Robust robot tracking for next-generation collaborative robotics-based gaming environments

Robust Robot Tracking for Next-Generation Collaborative Robotics-based Gaming Environments

Giovanni Piumatti, Fabrizio Lamberti, *Senior Member, IEEE,*
Andrea Sanna, and Paolo Montuschi, *Fellow, IEEE*

Abstract—The collaboration between humans and robots is one of the most disruptive and challenging research areas. Even considering advances in design and artificial intelligence, humans and robots could soon ally to perform together a number of different tasks. Robots could also became new playmates. In fact, an emerging trend is associated with the so-called *phygital* gaming, which builds upon the idea of merging the physical world with a virtual one in order to let physical and virtual entities, such as players, robots, animated characters and other game objects interact seamlessly as if they were all part of the same reality. This paper specifically focuses on mixed reality gaming environments that can be created by using floor projection, and tackles the issue of enabling accurate and robust tracking of off-the-shelf robots endowed with limited sensing capabilities. The proposed solution is implemented by fusing visual tracking data gathered via a fixed camera in a smart environment with odometry data obtained from robot's on-board sensors. The solution has been tested within a phygital gaming platform in a real usage scenario, by experimenting with a robotic game that exhibits many challenging situations which would be hard to manage using conventional tracking techniques.

Index Terms—Online tracking, TLD, odometry, data fusion, gaming, projected environment, mixed reality, collaborative robots.

1 Introduction

In 2015, the World Economic Forum identified "Next-generation robotics" as one of the top 10 emerging technologies, by outlining how in the near future robots will allow humans to evolve through alliances with machines ever more intelligent, collaborative, adaptive and flexible [1]. Collaboration between humans and robots has the potential to change our everyday lives. In fact, from purely industrial and scientific tools used to replace humans for laborious or uncomfortable operations, robots are becoming true "companions", helping humans in a variety of tasks. Many so-called *service* robots are already on the market, ranging from household appliances, such as vacuum cleaners [2] or lawn mowers [3], to toys for both entertainment [4] and educational [5] purposes.

In most of these application scenarios, robots must autonomously perform several common tasks that are essential to their functioning. One such task is localization within the environment. The ability to accurately pinpoint a robot's position (and, usually, also orientation) is extremely important, as it enables high-level capabilities such as global path planning and goal-based navigation [6] that are the starting point for more complex tasks (such as collaborative ones).

- *The authors are with the Dipartimento di Automatica e Informatica of Politecnico di Torino, Corso Duca degli Abruzzi 24, I-10129, Torino, Italy. E-mail: see http://grains.polito.it/people.php*

A large amount of research has been devoted to enabling robots to self-localize, i.e., to estimate their own pose (position and orientation) using only their on-board sensors. This approach is considered the most advantageous for mobile robots as it allows them to navigate most of the environments. State-of-the-art algorithms such as Simultaneous Localization and Mapping (SLAM) [7] allow robots to autonomously build a map of a previously unknown environment and locate in it. However, self-localization algorithms often require sophisticated sensors and high computational power. Therefore, robots used, e.g., in consumer-level applications (such as gaming) are often built using cheaper sensors and simpler algorithms at the expense of accurate localization [8].

A different approach to localization consists of relying on smart environments, instrumented with cameras, beacons, and other types of devices that can be used to track the robot's movements. The main drawback of this approach is that the robot can only be localized in that specific environment or in similarly-instrumented ones [9]. On the positive side, this approach enables the implementation of rather sophisticated behaviors, such as dynamic obstacle avoidance [10], object detection/recognition, complex decision making [11], etc. in relatively simple and cheap robots. Moreover, with this approach multiple robots can be managed in a concurrent way, therefore reducing their sensory and computational needs and, hence, their cost. Lastly, a smart environment has the advantage of providing greater situational awareness

to robots operating in it, since its sensors will likely capture more information than those of a single robot. This is especially important, e.g., in safety-critical settings, where robots may need to be aware of what is happening outside their sensors' range in order to make the correct decisions.

The concept of smart environment has been already investigated in a number of works [12], where it is used, e.g., for people tracking, activity recognition, etc. In this paper, considering also the fact that enabling technologies are expected to gain ever more importance and become more affordable and ubiquitous in the coming years [13][14], the concept is specifically exploited in the context of gaming.

In particular, the work reported in this paper is part of the activities that are being carried out to create a platform for mixed reality gaming in instrumented environments. The platform exploits the concept of *phygital* gaming, which is the idea to merge the physical world and a digital game environment with the aim to create the "perception" of a unified reality where people, robots and digital objects can interact seamlessly in various ways (a brief history of phygital gaming and of so-called physically interactive robogames is provided in [15] and [16]). The platform is designed to project a dynamic playground on the floor, where virtual elements can ultimately be drawn. Physical entities, i.e., players and robots, are detected and tracked using one or more cameras, whereas a game engine manages the physics simulation (responsible, e.g., for handling the interaction between physical and virtual objects) and the game logic.

The overall design envisaged for the platform, the preliminary games created and the user studies carried out so far have been presented in previous works [17][18]. This paper specifically focuses on the robot tracking algorithm, which has been designed by also considering the outcomes of the above experiments.

The challenges for tracking mobile robots in this type of environments mainly come from the design requirements of the games themselves. The main problems are: the presence of a dynamic background and local changes in luminosity (due to the projected virtual environment), which may interfere with the detection of the target to be tracked; occlusions (e.g., caused by human players, which may happen as a result of certain game interactions between players and robots), fast movement and abrupt direction changes, as well as motion blur (since a fast-moving robot is expected to make the game more engaging).

Moreover, the setup of the environment is not, by design, particularly constrained. In order to keep installation easy, it should be possible to position cameras quite freely, as long as their combined fields of view encompass the entire playground area. Lastly, the gaming system is intentionally designed to leverage commercial off-the-shelf robots. Therefore, the tracking algorithm should make few assumptions on the physical aspect of the robots. Notwithstanding, many commercial robots are equipped with inexpensive sensors. For instance, some robots developed as gaming companions include inertial measurement units (IMUs) and wheel encoders. These sensors enable the robot to keep track of its own movements by providing an odometry, i.e., a rough estimate of its pose with respect to the starting pose, which could be used to improve tracking performance.

Based on the above considerations, the goal of this paper is to present a tracking algorithm that exploits data fusion techniques to meet the aforementioned requirements. The proposed algorithm is designed to complement image data coming from a fixed RGB camera in a loosely instrumented smart environment with any type of odometry obtained with robot's onboard sensors. The problem of tracking a robot within a projected environment is known in the literature and many solutions have been proposed already. However, to the best of the authors' knowledge, this is the first one not requiring severe assumptions either on the robot's available sensors or on the setup of the environment (e.g., on camera positioning).

The robot is firstly tracked using the state-of-the-art Tracking-Learning-Detection (TLD) algorithm [19]. TLD has been selected based on an a preliminary evaluation carried out by running a recent benchmark on online visual tracking [20] over a number of video sequences featuring challenges that may be found in real gaming scenarios. The main issue with visual tracking algorithms is represented by model drifting. Internally, algorithms such as TLD use a classifier trained on a few images of the target object, which are updated as time passes. Problems occur when the model starts to be updated with images that contain unwanted information, causing it to drift.

To cope with the above issue, TLD was modified to make model updates occur only once the candidate bounding box that is supposed to contain the target is validated against the robot's odometry. The resulting visual tracking algorithm balances drifts that might be introduced by on-board sensors. It also provides an updated transformation matrix between the global (absolute) and the robot's local (relative) coordinate systems. This matrix allows to transform points from the robot's (i.e., the odometry) coordinate system to the global one, thus enabling absolute localization even when visual tracking fails, e.g., due to occlusion of the target. A further benefit of this approach is that it allows to use a fast control loop, since the robot's internal sensors have typically a higher throughput than visual tracking. It is worth noting that even though design requirements for the algorithm described in this paper stem from its expected usage within a projected playground, the proposed approach could be applied to other types of smart environments, like smart homes, smart factories, etc. [11][21][22].

The rest of the paper is organized as follows: Sec-

tion 2 reviews the state of the art in visual tracking. Section 3 discusses the process that led to the selection of TLD. Section 4 presents the design and implementation of the proposed odometry-assisted visual tracking system. Section 5 reports on the experimental evaluation. Lastly, Section 6 concludes the paper.

2 Background

Robot localization and mapping are arguably among the most important problems to solve in autonomous and collaborative mobile robotics [23]. On their own, the two problems are not particularly complex, and many solutions have already been developed. For instance, given the map of an indoor environment, various approaches to the localization problem have been proposed, which can be divided in two categories: position tracking and global position estimation [24].

The first approach assumes that the initial pose of the robot is known (or defined arbitrarily). Then, sensor measurements are used to estimate how the pose evolves over time. The main difficulty lays in modeling and recovering from measurement errors. Measurement errors can grow unbounded, inducing drifts in the pose estimation. An algorithm representative of this category is optical flow [25]. The algorithm has found many applications in visual odometry or stereo vision, where it is used to reconstruct a 3D scene from two or more images taken from different viewpoints. To partially solve the problem of drifts, the authors of [26] suggest to use geometric beacons, i.e., distinctive features of the environment such as walls or corners, to limit uncertainty. By using an extended Kalman filter and an a priori map of the beacons' location, the proposed method is able to limit error growth along different dimensions, depending on the feature observed. For instance, observing a wall reduces uncertainty along the direction perpendicular to the wall but not along the parallel direction.

The main limitation of position tracking algorithms is that they are not able to determine the robot's pose globally, but are dependent on a manual initialization of the starting pose. More importantly, if at any point the algorithm loses track of the current position, it cannot, by definition, recover it.

Global position estimation algorithms were developed to cope with such limitations. These techniques use more sophisticated belief models such as, for instance, Markov processes [27]. The idea is to model robot's pose as a set of posterior probabilities calculated over the whole state space (i.e., all the possible poses). The state space must be discretized in order to compute the posteriors. Different approaches were developed, characterized by various granularities.

For instance, in [27], the environment is sampled in a dense grid. Each cell stores the conditional probability for the robot to be at that location, given the current sensor readings. Once the robot has been globally located, the state space is reduced to a grid surrounding the robot's current position. The main problem with this approach is that grid size, resolution and, therefore, accuracy in the position estimation have to be predetermined. Some works showed that it possible to use space subdivision to implement adaptive grids, whose resolution changes where needed [28]. A coarser approach involves topological instead of metric maps. In [29], the authors start with the assumption that the robot has access to robust, low-level navigation skills (such as going through a door or around a corner), and posit that the robot does not actually need to know its exact position, but just a coarse location, such as a room or a corridor. The algorithm exploits a pre-built topological map connecting areas of interest in the environment, sensor readings and high-level commands to refine the probability distribution as it moves. The accuracy of the estimated position depends, therefore, on the map resolution.

To overcome most of the limitations of global position estimation techniques, the authors of [30] propose the Monte Carlo (MC) approach. This approach represents the state space as a finite set of weighted particles, the weight being the probability of that state. This representation allows the algorithm to approximate any arbitrary probability distribution, making it very suitable for localization. At each time step, the algorithm updates the distribution by randomly extracting particles based on their weight and a motion model. Then, weights are adjusted based on sensor readings. The algorithm is able to maintain multiple hypothesis and to refine and resolve ambiguities as more information is gathered. It can determine the robot's position globally, and is able to recover during operation if tracking is lost.

In [28], the two classes of map-based algorithms (position tracking and global position estimation) are integrated into a single hybrid algorithm. The algorithm adapts its computational and memory requirements depending on its confidence in the robot's position. When confidence is high, the algorithm behaves like a position tracking algorithm, maintaining a small search space. When tracking is lost, or confidence degrades, the system transitions towards a global positioning technique. Although this algorithm is probably the most efficient and accurate one, similar to the other techniques reported above, it suffers from a major drawback: it requires a pre-existing map of the environment, which greatly reduces its ability to be deployed quickly and easily. In order to remove this requirement, the robot must be able to simultaneously map the environment as well.

The algorithms able to concurrently carry out mapping and localization within an environment are known, collectively, as SLAM. Specifically, probabilistic SLAM techniques estimate, at every time step, both the robot's pose and a map of the environment the robot is working in. Early approaches adopted

extended Kalman filters to model their beliefs [7]. More recently, influenced by [28] and [30], the authors of [31] adopted the same MC technique of [30] in their SLAM implementation. Its main benefits being the possibility to explicitly model non-linear probability distributions and its reduced computational complexity, the MC-based solution represents the state-of-the-art for localization and mapping. Nonetheless, SLAM still presents many open problems. In fact, most of the current implementations work correctly only in structured environments (e.g., indoors) and virtually all of them are based on a static world assumption (i.e., landmarks do not move). These assumptions limit the applicability of SLAM in real-world scenarios, where environments are rarely completely static.

Considering the main target application for which the tracking system reported in this paper was built, self-localization techniques are not particularly appropriate. Firstly, the mixed reality gaming system is expected, by design, to be used with small off-the-shelf toy robots, which are often equipped with low-cost sensors. Unfortunately, the tracking algorithms in the above review generally require high-quality sensors, such as laser range finders, sonars, etc. Secondly, the robot is going to operate in a highly dynamic environment, with moving people/robots in close proximity, which might interfere with its mapping capabilities. Finally, in order to correctly map the physical world to the digital one, the robot's position must be transformed into the projection's coordinate system. The transformation matrix between this coordinate system and the robot's coordinate system is dynamic and must be determined at every time step, a task not easily to be accomplished.

These considerations, coupled with the fact that the gaming system is expected to be deployed inside a smart environment, led to the choice of an external tracking mechanism. In particular, since the only constraint on the hardware available on the robots is that it exposes some sort of odometry, specialized methods such as magnetic or ultrasonic trackers are not applicable. Therefore, visual tracking techniques are arguably the most appropriate choice in this case.

The idea of exploiting external vision sensors to localize and track a robot has received little attention compared to the task of self-localization. For instance, in [10] and [13], a network of cameras is used to track both robot and obstacles in the environment. Images captured by the cameras are fed to a planning algorithm in order to implement path following and obstacle avoidance. A similar system is presented in [9] and [32], where on-board sensors are taken into consideration as well. The robot's sensors are used to integrate missing data when visual tracking fails (e.g., because the robot goes out-of-frame). In both the systems, the robot is outfitted with IR beacons, which help the vision system determine position and orientation with great accuracy. In [33], the authors propose a system that is able to autonomously learn a robot's appearance model and subsequently track it by using an external camera and the robot's odometry. In order to distinguish the robot from the background, the vision system uses a discriminative background model initialized with a single frame of the background. Such an approach requires the environment to be mostly static, otherwise features of the environment may be mistaken for features of the robot. The authors state their solution is robust against short-term occlusions, since a Kalman filter allows to predict the robot's position quite reliably over short periods of time. However, the algorithm's performance in such situations was not quantified, and it is not stated whether the algorithm is robust against out-of-frame events or false positives. A similar concept is proposed in [14], where authors present a large-scale smart environment composed of a scalable network of cameras and other sensing units. Besides implementing robot tracking, their smart environment is able to detect the movement of people or other objects.

Considering the domain of mixed reality robotic gaming, most of the existing works exploit custom sensors, markers or similar devices to assist tracking. For example, in [34], colored markers on the robots allow the tracking system to accurately determine their identity, position and orientation. Tracking is performed with an overhead camera, and occlusions are not handled (as they cannot occur). In [35], the robots are outfitted with simple light sensors. By projecting carefully-designed fiducial markers on the robots, after an initialization phase devoted to determine their starting position, the system is able to locate the robots in the projected area. The need for camera-projector calibration is bypassed, since tracking is performed in the same coordinate system as the projection. In [36] the authors use a combination of display-based tracking, special IR markers and depth cameras to track robots, pens or other objects all interacting within a projected environment.

All of these works show how it is possible to exploit a great number of different devices and techniques to track people, robots and objects. Such technologies are used by the various authors to enable complex interaction paradigms between humans and machines (both robots and computer systems). Although the interaction mechanisms proposed in the works above show great promise, the underlying enabling technologies (i.e., the tracking systems) require certain constraints on the objects being tracked (e.g., active IR transmitters, known appearance, etc.).

This paper proposes a solution to the robot tracking problem which does not impose constraints on the appearance of either the robot or the environment (for the purpose of tracking), and only requires the robot to provide some form of odometry. Odometry data are fused with position information provided by an online visual tracking algorithm. This choice avoids

the need of fiducial markers, special sensors or other devices on the robot, as the tracker is quickly trained on the robot's own appearance. The robot's orientation can be inferred from its motion, by considering its position in previous frames.

The robot's odometry, in particular, is exploited in different ways: it is used to validate the trajectory reported by the visual tracker in order to avoid performing the training with incorrect data; it is fused with visual tracking data in an extended Kalman filter to obtain a more robust estimate (the filter lets the algorithm compensate for the possibly different frame rates of odometry and visual tracking data, as well as to deal with occlusions and out-of-frame situations); it provides a search window for the visual tracker that reduces the probability to select a wrong target.

3 Requirements and Visual Tracking

By focusing on the target application, which requires tracking sensor-enabled mobile robots in a mixed reality-based gaming environment instrumented with a RGB camera, the first step in the design methodology pursued in this work was to define the fundamental requirements the system should implement. Requirements can be summarized as reported below.

- Real-time tracking: the tracking system must be able to work in real-time, i.e., it must be able to process data collected from available sensors as quickly as they are produced.
- Online learning: the algorithm must be able to adapt to changing conditions in lighting, background and robot's appearance. The algorithm should not require the user to perform supervised training before it can be used.
- Robustness: tracking must be robust against occlusions, rapid changes in robot's direction, false positives, etc. It should also be possible to automatically resume the tracking once the robot exits and reenters the camera's field of view.
- Accuracy: the estimated pose returned by the tracking system must match the robot's real pose within an arbitrarily small error range.

Real-time tracking is an essential requisite to correctly implement the illusion of mixed reality. For example, to simulate a collision between a robot and a virtual ball the tracker must be able to communicate the robot's position to the game engine as soon as it touches the virtual ball's edge. The physics engine will then simulate the collision and change the ball's trajectory accordingly. If the tracker is too slow, the simulated collision will start too late and the human observer will not be able to correlate cause and effect, thus breaking the virtual-real continuum. Furthermore, real-time tracking is necessary to implement the feedback controller driving the robot.

Online learning is the requisite that provides the system with the necessary flexibility to operate under different environmental conditions and to track arbitrary-looking robots. Additionally, online learning does not require lengthy training to take place before the tracking can start, which improves ease of use.

Robustness is another critical requisite. The system should keep tracking the robot as long as possible, by avoiding to lock onto wrong targets. Consequences of a false positive can be quite catastrophic. Any module that is using the visual tracker output would start receiving incorrect data. The game logic would believe the robot is in a position different than the actual one. If the robot is being driven by a feedback controller, it would start behaving erratically, since the received feedback would be incorrect.

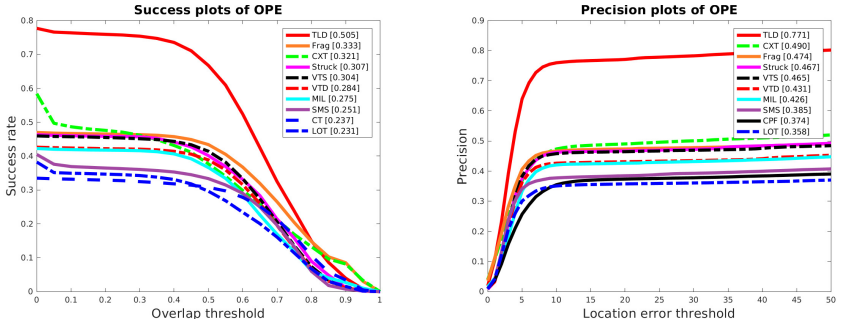
The last requirement is about accuracy. Although it may be regarded as the less critical requisite, it is nonetheless important to maintain the illusion of mixed reality. In fact, considering the typical size of a projected playground and of the robots involved, the acceptable error margin lies within a few centimeters.

The next step consisted in selecting a visual tracking algorithm able to satisfy the requirements presented above. There are many alternatives available, with different characteristics and performance. A possibility for exploring a rather comprehensive set of algorithms for online visual tracking is to consider the benchmark reported in [20]. Here, 29 algorithms were reviewed, their source code or binaries provided, and their overall performance assessed against a public set of video clips showing a number of critical features.

However, the specific scenario of robot tracking in a mixed reality playground requires the evaluation to be performed under more specific conditions. Hence, 20 video sequences depicting situations that could occur during real playing were recorded and used for running the benchmark. Sequences were recorded in the smart environment later used for the experimental evaluation (Section 5), capturing both camera images and robot odometry in a synchronized manner. The Orbotix Sphero 2.0[1] robot was used. Sequences (including tracking results) are available for download[2]. Raw data are available on request.

In each sequence, the following attributes are present in any combination (abbreviations used in sequence names are given): occlusion ("occl", the robot is occluded in a variable number of frames, and sometimes direction changes happen during occlusion); projection ("proj", the robot enters the projected area, where a moving pattern is displayed); out-of-frame ("oof", the robot goes out of the camera's field of view, sometimes returning from a different point); false positive ("fp", there is a second, identical robot in the field of view). Sequences are named after the set of attributes they present. Sequence "baseline" represents the reference, since attributes are all absent. Two further sequences are introduced, which are ex-

1. http://www.sphero.com/sphero
2. https://goo.gl/q8RWC7



(a) (b)

Fig. 1. Benchmark results (top 10 algorithms): a) success plot, sorted by AUC, b) precision plot, sorted by precision at 20px error threshold. Abbreviations used for algorithms are the same of [20].

ploited to evaluate the impact of changing lighting conditions ("lights_change") and reflections ("reflection") on the floor. Finally, two additional projection and false positive sequences are used to evaluate different conditions ("projection2", with a different pattern projected on the floor, and "fp2", where the robot collides with the second false positive robot).

Sequences were manually labeled to build the ground truth. Occlusions and out-of-frame events were labeled too. The robot's position in occluded frames was estimated with the recorded odometry.

Benchmarking was performed using an Intel Core i7-4810MQ at 2.8 GHz with 8GB RAM. Algorithms were executed using default parameters. Results obtained with one-pass evaluation (OPE) [20] are reported in Fig. 1. As done in [20], only the top 10 algorithms are shown, ranked by success and precision. The success plot (Fig. 1a) indicates, for each algorithm, the percentage of frames in which the tracked object's bounding box overlaps the ground truth by more than a threshold, varying the threshold itself. The precision plot (Fig. 1b) reports the percentage of frames in which the distance of the tracked object from the ground truth (considering bounding box's center) is less than a threshold. In the precision plot, algorithms are ranked by precision with a 20 pixel error threshold. In the success plot, the use of a specific threshold value to rank the algorithms is not deemed as significant [20]. Instead, a measure known as Area Under the Curve (AUC) of the plot is used. Further details on the evaluation process, e.g., the performance achieved on each sequence and average frame rate, are provided in Appendix A (included as supplementary material).

Given the requirements set, and considering the results obtained with the above sequences, it was decided to ground the design of the fused tracking system on the TLD algorithm. In fact, when the average performance over all the sequences is considered, TLD is by far the most robust alternative. Even though it appears to suffer in sequences with false positives and changing lighting conditions, its performance is only slightly below the best algorithms for those sequences. The average frame rate achieved by TLD (~ 34 Hz) is lower than with some of the other algorithms considered (see Table 1 in Appendix A). Nonetheless, it can still guarantee a sufficient tracking speed for real-time operations.

4 Odometry-assisted Tracking

For the purpose of better understanding how visual information gathered from the RGB camera is exploited in the design of the odometry-assisted tracking system proposed in this paper, it is necessary to introduce the basics of the visual tracking algorithm selected in Section 3. Hence, in the following, the key concepts behind the TLD technique will first be presented (for a complete formalization see [19]). Afterwards, the way odometry information is integrated in the overall tracking system will be shown.

4.1 The TLD Algorithm

The TLD algorithm is composed of four main modules (Fig. 2): a detector (a cascading classifier), a frame-to-frame tracker, an integrator and a learning component. When correctly trained, the detector is able to locate the target within the whole frame. The tracker, on the other hand, requires the bounding box from the previous frame to estimate its displacement in the next frame. The output of the detector and tracker are combined by the integrator into a single bounding box. The learning component manages to train the classifier, when appropriate. It is assisted by two "expert" processes (dashed block in Fig. 2), called "P-expert" and "N-expert", whose purpose is to analyze the detector's output and check for false positives (FP) and false negatives (FN). If any are found, they are relabeled and used to train the classifier.

The algorithm is initialized by providing the bounding box of the object to track in the initial frame. The learning component bootstraps the classifier by training it with synthesized positive patches extracted from the initial bounding box and negative patches

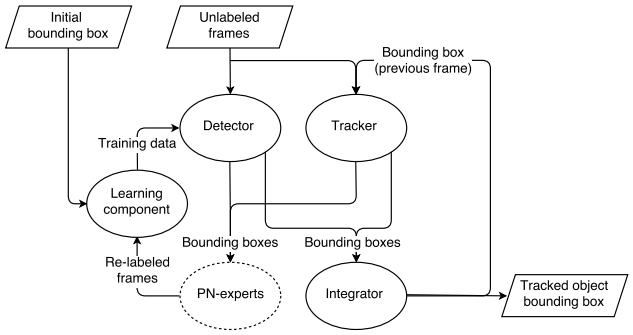


Fig. 2. Block diagram of the TLD algorithm [19]. Inputs are the initial bounding box containing the target and the unlabeled frames. The output is a bounding box centered on the tracked object.

from the background. In the next frames, the tracker and the PN-experts are used to validate the detector's results. The P-expert uses the tracker to check if the candidate bounding box was correctly classified. If the tracker's result was classified as negative (i.e., the detector failed to identify the target), a set of positive patches are synthesized and used for training. Similarly, the N-expert checks for false positives: any bounding box that was classified as positive and does not overlap with the tracker output is used to extract negative patches for training. Finally, the results of detector and tracker are combined in a single bounding box output by the integrator module. If the tracker fails to provide a bounding box for the current frame, the PN-experts are unable to validate the detector's output. In such cases, the algorithm outputs a result only if the detector reports a single bounding box candidate. Otherwise, if multiple potential candidates are identified or if neither the detector nor the tracker output a result, the object is declared as not visible.

In their formulation of the PN learning technique summarized above, the authors demonstrated that the number of errors in the two most common classes, i.e., FN and FP, drops as the quality of the P- and N-experts increases. Even though the PN-experts are automatic processes which can make mistakes themselves, it can be proved that they compensate each other's errors. As long as the number of errors made by one of them is lower than the number of patches correctly re-labeled by the other, the detector will converge to a state with less errors.

4.2 Integration of Odometry Information in TLD

After having analyzed the internal mechanisms of the TLD algorithm and its performance on the proposed sequences, it is possible to understand the reasons for tracking failure in different situations. In fact, experiments highlighted that performance suffers the most in the presence of projection. Specifically, TLD fails shortly after the robot enters the projection, possibly locking onto a patch on the floor. Usually it is able to recover when the robot moves outside the projected area, but not in all the cases. Tracking fails when neither the tracker nor the detector are able to provide a valid estimate of the robot's position. This behavior implies that the projection interferes with both processes. When the algorithm loses the target and starts tracking the floor, the detector is being trained with false positives, meaning the internal appearance model is being wrongly updated. If the robot exits the projected area, the detector can identify it and resume correct tracking only if the internal model was not completely overwritten by incorrect data. Additionally, in case of occlusions or out-of-frame events, it is possible for the TLD algorithm to detect and start tracking false positives. This situation can occur if there are image patches that are classified as positive (because they look similar to the robot in appearance). The N-expert is not able to validate the results, because the tracker did not provide an output (since the robot is not visible). Thus, if the detector identified only one patch as positive, that patch is considered to be the tracked object. In subsequent frames, the tracker will lock onto that patch and, in the worst-case scenario, reinforce the incorrect training.

It is evident from the analysis above that the main problem with the TLD algorithm is model degradation due to incorrect training. The PN-experts were designed, respectively, to reinforce the model when the classifier fails to identify it and to avoid its degradation. However, there are situations in which the available knowledge is not enough to determine whether to accept or reject a given patch. This paper proposes the design of an additional N-expert process capable of identifying false positives by exploiting the robot's own odometry.

The proposed N-expert should be able to discard false positives in those situations when the original N-expert was not. As a result, the classifier would be trained with a smaller number of false positives, improving its robustness. In the following, the approach will be presented using the level of detail used in Section 4.1. A complete formalization based on the notation used in [19] is provided in Appendix B.

The algorithm assumes that the setup of the environment and the camera intrinsics are known. With reference to Fig. 3, the *camera_footprint* coordinate system is obtained by projecting the *camera_frame* onto the ground plane, with the XY plane parallel to it. This coordinate system is used for extracting the robot's 3D position from its 2D projection on the image plane (it is assumed that the robot always lies on the ground plane). The robot reports its pose (*robot_frame* is a coordinate system affixed to it), relative to *odom_frame*, whose pose is arbitrary and usually set to the pose the robot is in when it is turned on. In order to be able to use odometry data to validate the detector's output, the proposed N-expert needs to know the transfor-

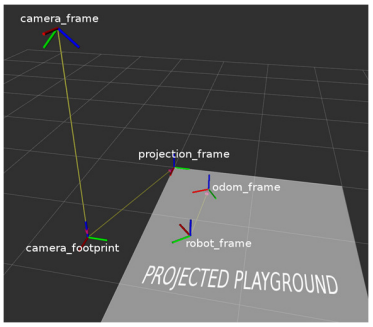


Fig. 3. Coordinate systems involved in a typical setup. Axes color coding: X red, Y green, Z blue. Yellow lines represent the available transformation matrices.

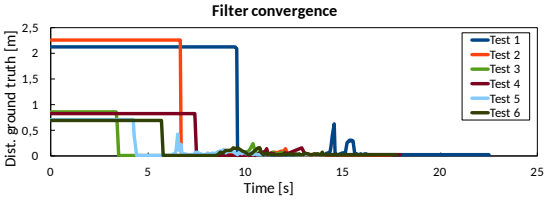


Fig. 4. Time required for the Kalman filter to converge.

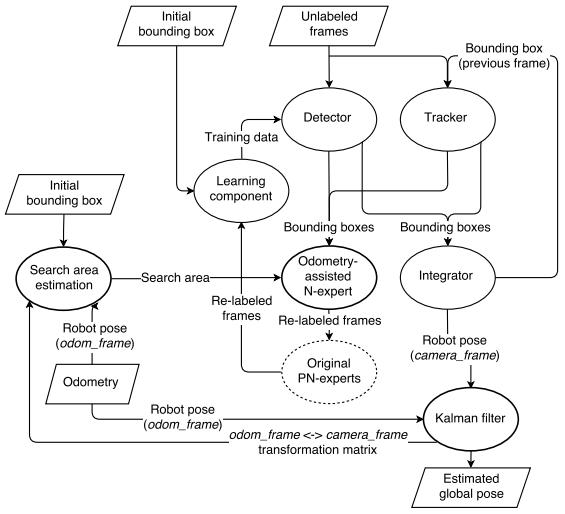


Fig. 5. Block diagram of the proposed algorithm. Inputs are those of TLD, plus odometry and camera calibration data. The output is the robot's pose estimated by the Kalman filter.

mation matrix between *odom_frame* and *camera_frame*. This matrix can be easily calculated once the robot's pose is known in both coordinate systems.

However, orientation in the camera coordinate system is hard to measure accurately, as it is estimated from the robot's movement. Therefore, data fusion and transformation matrix estimation are performed by an extended Kalman filter [37] in order to take into account both the robot's kinematic model and potential errors introduced by the odometry and visual tracking. The consequence is that, at startup, the transformation matrix, and thus the proposed N-expert, is not available, and will be ready only after the Kalman filter converges. To this aim, before actually considering the tracking as started, the robot is automatically moved long enough for the filter to reliably estimate its orientation in the *camera_frame*. This process is illustrated in Fig. 4 using measurements from several runs. Initially, the error is rather high (value depends on the physical position of the robot w.r.t. to the origin of the coordinate system). As soon as the target is defined for the visual tracker, error dramatically drops down, as filter covariance is low. As the robot starts moving, error fluctuates since robot's odometry is projected using a wrong orientation. After few seconds, error converges to an acceptable value (around 0.03 m). Based on the above considerations, it is important (and the algorithm relies on this assumption) that during the first few seconds when the robot starts moving, TLD is able to correctly track it with no false positives.

In Fig. 5, the block diagram of the proposed algorithm is shown. As it can be observed, it is an extension of the TLD architecture in Fig. 2. Once the Kalman filter converges, it is able to output the transformation matrix needed to project the robot's estimated position (as reported by the odometry) onto the image plane. The estimated position is then converted into a bounding box by rescaling the initial bounding box depending on the distance of the robot from the camera (applying the pinhole camera model).

Considering that the resulting bounding box was obtained by projecting the (estimated) odometry onto the image plane using an estimated transformation matrix, it is very likely that the target is not perfectly centered in the bounding box. For this reason, it is not possible to design an odometry-assisted P-expert, since it would require a much more reliable and accurate estimate of the robot's bounding box in the image plane in order to generate good training data. A N-expert, on the other hand, does not need exact knowledge of the robot's position to be able to discard most false positives: any match whose distance from the estimated position is over a threshold can be labeled as false positive. In fact, by simply knowing the "area" in which the target is likely to be, it can discard any false positive that does not overlap that area. Moreover, the target's displacement vectors (i.e., direction and magnitude of the displacement between two consecutive frames) in both *camera_frame* and *odom_frame* can be compared without the need to know the robot's position (i.e., the vectors' origin)[3].

The "search area" for the target is thus defined by its estimated bounding box and the confidence in such

3. The displacement vector in *camera_frame* must first be projected from the 2D image plane to the 3D ground plane in order to be comparable to the displacement vector in *odom_frame* (or viceversa). This is easily accomplished as both camera intrinsics and ground plane equation are known.

estimate. Once the robot's odometry is projected on the image plane and converted to a bounding box, the search area estimation component will enlarge the resulting bounding box by a factor inversely proportional to the Kalman filter's confidence (covariance) in its estimate of the transformation matrix. The less the filter is confident, the bigger the bounding box will be. This technique ensures that the target is very likely to be inside the search area at all times.

The proposed N-expert uses the obtained search area to validate the detector's output in different ways. First, any candidate bounding box that does not overlap the search area is re-labeled as negative. The label will be used during the learning process to avoid detecting the same object in the future. Next, displacement vectors are available, they are compared with the odometry displacement. A displacement vector is available when the previous iteration of TLD produced a bounding box. The displacement vector is simply the displacement of one of the bounding box's corners from the previous frame to the current one. The displacement vectors of two opposite corners are compared against those reported by the odometry (bounding box size is estimated again with the pinhole camera model). If the direction or magnitude of the displacement vectors differ by a set threshold, the result is re-labeled. This method ensures that the candidate target is moving consistently with the odometry readings. The reason for testing two corners rather than, e.g., only the bounding box center, is that, in this way, changes in the scale of the candidate bounding box can be taken into account. The thresholds for direction and magnitude are determined experimentally and depend on the accuracy of the robot's odometry and of the Kalman filter's underlying kinematic model.

The odometry-assisted N-expert's output is then fed to the original PN-experts and continues through the rest of the pipeline of the original TLD algorithm.

During runtime, it is possible for the tracked object to be occluded or to fall out of frame. In such cases, the only data source for the Kalman filter is the odometry: velocity is integrated to estimate a position, which, however, will drift over time. This situation can be identified by the growing covariance of the Kalman filter, which is an indicator of the filter's reliability. If covariance grows larger than a threshold, the N-expert is disabled completely and the algorithm reverts to the behavior of the original TLD. If tracking is correctly recovered by TLD, the Kalman filter will be able to correct its state estimate, output covariance will reduce and the odometry-assisted N-expert will be enabled again. This behavior allows the designed algorithm to keep tracking the robot when it is not visible, though accuracy degrades over time with a rate that depends on the quality of the robot's odometry and the accuracy of the Kalman filter's underlying model. Continuous tracking is highly desirable for the envisaged applications, as it will enable feedback-based controllers to drive the robot at all times.

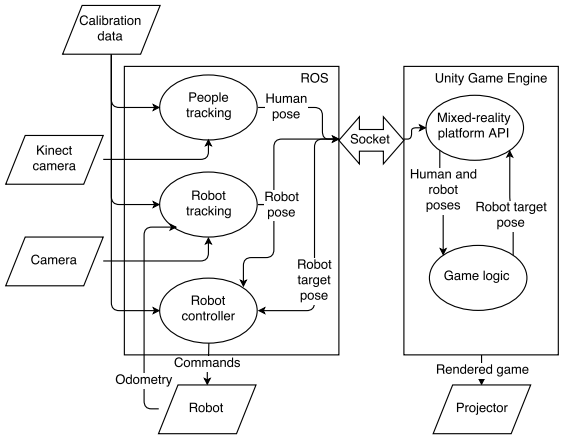


Fig. 6. High-level architecture of the mixed reality platform the proposed algorithm was developed for.

It is worth observing that, since the Kalman filter is updated with the result of the modified TLD algorithm, it can output an updated transformation matrix and estimated robot pose only after TLD has run. This implies that, during each iteration, the odometry-based N-expert is using the transformation matrix from the previous iteration. Hence, for the proper working of the designed algorithm it has to be assumed that the transformation matrix drifts slowly over time, i.e., that the error of the odometry between consecutive frames is negligible. After performing data fusion, the estimated global position of the robot is returned by the Kalman filter. Depending on how accurate its predictive model is, it is possible to run it at an arbitrary rate. This fact allows downstream applications to run decoupled from both the camera's and the tracking algorithm's frame rate.

5 Evaluation

In this section, experiments that have been carried out to validate the effectiveness of the devised technique are reported. In particular, the experimental setup is first introduced. Afterwards, results of quantitative measurements performed are reported and discussed.

5.1 Experimental Setup

As it was previously stated, the purpose for developing the proposed tracking algorithm was to support robot tracking in mixed reality gaming scenarios. In particular, the devised algorithm has been integrated in the platform proposed in [17][18], whose architecture is shown in Fig. 6.

The core system (Fig. 6, left) is implemented in the Robot Operating System (ROS) [38], and currently features tracking and control of the robot, and tracking of people (players in the playground). Games are

developed in the Unity game engine[4], using a custom API to interface with the mixed reality gaming system (Fig. 6, right). The decoupling between the core system and the game engine allows the two components to be run each on a dedicated machine, possibly in the cloud (in fact, the proposed architecture is fully aligned with the concept of "connected robotics" within the larger Internet of Things paradigm [39]).

Robot tracking information, obtained by fusing camera and odometry data, is passed to a feedback-based controller. The controller generates the necessary commands to move the robot on target. All that is required is to set a target position that the robot will be driven to. Beside the floor-facing camera used to track the robot, the platform exploits a second camera (framing the whole gaming area) and the Microsoft's Kinect SDK[5] to track the players' body and gestures, which can be used to implement natural interaction mechanisms in the game. All of these data sources are made available within the game engine.

A simple collaborative game was developed to test the devised tracking method under many critical conditions. Game design followed the guidelines of the Mechanics, Dynamics, Aesthetics (MDA) framework [40]. Gameplay examples are shown in Fig. 8–10.

The main aesthetic (i.e., experience) factor the game wants to transmit is a feeling of closeness and friendliness with the robot. As such, the characters of the game are their real counterparts: a robot and its human owner. Aesthetic patterns of *team accomplishments, team strategy identification and mutual experiences* [41] were deemed the most appropriate, as they instill a sense of cooperation and teamwork that should encourage the rise of the main aesthetic. To this end, the game was designed in such a way that the human and robot have to work together to be able to win. *Cooperation and team play* are dynamic patterns that come from the mechanics of *mutual enemies* and *mutual goal*. The game proposes a common enemy to both human and robot players, whose goal is to defeat it.

A sense of friendship with the robot is instilled by exploiting the mechanics of helplessness and the rising dynamics of rescue. At the beginning of the game, the robot character is low on battery (actually, the physical robot is not, battery level is a simulated element of the game). Thus, it cannot move on its own. The only way to make the robot move is for the player to "tilt" a projected floor they are moving onto and make the robot roll in the desired direction. For visualization purposes, the tilting plane is represented as a grid. The perspective setup of the virtual camera gives the illusion that the floor is tilting. The player controls tilt through his or her body position: moving towards the edge of the playground will tilt the plane in that direction. By collecting batteries that randomly pop up, the robot character gets charged more and more. On the one hand, the human player should feel closer to the robot, as through his or her actions he or she is "recharging" (*rescuing*) the robot. On the other hand, thanks to *asymmetric abilities* (human and robot player have different abilities) that are enabled as the battery gets charged, the robot is able to provide *delayed reciprocity* by helping the human player with its newly found powers. The mechanics of *asymmetric abilities* also gives rise to *team combos*, allowing, e.g., the player to directly drive the robot towards the enemy by pointing it using his or her hand. These elements contribute to the aesthetics of *team strategy identification* (e.g., by trying different *team combos*) and, when the strategy is successful, *team accomplishments*. The various game states are described in Appendix C.

The proposed game presents a challenging environment for tracking the robot. First, to display the tilting plane, the projection covers the whole playground, which may interfere with visual tracking. Additionally, the batteries the robot can collect are intentionally drawn on a white background, which can cause the visual tracking algorithm to lock on false positives. Furthermore, the robot and the player can move anywhere within the playground, thus occlusions are quite common. Finally, the robot is driven by a feedback-based controller (a Proportional Integral Derivative controller), which requires the estimated pose to be available at all times, or it will not output a control effort. A purely visual tracking algorithm could not, by definition, provide a position estimate when the robot is occluded. These challenges were designed to test the algorithm against the requirements laid out in Section 3 under real gaming conditions.

5.2 Data Collection and Analysis

In order to evaluate the performance of the proposed algorithm as done in the selection phase (Section 3), 10 sequences were recorded during gameplay. The game instances were run using the proposed algorithm, as it was expected to be capable to provide the most reliable tracking and, hence, the best gaming experience. Afterwards, the sequences were manually labeled to construct the ground truth. In addition, both the odometry and the initial transformation matrix between *camera_frame* and *odom_frame* were recorded.

Since the proposed algorithm was built upon TLD (which provided the best performance in the scenario of interest) and, as shown in Section 2, other techniques are not directly applicable to the considered setup, a comparative analysis was performed against original TLD as well as odometry-only tracking by using the metrics in [20]. Odometry-only tracking is implemented by projecting odometry data into the *camera_frame*, using the previously recorded transformation matrix. Results, which were collected by launching all the algorithms on the recorded game

4. https://unity3d.com/
5. https://developer.microsoft.com/en-us/windows/kinect

sessions under the same conditions illustrated in Section 3 are given in Fig. 7. Sequences with tracking results are available for download[6].

The proposed algorithm was run and evaluated in two different modes: a so called "robust" mode and a "continuous" mode. In robust mode, the final tracking output is the result of data fusion between TLD and odometry. If confidence in the result is not high enough (e.g., because odometry and TLD do not agree), the algorithm declares that the tracking is lost. However, if TLD finds a match again, the algorithm is able to recover (it reverts to pure TLD). In Fig. 7, results for this mode are referred to as ROB.

Continuous mode was developed in order to improve the experience while playing. In fact, in this mode the algorithm keeps track of the latest valid transformation matrix between the camera and the odometry coordinate systems. The algorithm works exactly as in robust mode, with the exception that tracking is never declared as lost. If it is not possible to determine the robot's position visually, the algorithm will output its position as reported by the odometry, by applying the latest transformation matrix available. The effect is that, if visual tracking is lost, the reported position will start to diverge from the robot's actual position. However, the output is continuous, which allows the feedback controller to keep driving the robot and the player to play the game, even though not with the best accuracy possible. In Fig. 7, this mode is referred to as CONT.

As it can be seen by looking at the plots, continuous mode achieves the best performance in terms of average location error (Fig. 7b), since it always outputs a position close to the robot. Performance is slightly worse when considering success rate (Fig. 7a), because the output bounding box does not overlap with the ground truth for most of the time. Robust mode, on the other hand, performs better in terms of overlap, since the output of the algorithm is the result of data fusion between the TLD and odometry position estimates. Robust mode is able to track a lower number of frames, but more accurately. Odometry-only and TLD-only tracking (ODOM and TLD, in Fig. 7, respectively) achieve the worst performance. The former shows the same characteristics as the proposed algorithm in continuous mode, but does not update the transformation matrix, thus suffering from a larger drift. The latter is very sensible to occlusions (once occluded, it is rarely able to recover the correct target), and false positives.

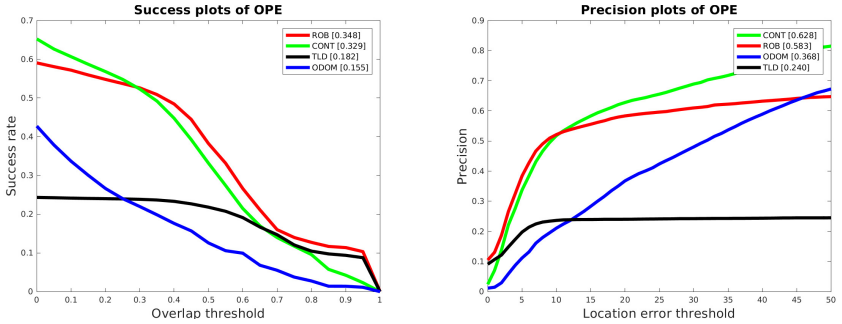
Fig. 8–10 report several representative frames from selected gaming sequences. In Fig. 8 it is possible to see how odometry-only tracking suffers from drift. At the beginning of the sequence (frame #98), before the robot starts moving all the algorithms are reporting the correct position. They are all initialized with the same bounding box and transformation matrix between *odom* and *camera_frame*. Then the robot starts moving, and after just a few frames the odometry-only tracking (blue box in the figure) drifts off the target. On the contrary, pure TLD (black) and the proposed algorithm (red and green) keep tracking the target accurately. The error that odometry reports is due to two factors: the uncertainty in the initial transformation matrix (small orientation errors in the matrix translate into large position errors), and the odometry's intrinsic drift. As said, the former factor can be easily dealt with by automatically driving the robot for few second before actually starting the game.

Fig. 9 shows the behavior of the algorithms when facing a possible false positive. As it can be seen, pure TLD is locking onto different false positives occurring at frames #269 and #649. In both cases, the proposed algorithm keeps tracking the robot correctly, whereas odometry-only tracking presents the usual drift.

Finally, Fig. 10 illustrates the behavior of the proposed algorithm when dealing with an occlusion. Before the occlusion occurs (frame #521), the proposed algorithm is tracking the robot accurately. Odometry-only tracking is slightly off-target, whereas pure TLD is reporting that tracking is lost. Throughout the whole sequence, pure TLD will not be able to recover tracking. Odometry-only tracking, as it can be expected, is not affected by occlusions and keeps reporting a position with an approximately constant error. As the robot moves behind the occluding player's body from frame #521 to frame #531, the proposed algorithm keeps tracking its position using the odometry data source. Differently than the odometry-only tracking, the proposed algorithm keeps updating the transformation matrix until the visual tracking source fails. At frame #540 the robot becomes visible again. As it can be noticed, the position reported by the proposed algorithm is very close to ground truth. Finally, at frame #546, the visual tracking source recovers the target, thus resetting the cumulated drift and reporting once again the correct position.

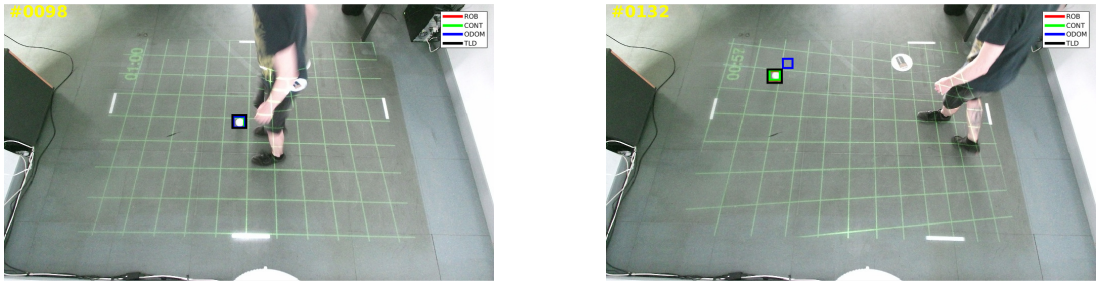
In terms of frame rate, the proposed algorithm can be tuned depending on actual needs. In general, performance is limited by the frame rate of the camera, of the modified TLD algorithm (comparable to that of the original TLD) and of the robot's odometry. Continuous mode is entirely decoupled from the visual tracking algorithm. Therefore, it runs at the frame rate of the robot's odometry (~ 50 Hz for the robot used in the experiments). It should be pointed out that, even though continuous mode can run up to the frame rate of the robot's odometry, the transformation matrix is updated at TLD's frame rate. Between updates, the algorithm is running purely on odometry data. Robust mode is tied to TLD's frame rate, which in turn is limited by the camera. In the experimental setup, the limit was the camera, which allowed the algorithm to run at about ~ 15 Hz on the considered machine.

6. https://goo.gl/9mXHbC



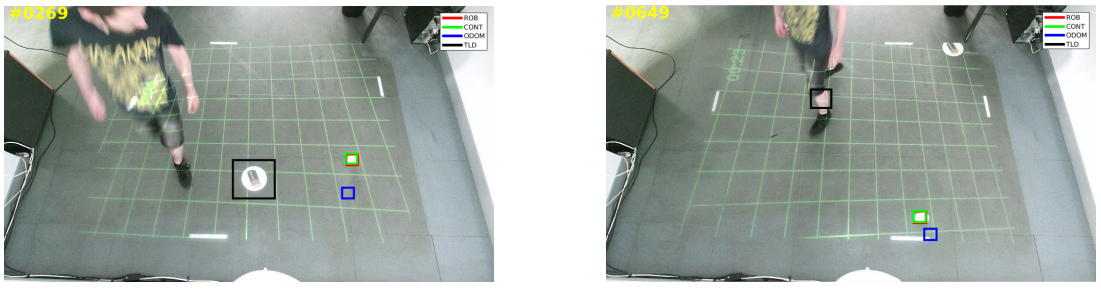
(a) (b)

Fig. 7. Results with the proposed algorithm (ROB and CONT mode), original TLD algorithm and odometry-only tracking: a) success plot, sorted by AUC, b) precision plot, sorted by precision at 20px error threshold.



(a) (b)

Fig. 8. Frames extracted from the gaming sequences showing the performance of the proposed algorithm (ROB and CONT mode) compared to the original TLD algorithm and odometry-only tracking: a) correct tracking with all the algorithms, b) drift experienced with the odometry-only tracking.



(a) (b)

Fig. 9. Frames extracted from the gaming sequences showing the performance of the proposed algorithm (ROB and CONT mode) compared to the original TLD algorithm and odometry-only tracking: a) and b) TLD is locked to a false positive, odometry-only tracking is affected by drift (as in Fig. 8).

6 Conclusion

This paper presented the design, realization and testing of an online robot tracking algorithm which exploits data fusion between visual tracking and odometry. The aim was to develop a robust robot tracking method for a mixed reality gaming platform. The platform allows human and robot players to play and interact within a projected environment. This particular setup presents a set of challenges (in terms of tracking) that need to be addressed in order to provide an effective gaming experience.

The proposed algorithm extends the well-known TLD online visual tracking technique by using the robot's own odometry in order to boost performance. In particular, the devised solution acts as a filter within the native learning mechanism of TLD, by validating potential targets against odometry data. This way, the online learning capabilities of TLD are improved, as the classifier is trained with a fewer number of false positives, thus reducing model drift. Additionally, odometry provides a search window for the detector to look for potential targets, as opposed to searching the whole frame, as the original TLD implementation actually does. Finally, odometry com-

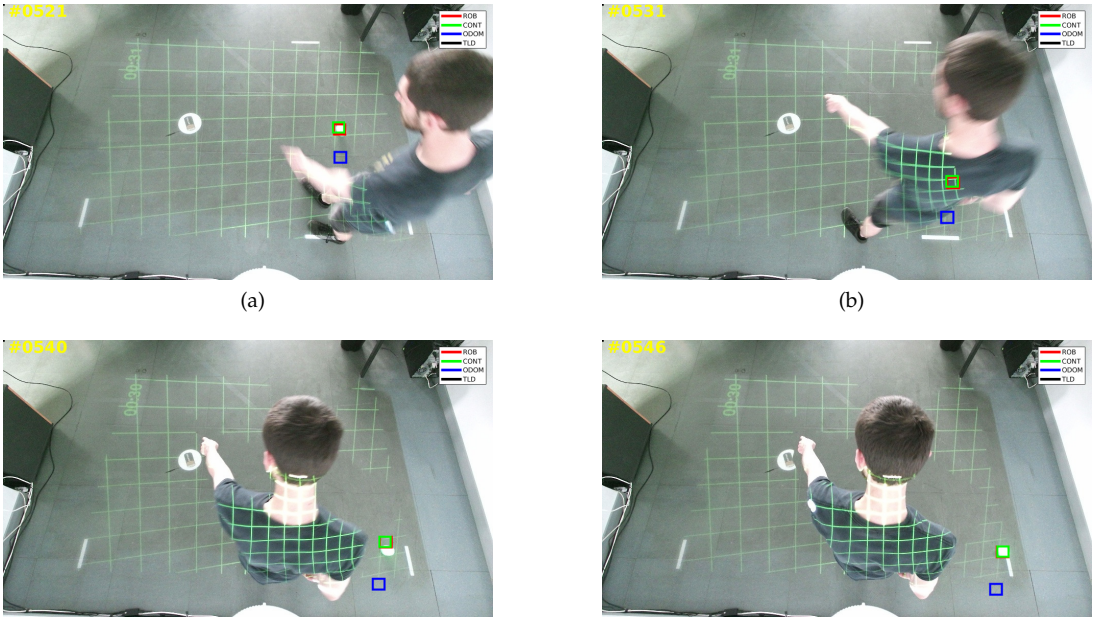


Fig. 10. Frames extracted from the gaming sequences showing the performance of the proposed algorithm (ROB and CONT mode) compared to the original TLD algorithm and odometry-only tracking: a) all the algorithms except TLT are tracking the target (with some drift, in the case of odometry-only tracking), b) target is occluded, odometry-only tracking still works with almost constant displacement, the proposed algorithm keeps tracking the target using the odometry source, c) target is visible again, tracking obtained with the proposed algorithm is close to actual robot's position, d) the visual tracking source recovers the tracking, drift cumulated by the proposed algorithm is reset and tracking is again on target.

plements visual tracking in case of occlusions or out-of-frame situations. At the same time, visual tracking allows to reset drifts introduced by the odometry (e.g., due to sensor noise, integration errors, slippage, etc.).

The proposed algorithm proved to outperform both the original TLD technique and odometry-only tracking, confirming the validity of the choices made.

The largely encouraging results motivate further research aimed to the characteristics of current design. In fact, at present only a part of TLD native functioning is considered, i.e., N-expert learning: the devised algorithm is programmed only to better discard false positives, thus reducing incorrect model updates in the classifier. False negatives could be validated as well, with a possible increase in tracking performance. However, a higher accuracy than the one provided by the odometry would be required. Furthermore, the algorithm is robust against changes in model appearance only as much as the original TLD. Similarly, proposed improvements are only effective when Kalman filter is converging; when it is not (e.g., after a long occlusion), performance reverts to that of TLD. Lastly, analysis of the recorded sequences revealed that the projected environment is the main cause of failure of the visual tracking algorithm.

For these reasons, future work will include the addition of depth information as a further data source to the algorithm, which will open the way to implementing a P-expert process. The N-expert proposed in this paper could benefit from depth information too, as it would allow the algorithm to recognize whether a potential candidate for tracking is in fact a projection on the floor rather than a solid object. Applicability of the designed method to other collaborative robotics domains mentioned in the initial sections of the paper will be explored as well.

ACKNOWLEDGMENTS

The authors wish to acknowledge TIM, which partially funded the activities and Giacomo P. Milano, who was involved in an early study on this subject.

REFERENCES

[1] World Economic Forum's Meta-Council on Emerging Technologies, "Top 10 Emerging Technologies of 2015," 2015.
[2] J. Forlizzi and C. Di Salvo. "Service robots in the domestic environment: A study of the roomba vacuum in the home," in *Proc. 1st Conf. on Human-Robot Inter.*, pp. 258–265, 2006.
[3] Y. Y. Huang, Z. L. Cao, S. J. Oh, E. U. Kattan, and E. L. Hall, "Automatic operation for a robot lawn mower," in *Proc. SPIE 0727, Mobile Robots I*, vol. 344, 1987.
[4] F. Masahiro, H. Kitano, and T. Doi, "Robot entertainment," Robots for kids: Exploring new technologies for learning, pp. 37–72, 2000.
[5] X. Yu and J.B. Weinberg, "From the Guest Editors - Robotics in education: New platforms and environments," *IEEE Robotics & Automation Magazine*, vol. 10, no. 3, pp. 3–3, 2003.

[6] A. Steinfeld, T. Fong, D. Kaber, M. Lewis, J. Scholtz, A. Schultz, and M. Goodrich, "Common metrics for human-robot interaction," in *Proc. 1st Conf. on Human-Robot Inter.*, pp. 33–40, 2006.
[7] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part I," *IEEE robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.
[8] K.Kawamura, R. T. Pack, M. Bishay, and M.Iskarous, "Design philosophy for service robots," *Robotics and Autonomous Systems*, vol. 18, no. 1, pp. 109–116, 1996.
[9] I. Fernández, M. Mazo, J. L. Lázaro, D. Pizarro, E. Santiso, P. Martín, and C. Losada, "Guidance of a mobile robot using an array of static cameras located in the environment," *Autonomous Robots*, vol. 23, no. 4, pp. 305–324, 2007.
[10] A. Hoover and B.D. Olsen, "Sensor network perception for mobile robotics," in *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 342–347, 2000.
[11] M. Broxvall, M. Gritti, A. Saffiotti, Beom-Su Seo, and Young-Jo Cho, "PEIS ecology: Integrating robots into smart environments." in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2006.
[12] D.J. Cook, "Multi-agent smart environments," *Journal of Ambient Intell. and Smart Environments*, vol. 1, no. 1, pp. 51–55, 2009.
[13] Joo-Ho Lee, N. Ando, T. Yakushi, K. Nakajima, T. Kagoshima, and H. Hashimoto, "Adaptive guidance for mobile robots in intelligent infrastructure," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 90–95, 2001.
[14] P. Steinhaus, M. Ehrenmann, and R. Dillmann, "MEPHISTO A Modular and Extensible Path Planning System Using Observation," in *Proc. Conf. on Comp. Vision Sys.*, pp. 361-375, 1999.
[15] P. Coulton, D. Burnett, A. Gradinar, D. Gullick, and E. Murphy, "Game design in an Internet of Things," *Transactions of the Digital Games Research Association*, vol. 1, no. 3, 2014.
[16] D. Martinoia, D. Calandriello, and A. Bonarini, "Physically Interactive RoboGames: Definition and design guidelines," *Robotics and Auton. Systems*, vol. 61, no. 8, pp. 739–748, 2013.
[17] M.L. Lupetti, G. Piumatti, and F. Rossetto, "Phygital play HRI in a new gaming scenario," in *Proc. 7th Int. Conf. on Intelligent Technologies for Interactive Entertainment*, 2015.
[18] G. Piumatti, F. Lamberti, and A. Sanna, "Spatial Augmented Reality meets robots: Human-Machine Interaction in cloud-based projected gaming environments," in *Proc. 34th IEEE Int. Conf. on Consumer Electronics*, pp. 1–4, 2017.
[19] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-learning-detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1409–1422, 2012.
[20] Y. Wu, J. Lim, and Ming-Hsuan Yang, "Online object tracking: A benchmark," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2411–2418, 2013.
[21] S. Coradeschi and A. Saffiotti, "Symbiotic robotic systems: Humans, robots, and smart environments," *IEEE Intelligent Systems*, vol. 21, no. 3, pp. 82–84, 2006.
[22] S. Karaman, A. D. Bagdanov, L. Landucci, G. D'Amico, A. Ferracani, D. Pezzatini, and A. Del Bimbo, "Personalized multimedia content delivery on an interactive table by passive observation of museum visitors," *Multimedia Tools and Applications*, vol. 75, no. 7, pp. 3787–3811, 2016.
[23] S. Thrun, "Robotic mapping: A survey," Exploring Artificial Intelligence in the New Millennium, pp. 1-35, 2002.
[24] J. Borenstein, L. Feng, and H. R. Everett, Navigating mobile robots: Systems and techniques. AK Peters, Ltd., 1996.
[25] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proc. 7th Int. Joint Conf. on Artificial Intelligence*, pp. 674–679, 1981.
[26] J. J. Leonard and H. F. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 376–382, 1991.
[27] W. Burgard, D. Fox, and D. Hennig, "Fast grid-based position tracking for mobile robots," in *Proc. Annual Conf. on Artificial Intelligence*, pp. 289-300, 1997.
[28] W. Burgard, A. Derr, D. Fox, and A. B. Cremers, "Integrating global position estimation and position tracking for mobile robots: the Dynamic Markov Localization approach," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 1998.
[29] A.R. Cassandra, L. Pack Kaelbling, and J.A. Kurien, "Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation," in *Proc. IEEE/RSJ Conf. on Int. Robots and Sys.*, 1996.
[30] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, no. 1, pp. 99–141, 2001.
[31] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *Proc. 18th Nat. Conf. on Artificial intelligence*, pp. 593–598. 2002.
[32] Y. Hada, E. Hemeldan, K. Takase, and H. Gakuhari, "Trajectory tracking control of a nonholonomic mobile robot using iGPS and odometry," in *Proc. IEEE Int. Conf. on Multisensor Fusion and Integration for Intelligent Systems*, 2003.
[33] D. Pizarro, M. Mazo, E. Santiso, M. Marron, D. Jimenez, S. Cobreces, and C. Losada, "Localization of mobile robots using odometry and an external vision sensor," *Sensors*, vol. 10, no. 4, pp. 3655–3680, 2010.
[34] H. Costa, P. Cebola, T. Cunha, and A. Sousa., "A mixed reality game using 3Pi robots-"PiTanks"," in *Proc. 10th Iberian Conf. on Information Systems and Technologies*, 2015.
[35] M. Kojima, M. Sugimoto, A. Nakamura, M. Tomita, H. Nii, and M. Inami, "Augmented Coliseum: An Augmented Game Environment with Small Vehicles," in *Proc. 1st IEEE Int. Workshop on Horizontal Interactive Human-Computer Systems*, 2006.
[36] J. Leitner, M. Haller, K. Yun, W. Woo, M. Sugimoto, and M. Inami, "IncreTable, a mixed reality tabletop game experience," in *Proc. Int. Conf. on Adv. in Comp. Ent. Tech.*, pp. 9–16, 2008.
[37] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," *Intelligent Autonomous Systems*, vol. 13, pp. 335–348, 2016.
[38] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source Robot Operating System," in *Proc. Works. on Open Source Soft.*, vol. 3, no. 3.2, 2009.
[39] Y. Hui, Z. Su, and S. Guo, "Utility based data computing scheme to provide sensing service in Internet of Things," *IEEE Transactions on Emerging Topics in Computing*, in press.
[40] R. Hunicke, M. LeBlanc, and R. Zubek, "MDA: A formal approach to game design and game research," in *Proc. AAAI Workshop on Challenges in Game AI*, vol. 4. no. 1, 2004.
[41] K. Bergström, S. Björk, and S. Lundgren, "Exploring aesthetical gameplay design patterns: camaraderie in four games," in *Proc. 14th Int. Academic MindTrek Conf.: Envisioning Future Media Environments*, pp. 17–24, 2010.

Giovanni Piumatti graduated in Computer Engineering from Politecnico di Torino. His research interests are in service robotics, computer graphics and gaming.

Fabrizio Lamberti (M'02-SM'14) is an Associate Professor at Politecnico di Torino. His research interests are in the areas of computer graphics and intelligent systems. He serves as an Associate Editor for IEEE Transactions on Emerging Topics in Computing and for IEEE Consumer Electronics Magazine. More info at http://staff.polito.it/fabrizio.lamberti/

Andrea Sanna is an Associate Professor at Politecnico di Torino. His research is in the areas of computer graphics, virtual reality, distributed computing, and computational geometry. He is a senior member of the ACM. More info at http://sanna.polito.it/

Paolo Montuschi Paolo Montuschi (M?90-SM?07-F?14) is a Professor at Politecnico di Torino. His interests include computer arithmetic, graphics and architectures. He is a HKN member and serves as Editor-in-Chief of the IEEE Transactions on Computers and chair of the Computer Society Awards Committee. More info at http://staff.polito.it/paolo.montuschi/