Practical method for the design of pretensioned fully grouted rockbolts in tunnels

Supervised global path planning for mobile robots with obstacle avoidance

Marina Indri, Corrado Possieri,
Fiorella Sibona, Pangcheng David Cen Cheng
Dipartimento di Elettronica e Telecomunicazioni
Politecnico di Torino
Corso Duca degli Abruzzi 24, 10129 Torino, Italy
{marina.indri,corrado.possieri,
fiorella.sibona, pangcheng.cencheng}@polito.it

Vinh Duong Hoang
School of Information and Communication Technology
Hanoi University of Science and Technology
Hanoi, Vietnam
vinhdh@outlook.com

***Abstract*—The presence of mobile agents in the industrial environment is growing, introducing specific safety issues in their path planning. This paper proposes the implementation of a three-level path planning procedure, which allows: (i) the imposition of a set of waypoints, tending to a safe path, generated by a supervisory planner on the basis of a static map of the environment (not necessarily fully updated), (ii) the generation of a global path including such waypoints exploiting a cost-based algorithm, taking into account also the obstacles not included in the static map, but detected at the beginning of the global planning phase, and (iii) the avoidance of dynamic obstacles appearing during the robot motion, thanks to the action of a local planner. The procedure has been experimentally tested to plan the motion of a differential mobile robot.**

***Index Terms*—Mobile robots, path planning, obstacle avoidance.**

I. Introduction and state of the art

Mobile robots are widely used in many applications, such as surveillance, transportation and automation in industry. In order to guarantee the safety of both the operators and the robots within the industrial environment, path planning with obstacle avoidance is required. While classical AGVs (Automated Guided Vehicles) are typically constrained to follow predefined paths, the adoption of autonomous mobile robots in the industrial context implicitly introduces some specific issues in the path planning operation, such as the existence of preferred areas or paths within the available free space for a safer simultaneous presence of humans, robots and machineries in the plant, and a more efficient organization of the working process itself.

In general, path planning for mobile robots is mainly carried out at two levels: global planning and local planning. The global planner computes (off-line) the path for a well known environment map, so allowing the robot to move from a starting point to the destination point avoiding the known static obstacles, while optimizing specific objectives, like searching the shortest path (so to reduce time traveling and energy consumption). The local planner updates (on-line) the path of the robot, taking into account the information coming from its sensors, with the goal of letting it follow the path generated by the global planner if possible, but modifying it to avoid unexpected obstacles, which were not taken into account.

Depending on the environment and the tasks to be performed, many researchers proposed and developed several algorithms for path planning.

For example, there are heuristic-based algorithms, such as the Dijkstra, A* and D* algorithms. The Dijkstra algorithm [1] computes the shortest path between two nodes in a map; however, its execution needs to evalutate too many nodes, making it an overall low-efficiency process. This issue is solved in the A* algorithm [2], a path planner based on Dijkstra, which takes into account a function that evaluates the distances between the nodes and their cost to reach the goal point. Nevertheless, A* takes relatively long computational time to execute, making it not suitable for performing sequential tasks in real time [3]. Such response time constrained tasks can be carried out by using D* [4], a dynamic re-planning algorithm based on the A*, considering the direction of the robot position as expressed by a series of states, whose values are updated each time the elements on the map change. Nonetheless, this still requires high computational effort, since the re-planning phase needs to calculate twice each state on the environment [5].

The computational burden of the three algorithms above can be mitigated by using probabilistic methods, such as the Probability Roadmap (PRM) [6] and the Rapidly-exploring Random Tree (RRT) [7]. In PRM, the free spaces on the world map are randomly sampled, and the planner tries to connect the generated points into a roadmap feasible motion. This method is commonly used in large maps, where the use of other algorithms may increase the computational cost. However, this method does not guarantee the shortest path [8]. In RRT, feasible trajectories are obtained growing expanding *trees* starting from the initial point, going through random points called *seeds*, until the tree is connected to the goal point. Since the release of the original RRT algorithm, researchers have proposed a variety of improved methods. For example, the method presented in [9] consists in "planting" a tree in both the starting and the goal points, and then expanding both trees in the whole world map until an intersection point is found; the algorithm presented in [10] controls the tree edges growth direction and density of the RRT* variant (an RRT version

The research activity has been partially supported by the HuManS – Human-centered Manufacturing Systems project, funded by Regione Piemonte within the MIUR-POR FESR 2014/2020 funding program.

converging to the shortest path introduced by S. Karaman and E. Frazzoli [11]). The main drawback of sampling based stochastic searches is that, in general, path cost is not taken into account, leading to a solution that is not guaranteed to be optimal [12].

Other planning algorithms are inspired by natural phenomena. One of the most popular is the Genetic Algorithm (GA) [13], based on the natural selection theory and applied as a path searching algorithm in the field of robotics. The cost function for computing the best path is structured similarly to a chromosome, where each location is considered as a gene [14].

Another widely used approach involves computing off-line the considered safe path and controlling the robot in such a way that it goes to the destination point following a set of waypoints. In this case, it is possible to use the feedback linearization technique in order to design a control system for path following, like in the fuzzy logic based control architecture proposed in [15]. Similarly, some researchers proposed navigation functions based on artificial potential fields to solve the robot motion planning [16], and a feedback control law ensuring that the robot reaches the destination point, avoiding obstacles [17].

This paper exploits the features of the algorithm developed in [18] (and validated there only in a pure theoretical context) to address the peculiar characteristics of path planning in an industrial-like scenario, proposing a possible implementation, executed by a differential robot, within a complete planning procedure based on three levels. In particular, the proposed solution integrates the common two-level planning hierarchy with a third level, which aims at reaching a given goal position traversing a virtual track previously identified as safe.

The paper is organized as follows: Section II presents the proposed procedure, at first providing some theoretical details about the supervisory planner, then describing the full/practical algorithm implementation including the obstacle avoidance capability. The testing of the procedure is unfolded in Section III, where the results obtained from three different test cases are reported. Finally, Section IV draws some conclusions and open issues.

II. Supervisory algorithm for path following with obstacle avoidance

The present section goes through the description of the developed and tested path planning procedure for path following with obstacle avoidance.

The scenario is that of a mobile robot roaming within a closed space, e.g., a warehouse or factory plant, where some predefined routes are considered safe for the mobile platform motion (no unexpected obstacle is assumed): the a-priori desired path computed by the algorithm, proposed in [18] and recalled in Section II-A, tends by construction to a curve representing a safe route that takes on the role of what guide tapes and wires represented in the most traditional industrial mobile navigation set-up.

The path planning algorithm proposed in this paper has a hierarchical structure based on three levels: the Supervisory Global Planner (SGP), the Global Planner (GP) and the Local Planner (LP). While traversing this hierarchy (Figure 1), we get to an enhanced robot environment awareness, a sort of transition from a "blind" planning (only based on the static map) to a dynamic-obstacles aware system.

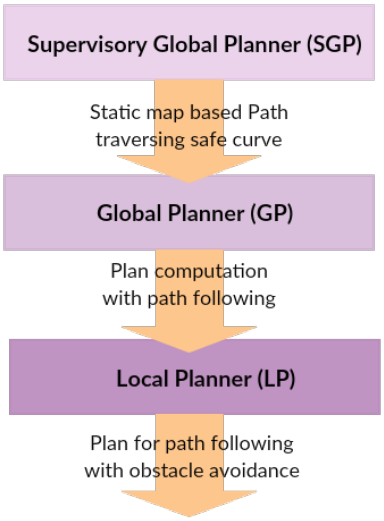

Fig. 1. Planning hierarchy schematics.

In fact, since the static map of the environment used at the supervising level could be not fully updated, the planned waypoints are integrated at a lower level within the GP, which lets the robot actually follow the planned trajectory (if possible) thanks to a cost-based algorithm. The avoidance of unexpected obstacles that may appear during the robot motion is left to the standard, lowest level LP, which further modifies on-line the path computed by the GP, when necessary.

A. Supervisory algorithm theoretical development

In this section, we review the algorithm proposed in [18] to design the path of a mobile robot in a known environment. The goal of this algorithm is to generate a path that is *collision free* (i.e., it does not intersect the curves describing the boundaries of the obstacles) and that tends to a preassigned algebraic curve, which is considered safe for motion.

In particular, assume that the aim of the supervisory algorithm is to obtain a path that tends to the planar curve

$$\mathcal{V} := \{x \in \mathbb{R}^2 : p(x) = 0\}, \tag{1}$$

where $p(x)$ is a given polynomial function. By using the algorithm given in [19] (omitted here for brevity), it is possible to compute two vector fields $\phi(x)$ and $\psi(x)$, whose entries are polynomials in x, such that the curve $\mathcal{V}$ given in (1) is attractive and invariant with respect to the dynamical system

$$\dot{\xi} = \phi(\xi) + \psi(\xi)\alpha(\xi), \tag{2}$$

where $\alpha(\xi)$ is an arbitrary function. Although the trajectories of system (2) converge to the set $\mathcal{V}$, they are not necessarily collision free. Thus, in order to ensure collision avoidance, the results given in [19] are coupled with classical navigation functions [16].

Let $b(x)$ be a function describing the boundaries of the obstacles (i.e., $b(\xi) = 0$ implies that the point ξ belongs to the boundary of one of the obstacles) and let

$$\mathcal{W} := \{x \in \mathbb{R}^2 : b(x) > 0\}$$

be the workspace of the mobile robot, which is assumed to be a connected set. Furthermore, we assume that

$$\mathcal{V} \cap \{x \in \mathbb{R}^2 : b(x) = 0\} = \emptyset,$$

i.e., that $\mathcal{V}$ is actually safe for motion. Thus, define

$$r(\xi) := \frac{p^2(\xi)}{b(\xi)}, \qquad \eta(\xi) := \left(\frac{\partial r(\xi)}{\partial \xi}\right)^\top.$$

Note that, by construction, the function $r(\xi)$ is nonnegative for all $\xi \in \mathcal{W}$, it is zero if $\xi \in \mathcal{V}$, and it tends to $+\infty$ if ξ tends to $\mathcal{W}$. This implies that, letting $\beta(\xi)$ be a nonnegative function such that $b(\xi) = 0 \implies \beta(\xi) \neq 0$, the trajectories of

$$\dot{\xi} = -\eta(\xi)\beta(\xi) \tag{3}$$

are collision free (see [18, Prop. 2]).

Hence, the supervisory algorithm is obtained coupling systems (2) and (3). Namely, letting $\zeta(\xi)$ be an arbitrary function (which is amenable for further optimization, see [17]) and k be a positive constant, it is possible to guarantee that the trajectories of the dynamical system

$$\dot{\xi} = b^2(\xi)\phi(\xi) + b^2(\xi)\psi(\xi)\zeta(\xi) - k\, p^2(\xi)\eta(\xi) \tag{4}$$

tend to $\mathcal{V}$ while avoiding collisions with the obstacles.

It is worth noticing that, if the objective is to steer the robot to $\mathcal{V}$ and, additionally, to let it stop for some $\bar{\xi} \in \mathcal{V}$, then such a goal can be pursued by designing the function $\zeta(\xi)$ in (4), following [19, Alg. 2].

B. Integration with ROS Global and Local Planners

In this section some high-level software details about the implementation of the SGP and the integration of the dynamic obstacle avoidance capability are given. In order to adapt the theoretical algorithm to a physical implementation, the supervisory algorithm has been executed using the real environment map, in place of a hypothetical one, taking into account the physical dimensions of the performing robot. Note that, at this level of description, we will omit map details and parameters tuning/adjustment specific to the chosen robot (more details about the physical set-up are given in Section II-C) to highlight the general validity of the approach.

The SGP algorithm has been written in MATLAB, while the navigation driving the robot exploits ROS (Robot Operating System) [20] tools. In particular, the ROS *Navigation Stack* [21] provides the user with off-the-shelf packages ready for mapping, localization, navigation and reference frames tracking. To ensure a correct interpretation of the supervisory algorithm output (i.e., a set of 2D desired points converging to the preassigned safe curve to be traversed on the static map), the ROS `/world` frame origin (reference frame for the whole ROS coordinate frames transform tree, managed by the `tf` package) has been aligned to the frame origin used in MATLAB. Then, the output plan has been conveniently packed in an array of desired positions stored on the ROS Parameter Server, exploiting the MATLAB Robotics System Toolbox™ [22]. The ROS framework provides the `actionlib` library stack [23], that allows the user to interact through a standardized interface with preemptable tasks, which in our case correspond to the desired poses for the mobile platform to reach. A custom Python ROS node is in charge of sending through a ROS Action Client a request to the Action Server, via a message containing the details of the next goal position to be reached. Note that ROS actions represent the ideal Client-Server-based mechanism for goal achieving, since while the whole operation is brought on, a feedback message about its status can be sent to the client node. A sketch of the software set-up is presented in Figure 2.

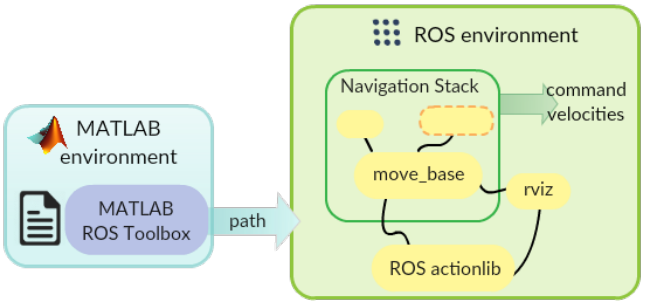

Fig. 2. Software setup for implementation of the proposed algorithm.

ROS package `move_base`, as part of the ROS navigation stack, provides several ready-to-use global planners. In our case, the GP main role is that of generating a plan that must be as close as possible to the path computed by the SGP. On the other hand, the LP algorithm is in charge of locally updating the robot trajectory, computing a short-term plan based on a *local cost map*, taking into account a predefined portion of the GP-generated long-term plan, and using sensors' data to guarantee obstacle avoidance of unexpected objects (either moving or fixed). Moreover, the whole planning performance depends on a set of *cost map parameters*, which can be accordingly tuned based on the final planning aim and on the physical characteristics of the robot.

By default, the global planner loaded by ROS adopts the Dijkstra's algorithm, which finds the shortest path from the starting pose to the desired destination. Another widely used algorithm for pathfinding is the A* algorithm. A* search algorithm is a combination between Dijkstra's algorithm and Best First Search algorithm, meaning it is a weight-based process, or informed algorithm, where the graph is explored and expanded only in nodes resulting convenient, based on an assigned cost with heuristic content. Indeed, A* inherits the benefits of a uniform-cost search from Dijkstra's, adding heuristics to efficiently find an optimal solution in less time [24].

We have developed a GP algorithm built upon the A* cost-based structure: the default global planner of the `navfn` package has been replaced exploiting the ROS plugin mechanism, in order to have access to a simpler standalone piece of code. As the ROS global planner plugin implementation guidelines

recommend, we have used the ROS *cpp* library to adhere to the `nav_core::BaseGlobalPlanner` C++ interface provided by the `nav_core` package (by overriding some specific methods) and employed by the `move_base` package to drive the mobile platform. With the purpose of forcing the robot to follow the set of waypoints computed by the SGP within the GP, the proposed algorithm assigns low costs to the preferred points to be traversed.

Before going through the description of the resulting algorithm, we will briefly outline the cost mechanism of the A* search algorithm. Consider a pathfinding problem, where an optimized path must be found between two points on a 2D cost map: at each main loop, starting from the source cell the algorithm expands the most suitable, and for construction most "convenient", cell depending on a function $f(c)$, defined as

$$f(c) = g(c) + h(c) \tag{5}$$

where c is the currently expanded cell, $g(c)$ denotes the cost from the source position to the current one, and $h(c)$ represents the heuristic estimated cost from the current cell to the goal (destination) cell. At each iteration the expanded cell, i.e., the cell whose neighbour cells are visited and their costs computed (Figure 3), is the one having the lowest $f(c)$.

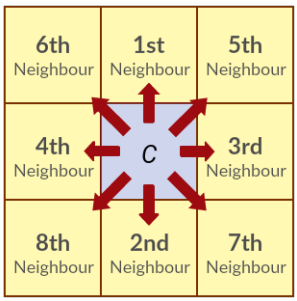

Fig. 3. Expansion of the most convenient cell.

Therefore, the $f(c)$ function encloses an estimated path cost from the source cell to the destination cell, traversing cell c. While $g(c)$ can be easily calculated, $h(c)$ computation usually involves an approximation method, since the exact heuristics computation, i.e., computing the value attained by the cost-to-go function at cell c, would take an excessive amount of time. Among the available approximated heuristic functions (e.g., Manhattan, Euclidean and diagonal distances), the Euclidean distance has been chosen; thus, the function $h(c)$ has been defined as:

$$h(c) = \sqrt{(c.x - dest.x)^2 + (c.y - dest.y)^2} \tag{6}$$

where $dest$ is the destination cell and the expressions $.x$ and $.y$ extract the fields corresponding to the x and y coordinates of the considered cell, respectively.

In order to perform path following, the idea is to use a cost-based algorithm that acts as A* when far from the SGP-computed path, while a low cost is associated to the positions produced by the SGP, so to guarantee their presence in the output plan. In particular, the passage on these positions is favoured by the introduction of a new cost h_{WP}, defined as

$$h_{WP}(c) = \sqrt{(c.x - WP.x)^2 + (c.y - WP.y)^2} \tag{7}$$

where WP is the next expected waypoint and the expressions $.x$ and $.y$ extract the fields corresponding to the x and y coordinates of the considered cell, respectively. The quantity in (7) is the Euclidean distance from the next expected waypoint, and assigns costs which penalize positions far from the desired path, based on four possible cases:

1) The currently visited neighbour cell n is a waypoint.
2) The currently expanded cell c is a waypoint.
3) All waypoints have been already traversed.
4) None of the previous conditions is valid.

Note that we consider all discrete points making up the supervisory algorithm output path as waypoints that we would like the robot to traverse when in their proximity.

More details on how the costs have been assigned to achieve an overall equilibrated and suitable cost system are given in Algorithm 1, reporting the pseudocode for the proposed algorithm. Handling of the above enumerated possible cases can be found at lines 14, 16, 18 and 20, respectively. Note that highlighted lines point out edits with respect to the original A*, to have a direct comparison. The inputs to the algorithm are the environment map and the start and destination cells, while the output is represented by the computed path. The proposed GP boasts the capability of reaching a goal position not necessarily in the shortest way, but as safely as possible.

The employed local planner exploits the `TrajectoryPlannerROS` wrapper, which adheres to the `nav_core::BaseLocalPlanner` interface. This local planner version implements the so called *trajectory rollout* algorithm: the robot's control space is discretely sampled and, for each sampled velocity a forward simulation is performed using the current robot state as starting point, to predict the robot motion if the considered velocity were applied for a restricted period of time. Each simulated trajectory is assessed on the basis of some evaluation parameters, e.g., proximity to the global path, to obstacles, to the goal, and the speed. Among the evaluated trajectories, those which would cause a collision with obstacles are discarded, while the trajectory obtaining the highest score is chosen and the relative velocities are sent to the mobile robot. This procedure is looped at each motion step [25]. The LP represents the last planning level and improves the robot's awareness about its surroundings, ensuring a reaction to obstacles that are not present in the static map, or are unknown when the global planning path is computed.

Moreover, in order to ensure that the resulting path tracks as precisely as possible the desired algebraic curve, the knowledge of the LP about the GP long term path is reduced, influencing the short-term plan generation, since a wider overview on the tracked plan would result in local path optimization.

C. Hardware/Software setup and testing

In order to test the supervisory algorithm with path following and obstacle avoidance we have decided to employ a Pioneer 3DX mobile robot [26] equipped with a SICK LMS200 laser range finder [27] with 10-meter range and

scanning angle of 180 degrees, a Raspberry Pi [28] 3 Model B mounting an ARM Cortex-A53 (x4 core) CPU (1.2 GHz) and 1-GB RAM, which takes the role of a processing unit for receiving data and controlling the robot (Figure 4).

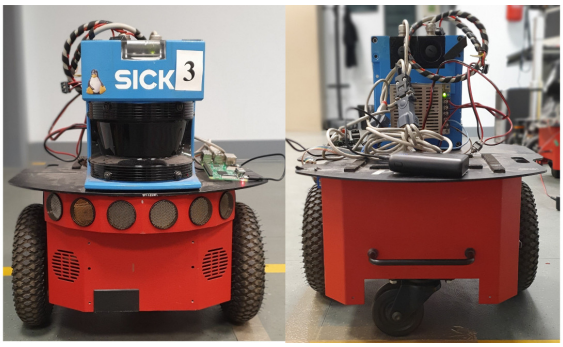

Fig. 4. Complete robot set-up used for the practical execution of the algorithm.

Note that the physical specifications of the robot (Figure 5) are crucial for modelling the kinematic equations necessary for executing the supervisory algorithm, to suitably adapt ROS visualization and map inflation parameters, and to generate proper commands taking into account the maximum allowed speeds and accelerations.

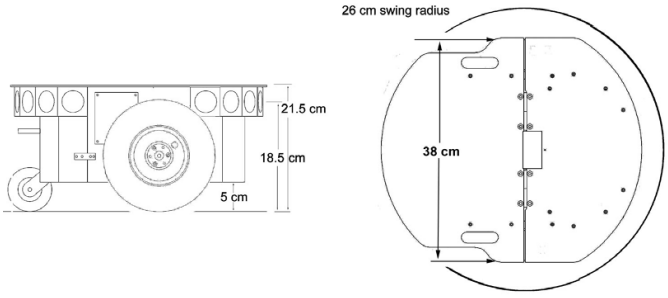

Fig. 5. The Pioneer3Dx mobile base dimensions [26].

Figure 6 represents the interactions between hardware components: the Raspberry Pi receives environment states via laser scan data and sends proper commands to the robot. Simultaneously, the robot states and environment information are exchanged between the Raspberry Pi board and a computer: on both processing units are running specific processes (nodes) of the ROS framework, respectively the ROS master node and sensor drivers nodes on the former, and navigation and visualization nodes on the latter, making up a typical distributed ROS system.

As already said, the first step for the SGP theoretical to practical transition has been that of feeding the algorithm implemented in MATLAB with the real map of the environment in which the robot can move, thus conservatively isolating fixed obstacles, by describing their boundaries. The map has been generated by ROS through the `gmapping` [29] package, which provides laser-based SLAM (Simultaneous Localization and Mapping) built upon the OpenSlam's Gmapping algorithm [30]. The output map (saved as a *.pgm* file) describes the environment according to the binary occupancy grid format,

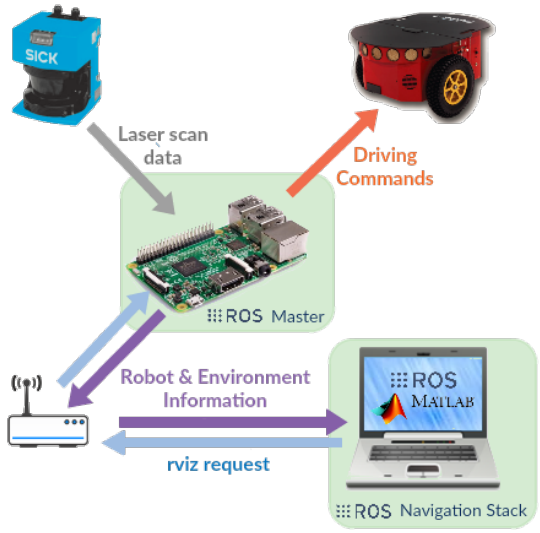

Fig. 6. General schema mapping hardware to the software distributed system.

in which a black-coloured cell represents an occupied area (obstacles), while a white one indicates free space. Such described map is interpreted by the ROS *rviz* tool [31] to build up the cost map that inflates costs, based on the occupancy grid information and physical features of the performing robot. In our case the original map of the whole research laboratory has been restricted to an area where unexpected obstacles are less probable, to fall within the assumed conditions, leading to a 160x190 cells grid with a resolution of 0.05 m/cell (Figure 7).

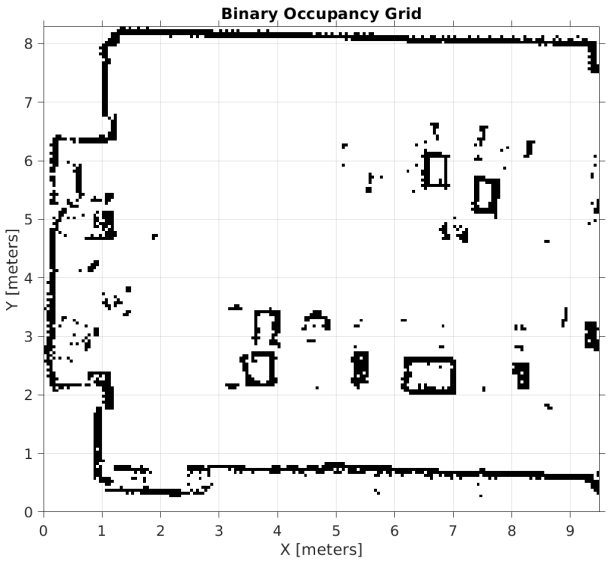

Fig. 7. Employed portion of the mapped laboratory in Politecnico di Torino.

III. Experimental Results

In this section the obtained results are presented through the analysis of three test cases: in the first one no unexpected

obstacles (i.e., not present in the static map) are involved, in the second one we go through the algorithm execution in the presence of an unknown object (not included in the static map, but detected before the GP path computation), and finally the third one deals with the appearance of an obstacle during the robot motion (i.e., after the GP has computed the path). The explored test cases represent a preliminary set of experiments: the authors want to demonstrate the capability of the robot of reaching the desired goal position traversing the safest path as soon as possible. Notice that, in general, the overall algorithm execution time depends on (i) the chosen GP and LP, (ii) the computational capability of the computer executing the navigation and planning instructions, and (iii) the goal to be reached. The given execution times (relative to each specific test case) are to be considered for our specific choice of planners, for reaching a specific goal position, and running the heavier algorithm nodes on a high-performing computer.

All the considered test cases results take advantage of the *rviz* visualization tool for directly comparing the planned path with the desired SGP trajectory. As mentioned, the LP local cost map, i.e., the portion of the global plan of which the LP has knowledge, has been restricted. The chosen values for the cost map dimensions represent a tradeoff for ensuring (i) a faithful tracking of the GP plan and (ii) obstacle avoidance. For all the tests, the starting and destination points (expressed in meters) with the format (x, y) have been set to $(4, 7)$ and $(2.5, 2)$, respectively (Figure 8); the blue dashed line in the figure indicates the desired safe curve to which the path planned by the SGP (green solid line) tends, avoiding any intersection with the boundary curves (red lines) of the obstacles.

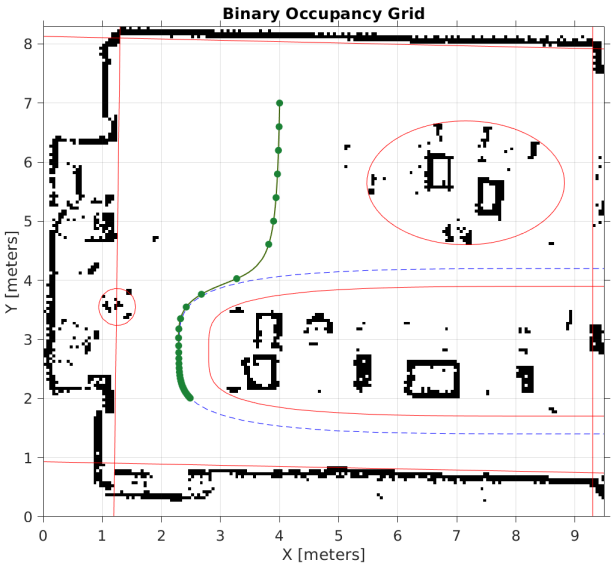

Fig. 8. MATLAB simulation plot for the SGP algorithm. A subset of the computed waypoints is highlighted with green dots.

Furthermore, with the aim of demonstrating the achieved results, the execution of the algorithms in all test cases has been collected in the video that can be found in [32].

A. Test Case 1: absence of unexpected obstacles

The results obtained from the execution of the proposed algorithm in the first test case are reported in Figure 9.

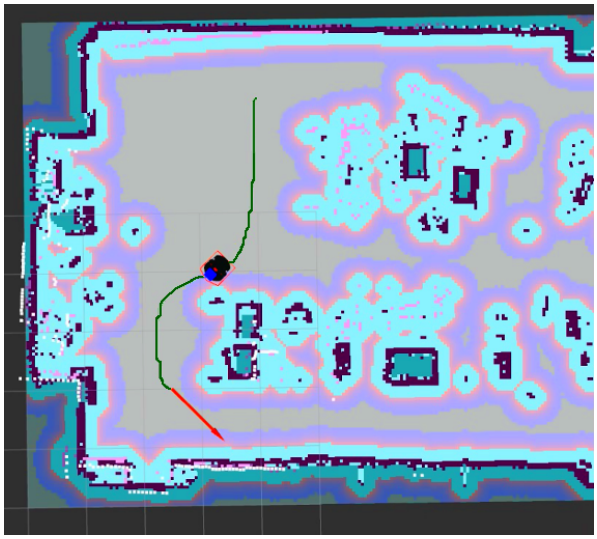

Fig. 9. *rviz* view during Test Case 1 execution.

As it can be seen, the GP (green line) faithfully tracks the SGP generated path, and the LP also follows precisely the GP as a consequence of the ad-hoc tuning performed on the local cost map parameters. Consider that, if the A* algorithm were used, it would generate a plan going straight to the goal position: however, we search for an optimal solution (in terms of safety) that leads the robot to reach a safe virtual track as soon as possible, making it part of the plan. This expected behaviour can be viewed within the "Test Case 1" section of the video: the time needed to reach the goal position is $43.05\ s$.

B. Test Case 2: behaviour with obstacles not in static map

The resulting behaviour generated by the execution of the proposed algorithm in the second test case is reported in Figure 10.

The reported screenshot shows that the GP-generated path deviates from the desired path since the sensor data detected an unexpected obstacle; being the global cost map updated with this new information, the traversed path going backwards, i.e., from the destination point to the starting point, will be the same. The test case execution is shown in the "Test Case 2" video section: the execution time is $57.34\ s$.

C. Test Case 3: behaviour with dynamic obstacles

The execution results of the proposed algorithm in the third test case are reported in Figure 11.

As shown, at first the GP computes a path coherent with the desired one, due to the absence of new obstacles from the beginning. Instead, when an unexpected obstacle is detected by the laser range finder during the robot motion, the LP re-plans the path in order to avoid it, trying to go back to the desired curve, when possible. The present test case has been considered in order to show that, while the robot moves towards the safety curve, if something or someone enters the

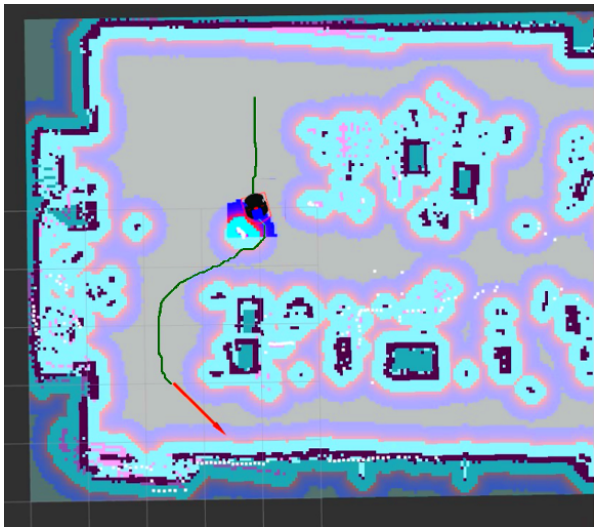

Fig. 10. *rviz* view during Test Case 2 execution.

scene, a safe behaviour is preserved, as shown in the "Test Case 3" section of the video, in which a human operator appears at 05:55 and the reaction of the LP lets the robot avoid him. The overall execution time in this case is 52.39 s.

IV. Conclusions and Future Works

This paper presents a global planning algorithm ensuring path following and obstacle avoidance, implemented by modifying the well-known A* algorithm to follow a set of waypoints, tending to a safe path, generated by a supervisory planner. Thanks to a three-level planning procedure, the goals of the supervisory planner are imposed, while taking into account possible differences between the real scenario and the a-priori map used by the SGP, as well as unexpected obstacles to be avoided on-line. The overall planner algorithm has been experimentally tested employing a differential robot to demonstrate its usability in practice.

The presented algorithm can be seen as an improvement of the global planning in the sense that introduces the capability of following a set of pre-defined waypoints while ensuring a collision free motion. Indeed, the traditional global planning algorithms with path tracking do not include the capability of avoiding unpredicted obstacles, which is a fundamental requirement in the upcoming industrial scenario that features human operators and mobile platforms.

Furthermore, the developed algorithm employs the A* path search (including obstacle avoidance) when the mobile robot is located significantly far away from the waypoints. In such a way, whenever the platform is sufficiently close to one of the desired waypoints, it is "attracted" by the path computed by the SGP. Nevertheless, one of the waypoints can be reached by simply sending a specific goal position to the mobile base, through the Action server provided by ROS.

The role of this work is mainly the implementation of a laboratory real robot demonstrator executing the SGP algorithm. At this stage, the SGP plan computation is performed off-line, but a possible improvement would include the algorithm

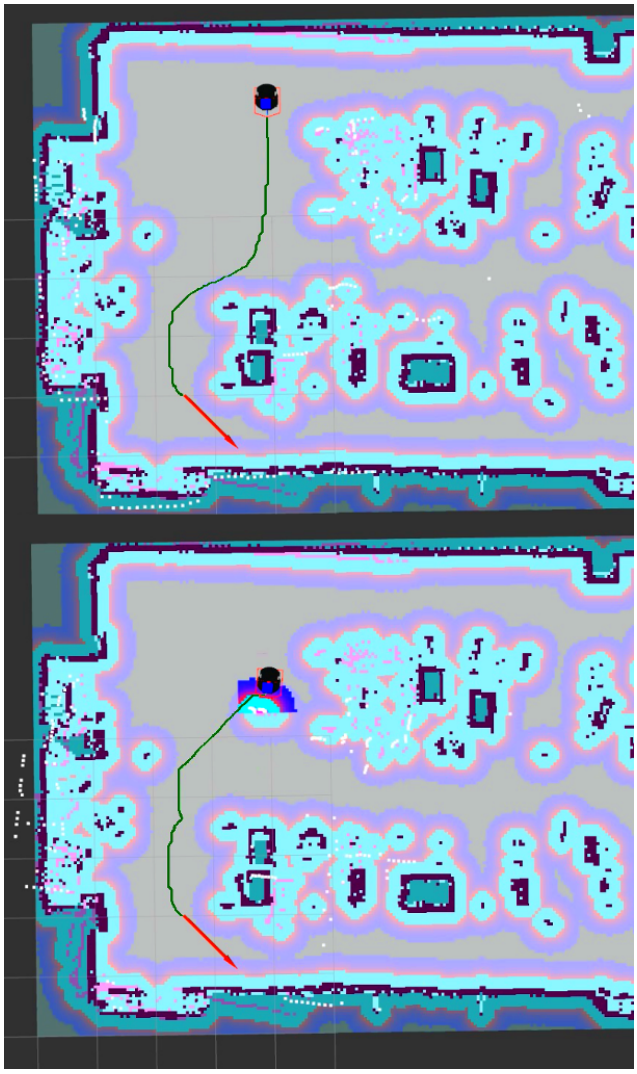

Fig. 11. *rviz* view during Test Case 3 execution. The top figure reports the plan before the obstacle appears, while the bottom one shows the re-planning action.

execution with an on-line re-planning behaviour involving the SGP as well.

Moreover, one of the objectives of our future work is to extend the proposed navigation paradigm to multi-agent scenarios. Indeed, by exploiting the results given in [17], the supervisory algorithm reviewed in Section II-A can be adapted to coordinate multiple mobile robots with the aim of patrolling selected paths with a prescribed formation, while avoiding collisions with each other and with fixed obstacles in the environment. Such an extension of the proposed planning procedure is aimed at meeting the main requirements of the Smart Factories of the future, in which a growing presence of autonomous mobile robots is expected to enhance the flexibility of the production lines.

Acknowledgements

The authors would like to thank Stefano Primatesta for his invaluable suggestions in the experimental application of the approach.

Algorithm 1: Proposed algorithm with path following.

Input: map, start position, goal position
Output: path
`/* Set-up cell expansion lists */`
openList ; `// Declare the open list`
closedList ; `// Declare the closed list`
`/* Insert starting cell in the open list */`
openList.insert(start)
while *openList is not empty* **do**
  `/* Pop cell` c `with lowest` $f(c)$ `off the openList */`
  c = openList.pop()
  `/* Push cell` c `into the closedList */`
  closedList.push(c)
  foreach *neighbor* n *of cell* c **do**
    if *neighbor* n *is the destination cell* **then**
      n.parent = c ; `// Assign parent node`
      Stop searching
    else if *neighbor* $n \notin$ *closedList* ***and*** *is not blocked* **then**
      if n *is a waypoint* **then**
        Assign minimum values to $n.g$, $n.h$ and $n.h_{WP}$
      else if c *is a waypoint* **then**
        Assign minimum values to $n.h$ and $n.h_{WP}$
      else if *waypoints have been traversed all* **then**
        Assign minimum values to $n.h_{WP}$
      else
        if ($n \in$ *openList)* ***and*** (*new* $f(n) >$ *old* $f(n)$) **then**
          Ignore n
        else
          $n.g$ = $n.g$ + distance from c to n
          $n.h$ = distance from n to destination cell
          $n.h_{WP}$ = distance from n to next waypoint
          `/* Compute` $f(n)$ `*/`
          $n.f = n.g + n.h + n.h_{WP}$
          `/* Assign parent cell */`
          n.parent = c
          `/* Insert cell` n `in the openList */`
          openList.insert(n)
`/* Trace the output path from the destination cell */`
p = dest
while p.*parent is not null* **do**
  Path.push(p)
  p = p.parent
Path.push(p)

References

[1] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[2] J. Yao, C. Lin, X. Xie, A. J. Wang, and C.-C. Hung, "Path planning for virtual human motion using improved A* star algorithm," in *2010 Seventh international conference on information technology: new generations*. IEEE, 2010, pp. 1154–1158.

[3] M. Korkmaz and A. Durdu, "Comparison of optimal path planning algorithms," in *2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET)*. IEEE, 2018, pp. 255–258.

[4] S. Koenig and M. Likhachev, "D^* Lite," *Aaai/iaai*, vol. 15, 2002.

[5] D. Ferguson, M. Likhachev, and A. Stentz, "A guide to heuristic-based path planning," in *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*, 2005, pp. 9–18.

[6] L. Kavraki and J.-C. Latombe, "Randomized preprocessing of configuration for fast path planning," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. IEEE, 1994, pp. 2138–2145.

[7] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[8] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised*. Springer, 2017, vol. 118.

[9] J. Kuffner and S. LaValle, "An efficient approach to path planning using balanced bidirectional RRT search," *Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep*, 2005.

[10] E. Taheri, M. H. Ferdowsi, and M. Danesh, "Fuzzy Greedy RRT Path Planning Algorithm in a Complex Configuration Space," *International Journal of Control, Automation and Systems*, vol. 16, no. 6, pp. 3026–3035, 2018.

[11] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[12] Z. Tahir, A. H. Qureshi, Y. Ayaz, and R. Nawaz, "Potentially guided bidirectionalized RRT* for fast optimal path planning in cluttered environments," *Robotics and Autonomous Systems*, vol. 108, pp. 13–27, 2018.

[13] J. Tu and S. X. Yang, "Genetic algorithm based path planning for a mobile robot," in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, vol. 1. IEEE, 2003, pp. 1221–1226.

[14] B. Hernández and E. Giraldo, "A Review of Path Planning and Control for Autonomous Robots," in *2018 IEEE 2nd Colombian Conference on Robotics and Automation (CCRA)*. IEEE, 2018, pp. 1–6.

[15] G. Antonelli, S. Chiaverini, and G. Fusco, "A fuzzy-logic-based approach for mobile robot path tracking," *IEEE Transactions on Fuzzy Systems*, vol. 15, no. 2, pp. 211–221, 2007.

[16] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, 1992.

[17] C. Possieri and M. Sassano, "Patrolling and collision avoidance beyond classical navigation functions," in *European Control Conference (ECC)*. IEEE, 2018, pp. 1821–1826.

[18] ——, "Motion Planning, Formation Control and Obstacle Avoidance for Multi-Agent Systems," in *2018 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, 2018, pp. 879–884.

[19] C. Possieri and A. Tornambè, "On polynomial vector fields having a given affine variety as attractive and invariant set: application to robotics," *International Journal of Control*, vol. 88, no. 5, pp. 1001–1025, 2015.

[20] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[21] ROS Navigation Stack Official Documentation. Accessed: April 2019. [Online]. Available: http://wiki.ros.org/navigation

[22] MATLAB Robotics System Toolbox ROS Interface Documentation. Accessed: April 2019. [Online]. Available: https://www.mathworks.com/help/robotics/robot-operating-system-ros.html

[23] ROS actionlib Official Documentation. Accessed: April 2019. [Online]. Available: http://wiki.ros.org/actionlib

[24] S. Baldi, N. Maric, R. Dornberger, and T. Hanne, "Pathfinding Optimization when Solving the Paparazzi Problem Comparing A* and Dijkstra's Algorithm," in *2018 6th International Symposium on Computational and Business Intelligence (ISCBI)*. IEEE, 2018, pp. 16–22.
[25] B. P. Gerkey and K. Konolige, "Planning and control in unstructured terrain," in *ICRA Workshop on Path Planning on Costmaps*, 2008.
[26] M. Inc, *Pioneer 3 Operations Manual*.
[27] S. A. Waldkirch, *PLMS200/211/221/291 Laser Measurement Systems*.
[28] Raspberry Pi site. Accessed: April 2019. [Online]. Available: https://www.raspberrypi.org/
[29] ROS gmapping Official Documentation. Accessed: April 2019. [Online]. Available: http://wiki.ros.org/gmapping
[30] G. Grisetti, C. Stachniss, W. Burgard *et al.*, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, p. 34, 2007.
[31] ROS rviz Official Documentation. Accessed: April 2019. [Online]. Available: http://wiki.ros.org/rviz
[32] Test Cases with the Pioneer3DX. Accessed: April 2019. [Online]. Available: https://youtu.be/ZnBrdG6uKfw