Robotica: page 1 of 19. © Cambridge University Press 2020
doi:10.1017/S0263574719001838

Navigation of Semi-autonomous Service Robots Using Local Information and Anytime Motion Planners

Guilherme A. S. Pereira†* and Elias J. R. Freitas‡

†*Department of Mechanical and Aerospace Engineering, West Virginia University, 1374 Evansdale Drive, Morgantown, WV 26506-6070, USA*
‡*Federal Institute of Minas Gerais, Rua Jose Benedito 369, Santa Efigenia, Itabirito, MG 35450-000, Brazil*

(Accepted December 14, 2019)

SUMMARY
This paper deals with the problem of navigating semi-autonomous mobile robots without global localization systems in unknown environments. We propose a planning-based obstacle avoidance strategy that relies on local maps and a series of short-time coordinate frames. With this approach, simple odometry and range information are sufficient to make the robot to safely follow the user commands. Different from reactive obstacle avoidance strategies, the proposed approach chooses a good and smooth local path for the robot. The methodology is evaluated using a mobile service robot moving in an unknown corridor environment populated with obstacles and people.

KEYWORDS: Service robots; Navigation; Obstacle avoidance; Probabilistic planners.

1. Introduction

With the recent development of robotics, service robots are becoming closer to people, being increasingly present in our work, study and leisure environments.[1] A peculiarity of some of these robots is that they can be directly commanded by a person, as is the case of intelligent wheelchairs,[2] robots that must follow people[3] or driving assistant systems of automobiles.[4] This paper is mainly interested in semi-autonomous robots that have the human being responsible by the mission control and the robot itself responsible for other specific tasks, specially safety.

Because they are controlled by a person, semi-autonomous robots differ from other service robots in at least three points: (i) they do not need a global map of their workspace (except when mapping is part of their tasks); (ii) they do not need to know the main objective of the mission or their final metric destination; (iii) they do not need to globally localize themselves in the workspace. In spite of these differences, semi-autonomous robots still need to avoid collisions with obstacles, other robots and people, preserving themselves, their workspace and their users. Therefore, the biggest challenge for the development of semi-autonomous service robots is to guarantee the robot safe navigation in unknown and changing environments.

For the sake of motivation, consider, for example, the situation where a robot enters a building for the first time with the objective of delivering a package to a given office. No global map of the environment is available (unknown environment), people are moving around the building (changing environment) and no global information is available (e.g. GPS data). Since the robot does not know how to reach the office, it needs to ask for human help. In this case, a human (e.g. the doorman) could lead the robot to its destination, what would require the robot to track the human while avoiding possible obstacles. More interesting, the robot could receive from the doorman a set of high-level,

* Corresponding author. E-mail: guilherme.pereira@mail.wvu.edu

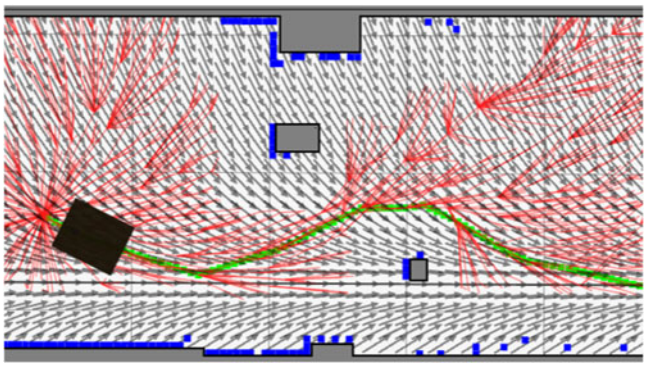

Fig. 1. An illustration of the proposed methodology. The black rectangle represents the semi-autonomous service robot, the small blue squares are laser sensor readings and the vector field is represented by arrows. An anytime motion planner (red graph) is used to locally compute the shortest collision free path that follows the vector field as close as possible.

human-like commands such as follows: follow this corridor until the second intersection, turn right and enter the second door; and execute it autonomously. Notice that the later option is similar to the way a person would behave in an unknown building. Observe that moving toward the destination using such a strategy is viable even without a global map and global localization, requiring only basic sensing capabilities such as corridor, intersection and door detection. One of the objectives of this work is then to propose and evaluate a robot navigation strategy that solves this kind of problem.

Since the beginning of robotics, navigation strategies for robots have been proposed.[5–8] These strategies were classified as reactive, when the robot reacts to the presence of recent detected obstacles without trying to predict the consequences of its action, or deliberative, when the robot, using maps of the environment, plans a path that not only avoids the obstacle, but also explicitly considers its mission.

Because reactive strategies are very efficient and do not require global localization or complex models for obstacles and workspace,[9,10] they are the first choice to guide semi-autonomous robots.[11–15] However, since these strategies do not consider the future, they can lead the robot to undesirable situations that may result in long paths or even in a deadlock. Deliberative strategies, such as refs. [16] and [17], on the other hand, plan or re-plan paths that consider not only the closest obstacle, but also a future horizon.[18] These approaches may generate optimal safe paths, but in opposition to reactive strategies, depend on global maps of the environment and global localization.[31] These requirements, which are inherited by hybrid approaches that combine deliberative and reactive methods,[19] may be hard to be fulfilled by some semi-autonomous robots, such as smart wheelchairs moving in unknown environments.[11]

The main contribution of this paper is the proposition of a novel and efficient navigation architecture for semi-autonomous service robots that uses motion planners and does not require global localization nor global maps. To avoid the requirement for global localization, the proposed strategy uses a sequence of short-time local coordinate frames. These frames, which are forgotten in hundreds of milliseconds, are only used by the robot to plan a path in its field of view and to follow the last path planned. Because the time between two coordinate frames in the sequence is very small, low uncertainty coordinate transformations between these frames may be obtained with standard wheel odometry. If complete path planners are employed, our strategy guarantees static obstacles avoidance and, within a given region of the sensor's field, may also allow the avoidance of moving obstacles. Since we are considering semi-autonomous robots, we use artificial vector fields to encode the user commands as desirable robot behaviors.[9] These fields may include cultural and social behaviors expected for the robot. For each human command, the planner uses a vector field in its optimization functional to find the safe path that better follows the required behavior. An illustration of the proposed method is shown in Fig. 1. In this figure, the arrows represent a vector field that encodes the "go straight" user command while the red lines represent the search tree used by the planner, from which was found the green path to be followed. In this example, the robot is required to follow the corridor close to the right wall, which is the social rule in several countries. The planner used in this example was the one proposed in ref. [20], with a few modifications. This planner is based

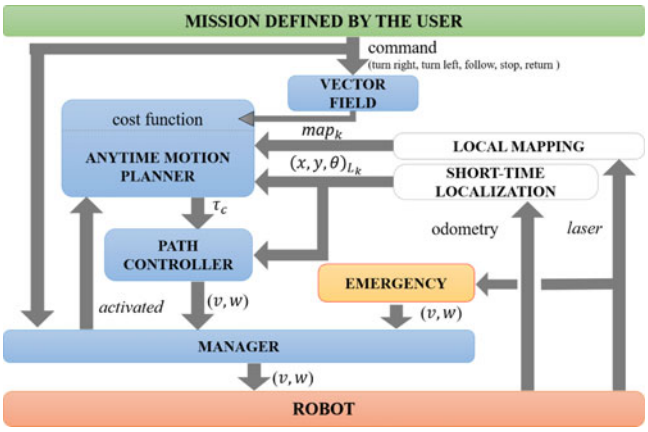

Fig. 2. Navigation architecture proposed in this work.

on RRT* (the asymptotically optimal version of the rapidly exploring random tree (RRT)[21]), which generates the red tree in Fig. 1. Although we have successfully used this planner in our work, it could be replaced in our approach by any other *anytime motion planner*. In this context, an anytime motion planner is a planner that rapidly computes a feasible path, and optimizes this path if more time is available. By the authors' knowledge, the proposition of this paper, which was motivated by practical difficulties encountered in the operation of robots in different unknown environments, is new. All other motion planning-based approaches, such as ref. [22], which also considers social rules, require global metric localization, what in practical situations is difficult to obtain.

Given this, the authors believe that this paper presents the following contributions:

- an architecture for semi-autonomous navigation of service robots in unknown workspaces that does not require a global map or a global localization system;
- an approach based on a series of local coordinate frames to allow the use of motion planners for obstacle avoidance without global localization;
- the use of vector fields to encode simple human commands and their local modification using anytime motion planners.

Next section is dedicated to describe the proposed methodology in details. Section 3 shows the experimental setup used to validate the method. Sections 4 and 5 show the experimental results that illustrate our approach, respectively, using simulations and an actual robot. Finally, conclusions and future work are discussed in Section 6.

2. Methodology

The navigation approach proposed in this paper may be pictured as the block diagram of Fig. 2. In this figure, it is possible to see that a human user defines the next mission to be executed by the robot. Since we assume a nontechnical user, all missions are defined by a sequence of simple human-like commands such as "turn left," "turn right," "go straight" and "stop." These commands could be given by one of several possible interfaces, including voice, touch screens, brain machine interfaces and eye or head trackers. These interfaces are very useful when the user is close to or even on the robot, as is the case of an intelligent wheelchair, or controlling it remotely based on images from a camera mounted on the robot, as the ones available on exploration robots. In situations like these, we assume that, except for some preemptive commands, such as "stop," the robot waits for a new command after the execution of the previous one is finished. To prevent the robot to stop between two commands, a small buffer with up to three commands may be used to store a sequence of user commands. With this approach, notice that the user is part of the robot's control loop and is the main responsible for the definition of its main mission. If the user is remote and knows the sequence of commands a priori, as is the case of delivery robots in a map known by the user, for example, the commands may be also given as a sequential list, where one command is executed after the previous one is completed. This second strategy was used in our experiments to facilitate reproducibility.

Notice in Fig. 2 that the top arrow referred as *command* is divided in two. Complicated and long duration commands are encoded using vector fields that are then used by the planner, as will be explained in Section 2.3. Other simple commands, such as the "stop" command, are executed directly by a mission MANAGER, which is the block responsible for the low-level interaction with the robot. The MANAGER is also responsible for stopping the robot in case of emergency, which may be detected by the block EMERGENCY using the robot's sensors, and for sending sensor data to other blocks of the architecture.

Central to our architecture is the block in Fig. 2 called ANYTIME MOTION PLANNER, which generates a path (τ_c) to be followed by the robot. This path will be a function of VECTOR FIELD, local map (map_k) and robot local localization (x, y, θ), all represented in the same coordinate frame L_k, valid for the small time interval k. We propose that the local map is built using the latest laser data only, with no past values, and considering that the robot is approximated by a disk whose radius is given by its width. So we can say that this map is a memory-free local map. Our experiments will show the advantages of this map over maps with memory. The SHORT-TIME LOCALIZATION block generates the robot localization with respect to L_k. Since L_k is a frame that will be only used for a small amount of time, this block relies on odometry only to compute $(x, y, \theta)_{L_k}$. The path computed by the planner is used by a PATH CONTROLLER that computes the robot's required linear and angular velocities (v, ω), which are sent to the robot through the mission MANAGER. To make it easier to understand the core behavior of our method, we present next an algorithm with a detailed sequence of operations executed.

2.1. Algorithm

Algorithm 1 summarizes the strategy proposed in this paper. The inputs of this algorithm are the desirable time of planning t_s and the user command codified as the vector field $\mathbf{u}(\cdot)$. Consider that the time is discretized in small intervals t_k, $k = 1, 2, \ldots$, where the period $\Delta t = t_k - t_{k-1}$ corresponds to the time t_s given to the planner to compute a path. Since the planner is anytime, the larger is t_s, the better is the path. On the other hand, we show in the next section that smaller values of t_s increase the safeness of the path in changing environments. Along the execution of the algorithm, a sequence of local coordinate frames $\{L_k\}$, one for each time step t_k will be generated (see line 6 of Algorithm 1).

The main loop of the algorithm, whose termination is controlled by the mission MANAGER, first generates a new coordinate frame $\{L_k\}$ and transforms the previous computed path and the previous local map, both obtained with respect to frame $\{L_{k-1}\}$, to the newly created frame $\{L_k\}$ (lines 7 and 8). Assuming that $\{L_{k-1}\}$ and $\{L_k\}$ are very close in time and space, a homogeneous transformation matrix $T_{k-1,k}$ that represents the position and orientation of $\{L_{k-1}\}$ with respect to $\{L_k\}$ can be computed using the robot's odometry. Once $T_{k-1,k}$ is computed, path and local map transformations in lines 7 and 8 of Algorithm 1 are executed by pre-multiplying each point of the path and of the map by $T_{k-1,k}$, as illustrated in Fig. 3.

After these transformations, whose time of processing may be neglected if compared to t_s, a new instance of the planner is launched (line 11). This planner generates path τ_s by taking as inputs the local map recently transformed to $\{L_k\}$, the origin of the new path $\mathbf{o}_c$, also represented with respect to $\{L_k\}$, the vector field $\mathbf{u}(\mathbf{p})$, which represents the human command, and the time of planning t_s. With this information, the resultant path τ_s follows the vector field and avoids all obstacles represented in $\mathtt{map}_k$, as will be discussed in Section 2.3.

The reader may be missing a goal position for the planner in line 11. In fact, since a semi-autonomous robot does not always know its precise destiny, in case the goal is required by the planner used, it can be chosen arbitrarily and without lost of generality as a position in front of the robot, at the limit of its field of view. The planner used in the experiments of this paper does not require a goal position. Because any anytime motion planner can be used in our approach, we choose not to show a pseudocode of the Planning function in line 11. The pseudo-code of the planner used in our experiments can be found in ref. [20].

The origin position, $\mathbf{o}_c$, is obtained by integrating (simulating) the previous computed path (computed with respect to $\{L_{k-1}\}$ and transformed to $\{L_k\}$), for a time t_s starting at the actual robot position $\mathbf{p_r}$ with respect to $\{L_k\}$ (line 9). For this integration, it is considered the average robot speed $\mathbf{v_r}$. By choosing a position in the previous path as the beginning of the next one, we guarantee continuity of the path and also prevent the planner to compute consecutive paths in different homotopic classes

Algorithm 1 Navigation strategy

Data: t_s, $\mathbf{u}(\cdot)$

1 $\mathtt{k} = 0$; $v = 0.0$; $\omega = 0.0$; $\mathbf{v_r} = (v, \omega)$; $\tau_0 \leftarrow \emptyset$;
2 L_0 = GetNewFrame();
3 $\mathtt{map}_0$ = GetMap(L_0);
4 $\mathtt{k} \leftarrow \mathtt{k} + 1$;
5 **while** *activated* **do:**
6 L_k = GetNewFrame();
7 τ_c = TransformPath(τ_{k-1},L_k);
8 $\mathtt{map}_k$ = TransformMap($\mathtt{map}_{k-1}$,L_k);
9 $\mathbf{o}_c$ = Integrate(τ_c, $\mathbf{p_r}$, t_s, $\mathbf{v_r}$);
10 **while** $t \leq t_s$ **do in parallel:**
11 τ_s = Planning($\mathtt{map}_k$, $\mathbf{o}_c$, $\mathbf{u}(\mathbf{p})$, t_s);
12 $\mathbf{v_r}$ = FollowPath(τ_c);
13 **end-parallel**
14 τ_k = ConcatenatePath(τ_c, τ_s, $\mathbf{p_r}$, $\mathbf{o}_c$);
15 $\mathtt{map}_k$ = GetMap(L_k);
16 $\mathtt{k} \leftarrow \mathtt{k} + 1$;
17 **end-while**

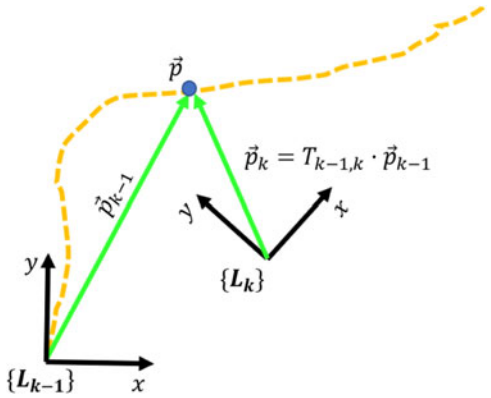

Fig. 3. Illustration of the homogeneous transformation used in Algorithm 1, where $\vec{p}$ is a point in the path, and $\vec{p}_{k-1}$ and $\vec{p}_k$ are representations of this point, respectively, with respect to L_{k-1} and L_k. The homogeneous transformation matrix that relates these two coordinate frames is $T_{k-1,k}$.

(e.g. at t_{k-1} the path avoids an obstacle from the right and at t_k from its left). Notice that this strategy defines a path between the current robot position $\mathbf{p_r}$ and $\mathbf{o_c}$ to be followed by the robot while the new path is computed. This path is known as *committed path*.[23]

While a new planning problem is solved in line 11 of Algorithm 1, a path follower controller drives the robot along the committed path (line 12). Notice that the committed path is already in $\{L_k\}$, so the controller uses robot positions and orientations with respect to this frame to compute the robot velocities.

After t_s seconds, the part of the committed path that was not followed by the robot is concatenated with the new path τ_s (line 14) and a new local map containing the most recent sensor information is obtained (line 15). The process then restarts by creating a new coordinate frame $\{L_{k+1}\}$ and representing the new path and new local map with respect to this frame.

It is important to notice that this process only works in practice because a transformation $T_{k-1,k}$ between the current and the previous coordinate frame is known. Since t_s is usually very small (less than 2 s), odometry information may be sufficient to compute this transformation with low uncertainty. We evaluate this affirmation in Section 5.

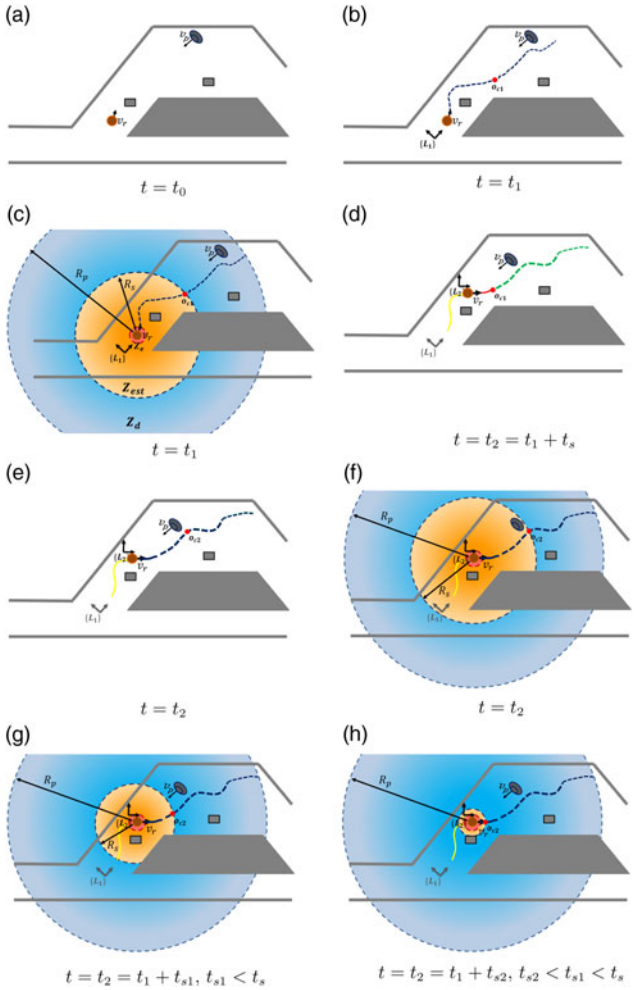

Fig. 4. An illustration of the navigation strategy. The robot is represented as a circle and has speed $\mathbf{v_r}$. Obstacles are represented by gray color. The dark gray ellipse is a moving obstacle. Notice that the robot cannot differentiate moving and static obstacles. The complete path to be followed by the robot is the dashed blue line. Radius R_s defines a Static Zone (Z_{est}) and radius R_p represents the planning radius. The Dynamic Zone (Z_d) is externally delimited by R_p and internally by R_s. The Emergency Zone is represented by a circle around the robot.

2.2. Analysis

An illustration of Algorithm 1 is shown in Fig. 4. In Fig. 4(a), we show a workspace containing static and moving obstacles and a robot moving with velocity $\mathbf{v_r}$. Assuming that the robot already computed a path τ_0 at $t = t_0$, at t_1, a coordinate frame $\{L_1\}$ is created and τ_0 is transformed to this frame, which is only valid until $t_2 = t_1 + t_s$ (Fig. 4(b)). While the robot moves using odometry with respect to $\{L_1\}$, the planner starts computing a new path, which starts in a point in front of the robot $\mathbf{o}_{c1}$. For the computation of this path, we assume in this example that the human command was "go straight." As will be discussed in Section 2.3, a vector field (not shown in this figure) is then created to represent this command. This field would enforce the robot to move straight if no obstacles were present in the workspace.

To analyze the algorithm behavior, we define three regions, as shown in Fig. 4(c): (i) static zone (Z_{est}); (ii) emergency zone (Z_e); and (iii) dynamic zone (Z_d). The static zone, which is a disk of radius R_s centered on the robot position at the beginning of the loop, is a region in which the robot is following the committed path. Therefore, R_s is directly related to point $\mathbf{o}_c$, found by simulating the robot movements using the previous path. In the case of no moving obstacles in this region, our method guarantees the absence of collisions if the robot follows the planned path. To try to prevent collisions with a possible moving obstacle inside this region, we define an emergency zone that moves along the robot. If an obstacle is detected inside this region, the robot stops by activating its

emergency system. Notice that this behavior is not sufficient to prevent a collision with a moving obstacle since this obstacle can be still moving toward the robot. It is expected, however, that some moving obstacles, such as people and other robots, would also stop or change their paths to avoid the collision. The dynamic zone is also centered on the robot position at the beginning of the loop and is delimited internally by R_s and externally by the planning radius R_p. Radius R_p is in fact a parameter of the planner, which is chosen as a function of computational capacity (larger regions require more computation), speed and number of obstacles and/or the sensor's field of view. If the robot keeps its previous velocity, the proposed algorithm will guarantee that the robot will not enter the dynamic region in the planning interval t_s. Therefore, there will be no collisions with the obstacles in this region. It is interesting to notice that, even if the robot does not enter the dynamic zone, the planned path consider obstacles in such a zone. Since this path will be used to define the starting point of the next one, this is an important characteristic to allow obstacle avoidance.

After t_s seconds (see Fig. 4(d)), the robot have followed the part of the path drawn in yellow but did not reach the expected point $\mathbf{o}_{c1}$. At this time, a new path starting in $\mathbf{o}_{c1}$ is delivered by the planner. The algorithm thus concatenates the part of the old committed path not followed by the robot (red path) with the new path (green path), composing a path τ_k, $k = 1$ that starts in the current robot position. Fig. 4(e) shows the new coordinate frame $\{L_2\}$ with respect to which the new complete path τ_1 is transformed using transformation $T_{1,2}$. A new planner is then launched with start position $\mathbf{o}_{c2}$ computed by simulating the new path.

To better analyze the effect of the planning time t_s for the behavior of the proposed algorithm, notice that, in the extreme case, when the robot moves in a straight path with maximum velocity v_{max} the radius of the static region is given by

$$R_s = v_{max} \cdot t_s \,. \tag{1}$$

Thus, if the robot speed and/or the planning time are increased, the static zone is also increased. Since in this region the robot follows a plan computed t_s seconds ago (the committed path), this plan only consider obstacles that became static during this time. If t_s is reduced, the static zone is also reduced, and thus the planned path would increase the probability of avoiding moving obstacles. To illustrate this behavior, see Fig. 4(f), which shows a situation where the moving obstacle is inside the static zone, what may cause the planner-based obstacle avoidance strategy to fail. This happens because the robot makes no distinction between moving and static obstacles and always follows the path assuming that it is free of collisions. In Fig. 4(g) and (h), when smaller planning times were used, the static zone became smaller and we again have obstacle avoidance within this region. Notice that the same behavior could be achieved if we had reduced the robot speed, what would certainly compromise the efficiency of the mission. It is very important to mention, however, that our approach cannot guarantee that no collisions with movable obstacles will happen. As pointed by ref. [24], collision with moving obstacles would only be guaranteed if the dynamics of both the robot and the obstacle are considered and the planning horizon is infinite.

We then conclude that the proposed navigation strategy guarantees obstacle avoidance for static obstacles, assuming that these obstacles are detected by the robot's sensors. For moving obstacles, the strategy depends on the planning time and on the speed and acceleration of both robot and obstacles. Assuming that high robot speeds are desirable and that obstacle speeds cannot be controlled, it is then necessary to reduce the planning time t_s to increase the probability of moving obstacle avoidance. However, it is important to mention that anytime planners still need a minimum amount of time to compute its first path. Also, if the planning time is very close the minimum, the quality of the path could be highly compromised. Thus, the reduction of t_s can be done by the development of more efficient path planning algorithms or by the use of more powerful processors. This will allow the robot to move fast and still be able to avoid some moving obstacles by following the planned path. However, as mentioned before, collision avoidance for moving obstacles is not guaranteed by the methodology, which would only be possible if the future trajectory of the obstacle is known and considered by our planner.[24]

2.3. Motion planner

The motion planner considered in this work must be (i) complete: to provide a path if a path exists; (ii) anytime:[23] to provide the best path up a certain interval of time t_s; and (iii) able to optimize a cost

functional based on a vector field that encodes the user command. Therefore, any path planner with these characteristics could be promptly used in our approach.

For an example of vector field, consider the command "go straight," which basically means that the robot must move straight in the direction it is pointing to. To simplify the definition of a vector field for this command, we define a coordinate frame as follows: the origin of the coordinate frame is the position of the robot at the moment it receives the command; the x axis points to the front of the robot at the same moment; and the y axis points to the left of the robot. A planar vector field $\mathbf{u}(\mathbf{p}) = [u_x, u_y]^T$, where $\mathbf{p}$ is the robot position, may be then defined as

$$\mathbf{u}(\mathbf{p}) = \begin{bmatrix} v \\ -k\,y \end{bmatrix}, \tag{2}$$

where, v is the predefined robot's forward speed and k is a constant that determines how fast the robot will correct its lateral displacement y. An important characteristic of vector fields such as the one in Eq. (2) is that they can be computed very efficiently given the robot position with respect to its coordinate frame. Since we assume that the robot is always very close to the origin of its coordinate frame, once a new frame is created every t_s seconds, $\mathbf{p}$ can be computed directly from the robot's odometry. Other user commands, such as "turn left" and "turn right" would define different vector fields. A vector field for these commands could be, for example, one that would make a robot to follow a circle. Although fast to be computed, the derivation of such a field is more elaborated than the one in Eq. (2) and is omitted here for the sake of focus. This and other fields can be derived with the methodology proposed in ref. [25]. It is important to mention that, during the execution of Algorithm 1, the several vector fields available are switched in function of the user command.

Let a path to be the continuous function $\tau : [0, 1] \to \mathcal{P}$, where $\mathcal{P}$ is the space of robot positions in frame $\{L_k\}$. To find a short path that follows the vector field, we search for a path that minimizes the following cost functional[1]:

$$F[\tau, \mathbf{u}\,] = \int_0^1 \left(1 - \frac{\tau'(s)}{\|\tau'(s)\|} \cdot \frac{\mathbf{u}\,(\tau(s))}{\|\mathbf{u}(\tau(s))\|}\right) \|\tau'(s)\|\, ds\,, \tag{3}$$

where the upper comma stands for the derivative with respect to the spacial parameterization variable s of the path τ, operator $\|\cdot\|$ represents the Euclidean norm and the center dot is the scalar product. Notice that this functional is a function of both the length of the path and of on how "close" the path is from the vector field. In fact, $F[\tau, \mathbf{u}]$ is greater than or equal to zero, and is zero if, and only if, the path is parallel to the field ($\tau'(s) = \alpha\,\mathbf{u}(\tau(\mathbf{s}))$, for $\alpha > 0$). Using this functional, our local path planning problem can be written as

$$\begin{aligned} \underset{\tau}{\text{minimize}} \quad & F[\tau, \mathbf{u}] \\ \text{subject to:} \quad & \tau(0) = \mathbf{o}_c\,, \\ & \|\tau(1) - \tau(0)\| = R_p\,, \\ & \tau(s) \in \mathcal{P}_{\text{free}}, \forall s \in [0, 1]\,, \end{aligned} \tag{4}$$

where $\mathbf{o}_c$ is the initial path position, R_p is the radius that defines the disk-shaped planning region and $\mathcal{P}_{\text{free}}$ is the space of positions where the robot is free of collisions.

The previous optimization problem presents a major difference in relation to the standard motion planning problem, which is the replacement of the goal position constraint by one constraint that enforces the distance between the initial and final positions of the path. Therefore, since there is no definition of a goal position, it is the role of Functional (3) to dictate the direction of the movement.

In this paper, we chose RRT* to solve the Optimization Problem (4) although, as discussed before, any anytime motion planner could be applied. By randomly sorting positions (nodes) in the search space, in our case the intersection between a disk-shaped region of radius R_p and $\mathcal{P}_{\text{free}}$, RRT*

[1]This functional is exactly the one proposed in ref. [20] when variables a and b in ref. [20] are set to 1. The functional is also similar to the *upstream cost* proposed in ref. [26], which is specific for curves of unit derivative.

Fig. 5. MARIA, the semi-autonomous service robot used in the experiments of this paper. The main components of the robot are shown in evidence: a SICK laser range sensor, a notebook, encoders and bumpers.

constructs a tree that starts at $\mathbf{o}_c$ and has nodes covering the free part of the space. An RRT* tree is illustrated in Fig. 1. Because we do not specify a goal position, in our method, differently from previous implementations of RRT*, once the tree is computed, our solution will be a path between $\mathbf{o}_c$ and one of the nodes that are approximately at distance R_p from $\mathbf{o}_c$. Among all possible paths, we choose the one with the smallest path cost. A deep discussion about RRT* can be found in ref. [21]. Details of our specific implementation are in refs. [20] and [27].

3. Experimental Setup

We have implemented the proposed methodology in C++ using OMPL[28] and ROS.[29] It was then simulated using StageROS and tested in practice with our semi-autonomous service robot MARIA, shown in Fig. 5. This robot was built over a Pioneer3-AT base equipped with encoders and gyroscope for odometry, bumpers for collision detection and a SICK-LMS 100 planar laser sensor for obstacle detection and avoidance. The robot base measures 0.625 m $\times$ 0.49 m. During path computation, the robot shape was approximated by a disk of radius 0.65 m to simplify collision detection.

The application that we explore in our experiments is the one where a robot enters an unknown building composed of straight corridors that intersect each other in right angles. The corridors may contain static obstacles and people. We assume that the user may give five different commands to the robot: "go straight," "turn right," "turn left," "return" and "stop." The first three commands may have a parameter in meters that indicates how much the robot must move. For example, "turn right 3" means that the robot must turn right in the first corridor found at the right and move 3 m in that corridor. In our experiments, to avoid command misinterpretation, thus guaranteeing experimental reproductibility, the commands are given to the robot using a sequential list, where a command is executed after the previous one is completed.

Our experiments were conducted in a building with 2.35 m wide corridors with doors, that could be open or closed, and some pillars close to the walls. We artificially added some obstacles to the environment to test our approach. To detect corridors, intersections and obstacles, our software uses laser data only. In these data, the longest straight lines, even with small gaps, are assumed to be the corridor walls. Larger gaps on these lines are assumed to be corridor intersections. Every other laser data not classified as corridor walls are assumed to be an obstacle. We do not differentiate or classify the obstacles.

3.1. Coordinate frames definition

Our method does not specify how the short-time local coordinates are defined. In our experimental setup, we chose to define these frames such that the origin is the center of the corridor (determined with the help of the laser sensor), x is tangent to the corridor pointing in the forward direction and y is orthogonal to it, pointing toward the left wall. Notice that this definition of the coordinate frame relies on the detection of the corridor, what is done by detecting the longest lines in the laser sensor data. This definition will facilitate the vector field computation, as shown next.

3.2. Artificial vector field design

The idea of this paper is to encode user commands as artificial vector fields. In our experimental setup, an important command is "go straight," which basically means that the robot must follow the corridor in the direction it is pointing to. Assuming that the coordinate frame is defined as explained in the previous subsection, $\mathbf{u}(\mathbf{p})$ is defined as

$$\mathbf{u}(\mathbf{p}) = \begin{bmatrix} 1 \\ k(d_0 - d) \end{bmatrix}, \tag{5}$$

where k is a constant scalar that determines how fast the robot will move toward d_0, which specifies the desirable distance from the right wall and d is the robot's current distance to this wall. Here we assume that d_0 is fixed and specified based on cultural and social conventions. It is also important to mention that only the direction of $\mathbf{u}(\mathbf{p})$ will be used by our motion planner, once it is normalized in the optimization functional (3).

The other commands can be executed with the same vector field. For example, the command *turn right* may be implemented by a simple angle controller that manipulates the robot angular velocity ω. After the robot turns 90° to the right in the next intersection found, it then follows the corridor using the previous vector field. When this command has a parameter, after the robot turns 90°, the vector field is used to make the robot to follow the corridor for the specified distance.

3.3. Path following

To follow the computed path, a controller is necessary. In our implementation, the path is parametrized by a sequence of waypoints. To follow the path, the robot must move from one waypoint to the next in the sequence. To do that, we use a controller that minimizes the error vector $\mathbf{e} = [e_x, e_y]^T$ from the current robot position to the closest waypoint in the path. Since the robot has a differential drive base and also has a low-level velocity controller with linear and angular velocities as inputs (v and ω, respectively), in this paper we use a simple static feedback linearization technique[30] to minimize $\mathbf{e}$. Once $\|\mathbf{e}\|$ is smaller than a given radius γ, the robot moves toward the next waypoint, by computing $\mathbf{e}$ in function of this new waypoint.

3.4. Path planning and time of planning

As mentioned in Section 2.3, our implementation uses RRT* with cost functional (3). To validate and justify the use of this planner and also choose the ideal time of planning t_s, before experiments with a service robot, we executed a numerical evaluation of this and other planners. We performed 100 runs, in the same conditions, with three planners: VF-RRT,[26] RRT* with Euclidean cost function and RRT* with functional (3). VF-RRT is a modification of RRT that implicitly optimize the functional (3). It is not an anytime planner, stopping the search as soon it finds a path that solves the problem.

Table I shows numerical comparative results obtained in a simulated corridor with 50 circular obstacles generated randomly and the vector field in Eq. (5). In those experiments, we set our parameters as searching region radius $R_p = 5$ m; field gain $k = 0.5$; and desired distance to the wall $d_0 = 0.7$ m. For each planner and for four different planning times, we computed (i) the Euclidean cost of the path; (ii) the upstream cost of the path given by functional (3); and (iii) the smoothness of the path, computed by OMPL as

$$\texttt{smoothness} = \sum_{i=2}^{n-1} \left(\frac{2(\pi - \arccos(a_i^2 + b_i^2 + c_i^2))}{a_i + b_i} \right)^2, \tag{6}$$

Table I. Comparison between planners for an environment with 50 obstacles.

Planning time		**VF-RRT**	**RRT* (Euclidean)**	**RRT* (upstream)**
$t_s = 0.45$ s	Euclidean cost	6.85	**5.54**	5.62
	Upstream cost ($\times 10^{-3}$)	1497.58	319.23	**252.24**
	Smoothness	320,356.45	**163.92**	299.68
$t_s = 1$ s	Euclidean cost	6.86	**5.52**	5.59
	Upstream cost ($\times 10^{-3}$)	1506.85	269.95	**213.5**
	Smoothness	295,188.52	**150.6**	238.33
$t_s = 2$ s	Euclidean cost	6.83	**5.5**	5.57
	Upstream cost ($\times 10^{-3}$)	1478.72	240.83	**190.31**
	Smoothness	303,344.8	**130.66**	174.84
$t_s = 5$ s	Euclidean cost	6.85	**5.49**	5.56
	Upstream cost ($\times 10^{-3}$)	1501.86	214.64	**172.35**
	Smoothness	308,461.22	**128.77**	173.24

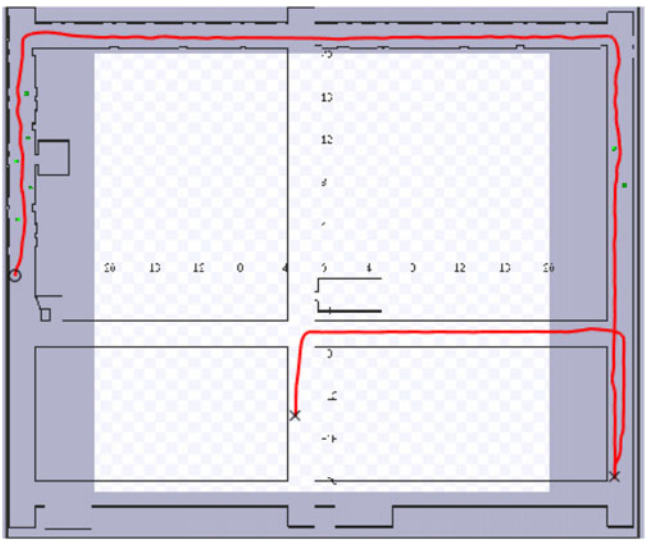

Fig. 6. Overall behavior of the navigation strategy simulated in StageRos. The path performed by the robot is represented by red line, its initial position is marked by a black circle and the targets by black "*x*"s. The green cubes represent statics obstacles in the environment.

where $a_i = \texttt{dist}(s_{i-2}, s_{i-1})$, $b_i = \texttt{dist}(s_{i-1}, s_i)$, $c_i = \texttt{dist}(s_{i-2}, s_i)$, s_i is the ith element of the path and $\texttt{dist}(s_i, s_j)$ is the Euclidean distance between two elements of the path.

Bold numbers in Table I indicate the best results found for each t_s. For times smaller than $t_s = 0.45$ s, both instances of RRT* failed a few times to return a solution when we used a Intel Core I7@2.4 GHz,8 GB RAM computer. Notice in the last column of the table that as larger is the planning time, the smoother and closer to the vector field is the path. Also, observe that RRT* with Euclidean cost function generates the smoothest paths but, as expected, does not follows the vector field (see the upstream cost). VF-RRT, which does not explicitly optimize any functional, presents the worst overall results.

4. Simulated Results

In our first set of experiments, we used the StageRos simulator. The workspace simulated in StageROS, shown in Fig. 6, is very close to the actual building floor used in the real-world experiments. In all simulations, the parameters of the method were set to be searching region radius

$R_p = 5$ m; field gain $k = 0.5$; desired distance to the wall $d_0 = 0.7$ m; and time of the committed path $t_s = 1.0$ s.

To evaluate the overall performance of the navigation strategy, the user defined two targets and the following sequence of commands to achieve them: (1) "go straight"; (2) "turn right"; (3) "go straight"; (4) "turn right"; (5) "go straight"; (6) "stop"; (7) "return"; (8) "turn left"; and (9) "turn left 5 (meters)". Figure 6 shows the path performed by the robot (red line) from the initial position (black circle) to the targets (black x). Notice that the robot was not aware of the targets' localization and was only following the human user commands. At each intersection of corridors, a new command is launched and performed by the robot. Notice that the robot successfully reaches the targets (even not knowing its own or the targets global position), avoiding the static obstacles present in the environment (green cubes) and remaining close to the right wall, as defined by the artificial vector field in Eq. (5). A video showing this simulation can be found at https://youtu.be/kl01d0wOE3s.

Additionally, to show that the proposed approach can deal with wider and more cluttered environments, we performed a simulation in a workspace composed of 7.5-m wide corridors populated by 20 obstacles of approximately the size of the robot, including two moving obstacles. In this simulation, the robot received three commands: (1) "go straight", (2) "turn right" and (3) "go straight." A video of this simulation can been seen at: https://youtu.be/zK95sSOQ7QE.

4.1. Comparison with potential function

To compare the proposed method with a reactive approach, we chose to implement the traditional method based on potential functions described in ref. [31]. In this method, the robot's velocity is determined by the sum of the gradient of an attractive potential function induced by the target and the gradient of repulsive potential functions induced by the obstacles. In our case, the normalized version of the vector field in Eq. (5) is used as the attractive field. The gradient of the repulsive potential function is obtained through the sum of the contribution of the ith obstacle detected by the laser as

$$\nabla U_i(d_i) = \begin{cases} k_r \left(\dfrac{1}{Q^*} - \dfrac{1}{d_i} \right), & d_i \leq Q^* \\ 0, & d_i > Q^* \end{cases}, \tag{7}$$

where d_i is the distance between the robot and obstacle i, Q^* is the radius of influence of the obstacle and k_r is a positive gain.

We have simulated the navigation of the robot in a corridor with obstacles 10 times, obtaining the paths executed by the robot using both, the strategy proposed in this work and potential functions. In the two situations, the robot was executing command "go straight." Figure 7 shows a comparison of two typical paths. Table II shows the results of the average values for (i) Euclidean cost; (ii) upstream cost; and (iii) smoothness.

It can observed by Table II that the distance traversed by the robot using the two strategies is statistically the same, if it is considered a T-test with significance level of 5%. However, we can notice that the upstream cost is much smaller for the proposed strategy, meaning that the robot more precisely followed the user command encoded by the vector field. Also, the proposed approach generated much smoother paths. This behavior can be also visualized in this comparison video: https://youtu.be/JyNZmqzTzp0. Smoothness would be an essential metric if the robot is carrying a fragile object or even a person, as is the case of wheelchairs.

We then conclude that a reactive strategy using potential functions is easy to implement but present poorer results. In addition, depending on the relative position of the obstacles, the reactive strategy can lead the robot to a local minimum, where the sum of each contributions ∇U_i and the attractive field is zero, stopping the robot indefinitely. This will never happen with the proposed approach if a free path exists within the robot's field of view. The reason for this is the use of complete planners in our algorithm.

4.2. Comparison with SLAM

In this section we evaluate our methodology with two kinds of maps: a memory-free local map generated directly with the newest laser scan, as proposed in our methodology (Section 2), and a global map obtained using Simultaneous localization and mapping (SLAM). We used the SLAM

Table II. Comparison between potential function and our strategy. The `p-value` was computed using the T-Test applied to each criteria.

	Potential function	**Our strategy**	**`p-value`**
Euclidean cost	20.97	21.05	0.62
Upstream cost	1.97	0.31	1.82e-09
`Smoothness`	362,868.13	17.71	6.92e-03

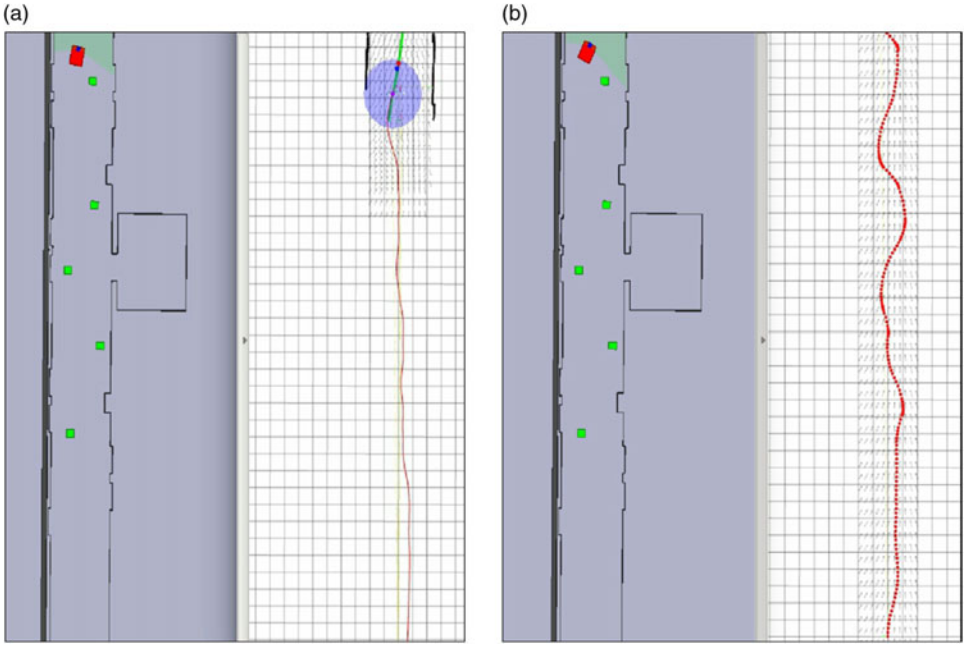

Fig. 7. Comparison between (a) the proposed methodology and (b) a reactive approach based on potential functions. In red is the simulated robot path.

implementation given by *hector_slam*, which was configured with the parameters suggested by the authors.[32] This package generates an occupancy grid which we binarized such that cells with an occupancy probability smaller or equal to 0.5 were considered free space.

To compare the two mapping versions, we also used a corridor where the robot is executing the "go straight" command. Once the robot starts its movement, two rectangular obstacles representing the legs of a person also start moving toward the robot. A sequence of snapshots of a simulation using SLAM is shown in Fig. 8. A complete simulation can be seen at https://youtu.be/pWG7zFOviDI.

Notice that SLAM causes a delay in the update of the actual position of the moving obstacles. This happens because the mapping algorithm needs a sequence of measurements to define the occupancy probability of a cell. Thus, even when the obstacle is very close to the robot, the global map indicates it is about 2 m way, as shown in Fig. 8(d). This erroneous information may lead to a collision, as shown in Fig. 8(e).

This experiment was repeated 40 times for the memory-free local map and 40 times for the global map generated using SLAM. For each method, the obstacle velocity was set to assume four different speeds. Thus, for each speed, 10 experiments were executed. For all simulations, the robot velocity was set to be 0.9 m/s, which is the maximum speed of our robot MARIA. Table III shows the rate of success that is the percentage of trials without collisions, for each situation. Observe that the memory-free local map increases the probability of moving obstacle avoidance. Besides, another advantage of the memory-free map is the lower memory requirement, once the global map created using SLAM grows as the robot moves around the workspace. On the other hand, SLAM creates a

Table III. Comparison of the rate of success of the methodology using our approach (memory-free local map) and SLAM. The planning time was 1.0 s, the robot speed was 0.9 m/s and the obstacle speed was v_{ob}.

	v_{ob} **(m/s)**			
Mapping	**0.10**	**0.25**	**0.40**	**0.50**
SLAM (%)	100	60	20	0
Memory-free local map (%)	100	100	90	70

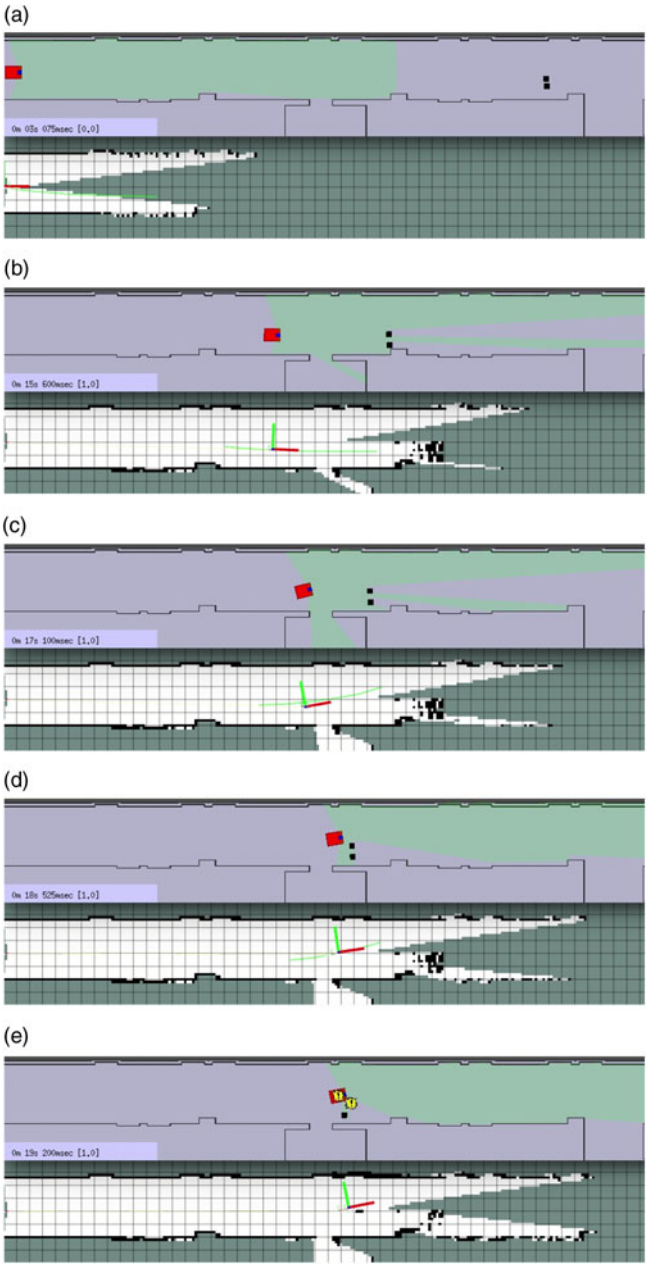

Fig. 8. Moving obstacle avoidance using a global map obtained using SLAM. Each snapshot shows a stage simulation and the global occupancy grid created by *hector_slam* package. The robot speed was configured to be 0.9 m/s, the obstacle velocity 0.5 m/s and the planning time 1.0 s.

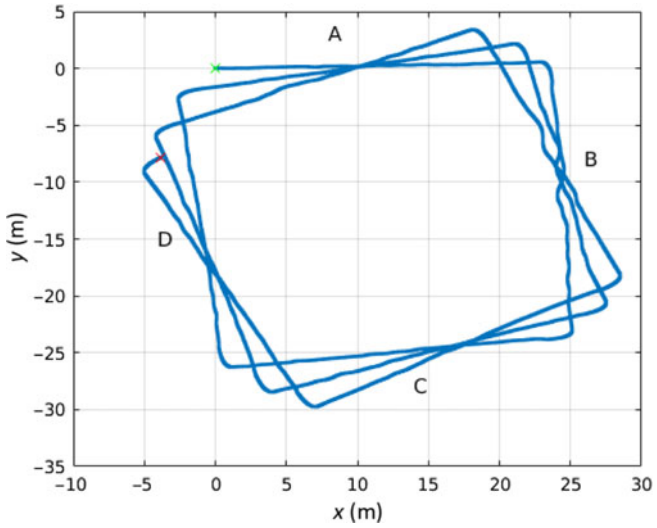

Fig. 9. Robot's odometry obtained during the execution of three laps in a closed-loop environment, composed of four corridors identified by A, B, C and D. The complete path measured approximately 300 m. We have added traffic cones to corridor B to complicate the passage of the robot. The robot started at the green point (corridor A) and stopped at the red dot.

very detailed global map, what facilitates wall detection. In a global map, a sequence of short-time coordinate frames is not required any more, although this strategy would make the system more robust to localization errors, as we show in next section.

5. Actual Robot Experiments

In this section, we show a series of experiments performed with MARIA, described in Section 3. We used the same code and parameters of the simulations.

Our first experiment evaluates the robustness of the proposed approach to odometry uncertainties. We have performed an experiment where the robot was commanded to circulate a given closed circuit inside the building. The user defined a simple sequence of two commands: "go straight" and "turn right," which was repeated 11 times, forcing the robot to complete three complete turns. The robot velocity was set to its maximum: 0.9 m/s. Besides the natural obstacles of the building, such as pillars and decorative pots, we have introduced some traffic cones to the robot path. No people were present during this experiment.

The odometry of the robot during the experiment is shown in Fig. 9. Snapshots of the experiment are shown in Fig. 10 and a complete video is at https://youtu.be/loS_TxmtQMI. After the experiment, we observed that the robot finished its mission very close to the starting point, what cannot be observed by the robot odometry, which presents a fairly large integration error. This error did not prevented the robot to complete its mission, since the methodology uses short-time coordinate frames. In this experiment, each coordinate frame was valid for only 1.0 s, during which the odometry presents very good results.

To show how the system behaves in the presence of moving obstacles, we run an experiment where a person moved against the robot while it was following a "go straight" command. Again, the robot was moving at $v_{max} = 0.9$ m/s but now with a planning time $t_s = 0.45$ s. Time reduction was necessary to allow the avoidance of a person moving at approximately 1.0 m/s. At this point, it is important to mention that, since we used an anytime motion planner, a reduction of the planning time may generate worse, but still complete paths.

This experiment was repeated 10 times and the robot was able to avoid the person in all trials. Figure 11 presents snapshots of the experiment. A complete video can be seen at https://youtu.be/DONxBJJ5OT8. Notice in Fig. 11(a) that a person starts its movement but because it is still outside the robot's planning radius (not shown in the figure), the planner ignores its presence. In Fig. 11(b) and (c), the person enters the robot's planning radius, causing the planning to change the path. Notice in Fig. 11(d) that the planned path considers the avoidance of an obstacle that was behind

Fig. 10. Snapshots of the experiments in Fig. 9. (a) Robot in corridor A; (b) robot in corridor B avoiding artificial obstacles; (c) robot in corridor C; and (d) robot in corridor D.

the actual obstacles detected by the laser. This happens because this path was generated with information obtained 0.45 s before. In this way, obstacle avoidance is only possible because the robot has started the avoidance some time before. It is then clear that, if the person had changed its direction or had increased its speed after path computation, avoidance would not be possible. This would happen because the person would be moving inside the robot's static zone, where it is assumed that all obstacles are static. However, except for sudden jumps and other very unique movements, we believe that the situation of this experiment, where person and robot are moving fast toward each other, is among the worst case scenarios that may happen in a real-world situation. Although we cannot guarantee obstacle avoidance without stopping the robot in these cases, we believe that the proposed methodology will perform well in the majority of cases. Remember that a collision caused by the robot will never happen, once it will stop when the person enters its emergency zone.

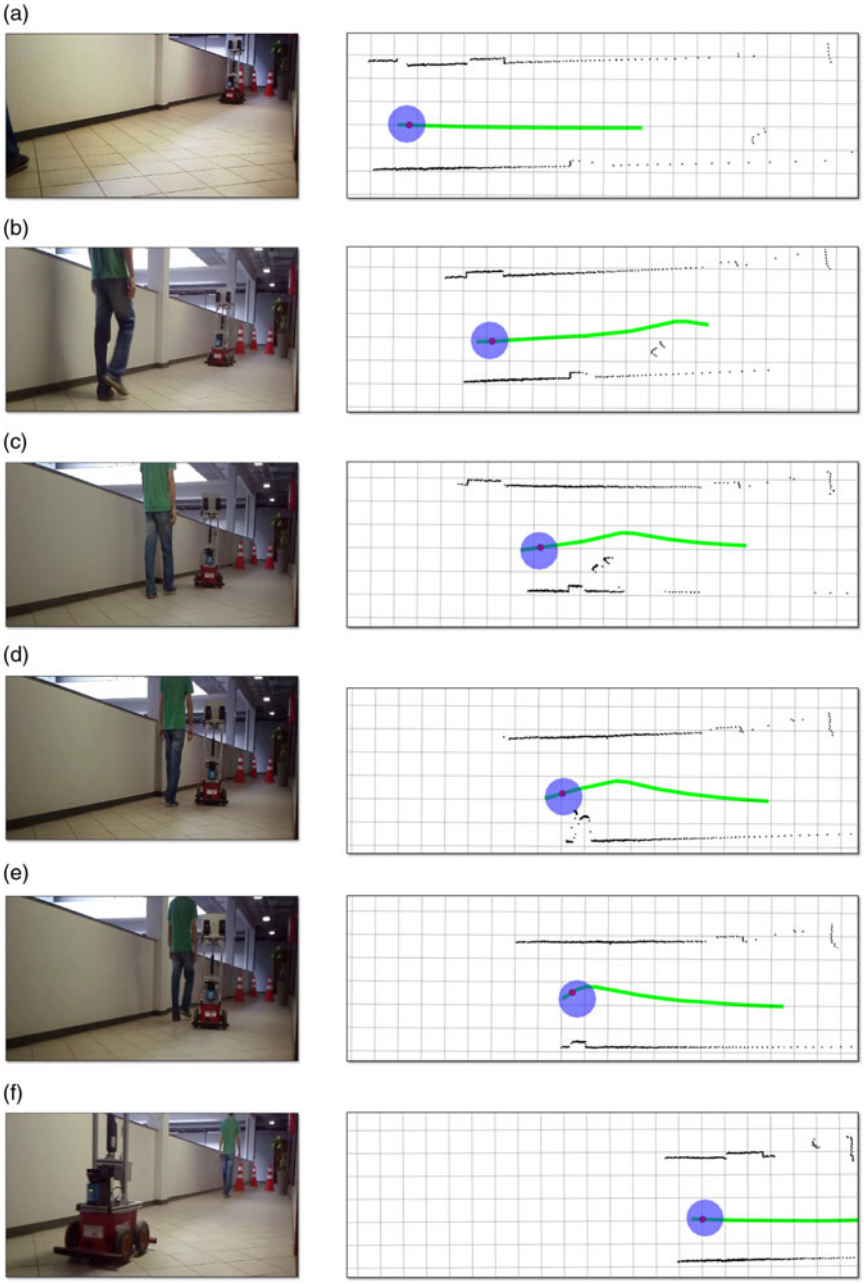

Fig. 11. Moving obstacle avoidance. At the right-hand side are snapshots of the experiment where a person moves toward the robot. At left are Rviz images showing the robot as a red dot, the planned path as a green line, laser data as black dots and the robot's static zone in blue.

6. Conclusions

This paper presented an architecture for safe navigation of semi-autonomous service robots that is based on motion planners and does not require global localization. The proposed architecture assumes that a human user is the part of the robot's control loop, although in this work we did not have the objective of studying the several aspects of human–robot interaction, which have been extensively covered by several other works.[33]

The proposed strategy is especially useful for robots that must follow user commands through safe and smooth paths, but do not have a global map of the environment or a global localization system. The basic idea of the strategy is to keep a local coordinate frame only by the time needed by the robot to follow part of the last computed path and, at the same time, to compute a new path that considers possible changes in the workspace. After the new path is computed, a new coordinate

frame is defined and the path is transformed to the new coordinates. The principle of the strategy is that two coordinate frames are very close to each other and, therefore, it is possible to define a coordinate transformation between them using odometry only.

Since our method was designed to be used with semi-autonomous robots, we proposed the codification of user commands as simple vector fields. This allowed the use of previous methodologies that integrates optimal planners and vector fields as the planner used in our method. With these strategies, specially,[20] which was used in the experiments of this paper, the robot is able to follow the vector field as close as possible and also to avoid unknown obstacles.

The experimental comparison between our strategy and a reactive approach shows that the former, which is based on optimal planners, generates smoother and faster robot paths. Also, because the planners are also complete or probabilistically complete, the proposed methodology is immune to local minima, which usually cause the robot to stop when reactive approaches are used. On the other hand, when our strategy, which does not require global localization, was compared to a SLAM-based alternative, we show that it can present comparable results in terms of the quality of the path, in spite of its simplicity and smaller computational cost. Additionally, our memory-free strategy has presented better results in the presence of moving obstacles.

By analyzing the proposed algorithm, we observe that a robot running our strategy is able to avoid static obstacles and, depending on the relationship among obstacle speed, robot speed and planning time, we can also avoid moving obstacles, including people, as shown in our experiments. Therefore, future works may include increasing the efficiency of the planners, what can be done, for example, by implementing some of its functions in GPU or other dedicated hardware. The prediction of the obstacle motion is something that could also improve the approach.

Another extension of the work would be the inclusion of differential and/or dynamic constraints in the optimization problem. This would allow the methodology to be applied to faster robots. Also, this modification would potentially make the ride of intelligent wheelchairs users more comfortable, since we would be able to limit the chair accelerations and jerks.

Acknowledgments

G. Pereira and E. Freitas were with the Graduate Program of Electrical Engineering, Federal University of Minas Gerais (UFMG) during the realization of this work. This work was partially supported by CNPq/Brazil, grant 465755/2014-3 (National Institute of Science and Technology) and FAPEMIG/Brazil, grant TEC-APQ-00850-15. G. Pereira held a research productivity fellowship from CNPq.

References

1. F. Dayoub, T. Morris and P. Corke, "Rubbing shoulders with mobile service robots," *IEEE Access* **3**, 333–342 (2015).
2. R. C. Simpson, E. F. LoPresti and R. A. Cooper, "How many people would benefit from a smart wheelchair?" *J. Rehabil. Res. Dev.* **45**(1), 53 (2008).
3. J. Müller, C. Stachniss, K. O. Arras and W. Burgard, "Socially Inspired Motion Planning for Mobile Robots in Populated Environments," *Proceedings of the International Conference on Cognitive Systems* (Springer, 2008) pp. 85–90.
4. I. Skog and P. Handel, "In-car positioning and navigation technologies—a survey," *IEEE Trans. Intell. Transp. Syst.* **10**(1), 4–21 (2009).
5. F. Andresen, L. Davis, R. Eastman and S. Kambhampati, "Visual Algorithms for Autonomous Navigation," *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2 (IEEE, 1985) pp. 856–861.
6. D. Keirsey, E. Koch, J. McKisson, A. Meystel and J. Mitchell, "Algorithm of Navigation for a Mobile Robot," *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 1 (IEEE, 1984) pp. 574–583.
7. O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robot. Res.* **5**(1), 90–98 (1986).
8. H. P. Moravec, "Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover," *Technical Report*, DTIC Document (1980).
9. R. C. Arkin, *Behavior-Based Robotics* (MIT Press, Cambridge, 1998).
10. T. Balch, "Avoiding the Past: A Simple but Effective Strategy for Reactive Navigation," *Proceedings of the IEEE International Conference on Robotics and Automation* (IEEE, 1993), pp. 678–685.
11. E. Baklouti, N. B. Amor and M. Jallouli, "Reactive control architecture for mobile robot autonomous navigation," *Robot. Auto. Syst.* **89**, 9–14 (2017).

12. J.-C. Chien, Z.-Y. Dang and J.-D. Lee, "Navigating a service robot for indoor complex environments," *Appl. Sci.* **9**(3), 491 (2019).
13. S. F. Chik, C. F. Yeong, E. L. M. Su, T. Y. Lim, Y. Subramaniam and P. J. H. Chin, "A review of social-aware navigation frameworks for service robot in dynamic human environments," *J. Telecommun. Electr. Comput. Eng.* **8**(11), 41–50 (2016).
14. B. Graf, "Reactive Navigation of an Intelligent Robotic Walking Aid," *Proceedings 10th IEEE International Workshop on Robot and Human Interactive Communication*. ROMAN IEEE, 2001 (Cat. No. 01TH8591) (2001) pp. 353–358.
15. G. Pires and U. Nunes, "A wheelchair steered through voice commands and assisted by a reactive fuzzy-logic controller," J. Intell. Robot. Syst. **34**(3), 301–314 (2002).
16. H. Yang, J. Lim and S.-E. Yoon, "Anytime RRBT for Handling Uncertainty and Dynamic Objects," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2016) pp. 4786–4793.
17. G. A. S. Pereira, L. C. A. Pimenta, A. R. Fonseca, L. D. Q. Corrêa, R. C. Mesquita, L. Chaimowicz, D. S. C. De Almeida and M. F. M. Campos, "Robot navigation in multi-terrain outdoor environments," *Int. J. Robot. Res.* **28**(6), 685–700 (2009).
18. M. Juliá, A. Gil and O. Reinoso, "A comparison of path planning strategies for autonomous exploration and mapping of unknown environments," *Auto. Robot.* **33**(4), 427–444 (2012).
19. F. Tian, S.-R. Ge and H. Zhu, "A Navigation Control Strategy with Hybrid Architecture for Rescue Robot," *2010 IEEE International Conference on Intelligent Computing and Intelligent Systems*, vol. 1 (IEEE, 2010) pp. 675–680.
20. G. A. S. Pereira, S. Choudhury and S. Scherer, "A Framework for Optimal Repairing of Vector Field-Based Motion Plans," *Proceedings of the International Conference on Unmanned Aerial Systems* (IEEE, 2016) pp. 261–266.
21. S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.* **30**(7), 846–894 (2011).
22. J. Minguez, F. Lamiraux and J.-P. Laumond, "Motion planning and obstacle avoidance," In: Siciliano B., Khatib O. (eds) Springer Handbook of Robotics (Springer, Berlin, Heidelberg, 2008) pp. 827–852.
23. S. Karaman, M. R. Walter, A. Perez, E. Frazzoli and S. Teller, "Anytime Motion Planning Using the RRT*," *Proceedings of the IEEE International Conference on Robotics and Automation* (IEEE, 2011) pp. 1478–1483.
24. T. Fraichard, "A Short Paper About Motion Safety," *Proceedings 2007 IEEE International Conference on Robotics and Automation* (IEEE, 2007) pp. 1140–1145.
25. V. M. Gonçalves, L. C. A. Pimenta, C. A. Maia, B. C. O. Dutra and G. A. S. Pereira, "Vector fields for robot navigation along time-varying curves in n-dimensions," *IEEE Trans. Robot.* **26**(4), 647–659 (2010).
26. I. Ko, B. Kim and F. C. Park, "Randomized path planning on vector fields," *Int. J. Robot. Res.* **33**(13), 1664–1682 (2014).
27. G. A. S. Pereira, S. Choudhury and S. Scherer, "Kinodynamic Motion Planning on Vector Fields Using RRT*," *Technical Report CMU-RI-TR-16-35* (Robotics Institute, Pittsburgh, PA, 2016).
28. I. A. Şucan, M. Moll and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.* **19**(4), 72–82 (2012). http://ompl.kavrakilab.org.
29. 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," *ICRA Workshop on Open Source Software* (2009).
30. B. d'Andréa Novel, G. Campion and G. Bastin, "Control of nonholonomic wheeled mobile robots by state feedback linearization," *Int. J. Robot. Res.* **14**(6), 543–559 (1995).
31. H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations, Intelligent Robotics and Autonomous Agents Series* (MIT Press, Cambridge, 2005).
32. S. Kohlbrecher, J. Meyer, O. von Stryk and U. Klingauf, "A Flexible and Scalable Slam System with Full 3d Motion Estimation," *Proceedings of IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)* (IEEE, 2011).
33. T. B. Sheridan, "Human–robot interaction: Status and challenges," *Human Factors* **58**(4), 525–532 (2016).