,682
Zone 4	5981	17,009	3244	31,361	2	552	11	26,134	38,141

11.2 Methodology

In this work, unsupervised domain adaptation is considered in the field of land cover classification from satellite images. The study aims to tackle the problem of low generalization capability of classifiers only trained on a peculiar geographical region dataset. Moreover, the lack of rich available datasets of labeled satellite images increases the interest towards this challenge. In particular, the proposed methodology is intended to investigate the application of representation learning (RL) techniques for domain adaptation when dealing with multi-temporal data. For this purpose, a

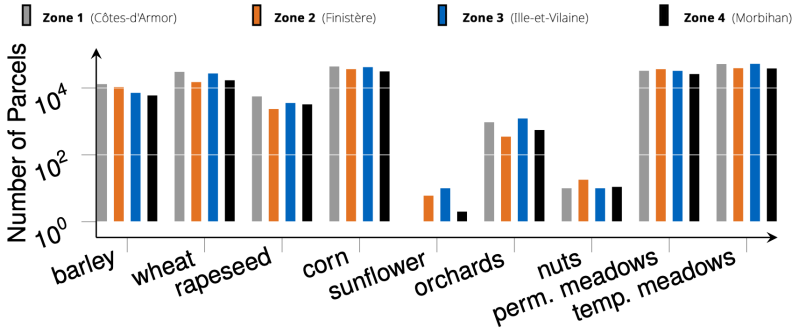


Fig. 11.3 Class frequencies divided in the four NUTS-3 regions of Brittany. The respective number of parcels highlights the strong class imbalance, reflecting the substantial imbalance in real-world crop-type-mapping datasets. However, samples per class in the four regions are equally divided.

Transformer Encoder-based classifier is adapted to a Domain-Adversarial Neural Networks (DANN) architecture and trained accordingly.

In this section, a thorough description of the methodology is provided. First, we frame domain adaptation with the DANN method. Then, we briefly explain the Transformer Encoder structure with self-attention adopted for the multi-temporal crops classification. Finally, we describe the resulting architecture of the attention-based DANN, which is used to train a classifier with improved domain generalization.

11.2.1 Domain-Adversarial Neural Networks

Classifiers obtained with Deep Neural Networks often suffer from a lack of generalization related to possible variations in the appearance of the same objects. This problem is usually identified as a domain gap. In the land cover classification task, this situation is very recurrent and can be associated with the spectral shift affecting the data collected in different regions at different times. The shift is often related to photogrammetric distortion or visual differences in the appearance of lands. Furthermore, when dealing with satellite images, a dataset usually needs to be created by labeling images for a specific region to train a classification model. Despite this time-expensive procedure, standard training does not guarantee satisfying performance on images of different regions.

Domain-Adversarial Neural Networks (DANN) is a representation learning technique that allows a classifier to generalize better from a *source domain* to a *target domain*. This specific domain adaptation method consists of adding a branch to the original

feed-forward architecture of the classifier and carry out an adversarial training. From a generic perspective, it is possible to identify three main components of the DANN: a *feature extractor* with parameters θ_f, a *label predictor* with parameters θ_y, and a *domain classifier* with parameters θ_d. The feature extractor is the first block of the DANN model. It is responsible for learning the function $G_f : X \rightarrow \mathbb{R}^d$, which maps the input samples X to a d-dimensional vector containing the extracted features. The label predictor function, $G_y(G_f(X))$, compute the label associated with the predicted class of the sample. The domain discriminator function $G_d(G_f(X))$ distinguishes between source and target domains given the extracted features. The combination of feature extractor and label predictor gives us the complete classifier model. The domain classifier is composed of a secondary branch, similar to the label predictor, which receives the extracted feature vector by the first block of the network.

Given these three main elements, the expression of the total loss used to train DANN is obtained by the following expression, according to the authors [238]:

$$\mathcal{L}\left(\theta_f, \theta_y, \theta_d\right) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_y^i\left(\theta_f, \theta_y\right) - \lambda\left(\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_d^i\left(\theta_f, \theta_d\right) + \frac{1}{n'}\sum_{i=n+1}^{N}\mathcal{L}_d^i\left(\theta_f, \theta_d\right)\right) \tag{11.1}$$

The first term $\mathcal{L}_y$ is the label predictor loss, while the second one involves the domain discriminator loss $\mathcal{L}_d$. The hyper-parameter λ can be tuned to weigh the contribution of the two learning terms. A more detailed analysis of the choice of λ is proposed in the experiments section. n and n' are respectively the numbers of samples from the source and the target domains. Totally, we have $N = n + n'$ samples used in the training. The expression of the total loss function also describes the principal goals of DANN: first, we want to obtain a label predictor with low classification risk. Second, we are adding a regularization term for the domain adaptation. To this extent, we aim to find a set of parameters of the feature extractor θ_f that can map a generic input sample from either source or target domain to a new latent space of features, where the domain gap is reduced. On the other hand, the classification performance has not to be affected. For this reason, the extracted features should be discriminative as well as domain-invariant. According to this goal, the optimal choice of parameters θ_f and θ_y is represented by the one which minimizes the total loss function, keeping $\hat{\theta}_d$ unchanged. By contrast, the domain discriminator parameters θ_d are updated to

maximize the loss while not changing the other ones.

$$\left(\hat{\theta}_f, \hat{\theta}_y\right) = \underset{\theta_f, \theta_y}{\operatorname{argmin}} \mathcal{L}\left(\theta_f, \theta_y, \hat{\theta}_d\right) \tag{11.2}$$

$$\hat{\theta}_d = \underset{\theta_d}{\operatorname{argmax}} \mathcal{L}\left(\hat{\theta}_f, \hat{\theta}_y, \theta_d\right). \tag{11.3}$$

In the original paper of DANN, the parameters of each piece of the neural network model are updated with a classical Stochastic Gradient Descent (SGD) optimizer. Here instead we use Adam [50]. Parameters θ_f,θ_y and θ_d are updated according to its rules.

$$\theta_f \quad \longleftarrow \quad \theta_f - \eta \left(\frac{\hat{m}_{f,y}}{\sqrt{\hat{v}_{f,y}} + \varepsilon} - \lambda \frac{\hat{m}_{f,d}}{\sqrt{\hat{v}_{f,d}} + \varepsilon} \right) \tag{11.4}$$

$$\theta_y \quad \longleftarrow \quad \theta_y - \frac{\eta}{\sqrt{\hat{v}_y} + \varepsilon} \hat{m}_y \tag{11.5}$$

$$\theta_d \quad \longleftarrow \quad \theta_d - \frac{\eta}{\sqrt{\hat{v}_d} + \varepsilon} \hat{m}_d \tag{11.6}$$

As explained in Chapter 1, the first (mean) and the second (uncentered variance) moments of Adam $\hat{m}$ and $\hat{v}$ are estimated as exponentially moving averages computed with the gradients obtained from each mini-batch. For the specific case of DANN, gradients used to estimate the Adam moments change for each element G_f, G_y, G_d of DANN structure. For example, the feature extractor gradients $(\partial \mathcal{L}_y^i / \partial \theta_f)$ and $(\partial \mathcal{L}_d^i / \partial \theta_f)$ are used to compute $\hat{m}_{f,y}$ and $\hat{m}_{f,d}$. Diversely, gradients obtained from label predictor $(\partial \mathcal{L}_y^i / \partial \theta_y)$ and domain discriminator $(\partial \mathcal{L}_d^i / \partial \theta_d)$ are only used to update their respective momentum $\hat{m}_y$ and $\hat{m}_d$.

The feature extractor and the domain discriminator play adversarial roles during the training process. A satisfying feature extractor can fool the domain discriminator by forwarding a vector of domain-invariant features. The role of the domain discriminator is to improve and evaluate this ability. A key intuition in the DANN method is to carry out the adversarial training with a standard backpropagation of the gradients, thanks to a custom Gradient Reversal Layer between the feature extractor and the domain discriminator. This particular layer does not add other parameters to the model but changes the sign of the upstream gradients. The GRL operation can be formulated with $\mathcal{R}(\mathbf{x})$ in the following mathematical expressions for the forward and

backpropagation step:

$$\mathcal{R}(\mathbf{x}) = \mathbf{x} \tag{11.7}$$

$$\frac{d\mathcal{R}}{d\mathbf{x}} = -\mathbf{I} \tag{11.8}$$

where $\mathbf{I}$ is the identity matrix. Hence, by performing optimization steps on the resulting DANN architecture, we can update parameters to reach saddle points of the total loss function reported in (11.1).

11.2.2 Classification of Multi-Spectral Time Series with Self-Attention

Self-Attention, popularized by the Transformer model in 2017 [40], has provided a considerable boost in machine translation performance while being more parallelizable and requiring significantly less time to train. Nevertheless, the introspection capability behind the success of Transformers is not limited only to natural language processing, but can be adapted to any time series analysis to filter data and focus on more relevant repressions aspects. In Chapter 1, an theoretical introduction to Self-Attention and to Vision Transformer [67] model can be found. Here, a summary of the main operation performed is provided adapting to the case of satellite imagery.

A single sample pixel i-th of multi-spectral, multi-temporal acquisition can be represented as a matrix $\boldsymbol{X}^{(i)} \in \mathbb{R}^{t \times b}$ where t is the temporal dimension and b is given by the number of spectral bands. Therefore, it is a 1D sequence of tokens, $(\boldsymbol{x}_0, ..., \boldsymbol{x}_t)$, with $\boldsymbol{x}_t \in \mathbb{R}^b$, that can be easily linearly projected to feed a standard Transformer encoder. The encoder can map a temporal input sequence $\boldsymbol{X}_{t \times b}$ in a continuous representation $\boldsymbol{X}^L_{t \times d_{model}}$, where L is the output layer of the Transformer model and d_{model} is the constant latent dimension of the projection space. Self-attention, through local multi-head dot-product self-attention blocks, can easily manipulate the temporal sequence finding correlations between different time-steps and completely avoiding the use of recurrent layers. The dot-product self-attention operation is composed on a trainable associative memory with key and value vector pairs of dimensions d. For a sequence of t query vectors, arranged in a matrix $\mathbf{Q} \in \mathbb{R}^{t \times d}$, the self-attention operation is described by the following operation:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}(\mathbf{Q}\mathbf{K}^T / \sqrt{d})\mathbf{V} \tag{11.9}$$

where the Softmax function is applied over each row of the input matrix and $\mathbf{K} \in \mathbb{R}^{t \times d}$ and $\mathbf{V} \in \mathbb{R}^{t \times d}$ are the key and value vector matrices, respectively. Query, key and values matrices are themselves computed from a sequence of t input vectors with dimension d_{model} using linear transformations: $\mathbf{Q} = XW_{\mathbf{Q}}$, $\mathbf{K} = XW_{\mathbf{K}}$, $\mathbf{V} = XW_{\mathbf{V}}$ where $X \in \mathbb{R}^{t \times d_{model}}$. Finally, multi-head dot-product self-attention is defined by considering applying h self-attention functions to the input X. Each head provides a sequence of size $t \times d$. These h sequences are rearranged into a $t \times dh$ sequence that is linearly projected into $t \times d_{model}$.

Subsequently, after the transformer encoder, the output representation, $\boldsymbol{X}^{L}_{t \times d_{model}}$,can be exploited to perform a classification of the input sequence. Indeed, that can be achieved by further processing the output encoder matrix and feeding a classification head trained to map the hidden representation to one of the k classes. In conclusion, a Transformer encoder can be repurposed to process a multi-spectral input sequence and find valuable correlations between the different time-steps to perform LC&CC with a high level of accuracy.

11.2.3 DANN for Land Cover and Crop Classification

We employ DANN in conjunction with self-attention-based models to bridge the domain gap between different geographical regions. The overall architecture of the adopted methodology is shown in Fig. 11.4. First, an input sequence $\boldsymbol{X}_{t \times b}$ is linearly projected to the constant latent dimension of the Transformer model d_{model}. Moreover, a Transformer encoder does not contain recurrence or convolution to make use of the order of the sequence. Therefore, some positional encoding is injected about the relative or absolute position of the tokens in the sequence. The positional encodings have dimension d_{model} as the projected sequence, so that the two can be summed. Guided by experimentation, as in [67], we adopt a learnable positional encoding instead of the sine and cosine functions with different frequencies of [40]. The resulting pre-processed input sequence $\boldsymbol{X}^{l_0}_{t \times d_{model}}$ feeds the Transformer encoder, parameterized by Θ_f, that provides as output a continuous representation $\boldsymbol{X}^{L}_{t \times d_{model}}$. Subsequently, we make use of the max function, over the temporal axis, to extract a token, $\boldsymbol{x}^{L}_{d_{model}}$, from the output sequence.

The extracted representation constitutes the input for either the LC&CC and domain multi-layer perceptron classifiers. The first network provides a probability distribution over the k different classes, $\hat{\boldsymbol{y}}_k$. On the other hand, the domain classifier

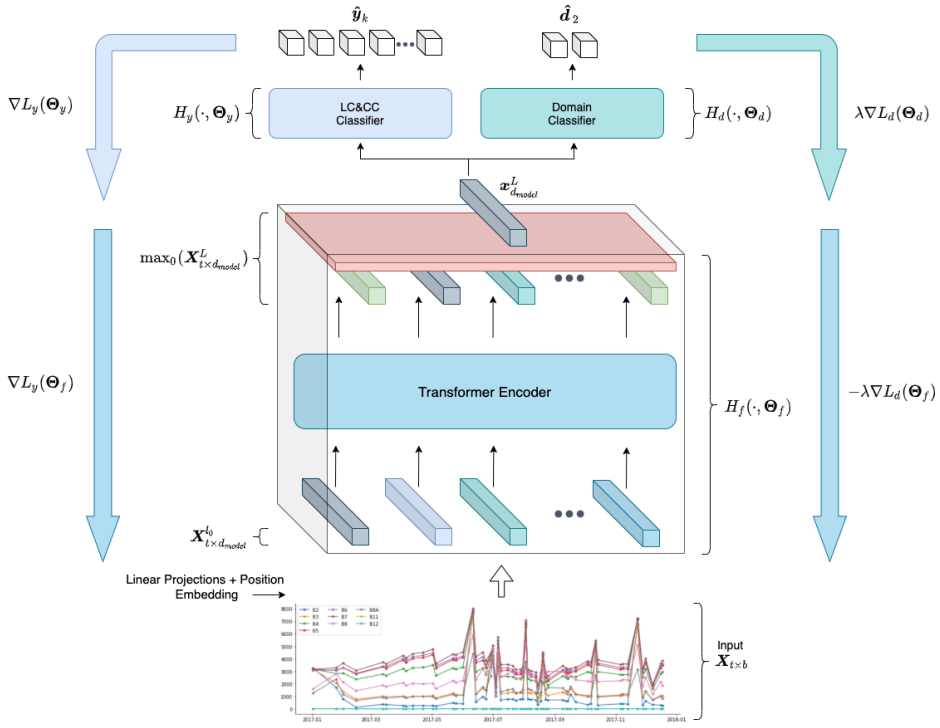


Fig. 11.4 Overview of the overall framework to train a Transformer encoder with domain-adversarial training. The multi-spectral temporal sequence $\boldsymbol{X}_{t\times b}$ is first linearly projected and fused with a position encoding. Subsequently, the self-attention-based model manipulates the input series and, through a max operation applied to the last layer of the encoder, is possible to extract a token $\boldsymbol{x}^L_{d_{model}}$ from the output sequence. Finally, gradients derived by LC&CC and Domain classifiers train the network while keeping close the distribution of source and target domains.

outputs the probability, $\hat{\boldsymbol{d}}_2$, that the extracted representation $\boldsymbol{x}^L_{d_{model}}$ belongs to the target or source domain. Using the cross-entropy loss function for both classifiers, it is possible to compute the respective gradients and update the weights, Θ_f of the feature extractor. Indeed, inverting the sign of the gradients, $\nabla L_d(\Theta_d)$, derived from the domain classifier, and multiplying them for a scale factor λ, we can increasingly reduce the distance between the latent space of the two domains while training the encoder on the classification task. Overall, the proposed training framework provides an effective solution to transfer the acquired knowledge of a model to a diverse region, exploiting only the original nature of the data.

11.3 Experiments and Discussion

We experiment with the proposed methodology on the four regions of the multi-temporal satellite BreizhCrops dataset presented in Section 11.1. As explained in the same section, we indicate this dataset as an optimal choice to train and test new domain adaptation methods exploiting labeled multi-temporal data. The first main objective of the conducted experimentation is to investigate how the classification performance of a state-of-the-art model for LC&CC model is affected by a lack of generalization towards different geographical regions. Then, we clearly highlight how adversarial training can mitigate the domain gap and significantly boost performance for source and target regions with marked distribution distance. It is important to remark that the method relies on the availability of samples of both source and target domains, whereas only source labels are required, not allowing direct applicability of transfer learning techniques. Finally, in the last part of the section, obtained results are discussed and inspected through dimensionality reduction techniques, validating the proposed method for practical use.

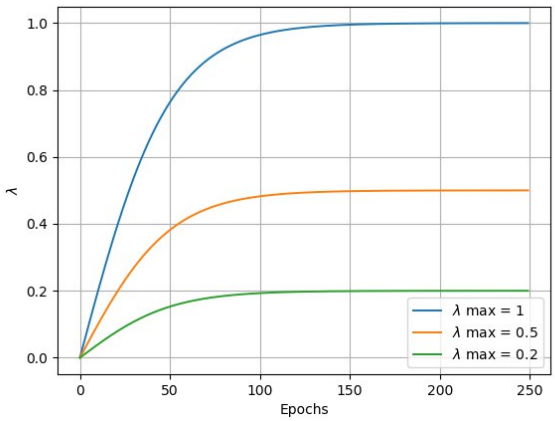


Fig. 11.5 λ scheduling: the value of the domain adaptation parameter λ is changed during training according to an exponentially growing trend. This allows the feature extractor to learn basic features during the initial epochs. Different final λ_{max} values are tested to study the right level of adaptation required in the different cases: 1, 0.5 and 0.2. $\lambda_{max} = 0.2$ is the best choice for an overall adaptation improvement of the classifier in the different regions. The parameter γ influences the slope of the curve and it is kept constant to 10 to let λ reach the desired value in a suitable number of epochs.

11.3.1 Experimental Settings

We carried out a complete set of experiments to compare the Transformer encoder classifier performance with and without DANN. The standard classifier is trained separately on each of the single regions of the dataset, then tested on the other ones. By contrast, DANN models are trained on each source-target pair to gain the desired adaptation capability and tested an all the regions except for the source domain. No validation set is used for model selection. Tests are always performed with the model resulting from a fixed number of training epochs.

In the final architecture, the classifier model comprises a transformer encoder feature extractor and a final classification stage. In all experimentation, the transformer encoder receives as input a batch of 256 tensors with $t = 45$ temporal steps and $b = 10$ spectral bands in the image samples. Moreover, to linearly project the temporal sequence to the constant latent dimension of the encoder, the input is first passed to a dense layer with 64 units. Therefore, d_{model} is equal to 128. On the other hand, the multi-head attention Transformer encoder is defined with several layers and attention heads equal to $n_{layers} = 3$ and $n_{heads} = 2$. Finally, the dimension of internal fully connected layers $d_{inner} = 128$. Rectified linear units is the non-linear activation function used for all neurons of the encoder.

The LC&CC classification stage is a simple multi-layer perceptron head composed of a normalization layer, a fully connected layer with 128 units, ReLU as activation function, and a final layer with $k = 9$ neurons. On the other hand, for the DANN experimentation, the domain predictor is identical to the multi-layer perceptron head of the LC&CC classifier, with 128 units and a ReLU activation. However, the number of neurons in the final layer is set to $d = 2$, since we always perform a single target domain adaptation.

A cross-entropy loss function is chosen to train both the classifiers. The parameters of both models are updated using Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 1 \times 10^{-7}$. A fixed number of epochs is always set to 250. The learning rate value is changed during training according to an exponential decay policy from a starting value of 0.001, with a decay scheduled for each epoch equal to 0.99^{epoch}. A key point in the experimental settings is related to the domain adaptation parameter λ. It acts as a regularization parameter, since it regulates the impact of the domain discriminator gradients on the feature extractor during training. Therefore, it can be considered to be the principal hyper-parameter to tune when using DANN. We

always use a scheduling policy for λ, as suggested in the original publication of DANN:

$$\lambda_t = \lambda_{max}\left(\frac{2}{1+e^{-\gamma t}} - 1\right) \tag{11.10}$$

where λ_{max} is the plateau value reached. This is the actual value of λ used for the second half of the training, which affects the final performance of the model in terms of generalization. The parameter $\gamma = 10$ defines the slope of the curve and it is fixed to such value to let λ_{max} be reached in a suitable number of epochs. A scheduled value of λ allows the feature extractor to learn the basic features for the classification during the first epochs. It then adjusts the mapping function to let the source and target domain feature distributions to overlap at the end of the training process. As shown in Fig. 11.5, different values of λ_{max} are tested to study the response of the model. To our knowledge, $\lambda_{max} = 0.2$ is the best value for a robust adaptation improvement of the classifier, at least among the set of tested λ_{max} values.

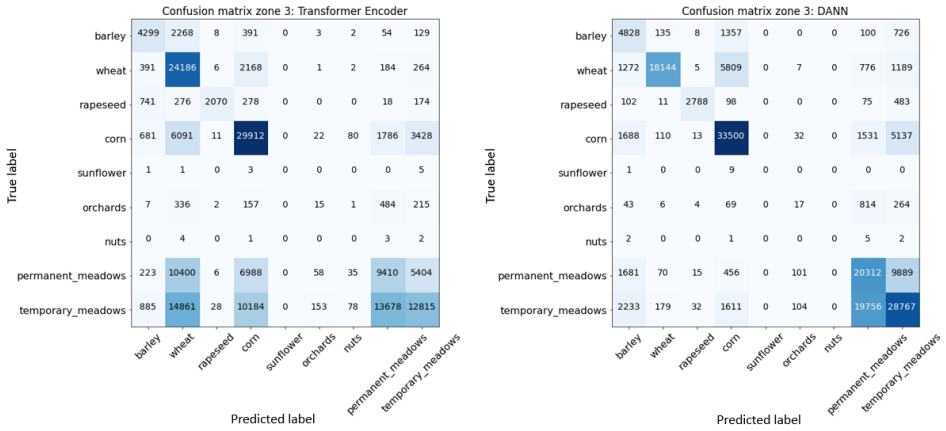


Fig. 11.6 Class-wise comparison of classification results on zone 3 (target), selecting zone 2 as source domain. Confusion matrix obtained with Transformer encoder trained on zone 2 and tested on zone 3 is shown in (**a**) on the left. Fig. (**b**) on the right shows classification results with DANN model tested on zone 3. The effect of DANN clearly mitigate the prediction error, with a particular focus on relevant classes such as Corn, Permanent and Temporary Meadows.

As already explained at the beginning of the section, the classifiers are trained and tested on all the possible combinations of regions to quantify the existing domain gap. The classification performance is evaluated using three different classification metrics, which are chosen among the ones proposed in the BreizhCrops dataset benchmarks: Accuracy, F1-score and K-score. This last metric is the Cohen's kappa [326], computed according $\kappa = (p_o - p_e)/(1 - p_e)$ where p_o and p_e are the

Table 11.2 Results of crops classification for the Transformer Encoder classifier trained with and without DANN using $\lambda_{max} = 0.2$. The two models are trained and tested on all the possible combinations of source/target domains available in BreizhCrops dataset. Accuracy, F1-Accuracy and K-score are the metrics used to compare the classification quality. Training accuracy is also reported for the Transformer encoder classifier. Maximum Mean Discrepancy computed on a subset of extracted features of source and target domain shows the successful reduction of features distance obtained with DANN.

Zone		**Transformer Encoder**					**DANN**			
Source Domain	**Target Domain**	**Train Accuracy**	**Test Accuracy**	**F1-Acc**	**K-Score**	**MMD**	**Test Accuracy**	**F1-Acc**	**K-Score**	**MMD**
1	2	0.8577	0.7877	0.5675	0.7229	0.1109	0.7628	0.5540	0.6950	0.0077
1	3	0.8577	0.7436	0.5266	0.6606	0.1620	0.7449	0.5080	0.6714	0.0183
1	4	0.8577	0.7941	0.5675	0.7294	0.0516	0.7960	0.5734	0.7343	0.0086
2	1	0.8951	0.7433	0.5309	0.6773	0.1577	0.7403	0.5161	0.6687	0.0208
2	3	0.8951	0.4967	0.3592	0.3642	0.6700	0.6505	0.4544	0.5483	0.0104
2	4	0.8951	0.6006	0.4395	0.4912	0.2536	0.7482	0.4832	0.6735	0.0416
3	1	0.8750	0.7767	0.5339	0.7122	0.1819	0.8045	0.5778	0.7488	0.0121
3	2	0.8750	0.6638	0.4594	0.5615	0.6254	0.7589	0.5334	0.6865	0.0277
3	4	0.8750	0.7348	0.5074	0.6504	0.1184	0.7968	0.5778	0.7338	0.0115
4	1	0.8870	0.7927	0.5551	0.7354	0.0339	0.8233	0.5822	0.7753	0.0039
4	2	0.8870	0.7600	0.5443	0.6870	0.0953	0.8003	0.5788	0.7399	0.0084
4	3	0.8870	0.7111	0.4961	0.6230	0.0960	0.7673	0.5443	0.6965	0.0062

empirical and expected probability of agreement on a label. In addition, we make use of Maximum Mean Discrepancy (MMD) metric, presented in Section 11.3.2, to quantitatively evaluate the distance between source and target distributions.

11.3.2 Maximum Mean Discrepancy

MMD is a statistical test originally proposed in [327] to determine a measure of the distance between two distributions. MMD is largely used in domain adaptation since it perfectly fits the need to understand whether the source and the target domain extracted features overlap. MMD can be directly exploited as a loss function for adversarial training of generative models or for domain adaptation purposes, as shown in [328, 329]. However, in this works we limit its usage to show the results of the Transformer Encoder DANN in terms of reduction of feature distances.
Formally, MMD is a kernel-based difference between feature means. Given a set of m samples X with a probability measure P, the feature mean can be expressed as:

$$\mu_p(\phi(X)) = [E[\phi(X_1)], \cdots, E[\phi(X_m)]]^T \tag{11.11}$$

Table 11.3 Comparison between Transformer Encoder Classifier with and without DANN, in terms of classification metrics reported in Table 11.2. This run of experiments is conducted with a scheduling of the adaptation parameter λ, with $\lambda_{max} = 0.2$.

Zone		Improvement [%]		
Source Domain	**Target Domain**	**Test Accuracy**	**F1-Accuracy**	**K-Score**
1	2	−3.1576	−2.3859	−3.8508
1	3	0.1762	−3.5378	1.6395
1	4	0.2296	1.0467	0.6773
2	1	−0.3996	−2.7935	−1.2698
2	3	30.9721	26.4916	50.5414
2	4	24.5690	9.9474	37.1046
3	1	3.5803	8.2152	5.1446
3	2	14.3204	16.1075	22.2539
3	4	8.4475	13.8791	12.8283
4	1	3.8705	4.8817	5.4228
4	2	5.3053	6.3384	7.6922
4	3	7.9018	9.7154	11.8067

where $\phi(X)$ is the feature map that maps X to a new feature space $\mathcal{F}$. If it satisfies the necessary theoretical conditions, a kernel-based approach can be used to compute the inner product of two distributions of samples $X \sim P$ and $Y \sim R$:

$$\langle \mu_P(\phi(X), \mu_Q(\phi(Y)\rangle_{\mathcal{F}} = E_{P,R}[\langle \phi(X), \phi(Y)\rangle_{\mathcal{F}}] = E_{P,Q}[k(X,Y)] \tag{11.12}$$

At this point the MMD can be defined as the distance between the feature means of $X \sim P$ and $Y \sim R$:

$$MMD^2(P,R) = \|\mu_P - \mu_R\|^2_{\mathcal{F}} \tag{11.13}$$

which can be expressed more in detail using Equation (11.12):

$$MMD^2(P,R) = E_P[k(X,X)] - 2E_{P,R}[k(X,Y)] + E_R[k(Y,Y)] \tag{11.14}$$

However, an empirical estimate of MMD needs to be computed since in a real case only samples are available instead of the explicit formulation of the distributions. It is possible to obtain the MMD expression by considering the empirical estimates of the feature means based on their samples:

$$MMD^2(X,Y) = \frac{1}{m(m-1)}\sum_i\sum_{j\neq i} k(\mathbf{x_i},\mathbf{x_j}) - 2\frac{1}{m\cdot m}\sum_i\sum_j k(\mathbf{x_i},\mathbf{y_j}) + \frac{1}{m(m-1)}\sum_i\sum_{j\neq i} k(\mathbf{y_i},\mathbf{y_j}) \tag{11.15}$$

where $\mathbf{x_i}$ and $\mathbf{y_i}$ in this case are the image samples from source and target domains, m is the number of samples of the considered subsets. Finally, we specifically use a gaussian kernel with the following expression:

$$k(\mathbf{x_i},\mathbf{x_j}) = \exp\left(\frac{-\|\mathbf{x_i}-\mathbf{x_j}\|^2}{2\sigma^2}\right) = \exp\left(\frac{-1}{\sigma^2}[\mathbf{x_i}^\intercal\mathbf{x_i} - 2\mathbf{x_i}^\intercal\mathbf{x_j} + \mathbf{x_j}^\intercal\mathbf{x_j}]\right) \tag{11.16}$$

11.3.3 Results and Applicability Study

In this section, we present the comparison results between the Transformer classifier with and without DANN, clearly highlighting the scenarios that present a definite advantage in applying adversarial training for training a classifier for LC&CC. From results in Tables 11.2 and 11.3, Fig. 11.6, it is possible to notice that DANN adversarial training allows the classifier to improve knowledge transferability to other domains for most of the cases. Nonetheless, we investigate a potential criterion to decide if the transfer of learning from source to target can be effectively improved by DANN. More in detail, since DANN aims to overlap feature distributions, we look at the extracted features from a subset of 10000 samples of each zone dataset that is considered representative of the total one. We use the set of extracted features to compute a numerical evaluation of the distance decrease, and to give a graphical visualization of the effect of DANN. From a quantitative perspective, we propose Maximum Mean Discrepancy as the feature distance metrics to detect suitable conditions where DANN is an appropriate methodology. To compute MMD without considering the clustering of classes, we only need unlabeled image samples. We use PCA algorithm to compute the principal components of the extracted features and we exploit them to provide 2D and 3D visualization of relevant cases. First, we can look at the MMD values obtained from both the Transformer encoder and DANN in Table 11.2. It is clear that DANN is always able to reduce the distance between

feature distributions. However, this is not always associated with an increase in classification performance. We realize that key information is contained in the MMD value obtained from source and target features, extracted by the standard classifier. This simple test is crucial and can also be done without labels. The best improvement with DANN is reached considering zone 2 as the source domain and selecting zone 3 as the target domain. The percentage improvement shown in Table 11.3, with an increase of more than 30% of accuracy, correlates with an initial MMD value for this specific case is equal to 0.6700, reduced by DANN to 0.0104. What can be deduced by this observation is that high values of the MMD indicate a lack of generalization of the classifier and a domain gap. It is also to consider that the geographical zones of interest are close to each other. Hence, it can be reasonable to find small domain gaps. A clear example is the case of zone 1, when chosen as source domain. This factor can be considered an additional difficulty of the study case. Therefore, it is possible that the same methodology applied to other regions on the planet, sharing the same categories of crops, can probably show greater results. Another peculiar case to be considered is: zones 4 (source) and 3 (target). The MMD value is low from the initial analysis of the case, without the intervention of DANN. However, a classification boost is always achieved.

We report a visual representation of the extracted features to add meaning to the previous considerations. In particular, Fig. 11.7–11.9 show the 2D principal components obtained from the peculiar cases defined below:

- **case 1**: zone 2 (source), zone 3 (target). In this case DANN shows the greatest improvements with an initial high value of MMD. Features are visually reported in Fig. 11.7: in (a,b) when extracted by standard Transformer encoder trained on the source domain, in (c,d) when extracted by DANN. The difference is visually clear. Features distributions are matched by DANN, with a resulting overlapping shape between source and target domain.
- **case 2**: zone 1 (source), zone 2 (target). In this case DANN shows the worst improvements with an initial low value of MMD. Features are visually reported in (a,b) of Fig. 11.8 when extracted by standard Transformer encoder, in (c,d) of the same Fig. 11.8 when extracted by DANN. They appear already similar also without DANN.
- **case 3**: zone 4 (source), zone 3 (target). In this case DANN shows noticeable improvements, regardless an initial low value of MMD. Features are visually

reported in (a,b) of Fig. 11.9 when extracted by standard Transformer encoder, in (c,d) of Fig. 11.9 when extracted by DANN. As with case 1, the difference is visually clear, and the effect of DANN can be easily appreciated.

Finally, case 1 and case 2 defined above are also considered for a 3D representation. Fig. 11.10 shows the obtained results. For each subplot in the figure, both source and target domain features are scattered. Thanks to this visual perspective, the effect of the DANN method is highlighted, considering both the worst and the best application scenario. In case 1, the difference between source and target features is shallow also without DANN, as shown in (a). By contrast, the situation from (c) to (d) is changed thanks to the adversarial training significantly. The proposed discussion underlines some interesting insights on the correlation between reducing the domain gap and improving a classifier performance. The isolated cases considered provide a good reference example to decide if it is a reasonable and convenient choice to adopt the proposed DANN methodology for multi-spectral temporal sequences for Land Cover classification.

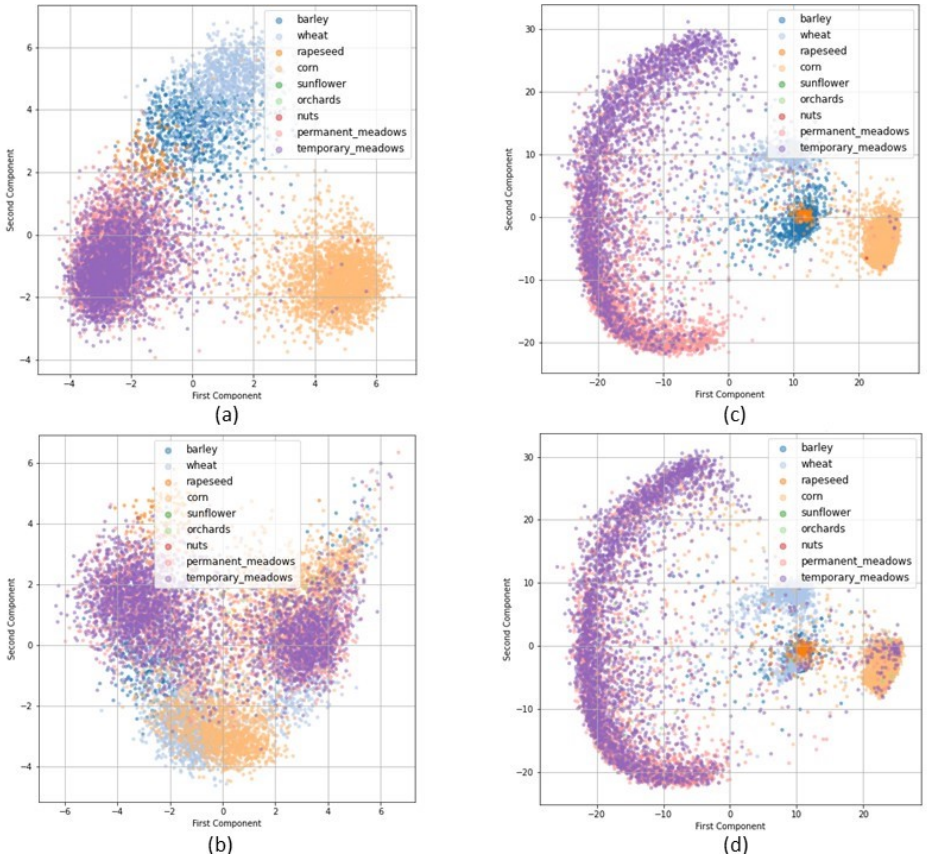


Fig. 11.7 2D feature visualization obtained with PCA, extracted with the Transformer Encoder trained on the source domain and with the Transformer DANN model trained on the specific source-target domains. A comparison between the 2D feature distributions is proposed for the case of zone 2 (source) and 3 (target). In (**a**,**b**) we have features extracted with the Transformer Encoder from source and target domains: (**a**) reports features of the source domain (zone 2) and (**b**) the ones extracted from the target domain (zone 3). In this case, features are mapped poorly in the target domain, with a consequent low accuracy in classification. In (**c**,**d**) the same features extracted with the Transformer DANN model are shown. The positive effect of DANN in terms of features overlapping is evident compared to (**a**,**b**).

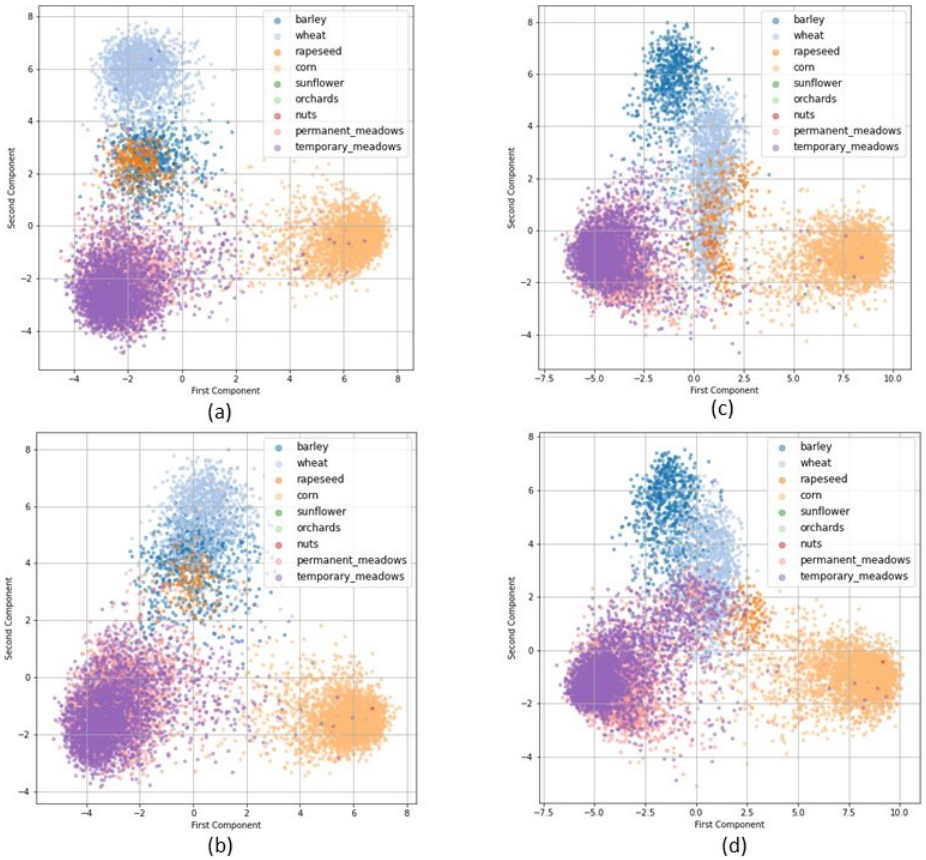


Fig. 11.8 2D feature visualization obtained with PCA, extracted with the Transformer Encoder trained on the source domain and with the Transformer DANN model trained on the specific source-target domains. A comparison between the 2D feature distributions is proposed for the case of zone 1 (source) and 2 (target). In (**a**,**b**) we have features extracted with the Transformer Encoder from source, (**a**), and target, (**b**), domain: a low MMD distance indicates no need for domain adaptation. In (**c**,**d**) the same features extracted with the Transformer DANN model are shown, with no substantial differences.

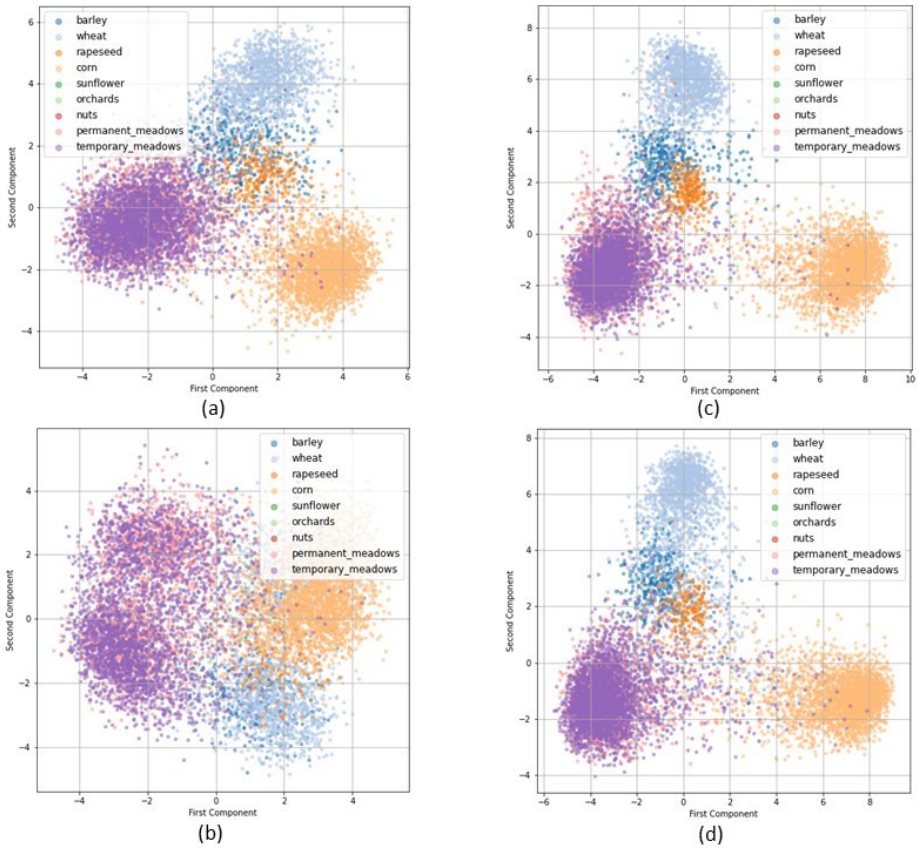


Fig. 11.9 2D feature visualization obtained with PCA, extracted with the Transformer Encoder trained on the source domain and with the Transformer DANN model trained on the specific source-target domains. A comparison between the 2D feature distributions is proposed for the case of zone 4 (source) and 3 (target). In (**a**,**b**) we have features extracted with the Transformer Encoder from source, (**a**), and target, (**b**), domains: regardless of an initial low MMD, the classifier accuracy can still be improved reducing the domain gap. In (**c**,**d**) the same features extracted with the Transformer DANN model are shown, with a clear improvement of the feature mapping, which result in very similar distributions from source to target domain.

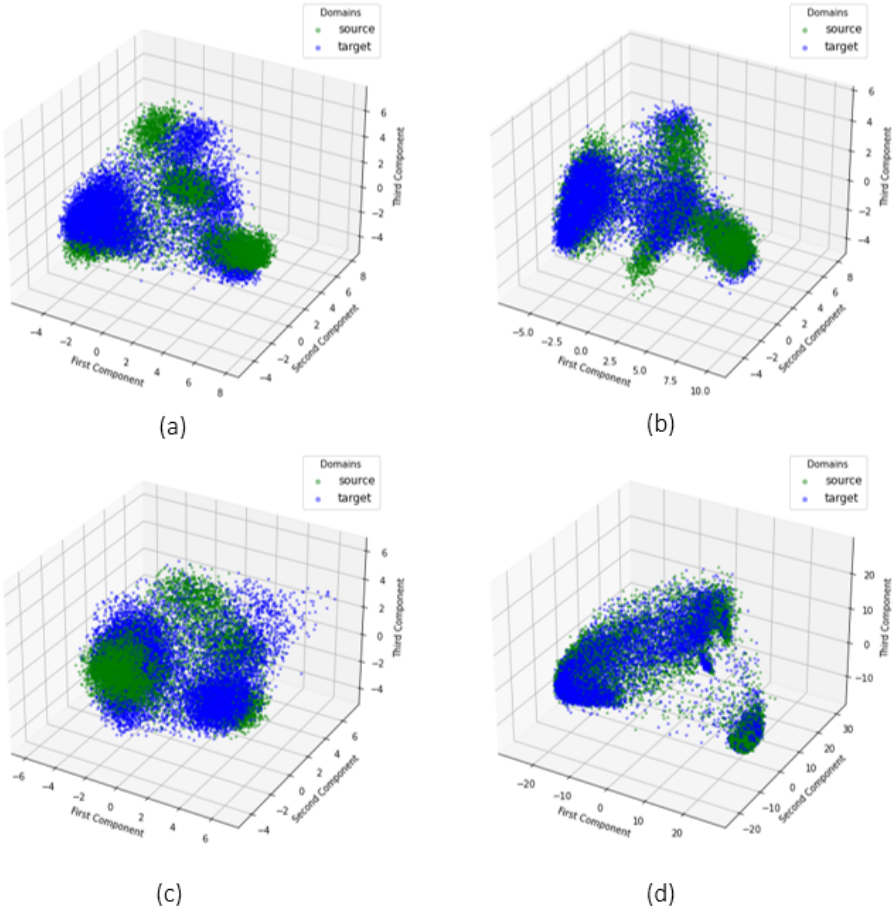


Fig. 11.10 3D feature visualization and comparison. (**a**,**b**) show the features extracted from zone 1 (source) and 2 (target). They are respectively obtained with transformer encoder and DANN. It is clear that the transformer encoder alone can correctly map features on both domains. By contrast, the improvement provided by DANN model is very evident in figures (**c**,**d**), representing the features extracted from zone 2 (source) and 3 (target), where the transformer encoder alone present both high values of MMD and low classification accuracy on target domain.

Chapter 12

Optimized Single-Image Super-Resolution at the Edge with Knowledge Distillation

In the last decade, Deep Learning (DL) techniques have pervaded robotic systems and applications, drastically boosting automation in both perception [330, 93], navigation and control [331, 332] tasks. The development of Machine Learning driven algorithms is paving the way for advanced levels of autonomy for mobile robots, widely increasing the reliability of both unmanned aerial vehicles (UAV) and unmanned ground vehicles (UGV) [330]. In this context, the successful transmission of images acquired by the robot to the ground station often assumes a significant relevance to the task at hand, allowing the human operators to get real-time information, monitor the state of the mission, take critical planning decisions and analyze the scenario. Moreover, unknown outdoor environments may present unexpected extreme characteristics which still hinder the release of unmanned mobile robots in the complete absence of human supervision. Complete or partial remote teleoperation remains the most reliable control strategy in uncertain scenarios. Indeed, irregular terrain, lighting conditions, and the loss of localization signal can lead navigation algorithms to fail. As a direct consequence of navigation errors, the robotic platform can get stuck in critical states where human intervention is required or preferred.

However, visual data transmission for robot teleoperation, monitoring, or online data processing requires a stable continuous stream of images, which may be drastically

affected by poor bandwidth conditions due to the long distance of the robot or by constitutive factors of the specific environment. Besides this, UAVs and high-speed platforms require the pilot to receive the image stream at a high framerate to follow the vehicle's motion in non-line-of-sight situations. A straightforward but effective solution to mitigate poor bandwidth conditions and meet high-frequency transmission requirements is reducing the transmitted image's resolution. On the other hand, heavy image compression with massive loss of detail can compromise image usability. To this end, we propose EdgeSRGAN, a novel deep learning model for Single-Image Super-Resolution (SISR) at the edge to handle the problem of efficient image transmission. Single-Image Super-Resolution, also referred to as super-sampling or image restoration, aims at reconstructing a high-resolution (HR) image starting from a single low-resolution (LR) input image, trying to preserve details and the information conceived by the image. Therefore SISR, together with image denoising, is an ill-posed underdetermined inverse problem since a multiplicity of possible solutions exist given an input low-resolution image. Recently, learning-based methods have rapidly reached state-of-the-art performance and are universally recognized as the most popular approach for Super-Resolution. Such approaches rely on learning common patterns from multiple LR-HR pairs in a supervised fashion. SRCNN [333] was the first example of a CNN applied to single-image super-resolution in literature. It has been followed by multiple methods applying standard deep learning methodologies to SISR, such as residual learning [334, 335], dense connections [336], residual feature distillation [337], attention [338–340], self-attention, and transformers [341–343]. All these works focus on content-based SR, in which the objective is to reconstruct an image with high pixel fidelity, and the training is based on a content loss, such as mean square error or mean absolute error. In parallel, other works proposed Generative Adversarial Networks (GAN) [312] for SISR to aim at reconstructing visually pleasing images. In this case, the focus is not on pixel values but perceptual indexes that try to reflect how humans perceive image quality. This is usually implemented using perceptual losses and adversarial training and is referred to as visual-based SR. SRGAN [344] first proposed adversarial training and was later followed by other works [335, 345, 346]. With robotic image transmission as a target application in mind, in this work, we particularly focus on visual-based SR, aiming to reconstruct visually pleasing images to be used by human operators for real-time teleoperation and monitoring.

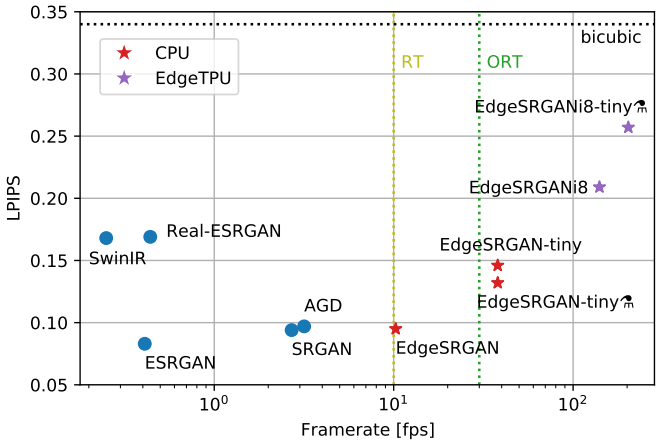


Fig. 12.1 LPIPS [347] results (lower is better) on Set5 [348] vs framerate (80×60 input) of different visual-oriented SISR methods for $\times 4$ upsampling. Real-time (RT) and over-real-time (ORT) framerates are marked as references. Our models, marked with $\star$, reach real-time performance with a competitive perceptual similarity index on the CPU. Edge TPU models can further increase inference speed far beyond real-time, still outperforming the bicubic baseline.

Our intuition relies on a lightweight neural network allowing us to send low-resolution images at a high transmission rate with scarce bandwidth and then reconstruct the high-resolution image on the pilot's mobile device. Moreover, the successful spread of edge-AI in different engineering applications [349–351] has shown encouraging results in moving the execution of DL models on ultra-low power embedded devices. Hence, we propose an edge-AI computationally efficient Super Resolution neural network to provide fast inference on CPUs and Edge TPU devices. To this aim, we adopt several optimization steps to boost the performance of our model while minimizing the quality drop. We refine the architecture of the original SRGAN [344] to speed up inference and perform model quantization. Nonetheless, we experiment with a teacher-student knowledge distillation technique for SISR to further enhance the reconstructed image of our tiny model. We take inspiration from the work of [352] and obtain a remarkable improvement for all the considered metrics. We perform experiments to validate the proposed methodology under multiple perspectives: numerical and qualitative analysis of our model reconstructed images and inference efficiency on both CPU and Edge TPU devices. As an example, as shown in Fig. 12.1, EdgeSRGAN achieves real-time performance with a competitive perceptual similarity index compared with other visual-oriented SISR

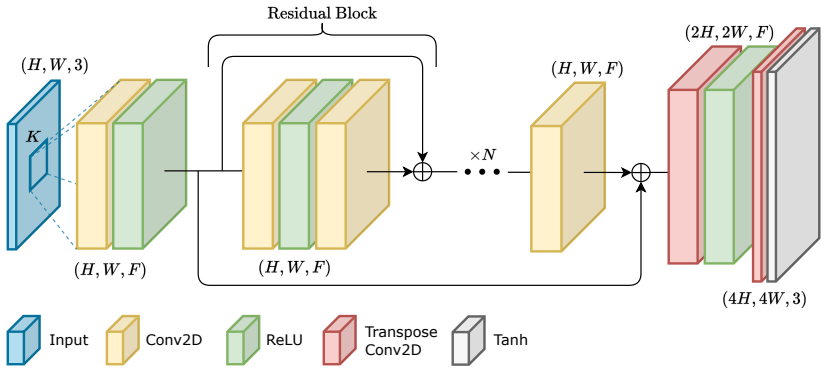


Fig. 12.2 EdgeSRGAN Generator Architecture.

methods. Moreover, we test the performance of our system for robotic applications. In particular, we focus on image transmission for teleoperation in case of bandwidth degradation, also performing tests with the popular robotic middleware ROS2. Other potential applications of efficient SISR can be found in underwater robotics perception [353, 354], robotic surgery [355, 356], and inspection of bridges cracks [357]. The code of the project is available at https://github.com/PIC4SeR/EdgeSRGAN.

12.1 Methodology

In this section, we introduce all the components of the proposed methodology. We choose to use an adversarial approach to obtain an optimal balance between pixel-wise fidelity and perceptual quality. For this reason, we take inspiration from three of the most popular GAN-based solutions for SISR: SRGAN [344], ESRGAN [358], and AGD [345]. The proposed method aims to obtain a real-time SISR model (EdgeSRGAN) with minimal performance drop compared to state-of-the-art solutions. For this reason, we mix successful literature practices with computationally-efficient elements to obtain a lightweight architecture. Then, we design the network training procedure to leverage a combination of pixel-wise loss, perceptual loss, and adversarial loss. To further optimize the inference time, we apply knowledge distillation to transfer the performance of EdgeSRGAN to an even smaller model (EdgeSRGAN-tiny). Furthermore, we study the effect of quantization on the network's latency and accuracy. Finally, we propose an additional inference-time network interpolation feature to allow real-time balancing between pixel-wise precision and photo-realistic textures.

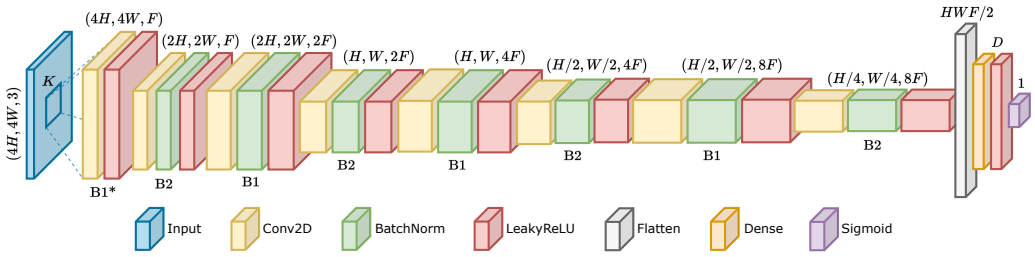


Fig. 12.3 EdgeSRGAN Discriminator Architecture. The model progressively reduces the spatial dimensions of the image by alternating blocks with strides 1 (B1) and 2 (B2). The first block (marked with *) does not apply batch normalization.

12.1.1 Network Architecture

As previously done by [358], we take the original design of SRGAN and propose some changes to both the architecture and training procedure. However, in our case, the modifications seek efficiency as well as performance. To obtain a lighter architecture, we reduce the depth of the model by using only $N = 8$ Residual Blocks instead of the original 16. In particular, we use simple residuals instead of the Residual-in-Residual Dense Blocks (RRDB) proposed by [358] as they are less computationally demanding. For the same reason, we change PReLU activation functions into basic ReLU. We also remove Batch Normalization to allow the model for better convergence without generating artifacts [358]. Finally, we use Transpose Convolution for the upsampling head instead of Sub-pixel Convolution [359]. Despite its popularity and effectiveness, Sub-pixel Convolution is computationally demanding due to the Pixel Shuffling operation, which rearranges feature channels spatially. We choose instead to trade some performance for efficiency and apply Transpose Convolutions taking precautions to avoid problems such as checkerboard artifacts [360]. The complete EdgeSRGAN architecture is described in Fig. 12.2. The adopted discriminator model is the same used in [344, 358], as it serves only training purposes and is not needed at inference time. Its architecture is described in Fig. 12.3.

12.1.2 Training Methodology

The training procedure is divided into two sections, as it is common practice in generative adversarial SISR. The first part consists of classic supervised training using pixel-wise loss. In this way, we help the generator to avoid local minima and

generate visually pleasing results in the subsequent adversarial training. We use the mean absolute error (MAE) loss for the optimization as it has recently proven to bring better convergence than mean squared error (MSE) [361, 335, 338, 358].

$$\mathcal{L}_{\text{MAE}} = ||\mathbf{y}_{\text{HR}} - \mathbf{y}_{\text{SR}}||_1 \tag{12.1}$$

where y_{HR} is the ground-truth high resolution image, y_{SR} is the output of the generator, and B is the batch size. We use the Peak Signal-to-Noise Ratio (PSNR) metric to validate the model. In the second phase, the resulting model is fine-tuned in an adversarial fashion, optimizing a loss that takes into account adversarial loss and perceptual loss. As presented in [344], the generator G training loss can be formulated as

$$\mathcal{L}_G = \mathcal{L}_G^P + \xi\,\mathcal{L}_G^A + \eta\,\mathcal{L}_{\text{MAE}}. \tag{12.2}$$

$\mathcal{L}_G^P$ is the perceptual loss introduce by [344], computed as the MSE between the feature representations $\phi(\cdot)$ of a reconstructed image y_{SR} and the reference image y_{HR}. The features are extracted using VGG19 [362] as backbone pre-trained on ImageNet:

$$\mathcal{L}_G^P = ||\phi(\mathbf{y}_{\text{HR}}) - \phi(\mathbf{y}_{\text{SR}})||_2 \tag{12.3}$$

Differently, the adversarial generator loss L_G^A is defined as:

$$\mathcal{L}_G^A = -\log(D(\mathbf{y}_{\text{SR}})) \tag{12.4}$$

where D is the discriminator. Using this loss, the generator tries to fool the discriminator by generating images that are indistinguishable from the real HR ones. ξ and η are used to balance the weight of different loss components. The weights of the discriminator D are optimized using a symmetrical adversarial loss, which tends to correctly discriminate HR and SR images.

$$\mathcal{L}_D = \log(D(\mathbf{y}_{\text{SR}})) - \log(D(\mathbf{y}_{\text{HR}})) \tag{12.5}$$

We optimize both models simultaneously, without alternating weight updates like in most seminal works on GANs. The overall training methodology is summarized in Fig. 12.4 summarizes the overall training methodology.

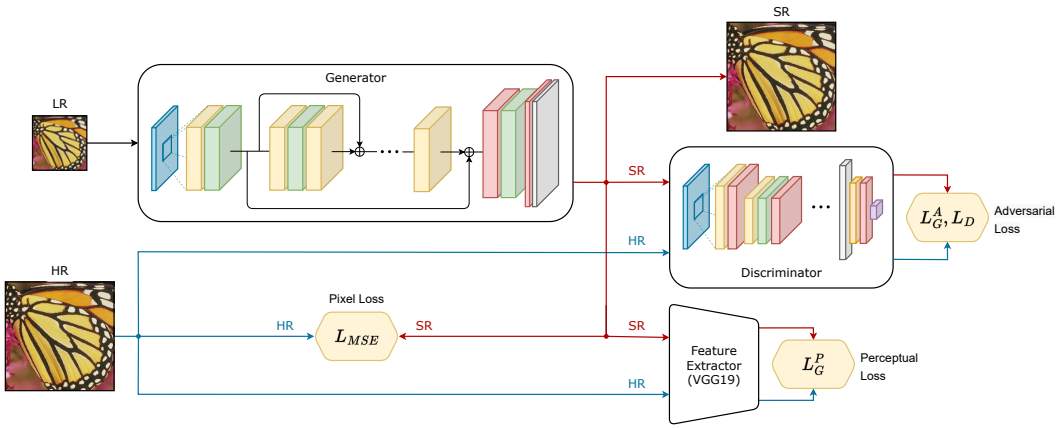


Fig. 12.4 EdgeSRGAN Training Methodology.

12.1.3 Knowledge Distillation

In Chapter 10, KD has been exploited to transfer knowledge with the aim of boosting the generalization of student models. Differently here, we use it to transfer knowledge from bigger models to simpler ones efficiently. KD has been applied in some SISR works to compress the texture reconstruction capability of cumbersome models and obtain efficient real-time networks [352, 363]. However, to the best of our knowledge, KD has never been applied to GAN SISR models. For this reason, we adapt an existing technique developed for SISR called Feature Affinity-based Knowledge Distillation (FAKD) [352] to the GAN training approach. The FAKD methodology transfers second-order statistical info to the student by aligning feature affinity matrices at different layers of the networks. This constraint helps to tackle the fact that regression problems generate unbounded solution spaces. Indeed, most of the KD methods so far have only tackled classification tasks. Given a layer l of the network, the feature map $\mathbf{F}_l$ extracted from that layer (after the activation function) has shape $C \times W \times H$, C is the number of channels, W and H are the width and the height of the tensor. We first flatten the tensor along the last two components obtaining the three-dimensional feature map $\mathbf{F}_l$ with shape $C \times (W \cdot H)$ which now holds all the spatial information along a single axis. We define the affinity matrix A_l as the product:

$$\mathbf{A}_l = \tilde{\mathbf{F}}_l^{\top} \cdot \tilde{\mathbf{F}}_l \tag{12.6}$$

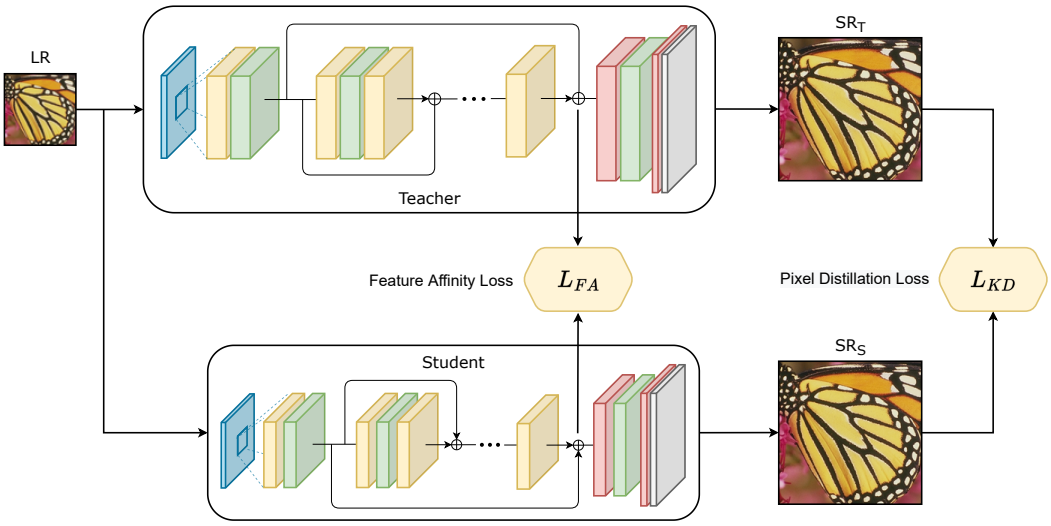


Fig. 12.5 EdgeSRGAN Distillation Process.

where $\cdot$ is the matrix multiplication operator and the transposition $\top$ swaps the last two dimensions of the tensor. $\tilde{\mathbf{F}}_l$ is the normalized feature map, obtained as

$$\tilde{\mathbf{F}}_l = \frac{\mathbf{F}_l}{||\mathbf{F}_l||_2} \tag{12.7}$$

Differently from [352], the norm is calculated for the whole tensor and not only along the channel axis. Moreover, we find better convergence using the euclidean norm instead of its square. In this way, the affinity matrix $\mathbf{A}_l$ has shape $(W \cdot H) \times (H \cdot W)$ and the total distillation loss L_{Dist} can be computed as:

$$\mathcal{L}_{\text{Dist}} = \lambda ||\mathbf{y}^T_{\text{SR}} - \mathbf{y}^S_{\text{SR}}||_1 + \frac{1}{N_L}\left(\sum_{l=1}^{N_L} ||\mathbf{A}^T_l - \mathbf{A}^S_l||_1\right) \tag{12.8}$$

where N_L is the number of distilled layers. Differently from [352], we sum the loss along all the tensor dimensions and average the result obtained for different layers. These modifications experimentally lead to better training convergence. We also add another loss component, weighted by λ, which optimizes the model to generate outputs close to the teacher's ones. In our experimentation, the distillation loss is added to the overall training loss weighted by the parameter γ. The overall distillation scheme is summarized in Fig. 12.5.

12.1.4 Model Interpolation

Following the procedure proposed in [358], we adopt a flexible and effective strategy to obtain a tunable trade-off between a content-oriented and GAN-trained model. This feature can be very useful for real-time applications, as it allows the SISR network to adapt to the user's needs promptly. Indeed, some real scenarios may need better perceptual quality, for example, when the remote control of a robot has to be performed by a human pilot. On the other hand, when images are used to directly feed perception, autonomous navigation, and mapping algorithms, higher pixel fidelity might be beneficial. To achieve this goal, we linearly interpolate model weights layer-by-layer, according to the following formula:

$$\theta_G^{\text{Int}} = \alpha\theta_G^{\text{PSNR}} + (1-\alpha)\theta_G^{\text{GAN}} \tag{12.9}$$

where θ_G^{Interp}, θ_G^{PSNR}, and θ_G^{GAN} are the weights of the interpolated model, the PSNR model, and the GAN fine-tuned model, respectively. $\alpha \in [0,1]$ is the interpolation weight. We report both qualitative and quantitative interpolation results for EdgeSRGAN in Section 12.2.3. We avoid the alternative technique of directly interpolating network outputs: applying this method in real time would require running two models simultaneously. Moreover, Wang et al.[358] report that this approach does not guarantee an optimal trade-off between noise and blur.

12.1.5 Model Quantization

To make EdgeSRGAN achieve even lower inference latency, we apply optimization methods to the model to reduce the computational effort at the cost of a loss in performance. Several techniques have been developed to increase model efficiency in the past few years [364], from which the employed method is chosen. We reduce the number of bits used to represent network parameters and activation functions with TFLite[1]. This strategy strongly increases efficiency with some impact on performance. We quantize weights, activations, and math operations through scale and zero-point parameters following the methodology presented by Jacob et al. [364]:

$$r = S(q-Z) \tag{12.10}$$

[1] https://www.tensorflow.org/lite/

where r is the original floating-point value, q is the quantized integer value, and S and Z are the quantization parameters (scale and zero point). A fixed-point multiplication approach is adopted to cope with the non-integer scale of S. This strategy drastically reduces memory and computational demands due to the high efficiency of integer computations on microcontrollers. For our experimentation, we deploy the quantized model on a Google Coral Edge TPU USB Accelerator.

Table 12.1 Framerate comparison of different methods for $\times 4$ and $\times 8$ upsampling, with two different input resolutions (80×60 and 160×120). The results are provided as mean and standard deviation of 10 independent experiments of 100 predictions each. Current content-oriented SISR state-of-art method SwinIR [343] is reported as a reference. Real-time and over-real-time framerates are in blue and red, respectively. The proposed solution is the only one compatible with EdgeTPU devices and allows reaching real-time performance in both conditions.

			Framerate (80×60) [fps]		**Framerate** (160×120) [fps]	
Method	**Scale**	**Params**	**CPU**	**EdgeTPU**	**CPU**	**EdgeTPU**
SwinIR [343]		11.9M	0.25 ± 0.01	-	0.06 ± 0.01	-
ESRGAN [358]		16.7M	0.40 ± 0.01	-	0.10 ± 0.01	-
Real-ESRGAN [346]		16.7M	0.44 ± 0.01	-	0.11 ± 0.01	-
SRGAN [344]	$\times 4$	1.5M	2.70 ± 0.08	-	0.95 ± 0.02	-
AGD [345]		0.42M	3.17 ± 0.12	-	0.88 ± 0.01	-
EdgeSRGAN		0.66M	10.26 ± 0.11	140.23 ± 1.50	2.66 ± 0.02	10.63 ± 0.03
EdgeSRGAN-tiny		0.09M	37.99 ± 1.42	203.16 ± 3.03	11.76 ± 0.20	20.57 ± 0.05
SwinIR [343]		12.0M	0.23 ± 0.01	-	0.06 ± 0.01	-
EdgeSRGAN	$\times 8$	0.71M	7.70 ± 0.31	14.26 ± 0.06	1.81 ± 0.04	-
EdgeSRGAN-tiny		0.11M	24.53 ± 1.28	41.55 ± 0.38	5.81 ± 0.29	-

12.2 Experiments

12.2.1 Experimental Setting

In this section, we define our method's implementation details and the procedure we followed to train and validate the efficiency of EdgeSRGAN optimally. As previously done by most GAN-based SISR works, we train the network on the high-quality DIV2K dataset [365] with a scaling factor of 4. The dataset contains 800 training samples and 100 validation samples. We train our model with input images of size 24x24 pixels, selecting random patches from the training set. We apply data augmentation by randomly flipping or rotating the images by multiples of 90°. We adopt a batch size of 16.

For the standard EdgeSRGAN implementation, we choose $N = 8$, $F = 64$, $K = 3$, and $D = 1024$, obtaining a generator with around 660k parameters and a discriminator of over 23M (due to the fully-connected head). The discriminator is built with $F = 64$, $K = 3$, $D = 512$, and with a coefficient for LeakyReLU $\alpha = 0.2$. We first train EdgeSRGAN pixel-wise for 5×10^5 steps with Adam optimizer and a constant learning rate of 1×10^{-4}. Then, the model is fine-tuned in the adversarial setting described in Section 12.1 for 1×10^5 steps. Adam optimizer is used for the generator and the discriminator with a learning rate of 1×10^{-5}, further divided by 10 after 5×10^4 steps. For the loss function, we set $\xi = 1 \times 10^{-3}$ and $\eta = 0$.

To obtain an even smaller model for our distillation experiments, we build EdgeSRGAN-tiny by choosing $N = 4$, $F = 32$, and $D = 256$. We further shrink the size of the discriminator by eliminating the first compression stage ($B1$) from each block (see Fig. 12.3). In this configuration, we also remove the batch normalization layer from the first B2 block to be coherent with the larger version. The obtained generator and discriminator contain around 90k and 2.75M parameters. The pre-training procedure is the one described for EdgeSRGAN, while the adversarial training is performed with the additional distillation loss ($\gamma = 1 \times 10^{-2}$, $\lambda = 1 \times 10^{-1}$) of Eq. 12.8. EdgeSRGAN is used as a teacher model, distilling its layers 2, 5, and 8 into EdgeSRGAN-tiny's layers 1, 2, and 4. The model is trained with a learning rate of 1×10^{-4}, which is further divided by 10 after 5×10^4 steps. For the loss function, we set $\xi = 1 \times 10^{-3}$ and $\eta = 0$.

Finally, we create a third version of our model to upscale images with a factor of 8. To do so, we change the first transpose convolution layer of EdgeSRGAN and EdgeSRGAN-tiny to have a stride of 4 instead of 2 and leave the rest of the architecture unchanged. The training procedure for these models is analogous to the ones used for the x4 models, with the main difference of adding a pixel-based component to the adversarial loss by posing $\eta = 1 \times 10^2$.

The optimal training hyperparameters are found by running a random search and choosing the best-performing models on DIV2K validation. During GAN training, we use PSNR to validate the models during content-based loss optimization and LPIPS [347] (with AlexNet backbone).

We employ TensorFlow 2 and a workstation with 64 GB of RAM, an Intel i9-12900K CPU, and an Nvidia 3090 RTX GPU to perform all the training experiments.

12.2.2 Real-time Performance

Since the main focus of the proposed methodology is to train an optimized SISR model to be efficiently run at the edge in real time, we first report an inference speed comparison between the proposed method and other literature methodologies. All the results are shown in Tab. 12.1 as the mean and standard deviation of 10 independent experiments of 100 predictions each. We compare the proposed methodology with other GAN-based methods [344, 358, 346, 345] and with the current state-of-the-art in content-oriented SISR SwinIR [343]. Since the original implementations of the GAN-based solutions consider $\times 4$ upsampling only, for the $\times 8$ comparison, we only report SwinIR. We select two different input resolutions for the experimentation, (80×60) and (160×120), in order to target (320×240) and (640×480) resolutions for $\times 4$ upsampling and (640×480) and (1280×960) for $\times 8$ upsampling, respectively. This choice is justified because (640×480) is a standard resolution provided by most cameras' native video stream. We also report the number of parameters for all the models.

For all the considered methods, we measure the CPU timings with the model format of the original implementation (PyTorch or TensorFlow) on a MacBook Pro with an Intel i5-8257U processor. The concept of real-time performance strongly depends on the downstream task. For robotic monitoring and teleoperation, we consider 10 fps as the minimum real-time framerate, considering over-real-time everything above 30 fps, which is the standard framerate for most commercial cameras. The proposed methodology outperforms all the other methods in inference speed and achieves real-time performance on the CPU in almost all the testing conditions. It is worth noting that AGD is specifically designed to reduce latency for GAN-based SR and has fewer parameters than EdgeSRGAN, but it still fails at achieving real-time without a GPU.

In addition, we report the framerate of the EdgeSRGAN int8-quantized models on an EdgeTPU Coral USB Accelerator. The proposed solution is the only one compatible with such devices and allows reaching over-real-time performance for (80×60) input resolution. It must be underlined how the $\times 8$ models with (160×120) input resolution cannot target the EdgeTPU device due to memory limitations.

Table 12.2 Quantitative comparison of different methods for content-oriented ×4 upsampling. Current SISR state-of-art method SwinIR [343] and bicubic baseline are reported as reference.
↑: higher is better, ↓: lower is better, †: trained on DIV2K [365] + Flickr2K [366] + OST [367]

	Set5 [348]			**Set14** [368]			**BSD100** [369]			**Manga109** [370]			**Urban100** [371]		
Method	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓
Bicubic	28.632	0.814	0.340	26.212	0.709	0.441	26.043	0.672	0.529	25.071	0.790	0.318	23.236	0.661	0.473
SwinIR [343]	32.719	0.902	0.168	28.939	0.791	0.268	27.834	0.746	0.358	31.678	0.923	0.094	27.072	0.816	0.193
SRGAN [344]	32.013	0.893	0.191	28.534	0.781	0.294	27.534	0.735	0.396	30.292	0.906	0.111	25.959	0.782	0.244
ESRGAN [358]†	32.730	0.901	0.181	28.997	0.792	0.275	27.838	0.745	0.371	31.644	0.920	0.097	27.028	0.815	0.201
AGD [345]	31.708	0.889	0.178	28.311	0.775	0.291	27.374	0.729	0.385	29.413	0.897	0.118	25.506	0.767	0.250
EdgeSRGAN	31.729	0.889	0.191	28.303	0.774	0.301	27.359	0.728	0.405	29.611	0.897	0.120	25.469	0.764	0.266
EdgeSRGAN-tiny	30.875	0.873	0.204	27.796	0.761	0.320	26.999	0.717	0.418	28.233	0.871	0.163	24.695	0.733	0.325

Table 12.3 Quantitative comparison of different methods for visual-oriented ×4 upsampling. Current SISR state-of-art method SwinIR [343] and bicubic baseline are reported as reference. ↑: higher is better, ↓: lower is better. †: trained on DIV2K [365] + Flickr2K [366] + OST [367].

	Set5 [348]			**Set14** [368]			**BSD100** [369]			**Manga109** [370]			**Urban100** [371]		
Model	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓
Bicubic	28.632	0.814	0.340	26.212	0.709	0.441	26.043	0.672	0.529	25.071	0.790	0.318	23.236	0.661	0.473
SwinIR [343]	32.719	0.902	0.168	28.939	0.791	0.268	27.834	0.746	0.358	31.678	0.923	0.094	27.072	0.816	0.193
SRGAN [344]	29.182	0.842	0.094	26.171	0.701	0.172	25.447	0.648	0.206	27.346	0.860	0.076	24.393	0.728	0.158
ESRGAN[358]†	30.459	0.852	0.083	26.283	0.698	0.139	25.288	0.649	0.168	28.478	0.860	0.065	24.350	0.733	0.125
Real-ESRGAN [346]†	26.617	0.807	0.169	25.421	0.696	0.234	25.089	0.653	0.282	25.985	0.836	0.149	22.671	0.686	0.214
AGD [345]	30.432	0.861	0.097	27.276	0.739	0.160	26.219	0.688	0.214	28.163	0.870	0.076	24.732	0.743	0.170
EdgeSRGAN	29.487	0.837	0.095	26.814	0.715	0.176	25.543	0.644	0.210	27.679	0.855	0.081	24.268	0.716	0.170
EdgeSRGAN-tiny	28.074	0.803	0.146	26.001	0.702	0.242	25.526	0.658	0.292	25.655	0.804	0.140	23.332	0.672	0.269
EdgeSRGAN-tiny-D	29.513	0.841	0.132	26.950	0.727	0.220	26.174	0.673	0.282	27.106	0.845	0.130	24.117	0.704	0.249

12.2.3 Super-Resolution Results

To present quantitative results on image super-resolution, we refer to content-oriented SR for models trained with content-based loss only and visual-oriented SR for models trained with adversarial and perceptual losses. Content-based loss (mean absolute error or mean squared error) aims to maximize PSNR and SSIM, while adversarial and perceptual losses aim to maximize visual quality. We test EdgeSRGAN models on five benchmark datasets (Set5 [348], Set14 [368], BSD100 [369], Manga109 [370], and Urban100 [371]) measuring PSNR, SSIM, and LPIPS. We follow the standard procedure for SISR adopted in [343], where the metrics are computed on the luminance channel Y of the YCbCr converted images. Also, S pixels are cropped from each image border, where S is the model scale factor.

Tab. 12.2 and Tab. 12.3 show the comparison with other methods for content-oriented and visual-oriented $\times 4$ SR, respectively. We report results of other GAN-based methodologies [344, 358, 346, 345] as well as the current content-oriented SOTA SwinIR [343] and bicubic baseline, as reference. Unlike what is usually found in literature, we refer to the OpenCV bicubic resize implementation instead of the one present in MATLAB. For visual-oriented SR, we also report the results of the distilled tiny model EdgeSRGAN-tiny-D. The proposed method reaches competitive results in all the metrics, even with some degradation for tiny models due to the considerable weight reduction. The distillation method helps EdgeSRGAN-tiny training by transferring knowledge from the standard model and decreasing the degradation due to the reduced number of parameters. Note that ESRGAN and RealESRGAN are trained on Flickr2K [366], and OST [367] datasets in addition to DIV2K. Tab. 12.4 reports results of the $\times 8$ models, together with SwinIR and bicubic. Also, in this case, the proposed models reach competitive results, and knowledge distillation helps to reduce performance degradation in the tiny model. As a final qualitative evaluation, Fig. 12.6 compares the super-resolved images obtained by EdgeSRGAN with the considered state-of-the-art solutions. Our model shows comparable results, highlighting more texture and details than networks trained with pixel loss (L_{MSE}) while remaining true to the ground truth image.

Network Interpolation

We report the results of network interpolation on the benchmark datasets in Fig. 12.8. We consider α values between 0 and 1 with a step of 0.1, with 0 implying

Table 12.4 Quantitative performance of the proposed method for ×8 upsampling. Current SISR state-of-art method SwinIR [343], and bicubic are reported as references. ↑: higher is better, ↓: lower is better.

		Set5 [348]			**Set14** [368]			**BSD100** [369]			**Manga109** [370]			**Urban100** [371]		
Model		**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓
Bicubic		24.526	0.659	0.533	23.279	0.568	0.628	23.727	0.546	0.713	21.550	0.646	0.535	20.804	0.515	0.686
SwinIR [343]		27.363	0.787	0.284	25.265	0.652	0.428	24.984	0.606	0.537	25.246	0.800	0.229	23.023	0.646	0.375
EdgeSRGAN	content	26.462	0.755	0.321	24.507	0.626	0.460	24.590	0.587	0.567	23.840	0.753	0.294	22.001	0.592	0.463
EdgeSRGAN-tiny	content	26.025	0.732	0.359	24.286	0.615	0.488	24.383	0.577	0.591	23.154	0.723	0.353	21.680	0.570	0.520
EdgeSRGAN	visual	25.307	0.680	0.228	23.585	0.558	0.348	23.547	0.514	0.386	22.719	0.680	0.257	21.102	0.522	0.374
EdgeSRGAN-tiny	visual	25.523	0.693	0.280	23.976	0.589	0.399	24.163	0.557	0.475	22.874	0.695	0.317	21.477	0.546	0.459

Table 12.5 Quantitative performance of the full-integer quantized models for ×4 and ×8 visual-based SR. ↑: higher is better, ↓: lower is better.

		Set5 [348]			**Set14** [368]			**BSD100** [369]			**Manga109** [370]			**Urban100** [371]		
Model	**Scale**	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓
EdgeSRGANi8	×4	27.186	0.721	0.209	24.714	0.475	0.342	23.675	0.484	0.438	25.601	0.712	0.221	22.802	0.580	0.341
EdgeSRGANi8-tiny-D	×4	27.330	0.710	0.257	24.807	0.562	0.390	23.837	0.485	0.481	25.299	0.696	0.286	22.580	0.538	0.454
EdgeSRGANi8	×8	24.433	0.602	0.312	22.846	0.477	0.440	22.609	0.422	0.492	22.227	0.603	0.342	20.525	0.433	0.499
EdgeSRGANi8-tiny	×8	24.956	0.642	0.333	23.487	0.532	0.461	23.591	0.494	0.544	22.445	0.632	0.386	21.125	0.489	0.548

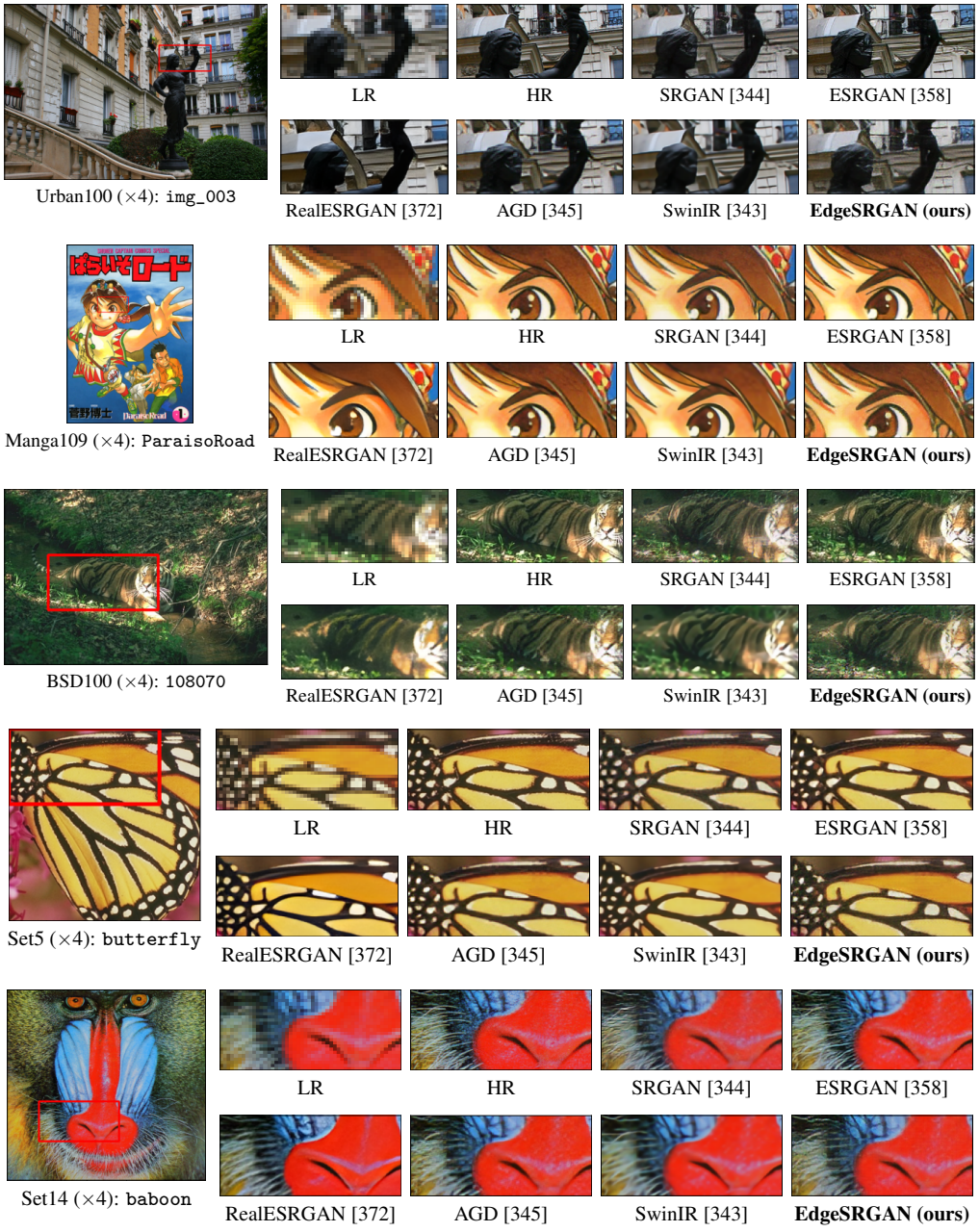


Fig. 12.6 Visual comparison of bicubic image SR (×4) methods on random samples from the considered datasets. EdgeSRGAN achieves results that are comparable to state-of-the-art solutions with $\sim 10\%$ of the weights.

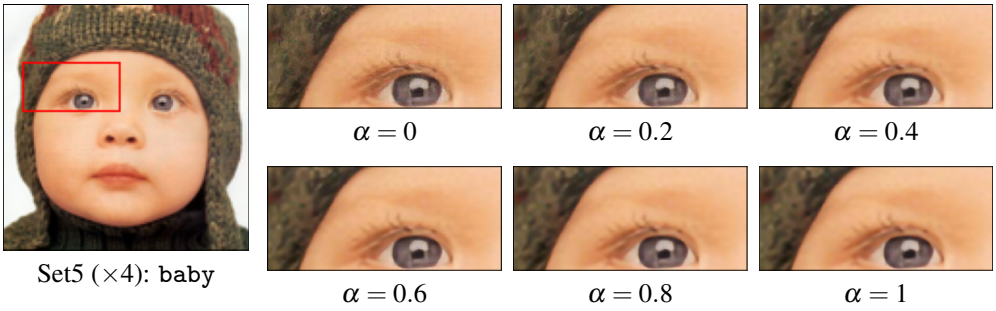


Fig. 12.7 Visual comparison of interpolated EdgeSRGAN for different values of α. Values closer to 1 generate outputs focused on content fidelity, while small values go towards visually pleasing results.

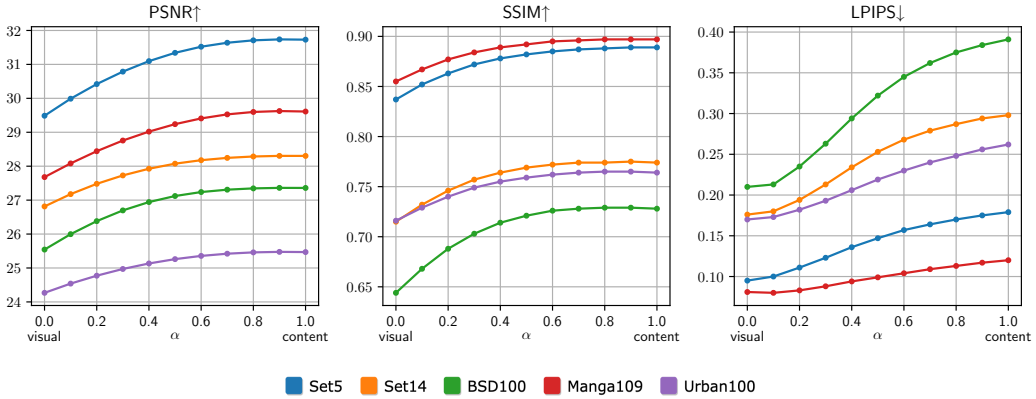


Fig. 12.8 EdgeSRGAN network interpolation results on the benchmark datasets for ×4 upsampling. Changing the network interpolation weight α, it is possible to select the desired trade-off between content-oriented and visual-oriented SR.
↑: higher is better, ↓: lower is better.

a full visual-oriented model and 1 a full content-oriented one. All results refer to the standard EdgeSRGAN model for ×4 upsampling. This procedure effectively shows how it is possible to choose the desired trade-off between content-oriented and visual-oriented SR simply by changing the interpolation weight α. An increase in the weight value causes an improvement of the content-related metrics PSNR and SSIM and a worsening of the perceptual index LPIPS. This behavior holds for all the test datasets, validating the proposed approach. This procedure can be easily carried out in a real-time application and only requires computing the interpolated weights once. Thus, it does not affect any way the inference speed. For an additional visual evaluation, Fig. 12.7 reports the outputs obtained for increasing values of α on a random dataset sample.

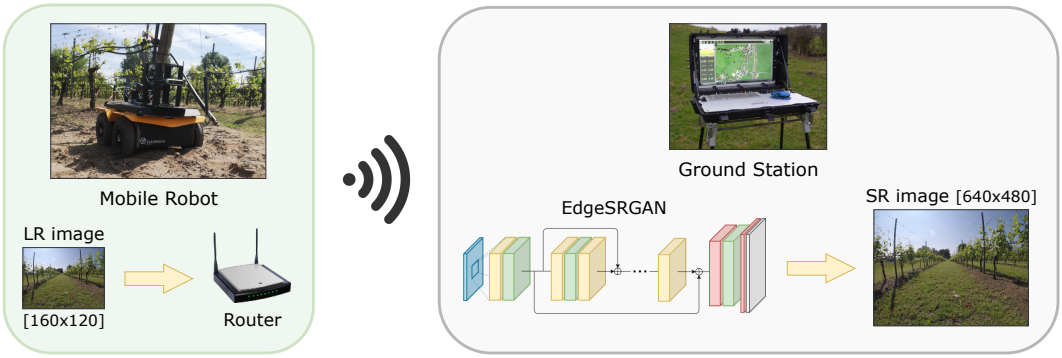


Fig. 12.9 Efficient image transmission system with EdgeSRGAN for mobile robotic applications in outdoor environments.

Network Quantization

To target Edge TPU devices and reach over-real-time inference results, we follow the quantization scheme of Eq. 12.10 for both weights and activations to obtain a full-integer model. Since quantized models must have a fixed input shape, we generate a full-integer network for each input shape of the testing samples. We use the 100 images from the DIV2K validation set as a representative dataset to calibrate the quantization algorithm. We refer to the int8-quantized standard model as EdgeSRGANi8. As for the tiny model, we optimize the distilled network EdgeSRGANi8-tiny-D. Results for the visual-oriented optimized models are shown in Tab. 12.5. Due to the full-integer models' reduced activation and weight, we experience a great increase in inference speed up to over-real-time at the cost of degradation in SR performance. All the proposed quantized models still outperform the bicubic baseline on the perceptual index LPIPS and therefore represent a good option for applications in which really fast inference is needed. A comparison of different models for visual-oriented $\times 4$ upsampling is shown in Fig. 12.1. We consider LPIPS performance on the Set5 dataset compared to framerate.

12.2.4 Application: Image Transmission for Mobile Robotics

Our real-time SISR can provide competitive advantages in a wide variety of practical engineering applications. In this section, we target a specific use case of mobile robotics, proposing our EdgeSRGAN system as an efficient deep learning-based solution for real-time image transmission. Indeed, robot remote control in unknown terrains needs a reliable transmission of visual data at a satisfying framerate, pre-

serving robustness even in bandwidth-degraded conditions. This requirement is particularly relevant for high-speed platforms and UAVs. Dangerous or delicate tasks such as tunnel exploration, inspection, or open space missions all require an available visual stream for human supervision, regardless of the autonomy level of the platform. In the last few years, the robotics community has focused on developing globally shared solutions for robot software and architectures and handling data communications between multiple platforms and devices. ROS2 [373] is the standard operative system for robotic platforms. It is a middleware based on a Data Distribution System (DDS) protocol where application nodes communicate with each other through a topic with a publisher/subscriber mechanism. However, despite the most recent attempts to improve the reliability and efficiency of message and data packet communications between different nodes and platforms, heavier data transmission, such as image streaming, is not yet optimized and reliable.

The typical practical setting used for robot teleoperation and exploration in unknown environments is composed of a ground station and a rover connected to the same wireless network. As shown in Fig. 12.9, we adopted this ground station configuration to test the transmission of images through a ROS2 topic, as should be done in any robotic application to stream what the robot sees or to receive visual data and feed perception and control algorithms for autonomous navigation and mapping. For this experiment, we use both an Intel RealSense D435i camera and a Logitech C920 webcam mounted on a Clearpath Jackal robot, together with a Microhard BulletPlus router for image transmission. The available image resolutions with RealSense cameras, the standard RGBD sensors for visual perception in robotics, are (320×240) and (640×480), whereas the framerate typically varies between 15 and 30 fps.

Despite the absence of strong bandwidth limitations, transmission delays, or partial loss of packets, the maximum resolution and framerate allowed by ROS2 communication are extremely low: we find that at 30 fps, the maximum transmissible resolution for RGB is (120×120) with a bandwidth of 20 Mb/s while reducing the framerate to 5 fps the limit is (320×240). This strict trade-off between framerate and resolution hinders the high-speed motion of a robotic platform in a mission, increasing the risk of collision due to reduced scene supervision. Even selecting *best effort* in the Quality of Service (QoS) settings, which manage the reception of packages through topics, the detected performances are always scarce.

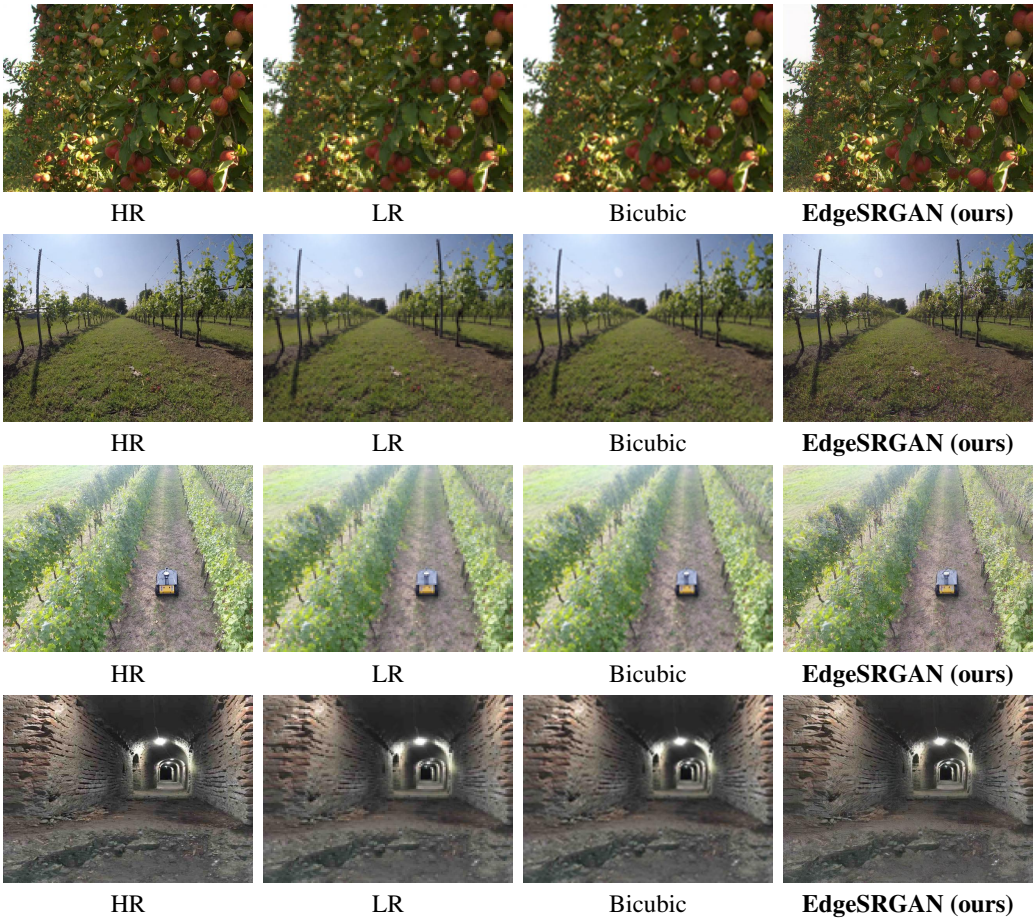


Fig. 12.10 Qualitative demonstration of applying EdgeSRGAN ($\times 4$) on real scenarios (zoom for more detail). From top to bottom: apple monitoring, navigation in vineyards, drone surveillance for autonomous rovers, and tunnel inspection.

Adopting our real-time Super-Resolution system ensures the timely arrival of RGB and depth images via ROS2. Thanks to the fast-inference performance of EdgeSRGAN, we can stream low-resolution images (80×60) at a high framerate (30 fps) and receive a high-resolution output: (320×240) with a x4 image upsampling and (640×480) with a x8 upsampling, showing a clear improvement on standard performance. Our system allows the ground station to access the streaming data through a simple ROS topic. Hence, it provides multiple competitive advantages in robotic teleoperation and autonomous navigation: high-resolution images can be directly exploited by the human operator for remote control. Moreover, they can be used to feed computationally hungry algorithms like sensorimotor agents, visual-odometry, or visual-SLAM, which we may prefer to run on the ground station

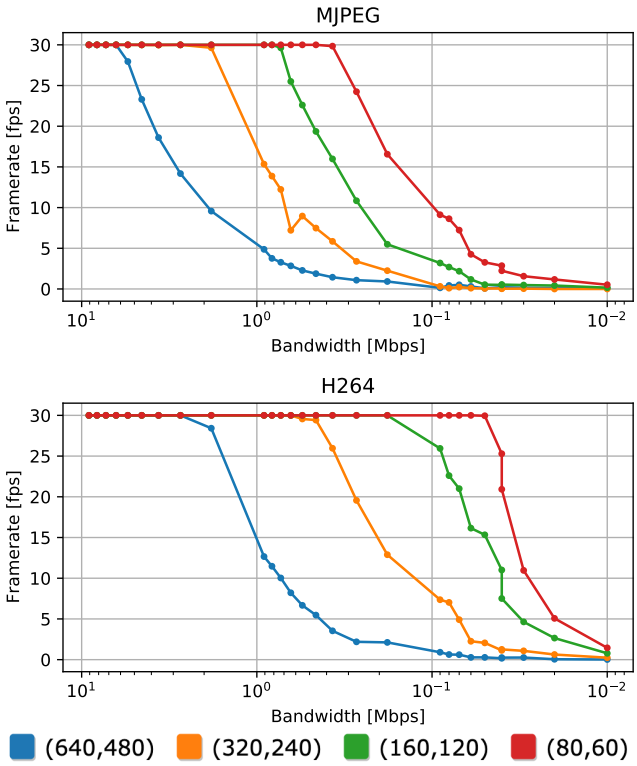


Fig. 12.11 Framerate results vs. bandwidth for video transmission at different input resolutions with MJPEG and H264 compression. Bandwidth is in log scale.

to save the constrained power resources of the robot and significantly boost the autonomy level of the mission. In Fig. 12.10, we report a qualitative comparison to highlight the effectiveness of EdgeSRGAN for real-world robotic scenarios. In particular, we consider apple monitoring, navigation in vineyards, drone surveillance for autonomous rovers, and tunnel inspection.

We also test video transmission performance in a more general framework to reproduce all the potential bandwidth conditions. We use the well-known video streaming library GStreamer[2] to transmit video samples changing the available bandwidth. We progressively reduce the bandwidth from 10 Mbps to 10 kbps using the Wondershaper library[3] and measure the framerate at the receiver side. We use 10 seconds of the standard video sample *smtpe* natively provided by GStreamer *videotestsrc* video source at 30 fps, and we encode it for transmission using MJPEG and H264 video compression standards. The encoding is performed offline to ensure that all the

[2] https://gstreamer.freedesktop.org/
[3] https://github.com/magnific0/wondershaper

available resources are reserved for transmission only. Indeed, most cameras provide hardware-encoded video sources without requiring software compression. To be consistent with the other experiments, we keep using (640×480) and (320×240) as high resolutions and (160×120) and (80×60) as low resolutions. Each experiment is performed 10 times to check the consistency in results. Fig. 12.11 presents the average framerate achieved with different bandwidths. Streaming video directly without any middleware, such as ROS2, ensures a higher transmission performance. However, as expected, streaming high-resolution images is impossible in the case of low bandwidth and the framerate quickly drops to very low values, resulting unsuitable for real-time applications. On the other hand, lower resolutions can be streamed with minimal frame drop, even with lower available bandwidths. H264 compression shows the same behavior as MJPEG but shifts to lower bandwidths. Indeed, H264 is more sophisticated and efficient, as it uses temporal frame correlation in addition to spatial compression. In a practical application with a certain bandwidth constraint, a proper combination of a low-resolution video source and an SR model can be selected to meet the desired framerate requirements on the available platform (CPU or Edge TPU). This mechanism can also be dynamically and automatically activated and deactivated depending on the current connectivity to avoid framerate drops and ensure a smooth image transmission.

Conclusions

Intelligent service robots are at the forefront of technological innovation to support humans in a wide range of activities. What we usually call intelligence refers to a service robot's ability to sense and understand the surrounding world, plan its behavior, and act effectively. AI and ML are key technologies to move robotics intelligence to the next level of autonomy and reach adaptive and reliable performance in unseen or adverse environmental conditions different from structured and controlled lab spaces. This thesis aims to meaningfully integrate these two recent worlds and advance mobile robots towards natural collaborators in our everyday lives. The problems and challenges highlighted in the thesis demonstrate that this future is not far, but there are still many points to be improved to reach complete, reliable solutions. Precision agriculture and indoor domestic assistance are the specific application contexts considered in this work. Part II focuses on the study of autonomous navigation solutions based on both classic and learning methods for social indoor environments. A prototype for a domestic assistance case study is also presented. There are many challenges offered by this type of robotic solution. For example, navigating a crowded environment and considering social rules are unsolved problems. From the platform perspective, executing navigation algorithms and multiple neural models on-board on the constrained resources of the robot is not a trivial task. Privacy issues can also derive from solutions running on cloud-based servers. Part III introduced a DL pipeline for autonomous operation in a row-based crop composed of a position-agnostic local controller and a waypoints generator. The overall solution represents an attempt to reduce the complexity and the costs of an outdoor robotic platform, trying to develop navigation strategies independent from a precise GNSS RTK positioning signal but only relying on an RGB-D camera. The challenges related to deploying DL models in general real contexts are analyzed in Part IV. Generalization of unseen conditions and out-of-distribution data is a

common problem in real-world settings. Nowadays, the development of novel DL methods is changing its paradigm, moving from custom dataset-specific models to huge foundation models to address multiple downstream tasks with a single solution. Among them, ChatGPT [374] for language processing, Segment Anything [375] for computer vision, and novel foundation models for robot grasping [376, 377] and navigation [378] are changing the common sense about AI. On the other hand, Edge-AI is pushing model's optimization techniques to fit low-power embedded devices. The application and methodology presented in Chapter 12 follow this technological direction, which remains the most suitable for robotics on-board execution of ML models. Generalization and optimization are, therefore, two fundamental aspects to move the next generation of AI toward a seamless integration with robotic platforms. This consideration holds for both autonomous navigation and perception.

Future Works

In this dissertation, different methodologies and experiments have presented and discussed in the real of DL and Service Robotics. However, many of them present unsolved challenges and leave space for further development. Considering the indoor social robotics field discussed in Part II, learning and planning techniques can significantly contribute to enhancing autonomous operation in dynamic, crowded environments where social rules also play a crucial role. The solution presented in Chapter 4 is an example of integration between classic and learning methods. These novel hybrid solutions convey different research toward an advanced autonomy characterized by robust and adaptive behaviors. Moreover, the localization of mobile robot can be enhanced with learning approaches as shown in Chapter 5. Future works may consider physics-based learned models for such tasks, extending the study to alternative positioning sensors. In Part III, autonomous robotic operations in vineyards are presented with experiments on the field and different approaches, focusing on the field's rows traversal. Precision agriculture is an active sector for service robotics, and the natural future contributions in this direction would be to further increase the robustness of the position-agnostic algorithms developed to avoid the usage of costly GNSS receivers and, above all, leverage the navigation capabilities to address specific agricultural tasks. Perception and manipulation can be combined on the robot to carry out fruit counting, plant monitoring, harvesting, pruning, and parasite detection. The study of novel backbone architectures and

training paradigms to enhance the generalization and inference performance of DL models will undoubtedly be the focus of research worldwide in the next decade. Knowledge distillation is a promising methodology for performing transfer learning with a twofold objective: training a general and optimized student model. Overall, all the research carried out and presented in this thesis paves the way for further development and investigation of both ML and autonomous navigation algorithms for the next generation of mobile service robots.

References

[1] Fortune Business Insights. Service robotics market size, share | growth drivers, 2023.

[2] Chinchane Amar and Mutreja Sonia. Service robotics market size, trends, industry revenue, 2023.

[3] Cheng Yuan, Bing Xiong, Xiuquan Li, Xiaohan Sang, and Qingzhao Kong. A novel intelligent inspection robot with deep stereo vision for three-dimensional concrete damage detection and quantification. *Structural Health Monitoring*, 21(3):788–802, 2022.

[4] Jorge Pena Queralta, Jussi Taipalmaa, Bilge Can Pullinen, Victor Kathan Sarker, Tuan Nguyen Gia, Hannu Tenhunen, Moncef Gabbouj, Jenni Raitoharju, and Tomi Westerlund. Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision. *Ieee Access*, 8:191617–191643, 2020.

[5] Filippo Cantucci and Rino Falcone. Autonomous critical help by a robotic assistant in the field of cultural heritage: A new challenge for evolving human-robot interaction. *Multimodal Technologies and Interaction*, 6(8):69, 2022.

[6] Fangrui Yin. Inspection robot for submarine pipeline based on machine vision. In *Journal of Physics: Conference Series*, volume 1952, page 022034. IOP Publishing, 2021.

[7] Evangelos Papadopoulos, Farhad Aghili, Ou Ma, and Roberto Lampariello. Robotic manipulation and capture in space: A survey. *Frontiers in Robotics and AI*, 8:686723, 2021.

[8] Jinming Wen, Li He, and Fumin Zhu. Swarm robotics control and communications: Imminent challenges for next generation smart logistics. *IEEE Communications Magazine*, 56(7):102–107, 2018.

[9] Mohd Saiful Azimi Mahmud, Mohamad Shukri Zainal Abidin, Abioye Abiodun Emmanuel, and Hameedah Sahib Hasan. Robotics and automation in agriculture: present and future applications. *Applications of Modelling and Simulation*, 4:130–140, 2020.

[10] Jane Holland, Liz Kingston, Conor McCarthy, Eddie Armstrong, Peter O'Dwyer, Fionn Merz, and Mark McConnell. Service robots in the healthcare sector. *Robotics*, 10(1):47, 2021.

[11] Saeid Nahavandi, Roohallah Alizadehsani, Darius Nahavandi, Shady Mohamed, Navid Mohajer, Mohammad Rokonuzzaman, and Ibrahim Hossain. A comprehensive review on autonomous navigation. *arXiv preprint arXiv:2212.12808*, 2022.

[12] Mauro Martini, Noé Pérez-Higueras, Andrea Ostuni, Marcello Chiaberge, Fernando Caballero, and Luis Merino. Adaptive social force window planner with reinforcement learning. *arXiv preprint arXiv:2404.13678*, 2024.

[13] Alessandro Navone, Mauro Martini, Simone Angarano, and Marcello Chiaberge. Online learning of wheel odometry correction for mobile robots with attention-based neural network. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, pages 1–6, 2023.

[14] Andrea Eirale, Mauro Martini, Luigi Tagliavini, Dario Gandini, Marcello Chiaberge, and Giuseppe Quaglia. Marvin: An innovative omni-directional robotic assistant for domestic environments. *Sensors*, 22(14):5261, 2022.

[15] Andrea Eirale, Mauro Martini, and Marcello Chiaberge. Human-centered navigation and person-following with omnidirectional robot for indoor assistance and monitoring. *Robotics*, 11(5):108, 2022.

[16] Andrea Eirale, Mauro Martini, and Marcello Chiaberge. Rl-dwa omnidirectional motion planning for person following in domestic assistance and monitoring. In *2023 9th International Conference on Automation, Robotics and Applications (ICARA)*, pages 86–90. IEEE, 2023.

[17] Simone Cerrato, Vittorio Mazzia, Francesco Salvetti, Mauro Martini, Simone Angarano, Alessandro Navone, and Marcello Chiaberge. A deep learning driven algorithmic pipeline for autonomous navigation in row-based crops. *arXiv preprint arXiv:2112.03816*, 2021.

[18] Mauro Martini, Andrea Eirale, Brenno Tuberga, Marco Ambrosio, Andrea Ostuni, Francesco Messina, Luigi Mazzara, and Marcello Chiaberge. Enhancing navigation benchmarking and perception data generation for row-based crops in simulation. In *Precision agriculture'23*, pages 451–457. Wageningen Academic, 2023.

[19] Mauro Martini, Simone Cerrato, Francesco Salvetti, Simone Angarano, and Marcello Chiaberge. Position-agnostic autonomous navigation in vineyards with deep reinforcement learning. In *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, pages 477–484, 2022.

[20] Alessandro Navone, Mauro Martini, Andrea Ostuni, Simone Angarano, and Marcello Chiaberge. Autonomous navigation in rows of trees and high crops with deep semantic segmentation. In *2023 European Conference on Mobile Robots (ECMR)*, pages 1–6, 2023.

[21] Alessandro Navone, Mauro Martini, Marco Ambrosio, Andrea Ostuni, Simone Angarano, and Marcello Chiaberge. Gps-free autonomous navigation in cluttered tree rows with deep semantic segmentation. *arXiv preprint arXiv:2404.05338*, 2024.

[22] Francesco Salvetti, Simone Angarano, Mauro Martini, Simone Cerrato, and Marcello Chiaberge. Waypoint generation in row-based crops with deep learning and contrastive clustering. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 203–218. Springer, 2022.

[23] Simone Angarano, Mauro Martini, Francesco Salvetti, Vittorio Mazzia, and Marcello Chiaberge. Back-to-bones: Rediscovering the role of backbones in domain generalization. *arXiv preprint arXiv:2209.01121*, 2022.

[24] Mauro Martini, Vittorio Mazzia, Aleem Khaliq, and Marcello Chiaberge. Domain-adversarial training of self-attention-based networks for land cover classification using multi-temporal sentinel-2 satellite imagery. *Remote Sensing*, 13(13):2564, 2021.

[25] Simone Angarano, Mauro Martini, Alessandro Navone, and Marcello Chiaberge. Domain generalization for crop segmentation with knowledge distillation. *arXiv preprint arXiv:2304.01029*, 2023.

[26] Simone Angarano, Francesco Salvetti, Mauro Martini, and Marcello Chiaberge. Generative adversarial super-resolution at the edge with knowledge distillation. *Engineering Applications of Artificial Intelligence*, 123:106407, 2023.

[27] Mauro Martini, Andrea Eirale, Simone Cerrato, and Marcello Chiaberge. Pic4rl-gym: a ros2 modular framework for robots autonomous navigation with deep reinforcement learning. In *2023 3rd International Conference on Computer, Control and Robotics (ICCCR)*, pages 198–202. IEEE, 2023.

[28] Luca Marchionna, Giulio Pugliese, Mauro Martini, Simone Angarano, Francesco Salvetti, and Marcello Chiaberge. Deep instance segmentation and visual servoing to play jenga with a cost-effective robotic system. *Sensors*, 23(2):752, 2023.

[29] F Ioli, E Bruno, D Calzolari, M Galbiati, A Mannocchi, P Manzoni, M Martini, A Bianchi, A Cina, C De Michele, et al. A replicable open-source multi-camera system for low-cost 4d glacier monitoring. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 48:137–144, 2023.

[30] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11), 2018.

[31] Mahmoud Hassaballah and Ali Ismail Awad. *Deep learning in computer vision: principles and applications*. CRC Press, 2020.

[32] Yoav Goldberg. *Neural network methods for natural language processing*. Springer Nature, 2022.

[33] Yifang Ma, Zhenyu Wang, Hong Yang, and Lin Yang. Artificial intelligence applications in the development of autonomous vehicles: A survey. *IEEE/CAA Journal of Automatica Sinica*, 7(2):315–329, 2020.

[34] Long Chen, Shaobo Lin, Xiankai Lu, Dongpu Cao, Hangbin Wu, Chi Guo, Chun Liu, and Fei-Yue Wang. Deep neural network based vehicle and pedestrian detection for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 22(6):3234–3246, 2021.

[35] Iqbal H Sarker. Deep cybersecurity: a comprehensive overview from neural network and deep learning perspective. *SN Computer Science*, 2(3):154, 2021.

[36] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[37] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[38] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[39] Yann LeCun et al. Generalization and network design strategies. *Connectionism in perspective*, 19(143-155):18, 1989.

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[41] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[42] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[43] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[44] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[45] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[46] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013.

[47] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[48] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[49] Primo.AI. Gradient descent optimization & challenges, 2023.

[50] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[51] Meenal V Narkhede, Prashant P Bartakke, and Mukul S Sutaone. A review on weight initialization strategies for neural networks. *Artificial intelligence review*, 55(1):291–322, 2022.

[52] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[53] Yoshua Bengio and Yves Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Advances in Neural Information Processing Systems*, 16, 2003.

[54] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[55] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[56] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[57] Perkgoz C. Alsaleh A. A space and time efficient convolutional neural network for age group estimation from facial images. *PeerJ Computer Science*, 2023.

[58] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[59] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[60] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[61] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[62] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[63] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[64] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*, 2017.

[65] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[66] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.

[67] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[68] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pages 353–374, 2023.

[69] Raghubir Singh and Sukhpal Singh Gill. Edge ai: a survey. *Internet of Things and Cyber-Physical Systems*, 3:71–92, 2023.

[70] Richard S Sutton, Andrew G Barto, et al. Reinforcement learning. *Journal of Cognitive Neuroscience*, 11(1):126–134, 1999.

[71] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[72] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[73] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

[74] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. *31st International Conference on Machine Learning, ICML 2014*, 1, 06 2014.

[75] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[76] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[77] Josep Aulinas, Yvan Petillot, Joaquim Salvi, and Xavier Lladó. The slam problem: a survey. *Artificial Intelligence Research and Development*, pages 363–371, 2008.

[78] Hamid Taheri and Zhao Chun Xia. Slam; definition and evolution. *Engineering Applications of Artificial Intelligence*, 97:104032, 2021.

[79] Prabin Kumar Panigrahi and Sukant Kishoro Bisoy. Localization strategies for autonomous mobile robots: A review. *Journal of King Saud University-Computer and Information Sciences*, 34(8):6019–6039, 2022.

[80] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. 1995.

[81] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[82] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[83] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[84] Steve Macenski, Tom Moore, David V Lu, Alexey Merzlyakov, and Michael Ferguson. From the desks of ros maintainers: A survey of modern & capable mobile robotics algorithms in the robot operating system 2. *Robotics and Autonomous Systems*, 168:104493, 2023.

[85] Harmish Khambhaita and Rachid Alami. Assessing the social criteria for human-robot collaborative navigation: A comparison of human-aware navigation planners. In *2017 26th IEEE international symposium on robot and human interactive communication (RO-MAN)*, pages 1140–1145. IEEE, 2017.

[86] Noé Pérez-Higueras, Roberto Otero, Fernando Caballero, and Luis Merino. Hunavsim: A ros 2 human navigation simulator for benchmarking human-aware robot navigation. *IEEE Robotics and Automation Letters*, 8(11):7130–7137, September 2023.

[87] Yuxiang Gao and Chien-Ming Huang. Evaluation of socially-aware robot navigation. *Frontiers in Robotics and AI*, 8:721317, 2022.

[88] Christoforos Mavrogiannis, Francesca Baldini, Allan Wang, Dapeng Zhao, Pete Trautman, Aaron Steinfeld, and Jean Oh. Core challenges of social robot navigation: A survey. *ACM Transactions on Human-Robot Interaction*, 12(3):1–39, 2023.

[89] Xuesu Xiao, Bo Liu, Garrett Warnell, and Peter Stone. Motion control for mobile robot navigation using machine learning: a survey. *arXiv preprint arXiv:2011.13112*, 2020.

[90] Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan P How. Socially aware motion planning with deep reinforcement learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1343–1350. IEEE, 2017.

[91] Reuth Mirsky, Xuesu Xiao, Justin Hart, and Peter Stone. Prevention and resolution of conflicts in social navigation–a survey. *arXiv preprint arXiv:2106.12113*, 2021.

[92] Santosh Balajee Banisetty, Vineeth Rajamohan, Fausto Vega, and David Feil-Seifer. A deep learning approach to multi-context socially-aware navigation. In *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*, pages 23–30. IEEE, 2021.

[93] Kai Zhu and Tao Zhang. Deep reinforcement learning based mobile robot navigation: A review. *Tsinghua Science and Technology*, 26(5):674–691, 2021.

[94] Zifan Xu, Xuesu Xiao, Garrett Warnell, Anirudh Nair, and Peter Stone. Machine learning methods for local motion planning: A study of end-to-end vs. parameter learning. In *2021 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 217–222. IEEE, 2021.

[95] Utsav Patel, Nithish Kumar, Adarsh Jagan Sathyamoorthy, and Dinesh Manocha. Dynamically feasible deep reinforcement learning policy for robot navigation in dense mobile crowds, 2020.

[96] Xuesu Xiao, Zizhao Wang, Zifan Xu, Bo Liu, Garrett Warnell, Gauraang Dhamankar, Anirudh Nair, and Peter Stone. Appl: Adaptive planner parameter learning. *Robotics and Autonomous Systems*, 154:104132, 2022.

[97] Zifan Xu, Gauraang Dhamankar, Anirudh Nair, Xuesu Xiao, Garrett Warnell, Bo Liu, Zizhao Wang, and Peter Stone. Applr: Adaptive planner parameter learning from reinforcement. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 6086–6092. IEEE, 2021.

[98] Masato Kobayashi, Hiroka Zushi, Tomoaki Nakamura, and Naoki Motoi. Local path planning: Dynamic window approach with q-learning considering congestion environments for mobile robot. *IEEE Access*, 11:96733–96742, 2023.

[99] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[100] Mehdi Moussaïd, Niriaska Perozo, Simon Garnier, Dirk Helbing, and Guy Theraulaz. The walking behaviour of pedestrian social groups and its impact on crowd dynamics. *PloS one*, 5(4):e10047, 2010.

[101] Mehdi Moussaïd, Dirk Helbing, Simon Garnier, Anders Johansson, Maud Combe, and Guy Theraulaz. Experimental study of the behavioural mechanisms underlying self-organization in human crowds. *Proceedings of the Royal Society B: Biological Sciences*, 276(1668):2755–2762, 2009.

[102] Sherif A. S. Mohamed, Mohammad-Hashem Haghbayan, Tomi Westerlund, Jukka Heikkonen, Hannu Tenhunen, and Juha Plosila. A Survey on Odometry for Autonomous Navigation Systems. *IEEE Access*, 7:97466–97486, 2019.

[103] Ke Wang, Sai Ma, Junlan Chen, Fan Ren, and Jianbo Lu. Approaches, challenges, and applications for deep visual odometry: Toward complicated and emerging areas. *IEEE Transactions on Cognitive and Developmental Systems*, 14(1):35–49, 2020.

[104] Danilo Tardioli, Luis Riazuelo, Domenico Sicignano, Carlos Rizzo, Francisco Lera, José L Villarroel, and Luis Montano. Ground robotics in tunnels: Keys and lessons learned after 10 years of research and experiments. *Journal of Field Robotics*, 36(6):1074–1101, 2019.

[105] Teresa Seco, María T Lázaro, Jesús Espelosín, Luis Montano, and José L Villarroel. Robot localization in tunnels: Combining discrete features in a pose graph framework. *Sensors*, 22(4):1390, 2022.

[106] Christian Tamantini, Francesco Scotto di Luzio, Francesca Cordella, Giuseppe Pascarella, Felice Eugenio Agro, and Loredana Zollo. A robotic healthcare assistant for covid-19 emergency: A proposed solution for logistics and disinfection in a hospital environment. *IEEE Robotics & Automation Magazine*, 28(1):71–81, 2021.

[107] Surbhi Gupta, R Sangeeta, Ravi Shankar Mishra, Gaurav Singal, Tapas Badal, and Deepak Garg. Corridor segmentation for automatic robot navigation in indoor environment using edge devices. *Computer Networks*, 178:107374, 2020.

[108] Beatriz Pérez-Sánchez, Oscar Fontenla-Romero, and Bertha Guijarro-Berdiñas. A review of adaptive online learning for artificial neural networks. *Artificial Intelligence Review*, 49(2):281–299, February 2018.

[109] Haoming Xu and John James Collins. Estimating the Odometry Error of a Mobile Robot by Neural Networks. In *2009 International Conference on Machine Learning and Applications*, pages 378–385, Miami, FL, USA, December 2009. IEEE.

[110] Huijun Li, Ying Mao, Wei You, Bin Ye, and Xinyi Zhou. A neural network approach to indoor mobile robot localization. pages 66–69, October 2020. ISSN: 2473-3636.

[111] Uche Onyekpe, Vasile Palade, Stratis Kanarachos, and Stavros-Richard G. Christopoulos. Learning Uncertainties in Wheel Odometry for Vehicular Localisation in GNSS Deprived Environments. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 741–746, Miami, FL, USA, December 2020. IEEE.

[112] Changhao Chen, Chris Xiaoxuan Lu, Johan Wahlström, Andrew Markham, and Niki Trigoni. Deep Neural Network Based Inertial Odometry Using Low-Cost Inertial Measurement Units. *IEEE Transactions on Mobile Computing*, 20(4):1351–1364, April 2021. Conference Name: IEEE Transactions on Mobile Computing.

[113] Siyu Chen, Yu Zhu, Xiaoguang Niu, and Zhiyong Hu. Improved Window Segmentation for Deep Learning Based Inertial Odometry. In *2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC)*, pages 1–7, Austin, TX, USA, November 2020. IEEE.

[114] Arno Solin, Santiago Cortes, Esa Rahtu, and Juho Kannala. Inertial Odometry on Handheld Smartphones, June 2018. arXiv:1703.00154 [cs, stat].

[115] Changhao Chen, Xiaoxuan Lu, Andrew Markham, and Niki Trigoni. IONet: Learning to Cure the Curse of Drift in Inertial Odometry, January 2018. arXiv:1802.02209 [cs].

[116] J.-S. Botero Valencia, M. Rico Garcia, and J.-P. Villegas Ceballos. A simple method to estimate the trajectory of a low cost mobile robotic platform using an IMU. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 11(4):823–828, November 2017.

[117] Martin Brossard, Axel Barrau, and Silvere Bonnabel. AI-IMU Dead-Reckoning. *IEEE Transactions on Intelligent Vehicles*, 5(4):585–595, December 2020.

[118] Mahdi Abolfazli Esfahani, Han Wang, Keyu Wu, and Shenghai Yuan. AbolDeepIO: A Novel Deep Inertial Odometry Network for Autonomous Vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 21(5):1941–1950, May 2020.

[119] Quentin Arnaud Dugne-Hennequin, Hideaki Uchiyama, and Joao Paulo Silva Do Monte Lima. Understanding the Behavior of Data-Driven Inertial Odometry With Kinematics-Mimicking Deep Neural Network. *IEEE Access*, 9:36589–36619, 2021.

[120] Jinglin Shen, David Tick, and Nicholas Gans. Localization through fusion of discrete and continuous epipolar geometry with wheel and IMU odometry. In *Proceedings of the 2011 American Control Conference*, pages 1292–1298, San Francisco, CA, June 2011. IEEE.

[121] Zhihuang Zhang, Jintao Zhao, Changyao Huang, and Liang Li. Learning End-to-End Inertial-Wheel Odometry for Vehicle Ego-Motion Estimation. In *2021 5th CAA International Conference on Vehicular Control and Intelligence (CVCI)*, pages 1–6, Tianjin, China, October 2021. IEEE.

[122] Kansu Oguz Canbek, Hulya Yalcin, and Eray A. Baran. Drift compensation of a holonomic mobile robot using recurrent neural networks. *Intelligent Service Robotics*, 15(3):399–409, July 2022.

[123] Simone Angarano, Vittorio Mazzia, Francesco Salvetti, Giovanni Fantin, and Marcello Chiaberge. Robust Ultra-wideband Range Error Mitigation with Deep Learning at the Edge. *Engineering Applications of Artificial Intelligence*, 102:104278, June 2021. arXiv:2011.14684 [cs, eess].

[124] Simone Angarano, Francesco Salvetti, Vittorio Mazzia, Giovanni Fantin, Dario Gandini, and Marcello Chiaberge. Ultra-low-power range error mitigation for ultra-wideband precise localization. In *Intelligent Computing: Proceedings of the 2022 Computing Conference, Volume 2*, pages 814–824. Springer, 2022.

[125] Christoph Käding, Erik Rodner, Alexander Freytag, and Joachim Denzler. Fine-Tuning Deep Neural Networks in Continuous Learning Scenarios. In Chu-Song Chen, Jiwen Lu, and Kai-Kuang Ma, editors, *Computer Vision – ACCV 2016 Workshops*, volume 10118, pages 588–605. Springer International Publishing, Cham, 2017. Series Title: Lecture Notes in Computer Science.

[126] Valentin Peretroukhin and Jonathan Kelly. DPC-Net: Deep Pose Correction for Visual Localization. *IEEE Robotics and Automation Letters*, 3(3):2424–2431, July 2018. arXiv:1709.03128 [cs].

[127] Ester Martinez-Martin and Angel P del Pobil. Personal robot assistants for elderly care: an overview. *Personal assistants: Emerging computational technologies*, pages 77–91, 2018.

[128] Alessandro Vercelli, Innocenzo Rainero, Ludovico Ciferri, Marina Boido, and Fabrizio Pirri. Robots in elderly care. *DigitCult-Scientific Journal on Digital Cultures*, 2(2):37–50, 2018.

[129] United Nations. Shifting demographics.

[130] Laurie L Novak, Juliann G Sebastian, and Tracy A Lustig. The world has changed: Emerging challenges for health care research to reduce social isolation and loneliness related to covid-19. *NAM perspectives*, 2020, 2020.

[131] Yang Shen, Dejun Guo, Fei Long, Luis A Mateos, Houzhu Ding, Zhen Xiu, Randall B Hellman, Adam King, Shixun Chen, Chengkun Zhang, et al. Robots under covid-19 pandemic: A comprehensive survey. *Ieee Access*, 9:1590–1615, 2020.

[132] Jordan Abdi, Ahmed Al-Hindawi, Tiffany Ng, and Marcela P Vizcaychipi. Scoping review on the use of socially assistive robot technology in elderly care. *BMJ open*, 8(2):e018815, 2018.

[133] David Gouaillier, Vincent Hugel, Pierre Blazevic, Chris Kilner, Jérôme Monceaux, Pascal Lafourcade, Brice Marnier, Julien Serre, and Bruno Maisonnier. Mechatronic design of nao humanoid. In *2009 IEEE International Conference on Robotics and Automation*, pages 769–774. IEEE, 2009.

[134] Masahiro Fujita. Aibo: Toward the era of digital creatures. *The International Journal of Robotics Research*, 20(10):781–794, 2001.

[135] Selma Šabanović, Casey C Bennett, Wan-Ling Chang, and Lesa Huber. Paro robot affects diverse interaction modalities in group sensory therapy for older adults with dementia. In *2013 IEEE 13th international conference on rehabilitation robotics (ICORR)*, pages 1–6. IEEE, 2013.

[136] Susel Góngora Alonso, Sofiane Hamrioui, Isabel de la Torre Díez, Eduardo Motta Cruz, Miguel López-Coronado, and Manuel Franco. Social robots for people with aging and dementia: a systematic review of literature. *Telemedicine and e-Health*, 25(7):533–540, 2019.

[137] Norina Gasteiger, Kate Loveys, Mikaela Law, and Elizabeth Broadbent. Friends from the future: A scoping review of research into robots and computer agents to combat loneliness in older people. *Clinical interventions in aging*, 16:941, 2021.

[138] Akihito Yatsuda, Toshiyuki Haramaki, and Hiroaki Nishino. A study on robot motions inducing awareness for elderly care. In *2018 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, pages 1–2, 2018.

[139] Zaid A Mundher and Jiaofei Zhong. A real-time fall detection system in elderly care using mobile robot and kinect sensor. *International Journal of Materials, Mechanics and Manufacturing*, 2(2):133–138, 2014.

[140] Gonçalo Marques, Ivan Miguel Pires, Nuno Miranda, and Rui Pitarma. Air quality monitoring using assistive robots for ambient assisted living and enhanced living environments through internet of things. *Electronics*, 8(12):1375, 2019.

[141] Jagriti Saini, Maitreyee Dutta, and Goncalo Marques. Sensors for indoor air quality monitoring and assessment through internet of things: a systematic review. *Environmental Monitoring and Assessment*, 193(2):1–32, 2021.

[142] Dragos Mocrii, Yuxiang Chen, and Petr Musilek. Iot-based smart homes: A review of system architecture, software, communications, privacy and security. *Internet of Things*, 1:81–98, 2018.

[143] Juan Angel Gonzalez-Aguirre, Ricardo Osorio-Oliveros, Karen L Rodríguez-Hernández, Javier Lizárraga-Iturralde, Ruben Morales Menendez, Ricardo A Ramírez-Mendoza, Mauricio Adolfo Ramírez-Moreno, and Jorge de Jesus Lozoya-Santos. Service robots: Trends and technology. *Applied Sciences*, 11(22):10702, 2021.

[144] Ioan Doroftei, Victor Grosu, and Veaceslav Spinu. *Omnidirectional mobile robot-design and implementation*. INTECH Open Access Publisher, 2007.

[145] Md Jahidul Islam, Jungseok Hong, and Junaed Sattar. Person-following by autonomous robots: A categorical overview. *The International Journal of Robotics Research*, 38(14):1581–1618, 2019.

[146] Shanee S Honig, Tal Oron-Gilad, Hanan Zaichyk, Vardit Sarne-Fleischmann, Samuel Olatunji, and Yael Edan. Toward socially aware person-following robots. *IEEE Transactions on Cognitive and Developmental Systems*, 10(4):936–954, 2018.

[147] Jibo Robot Website. https://jibo.com/, 2017.

[148] David Fischinger, Peter Einramhof, Konstantinos Papoutsakis, Walter Wohlkinger, Peter Mayer, Paul Panek, Stefan Hofmann, Tobias Koertner, Astrid Weiss, Antonis Argyros, et al. Hobbit, a care robot supporting independent living at home: First prototype and lessons learned. *Robotics and Autonomous Systems*, 75:60–78, 2016.

[149] Kunimatsu Hashimoto, Fuminori Saito, Takashi Yamamoto, and Koichi Ikeda. A field study of the human support robot in the home environment. In *2013 IEEE Workshop on Advanced Robotics and its Social Impacts*, pages 143–150. IEEE, 2013.

[150] Tetsuya Tanioka. Nursing and rehabilitative care of the elderly using humanoid robots. *The Journal of Medical Investigation*, 66(1.2):19–23, 2019.

[151] PAL Robotics. Tiago.

[152] William K Juel, Frederik Haarslev, Eduardo R Ramirez, Emanuela Marchetti, Kerstin Fischer, Danish Shaikh, Poramate Manoonpong, Christian Hauch, Leon Bodenhagen, and Norbert Krüger. Smooth robot: Design for a novel modular welfare robot. *Journal of Intelligent & Robotic Systems*, 98(1):19–37, 2020.

[153] Md Abdullah Al Mamun, Mohammad Tariq Nasir, and Ahmad Khayyat. Embedded system for motion control of an omnidirectional mobile robot. *IEEE Access*, 6:6722–6739, 2018.

[154] Paulo José Costa, Nuno Moreira, Daniel Campos, José Gonçalves, José Lima, and Pedro Luís Costa. Localization and navigation of an omnidirectional mobile robot: the robot@ factory case study. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, 11(1):1–9, 2016.

[155] Jun Qian, Bin Zi, Daoming Wang, Yangang Ma, and Dan Zhang. The design and development of an omni-directional mobile robot oriented to an intelligent manufacturing system. *Sensors*, 17(9):2073, 2017.

[156] Amazon. Introducing amazon astro – household robot for home monitoring, with alexa, 2021.

[157] Nexus 4wd mecanum wheel mobile robot on nexus official site. https://www.nexusrobot.com/product/4wd-mecanum-wheel-mobile-arduino-robotics-car-10011.html.

[158] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[159] The robot operating system official site. https://www.ros.org/.

[160] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. Exploring the performance of ros2. 2016.

[161] Changes between ros2 and ros1. https://design.ros2.org/articles/changes.html.

[162] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[163] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[164] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: realtime multi-person 2d pose estimation using part affinity fields. *IEEE transactions on pattern analysis and machine intelligence*, 43(1):172–186, 2019.

[165] George Papandreou, Tyler Zhu, Liang-Chieh Chen, Spyros Gidaris, Jonathan Tompson, and Kevin Murphy. Personlab: Person pose estimation and instance segmentation with a bottom-up, part-based, geometric embedding model. *CoRR*, abs/1803.08225, 2018.

[166] Alex Bewley, ZongYuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. *CoRR*, abs/1602.00763, 2016.

[167] Gabriel Skantze. Turn-taking in conversational systems and human-robot interaction: a review. *Computer Speech & Language*, 67:101178, 2021.

[168] Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*, 2019.

[169] Jawid Ahmad Baktash and Mursal Dawodi. Gpt-4: A review on advancements and opportunities in natural language processing. *arXiv preprint arXiv:2305.03195*, 2023.

[170] Axel Berg, Mark O'Connor, and Miguel Tairum Cruz. Keyword transformer: A self-attention model for keyword spotting. *arXiv preprint arXiv:2104.00769*, 2021.

[171] Alexey Andreev and Kirill Chuvilin. Speech recognition for mobile linux distrubitions in the case of aurora os. In *2021 29th Conference of Open Innovations Association (FRUCT)*, pages 14–21. IEEE, 2021.

[172] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

[173] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Céspedes, Steve Yuan, Chris Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

[174] Yinfei Yang, Daniel Cer, Amin Ahmad, Mandy Guo, Jax Law, Noah Constant, Gustavo Hernandez Abrego, Steve Yuan, Chris Tar, Yun-Hsuan Sung, et al. Multilingual universal sentence encoder for semantic retrieval. *arXiv preprint arXiv:1907.04307*, 2019.

[175] Zhaoyu Zhai, José Fernán Martínez, Victoria Beltran, and Néstor Lucas Martínez. Decision support systems for agriculture 4.0: Survey and challenges. *Computers and Electronics in Agriculture*, 170:105256, 2020.

[176] C Wouter Bac, Eldert J van Henten, Jochen Hemming, and Yael Edan. Harvesting robots for high-value crops: State-of-the-art review and challenges ahead. *Journal of Field Robotics*, 31(6):888–911, 2014.

[177] Ron Berenstein, Ohad Ben Shahar, Amir Shapiro, and Yael Edan. Grape clusters and foliage detection algorithms for autonomous selective vineyard sprayer. *Intelligent Service Robotics*, 3(4):233–243, 2010.

[178] Deepak Deshmukh, Dilip Kumar Pratihar, Alok Kanti Deb, Hena Ray, and Nabarun Bhattacharyya. Design and development of intelligent pesticide spraying system for agricultural robot. In *International Conference on Hybrid Intelligent Systems*, pages 157–170. Springer, 2020.

[179] GuoSheng Zhang, TongYu Xu, YouWen Tian, Han Xu, JiaYu Song, and Yubin Lan. Assessment of rice leaf blast severity using hyperspectral imaging during late vegetative growth. *Australasian Plant Pathology*, 49:571–578, 2020.

[180] Aijing Feng, Jianfeng Zhou, Earl D Vories, Kenneth A Sudduth, and Meina Zhang. Yield estimation in cotton using uav-based multi-sensor imagery. *Biosystems Engineering*, 193:101–114, 2020.

[181] Jayantha Katupitiya, Ray Eaton, and Tahir Yaqub. Systems engineering approach to agricultural automation: new developments. In *2007 1st Annual IEEE Systems Conference*, pages 1–7. IEEE, 2007.

[182] David Kohanbash, Abhinav Valada, and George Kantor. Irrigation control methods for wireless sensor network. In *2012 Dallas, Texas, July 29-August 1, 2012*, page 1. American Society of Agricultural and Biological Engineers, 2012.

[183] Daniel Bigelow and Allison Borchers. Major uses of land in the united states, 2012. *Economic Information Bulletin Number 178*, (1476-2017-4340):69, 2017.

[184] Benoît Thuilot, Christophe Cariou, Lionel Cordesses, and Philippe Martinet. Automatic guidance of a farm tractor along curved paths, using a unique cp-dgps. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, volume 2, pages 674–679. IEEE, 2001.

[185] O. Ly, H. Gimbert, G. Passault, and G. Baron. A fully autonomous robot for putting posts for trellising vineyard with centimetric accuracy. In *2015 IEEE International Conference on Autonomous Robot Systems and Competitions*, pages 44–49, 2015.

[186] Stewart J Moorehead, Carl K Wellington, Brian J Gilmore, and Carlos Vallespi. Automating orchards: A system of autonomous tractors for orchard maintenance. In *Proceedings of the IEEE International Conference of Intelligent Robots and Systems, Workshop on Agricultural Robotics*, 2012.

[187] Flavio Callegati, Alessandro Samorì, Roberto Tazzari, Nicola Mimmo, and Lorenzo Marconi. Autonomous tracked agricultural ugv configuration and navigation experimental results. In *Workshop on Small UAVs for Precision Agriculture*, 2018.

[188] Sam Marden and Mark Whitty. Gps-free localisation and navigation of an unmanned ground vehicle for yield forecasting in a vineyard. In *Recent Advances in Agricultural Robotics, International workshop collocated with the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, 2014.

[189] Md Shaha Nur Kabir, Ming-Zhang Song, Nam-Seok Sung, Sun-Ok Chung, Yong-Joo Kim, Noboru Noguchi, and Soon-Jung Hong. Performance comparison of single and multi-gnss receivers under agricultural fields in korea. *Engineering in agriculture, environment and food*, 9(1):27–35, 2016.

[190] André Aguiar, Filipe Santos, Luís Santos, and Armando Sousa. Monocular visual odometry using fisheye lens cameras. In *EPIA Conference on Artificial Intelligence*, pages 319–330. Springer, 2019.

[191] Shahzad Zaman, Lorenzo Comba, Alessandro Biglia, Davide Ricauda Aimonino, Paolo Barge, and Paolo Gay. Cost-effective visual odometry system for vehicle motion control in agricultural environments. *Computers and Electronics in Agriculture*, 162:82–94, 2019.

[192] Igor Nevliudov, Sergiy Novoselov, Oksana Sychova, and Serhii Tesliuk. Development of the architecture of the base platform agricultural robot for determining the trajectory using the method of visual odometry. In *2021 IEEE XVIIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, pages 64–68. IEEE, 2021.

[193] Yue Ma, Wenqiang Zhang, Waqar S Qureshi, Chao Gao, Chunlong Zhang, and Wei Li. Autonomous navigation for a wolfberry picking robot using visual cues and fuzzy control. *Information Processing in Agriculture*, 8(1):15–26, 2021.

[194] Vittorio Mazzia, Aleem Khaliq, Francesco Salvetti, and Marcello Chiaberge. Real-time apple detection system using embedded systems with hardware accelerators: An edge ai application. *IEEE Access*, 8:9102–9114, 2020.

[195] Helizani Couto Bazame, José Paulo Molin, Daniel Althoff, and Maurício Martello. Detection, classification, and mapping of coffee fruits during harvest with computer vision. *Computers and Electronics in Agriculture*, 183:106066, 2021.

[196] Maryam Rahnemoonfar and Clay Sheppard. Deep count: fruit counting based on deep simulated learning. *Sensors*, 17(4):905, 2017.

[197] Jinfan Xu, Jie Yang, Xingguo Xiong, Haifeng Li, Jingfeng Huang, KC Ting, Yibin Ying, and Tao Lin. Towards interpreting multi-temporal deep learning models in crop mapping. *Remote Sensing of Environment*, 264:112599, 2021.

[198] Diego Aghi, Vittorio Mazzia, and Marcello Chiaberge. Local motion planner for autonomous navigation in vineyards with a rgb-d camera-based algorithm and deep learning synergy. *Machines*, 8(2):27, 2020.

[199] Diego Aghi, Simone Cerrato, Vittorio Mazzia, and Marcello Chiaberge. Deep semantic segmentation at the edge for autonomous navigation in vineyard rows. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3421–3428. IEEE, 2021.

[200] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[201] Mostafa Sharifi and XiaoQi Chen. A novel vision based row guidance approach for navigation of agricultural mobile robots in orchards. In *2015 6th International Conference on Automation, Robotics and Applications (ICARA)*, pages 251–255, 2015.

[202] Josiah Radcliffe, Julie Cox, and Duke M. Bulanon. Machine vision for orchard navigation. *Computers in Industry*, 98:165–171, 2018.

[203] Peichen Huang, Lixue Zhu, Zhigang Zhang, and Chenyu Yang. An end-to-end learning-based row-following system for an agricultural robot in structured apple orchards. *Mathematical Problems in Engineering*, 2021, 2021.

[204] Diego Aghi, Vittorio Mazzia, and Marcello Chiaberge. Autonomous navigation in vineyards with deep learning at the edge. In *International Conference on Robotics in Alpe-Adria Danube Region*, pages 479–486. Springer, 2020.

[205] Md. Shaha Nur Kabir, Ming-Zhang Song, Nam-Seok Sung, Sun-Ok Chung, Yong-Joo Kim, Noboru Noguchi, and Soon-Jung Hong. Performance comparison of single and multi-gnss receivers under agricultural fields in korea. *Engineering in Agriculture, Environment and Food*, 9(1):27–35, 2016.

[206] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

[207] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018.

[208] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, Los Alamitos, CA, USA, jun 2018. IEEE Computer Society.

[209] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *ArXiv*, abs/1706.05587, 2017.

[210] Szilárd Aradi. Survey of deep reinforcement learning for motion planning of autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[211] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[212] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10674–10681, 2021.

[213] Giuseppe Riggio, Cesare Fantuzzi, and Cristian Secchi. A low-cost navigation strategy for yield estimation in vineyards. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2200–2205. IEEE, 2018.

[214] Pietro Astolfi, Alessandro Gabrielli, Luca Bascetta, and Matteo Matteucci. Vineyard autonomous navigation in the echord++ grape experiment. *IFAC-PapersOnLine*, 51(11):704–709, 2018.

[215] Oscar C Barawid Jr, Akira Mizushima, Kazunobu Ishii, and Noboru Noguchi. Development of an autonomous navigation system using a two-dimensional laser scanner in an orchard application. *Biosystems Engineering*, 96(2):139–149, 2007.

[216] Jurgen Zoto, Maria Angela Musci, Aleem Khaliq, Marcello Chiaberge, and Irene Aicardi. Automatic path planning for unmanned ground vehicle using uav imagery. In *International Conference on Robotics in Alpe-Adria Danube Region*, pages 223–230. Springer, 2019.

[217] Ivan Vidović and Rudolf Scitovski. Center-based clustering for line detection and application to crop rows detection. *Computers and electronics in agriculture*, 109:212–220, 2014.

[218] Lorenzo Comba, Alessandro Biglia, Davide Ricauda Aimonino, and Paolo Gay. Unsupervised detection of vineyards by 3d point-cloud uav photogrammetry for precision agriculture. *Computers and Electronics in Agriculture*, 155:84–95, 2018.

[219] Vittorio Mazzia, Francesco Salvetti, Diego Aghi, and Marcello Chiaberge. Deepway: a deep learning waypoint estimator for global path generation. *Computers and Electronics in Agriculture*, 184:106091, 2021.

[220] T Lei, C Luo, GE Jan, and Z Bi. Deep learning-based complete coverage path planning with re-joint and obstacle fusion paradigm. *Front. Robot. AI 9: 843816. doi: 10.3389/frobt*, 2022.

[221] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[222] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[223] Michael Tschannen, Olivier Bachem, and Mario Lucic. Recent advances in autoencoder-based representation learning. *arXiv preprint arXiv:1812.05069*, 2018.

[224] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, pages 281–297. Oakland, CA, USA, 1967.

[225] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[226] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3733–3742, 2018.

[227] Aaron Van den Oord, Yazhe Li, Oriol Vinyals, et al. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2(3):4, 2018.

[228] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[229] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[230] Simone Cerrato, Diego Aghi, Vittorio Mazzia, Francesco Salvetti, and Marcello Chiaberge. An adaptive row crops path generator with deep learning synergy. In *2021 6th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, pages 6–12. IEEE, 2021.

[231] Leslie Valiant. *Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World*. Basic Books (AZ), 2013.

[232] Dengxin Dai and Luc Van Gool. Dark model adaptation: Semantic image segmentation from daytime to nighttime. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3819–3824. IEEE, 2018.

[233] Georg Volk, Stefan Müller, Alexander von Bernuth, Dennis Hospach, and Oliver Bringmann. Towards robust cnn-based object detection through augmentation with synthetic rain variations. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 285–292. IEEE, 2019.

[234] Gilles Blanchard, Gyemin Lee, and Clayton Scott. Generalizing from several related classification tasks to a new unlabeled sample. *Advances in neural information processing systems*, 24:2178–2186, 2011.

[235] Krikamol Muandet, David Balduzzi, and Bernhard Schölkopf. Domain generalization via invariant feature representation. In *International Conference on Machine Learning*, pages 10–18. PMLR, 2013.

[236] Shiv Shankar, Vihari Piratla, Soumen Chakrabarti, Siddhartha Chaudhuri, Preethi Jyothi, and Sunita Sarawagi. Generalizing across domains via cross-gradient training. In *International Conference on Learning Representations*, 2018.

[237] Riccardo Volpi, Hongseok Namkoong, Ozan Sener, John C Duchi, Vittorio Murino, and Silvio Savarese. Generalizing to unseen domains via adversarial data augmentation. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[238] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030, 2016.

[239] Baochen Sun and Kate Saenko. Deep CORAL: Correlation alignment for deep domain adaptation. In *European conference on computer vision*, pages 443–450. Springer, 2016.

[240] Saeid Motiian, Marco Piccirilli, Donald A Adjeroh, and Gianfranco Doretto. Unified deep supervised domain adaptation and generalization. In *Proceedings of the IEEE international conference on computer vision*, pages 5715–5725, 2017.

[241] Haoliang Li, Sinno Jialin Pan, Shiqi Wang, and Alex C Kot. Domain generalization with adversarial feature learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5400–5409, 2018.

[242] Sentao Chen, Lei Wang, Zijie Hong, and Xiaowei Yang. Domain generalization by joint-product distribution alignment. *Pattern Recognition*, 134:109086, 2023.

[243] Yogesh Balaji, Swami Sankaranarayanan, and Rama Chellappa. Metareg: Towards domain generalization using meta-regularization. *Advances in Neural Information Processing Systems*, 31:998–1008, 2018.

[244] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Learning to generalize: Meta-learning for domain generalization. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[245] Marvin Mengxin Zhang, Henrik Marklund, Nikita Dhawan, Abhishek Gupta, Sergey Levine, and Chelsea Finn. Adaptive risk minimization: A meta-learning approach for tackling group shift. In *International Conference on Learning Representations*, 2020.

[246] Silvia Bucci, Antonio D'Innocente, Yujun Liao, Fabio M. Carlucci, Barbara Caputo, and Tatiana Tommasi. Self-supervised learning across domains. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5516–5528, 2022.

[247] Isabela Albuquerque, Nikhil Naik, Junnan Li, Nitish Keskar, and Richard Socher. Improving out-of-distribution generalization via multi-task self-supervised pretraining. *arXiv preprint arXiv:2003.13525*, 2020.

[248] Mohammad Mahfujur Rahman, Clinton Fookes, Mahsa Baktashmotlagh, and Sridha Sridharan. Correlation-aware adversarial domain adaptation and generalization. *Pattern Recognition*, 100:107124, 2020.

[249] Shiori Sagawa, Pang Wei Koh, Tatsunori B. Hashimoto, and Percy Liang. Distributionally robust neural networks. In *International Conference on Learning Representations*, 2020.

[250] Zeyi Huang, Haohan Wang, Eric P Xing, and Dong Huang. Self-challenging improves cross-domain generalization. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 124–140. Springer, 2020.

[251] Soroosh Shahtalebi, Jean-Christophe Gagnon-Audet, Touraj Laleh, Mojtaba Faramarzi, Kartik Ahuja, and Irina Rish. Sand-mask: An enhanced gradient masking strategy for the discovery of invariances in domain generalization. *arXiv preprint arXiv:2106.02266*, 2021.

[252] Daehee Kim, Youngjun Yoo, Seunghyun Park, Jinkyu Kim, and Jaekoo Lee. Selfreg: Self-supervised contrastive regularization for domain generalization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9619–9628, 2021.

[253] Mattia Segu, Alessio Tonioni, and Federico Tombari. Batch normalization embeddings for deep domain generalization. *Pattern Recognition*, 135:109115, 2023.

[254] Ishaan Gulrajani and David Lopez-Paz. In search of lost domain generalization. In *International Conference on Learning Representations*. Computer Vision Foundation, 2021.

[255] Vladimir N Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

[256] Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. Domain generalization: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[257] Ross Wightman, Hugo Touvron, and Herve Jegou. Resnet strikes back: An improved training procedure in timm. In *NeurIPS 2021 Workshop on ImageNet: Past, Present, and Future*, 2021.

[258] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 2019.

[259] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10347–10357. PMLR, 2021.

[260] Stéphane D'Ascoli, Hugo Touvron, Matthew L Leavitt, Ari S Morcos, Giulio Biroli, and Levent Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2286–2296. PMLR, 18–24 Jul 2021.

[261] Benjamin Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Hervé Jégou, and Matthijs Douze. Levit: a vision transformer in convnet's clothing for faster inference. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12259–12269, 2021.

[262] Chen Fang, Ye Xu, and Daniel N. Rockmore. Unbiased metric learning: On the utilization of multiple datasets and web images for softening bias. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2013.

[263] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M. Hospedales. Deeper, broader and artier domain generalization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[264] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[265] Sara Beery, Grant Van Horn, and Pietro Perona. Recognition in terra incognita. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 456–473, 2018.

[266] Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. Moment matching for multi-source domain adaptation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Computer Vision Foundation, 2019.

[267] Muhammad Ghifary, W Bastiaan Kleijn, Mengjie Zhang, and David Balduzzi. Domain generalization for object recognition with multi-task autoencoders. In *Proceedings of the IEEE international conference on computer vision*, pages 2551–2559, 2015.

[268] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.

[269] Shen Yan, Huan Song, Nanxiang Li, Lincan Zou, and Liu Ren. Improve unsupervised domain adaptation with mixup training. *arXiv preprint arXiv:2001.00677*, 2020.

[270] Mathieu Chevalley, Charlotte Bunne, Andreas Krause, and Stefan Bauer. Invariant causal mechanisms through distribution matching. *arXiv preprint arXiv:2206.11646*, 2022.

[271] Yangjun Ruan, Yann Dubois, and Chris J. Maddison. Optimal representations for covariate shift. In *International Conference on Learning Representations*, 2022.

[272] Rang Meng, Xianfeng Li, Weijie Chen, Shicai Yang, Jie Song, Xinchao Wang, Lei Zhang, Mingli Song, Di Xie, and Shiliang Pu. Attention diversification for domain generalization. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIV*, pages 322–340. Springer, 2022.

[273] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11), 2008.

[274] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[275] Chengjuan Ren, Dae-Kyoo Kim, and Dongwon Jeong. A survey of deep learning in agriculture: techniques and their applications. *Journal of Information Processing Systems*, 16(5):1015–1033, 2020.

[276] Zifei Luo, Wenzhu Yang, Yunfeng Yuan, Ruru Gou, and Xiaonan Li. Semantic segmentation of agricultural images: A survey. *Information Processing in Agriculture*, 2023.

[277] Somnath Mukhopadhyay, Munti Paul, Ramen Pal, and Debashis De. Tea leaf disease detection using multi-objective image segmentation. *Multimedia Tools and Applications*, 80:753–771, 2021.

[278] Hongbo Yuan, Jiajun Zhu, Qifan Wang, Man Cheng, and Zhenjiang Cai. An improved deeplab v3+ deep learning network applied to the segmentation of grape leaf black rot spots. *Frontiers in Plant Science*, 13, 2022.

[279] Ramesh Kestur, Avadesh Meduri, and Omkar Narasipura. Mangonet: A deep semantic segmentation architecture for a method to detect and count mangoes in an open orchard. *Engineering Applications of Artificial Intelligence*, 77:59–69, 2019.

[280] Hongxing Peng, Chao Xue, Yuanyuan Shao, Keyin Chen, Juntao Xiong, Zhihua Xie, and Liuhong Zhang. Semantic segmentation of litchi branches using deeplabv3+ model. *IEEE Access*, 8:164546–164555, 2020.

[281] Ehsan Raei, Ata Akbari Asanjan, Mohammad Reza Nikoo, Mojtaba Sadegh, Shokoufeh Pourshahabi, and Jan Franklin Adamowski. A deep learning image segmentation model for agricultural irrigation system classification. *Computers and Electronics in Agriculture*, 198:106977, 2022.

[282] Zhishuang Song, Zhitao Zhang, Shuqin Yang, Dianyuan Ding, and Jifeng Ning. Identifying sunflower lodging based on image fusion and deep semantic segmentation with uav remote sensing imaging. *Computers and Electronics in Agriculture*, 179:105812, 2020.

[283] Melissa Mozifian, Amy Zhang, Joelle Pineau, and David Meger. Intervention design for effective sim2real transfer. *arXiv preprint arXiv:2012.02055*, 2020.

[284] Sungha Choi, Sanghun Jung, Huiwon Yun, Joanne T Kim, Seungryong Kim, and Jaegul Choo. Robustnet: Improving domain generalization in urban-scene segmentation via instance selective whitening. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11580–11590, 2021.

[285] Suhyeon Lee, Hongje Seong, Seongwon Lee, and Euntai Kim. Wildnet: Learning domain generalized semantic segmentation from the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9936–9946, 2022.

[286] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. *arXiv:2304.02643*, 2023.

[287] Xingang Pan, Ping Luo, Jianping Shi, and Xiaoou Tang. Two at once: Enhancing learning and generalization capacities via ibn-net. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 464–479, 2018.

[288] Oren Nuriel, Sagie Benaim, and Lior Wolf. Permuted adain: Reducing the bias towards global statistics in image classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9482–9491, 2021.

[289] Yufei Wang, Haoliang Li, Lap-pui Chau, and Alex C Kot. Embracing the dark knowledge: Domain generalization using regularized knowledge distillation. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 2595–2604, 2021.

[290] Kyungmoon Lee, Sungyeon Kim, and Suha Kwak. Cross-domain ensemble distillation for domain generalization. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXV*, pages 1–20. Springer, 2022.

[291] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[292] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[293] Changyong Shu, Yifan Liu, Jianfei Gao, Zheng Yan, and Chunhua Shen. Channel-wise knowledge distillation for dense prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5311–5320, October 2021.

[294] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[295] Joshua D Rudd, Gary T Roberson, and John J Classen. Application of satellite, unmanned aircraft system, and ground-based sensor data for precision agriculture: A review. In *2017 ASABE Annual International Meeting*, page 1. American Society of Agricultural and Biological Engineers, 2017.

[296] Antonio Novelli, Manuel A Aguilar, Abderrahim Nemmaoui, Fernando J Aguilar, and Eufemia Tarantino. Performance evaluation of object based greenhouse detection from sentinel-2 msi and landsat 8 oli data: A case study from almería (spain). *International journal of applied earth observation and geoinformation*, 52:403–411, 2016.

[297] Aleem Khaliq, Vittorio Mazzia, and Marcello Chiaberge. Refining satellite imagery by using uav imagery for vineyard environment: A cnn based approach. In *2019 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*, pages 25–29. IEEE, 2019.

[298] Cristina Gomez, Joanne C White, and Michael A Wulder. Optical remotely sensed time series data for land cover classification: A review. *ISPRS Journal of Photogrammetry and Remote Sensing*, 116:55–72, 2016.

[299] Aleem Khaliq, Maria Angela Musci, and Marcello Chiaberge. Analyzing relationship between maize height and spectral indices derived from remotely sensed multispectral imagery. In *2018 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, pages 1–5. IEEE, 2018.

[300] George Büttner, Jan Feranec, Gabriel Jaffrain, László Mari, Gergely Maucha, and Tomas Soukup. The corine land cover 2000 project. *EARSeL eProceedings*, 3(3):331–346, 2004.

[301] Devis Tuia, Claudio Persello, and Lorenzo Bruzzone. Domain adaptation for the classification of remote sensing data: An overview of recent advances. *IEEE geoscience and remote sensing magazine*, 4(2):41–57, 2016.

[302] Jochem Verrelst, Luis Alonso, Juan Pablo Rivera Caicedo, José Moreno, and Gustavo Camps-Valls. Gaussian process retrieval of chlorophyll content from imaging spectroscopy data. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 6(2):867–874, 2012.

[303] Laurel Ballanti, Leonhard Blesius, Ellen Hines, and Bill Kruse. Tree species classification using hyperspectral imagery: A comparison of two classifiers. *Remote Sensing*, 8(6):445, 2016.

[304] Xin Huang, Sahara Ali, Sanjay Purushotham, Jianwu Wang, Chenxi Wang, and Zhibo Zhang. Deep multi-sensor domain adaptation on active and passive satellite remote sensing data. In *1st KDD Workshop on Deep Learning for Spatiotemporal Data, Applications, and Systems (DeepSpatial 2020)*, 2020.

[305] Otávio AB Penatti, Keiller Nogueira, and Jefersson A Dos Santos. Do deep features generalize from everyday objects to remote sensing and aerial scenes domains? In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 44–51, 2015.

[306] Vittorio Mazzia, Aleem Khaliq, and Marcello Chiaberge. Improvement in land cover and crop classification based on temporal features learning from sentinel-2 data using recurrent-convolutional neural network (r-cnn). *Applied Sciences*, 10(1):238, 2020.

[307] Ilke Demir, Krzysztof Koperski, David Lindenbaum, Guan Pang, Jing Huang, Saikat Basu, Forest Hughes, Devis Tuia, and Ramesh Raskar. Deepglobe 2018: A challenge to parse the earth through satellite images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 172–181, 2018.

[308] Chao Tian, Cong Li, and Jianping Shi. Dense fusion classmate network for land cover classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 192–196, 2018.

[309] Tzu-Sheng Kuo, Keng-Sen Tseng, Jia-Wei Yan, Yen-Cheng Liu, and Yu-Chiang Frank Wang. Deep aggregation net for land cover classification.

In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 252–256, 2018.

[310] Liangpei Zhang, Lefei Zhang, and Bo Du. Deep learning for remote sensing data: A technical tutorial on the state of the art. *IEEE Geoscience and Remote Sensing Magazine*, 4(2):22–40, 2016.

[311] Haokui Zhang, Ying Li, Yuzhu Zhang, and Qiang Shen. Spectral-spatial classification of hyperspectral imagery using a dual-channel convolutional neural network. *Remote sensing letters*, 8(5):438–447, 2017.

[312] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[313] Zhi He, Han Liu, Yiwen Wang, and Jie Hu. Generative adversarial networks-based semi-supervised learning for hyperspectral image classification. *Remote Sensing*, 9(10):1042, 2017.

[314] Mei Wang and Weihong Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153, 2018.

[315] Wen Ma, Zongxu Pan, Feng Yuan, and Bin Lei. Super-resolution of remote sensing images via a dense residual generative adversarial network. *Remote Sensing*, 11(21):2578, 2019.

[316] Nadir Mohamed Bengana and Janne Heikkila. Improving land cover segmentation across satellites using domain adaptation. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2020.

[317] Sailesh Conjeti, Amin Katouzian, Abhijit Guha Roy, Loïc Peter, Debdoot Sheet, Stephane Carlier, Andrew Laine, and Nassir Navab. Supervised domain adaptation of decision forests: Transfer of models trained in vitro for in vivo intravascular ultrasound tissue characterization. *Medical image analysis*, 32:1–17, 2016.

[318] Zhipeng Deng, Hao Sun, Shilin Zhou, and Kefeng Ji. Semi-supervised cross-view scene model adaptation for remote sensing image classification. In *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 2376–2379, 2016.

[319] Wei Liu and Finlin Su. A novel unsupervised adversarial domain adaptation network for remotely sensed scene classification. *International Journal of Remote Sensing*, 41(16):6099–6116, 2020.

[320] Bilel Benjdira, Yakoub Bazi, Anis Koubaa, and Kais Ouni. Unsupervised domain adaptation using generative adversarial networks for semantic segmentation of aerial images. *Remote Sensing*, 11(11):1369, 2019.

[321] Morvarid Karimpour, Shiva Noori Saray, Jafar Tahmoresnezhad, and Mohammad Pourmahmood Aghababa. Multi-source domain adaptation for image classification. *Machine Vision and Applications*, 31(6):1–19, 2020.

[322] Kanchan Bahirat, Francesca Bovolo, Lorenzo Bruzzone, and Subhasis Chaudhuri. A novel domain adaptation bayesian classifier for updating land-cover maps with class differences in source and target domains. *IEEE Transactions on Geoscience and Remote Sensing*, 50(7):2810–2826, 2011.

[323] Marc Rußwurm, Sébastien Lefèvre, and Marco Körner. Breizhcrops: A satellite time series dataset for crop type identification. In *Proceedings of the International Conference on Machine Learning Time Series Workshop*, 2019.

[324] Noel Gorelick, Matt Hancher, Mike Dixon, Simon Ilyushchenko, David Thau, and Rebecca Moore. Google earth engine: Planetary-scale geospatial analysis for everyone. *Remote sensing of Environment*, 202:18–27, 2017.

[325] Olivier Hagolle, Mireille Huc, David Villa Pascual, and Gerard Dedieu. A multi-temporal and multi-spectral method to estimate aerosol optical thickness over land, for the atmospheric correction of formosat-2, landsat, venμs and sentinel-2 images. *Remote Sensing*, 7(3):2668–2691, 2015.

[326] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[327] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

[328] Gintare Karolina Dziugaite, Daniel M Roy, and Zoubin Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. *arXiv preprint arXiv:1505.03906*, 2015.

[329] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Deep transfer learning with joint adaptation networks. In *International conference on machine learning*, pages 2208–2217. PMLR, 2017.

[330] Raul de Queiroz Mendes, Eduardo Godinho Ribeiro, Nicolas dos Santos Rosa, and Valdir Grassi Jr. On deep learning techniques to boost monocular depth estimation for autonomous navigation. *Robotics and Autonomous Systems*, 136:103701, 2021.

[331] Priya Roy and Chandreyee Chowdhury. A survey of machine learning techniques for indoor localization and navigation systems. *Journal of Intelligent & Robotic Systems*, 101(3):1–34, 2021.

[332] Xuesu Xiao, Bo Liu, Garrett Warnell, and Peter Stone. Motion planning and control for mobile robot navigation using machine learning: a survey. *Autonomous Robots*, pages 1–29, 2022.

[333] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2015.

[334] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1646–1654, 2016.

[335] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 136–144, 2017.

[336] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2472–2481, 2018.

[337] Jie Liu, Jie Tang, and Gangshan Wu. Residual feature distillation network for lightweight image super-resolution. In *European Conference on Computer Vision*, pages 41–55. Springer, 2020.

[338] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. Image super-resolution using very deep residual channel attention networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 286–301, 2018.

[339] Tao Dai, Jianrui Cai, Yongbing Zhang, Shu-Tao Xia, and Lei Zhang. Second-order attention network for single image super-resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11065–11074, 2019.

[340] Ben Niu, Weilei Wen, Wenqi Ren, Xiangde Zhang, Lianping Yang, Shuzhen Wang, Kaihao Zhang, Xiaochun Cao, and Haifeng Shen. Single image super-resolution via a holistic attention network. In *European conference on computer vision*, pages 191–207. Springer, 2020.

[341] Jiezhang Cao, Yawei Li, Kai Zhang, and Luc Van Gool. Video super-resolution transformer. *arXiv preprint arXiv:2106.06847*, 2021.

[342] Hanting Chen, Yunhe Wang, Tianyu Guo, Chang Xu, Yiping Deng, Zhenhua Liu, Siwei Ma, Chunjing Xu, Chao Xu, and Wen Gao. Pre-trained image processing transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12299–12310, 2021.

[343] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1833–1844, 2021.

[344] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[345] Yonggan Fu, Wuyang Chen, Haotao Wang, Haoran Li, Yingyan Lin, and Zhangyang Wang. Autogan-distiller: searching to compress generative adversarial networks. In *Proceedings of the 37th International Conference on Machine Learning*, pages 3292–3303, 2020.

[346] Xintao Wang, Liangbin Xie, Chao Dong, and Ying Shan. Real-esrgan: Training real-world blind super-resolution with pure synthetic data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1905–1914, 2021.

[347] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[348] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie-Line Alberi Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In *British Machine Vision Conference (BMVC)*, 2012.

[349] Jiasi Chen and Xukan Ran. Deep learning with edge computing: A review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019.

[350] Simone Angarano, Vittorio Mazzia, Francesco Salvetti, Giovanni Fantin, and Marcello Chiaberge. Robust ultra-wideband range error mitigation with deep learning at the edge. *Engineering Applications of Artificial Intelligence*, 102:104278, 2021.

[351] Di Liu, Hao Kong, Xiangzhong Luo, Weichen Liu, and Ravi Subramaniam. Bringing ai to edge: From deep learning's perspective. *Neurocomputing*, 2021.

[352] Zibin He, Tao Dai, Jian Lu, Yong Jiang, and Shu-Tao Xia. Fakd: Feature-affinity based knowledge distillation for efficient image super-resolution. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 518–522. IEEE, 2020.

[353] Shinnosuke Ooyama, Huimin Lu, Tohru Kamiya, and Seiichi Serikawa. Underwater image super-resolution using srcnn. In *International Symposium on Artificial Intelligence and Robotics 2021*, volume 11884, pages 177–182. SPIE, 2021.

[354] Md Jahidul Islam, Peigen Luo, and Junaed Sattar. Simultaneous enhancement and super-resolution of underwater imagery for improved visual perception.

In Marc Toussaint, Antonio Bicchi, and Tucker Hermans, editors, *Robotics*, Robotics: Science and Systems. MIT Press Journals, 2020.

[355] Ruoxi Wang, Dandan Zhang, Qingbiao Li, Xiao-Yun Zhou, and Benny Lo. Real-time surgical environment enhancement for robot-assisted minimally invasive surgery based on super-resolution. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3434–3440. IEEE, 2021.

[356] Andrew Brodie and Nikhil Vasdev. The future of robotic surgery. *The Annals of The Royal College of Surgeons of England*, 100(Supplement 7):4–13, 2018.

[357] Hyunjin Bae, Keunyoung Jang, and Yun-Kyu An. Deep super resolution crack network (srcnet) for improving computer vision–based automated crack detectability in in situ bridges. *Structural Health Monitoring*, 20(4):1428–1442, 2021.

[358] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *The European Conference on Computer Vision Workshops (ECCVW)*, September 2018.

[359] Wenzhe Shi, Jose Caballero, Ferenc Huszar, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[360] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[361] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for image restoration with neural networks. *IEEE Transactions on computational imaging*, 3(1):47–57, 2016.

[362] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[363] Yiman Zhang, Hanting Chen, Xinghao Chen, Yiping Deng, Chunjing Xu, and Yunhe Wang. Data-free knowledge distillation for image super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7852–7861, 2021.

[364] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[365] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 126–135, 2017.

[366] Radu Timofte, Eirikur Agustsson, Luc Van Gool, Ming-Hsuan Yang, and Lei Zhang. Ntire 2017 challenge on single image super-resolution: Methods and results. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 114–125, 2017.

[367] Xintao Wang, Ke Yu, Chao Dong, and Chen Change Loy. Recovering realistic texture in image super-resolution by deep spatial feature transform. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 606–615, 2018.

[368] Roman Zeyde, Michael Elad, and Matan Protter. On single image scale-up using sparse-representations. In *International conference on curves and surfaces*, pages 711–730. Springer, 2010.

[369] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pages 416–423. IEEE, 2001.

[370] Yusuke Matsui, Kota Ito, Yuji Aramaki, Azuma Fujimoto, Toru Ogawa, Toshihiko Yamasaki, and Kiyoharu Aizawa. Sketch-based manga retrieval using manga109 dataset. *Multimedia Tools and Applications*, 76(20):21811–21838, 2017.

[371] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. Single image super-resolution from transformed self-exemplars. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5197–5206, 2015.

[372] Honggang Chen, Xiaohai He, Linbo Qing, Yuanyuan Wu, Chao Ren, Ray E Sheriff, and Ce Zhu. Real-world single image super-resolution: A brief review. *Information Fusion*, 79:124–145, 2022.

[373] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[374] Partha Pratim Ray. Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. *Internet of Things and Cyber-Physical Systems*, 2023.

[375] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026, 2023.

[376] Yuchen Cui, Scott Niekum, Abhinav Gupta, Vikash Kumar, and Aravind Rajeswaran. Can foundation models perform zero-shot task specification for robot manipulation? In *Learning for dynamics and control conference*, pages 893–905. PMLR, 2022.

[377] Jiange Yang, Wenhui Tan, Chuhao Jin, Bei Liu, Jianlong Fu, Ruihua Song, and Limin Wang. Pave the way to grasp anything: Transferring foundation models for universal pick-place robots. *arXiv preprint arXiv:2306.05716*, 2023.

[378] Dhruv Shah, Ajay Sridhar, Nitish Dashora, Kyle Stachowicz, Kevin Black, Noriaki Hirose, and Sergey Levine. Vint: A foundation model for visual navigation. *arXiv preprint arXiv:2306.14846*, 2023.