Dynamic Trajectory Planning for Mobile Robot Navigation in Crowded Environments

Stefano Primatesta, Ludovico Orlando Russo, Basilio Bona
Dipartimento di Automatica e Informatica
Politecnico di Torino
Torino, Italy
Email: {name.surname}@polito.it

***Abstract*—This paper describes a trajectory planning algorithm for mobile robot navigation in crowded environments; the aim is to solve the problem of planning a valid path through moving people.**

The proposed solution relies on an algorithm based on the Informed Optimal Rapidly-exploring Random Tree (Informed-RRT*), where the planner continuously computes a valid path to navigate in crowded environments. While the robot executes the trajectory of the current path, this re-planning method always allows a feasible and optimal solution to be obtained.

Compared to other state-of-the-art algorithms, this solution does not compute the entire path each time an obstacle is detected, instead it evaluating the current solution validity, i.e., the presence of moving obstacles on the current path; in this case the algorithm tries to repair the current solution. Only if the current path is completely unacceptable is a new path computed from scratch. Thanks to its reactivity, our solution always guarantees a valid path that brings the robot to the desired goal position.

This dynamic approach is validated in a real case scenario where a mobile robot moves through a human crowd in a safe and reliable way.

I. Introduction

The presence of robots in our lives is growing. Service robotics aims at developing novel robotic solutions that cooperate with human beings. To this reason, it is necessary for a mobile robot to be able to navigate among people in a safe and reliable way.

Navigation in crowded environments is challenging, since they change very quickly. State-of-the-art approaches in robot navigation compute the path from the initial position to the goal position, assuming the environments to be static; if the robot then senses an obstacle on the computed path, it recomputes the entire path from scratch. In crowded environments path computation is performed several times, so that the robot stops and changes path too often to reach the goal in a reasonable amount of time.

Autonomous navigation is very important in Service Robotics, because it allows any desired position to be reached in a safe and reliable way. For example in [1], a robot navigates in autonomy for Data Center Monitoring. Instead, in [2] a mobile robot explores museum spaces and shows a real-time video to remote users.

Several approaches have been proposed to solve the problem of mobile robot navigation in crowded environments. In particular, there are two strands of thought: (i) prediction of people motion, or (ii) dynamic path planning.

Very important work has been done by P. Trautman; in [3], he introduced an approach to solve the freezing robot problem using Interacting Gaussian Processes (IGP) to model and predict people trajectory. In [4] he focused on Multiple Goal Interacting Gaussian Processes (MGIGP), which aim at estimating the behaviour of a crowd, and introduced a cooperation model between robot and people. In this way the robot can navigate in dense human crowds. This solution requires the environment to be structured, since a pedestrian tracking system must be put in place.

In [5], P. Henry at al. proposed using Inverse Reinforcement Learning (IRL) to learn some of the pedestrians' decisions. The robot is hence controlled so that the humans' predicted path is minimally disrupted.

However, these approaches require tracking all the people in a crowd. To do this the use of onboard robot sensors is not enough; it is often necessary to use external sensors to monitor the human crowd.

In dynamic Path Planning, important work has been done with Sample-Based methods, in particular using the RRTs-based algorithms (Rapidly-exploring Random Trees).

The original RRT was introduced by S. LaValle in [6] to solve a Static Path Planning problem. RRT is able to find a feasible solution, even in a high-dimensional state space, but it is not guaranteed to be optimal. For this reason, many improvements to the original RRT approach have been proposed in literature. One of these was introduced by Karaman and Frazzoli in [7], where the authors proposed an alternative method, known as RRT*. It is an RRT-based algorithm with asymptotic optimality. In [8], the authors proposed an Anytime Motion Planning using RRT*, where, after a first solution is found, it continues to search for an optimal path.

In general, Sample-Based algorithms are used in static Path Planning approaches. Dynamic path planning is more difficult, since the movements of obstacles are unknown.

In [9] Svenstrup et al. use a modified version of RRT to compute a path in a human environment: it estimates people's motion in order to have a better and natural path. A similar approach has been proposed in [10] where the authors use Probabilistic RRT to navigate in dynamic environments.

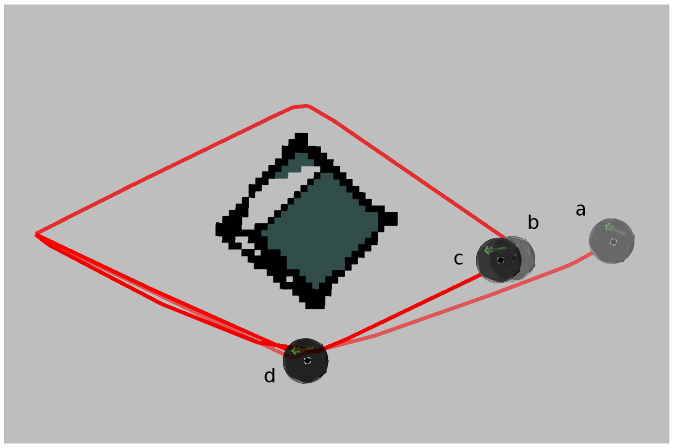


Figure 1. Example of indecision behaviour. In (a) the robot follows the solution path. During the execution, the algorithm computes new paths at every loop. The resulting indecision behaviour can be viewed between (b) and (c), where the robot changes direction toward different trajectories. However, after many loops the robot overcomes the obstacle (d).

Another solution to Dynamic Path Planning is the re-planning method. In [11] S. Yun uses a Genetic Algorithm with a re-planning approach to avoid obstacles dynamically. A similar approach is used in [12] by D. Ferguson et al. In this work, the authors introduce a re-planning method in RRTs algorithms. When a solution is interrupted by an obstacle, the algorithm removes the invalid segments and computes new ones, starting from the current exploration tree. In [13] and [14], the authors propose a reactive motion planning method capable of re-planning the path online when the environment changes during the execution.

In [15] the authors use bi-directional RRT (Bi-RRT) with a re-planning method to solve a Dynamic Path Planning problem. In this work a novel 2D-span re-sampling is used to repair an invalid path.

In any case, with the dynamic path planning approach there can be undesirable behaviour. When a trajectory is computed to go around an obstacle, the algorithm searches for an optimal path that always passes on the same side of the obstacle. Since RRT*-based algorithms find a near optimal solution, if the distance cost is similar on both sides, there can be indecision. In fact the algorithm can find solution paths on alternating sides, during different runtime loops. This causes indecision behaviour on the robot motion, which first follows one side, then the other. Figure 1 depicts this behaviour.

In this paper, we propose a new framework for dynamic motion planning in order to overcome the crowd, where people are assumed to be simple dynamic obstacles. A re-planning method is used to maintain a valid path from start to goal positions. Initially the algorithm computes the entire path; then, while the robot moves along this path, the algorithm continues to check, repair and improve the path, while the environment changes.

This approach uses a sample-based algorithm called Informed-RRT*. This algorithm was introduced in [16] by J. Gammel et al. and allows a fast computation of the solution path in a high-dimensional space to be performed.

The method permits to be maintained stable trajectories, producing a better movement of the robot and avoiding the indecision behaviour described above. In fact, since it computes a new path each time, it involves a continuous change of trajectories that the robot follows by changing direction. Another benefit is a reduction in CPU resources.

In this work we adopt a very simple approach to solve the path planning problem in highly dynamic environments. We do not perform motion estimation of dynamic obstacles. In fact, in a crowded environment it is very difficult to track all the people around the robot using only on-board sensors. Our solution does not require information from external sensors, in order to reduce the system complexity and extend the application scenarios.

The paper is organized as follows: in Section II the original Informed-RRT* is described. Then in Section III our method for efficient re-planning in dynamic environments is presented. Afterwords, in Section IV the proposed method is validated by experimental result in simulation and in a real environment.

II. The Informed-RRT* algorithm

In this section the original Informed-RRT*, introduced in [16] by Gammel et al., is discussed. Informed-RRT* performs stochastic research using a sampling-based method, such as the original Rapidly-exploring random tree (RRT). RRT is popular in motion planning because it efficiently finds solutions to single-query problems. Optimal RRT (RRT*) is an improvement that allows a near-optimal solution to be found. Indeed RRT is not optimal, because the existing tree biases future expansions during the exploration. Instead RRT* connects new states considering all near states, even if this means replacing an edge in an existing tree.

Informed-RRT* is a simple modification to RRT*; this method preserves the same completeness and optimality as RRT*, while improving the convergence rate and the final solution quality. Informed-RRT* improves RRT* when it optimizes the path length, i.e. it minimizes the distance cost function.

Let us define an optimal planning problem. Let $X \subseteq \mathbb{R}^n$ be the state space of the planning problem, $X_{obs} \subseteq X$ the invalid states, i.e., those where an obstacle is present, and $X_{free} = X \setminus X_{obs}$ the remaining valid states. We define $x_{start}, x_{goal} \in X_{free}$, respectively the initial state and the final state. Let Σ be the set of all paths, where a single path $\sigma : [0,1] \to X$ is a sequence of states. The path planning algorithm searches an optimal path σ^* from x_{start} to x_{goal} in X_{free} that minimizes a given cost function $c : \Sigma \to \mathbb{R} \geqslant 0$:

$$\begin{aligned} & \sigma^* = arg \min_{\sigma \in \sum} c(\sigma) \\ \text{subject to} \quad & \sigma(0) = x_{start} \\ & \sigma(1) = x_{goal} \\ & \forall s \in [0,1]\,,\ \sigma(s) \in X_{free}. \end{aligned} \tag{1}$$

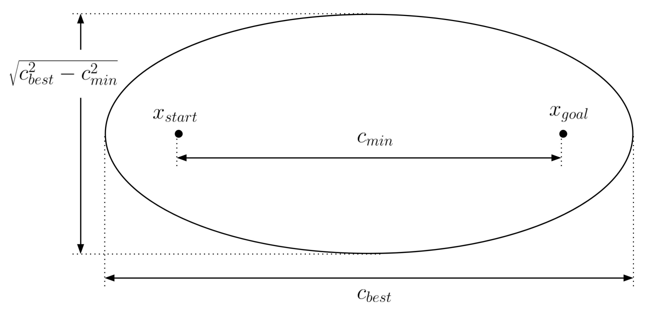


Figure 2. Ellipse that describes the ellipsoidal informed subset of states. The focal points are x_{start} and x_{goal}.

Now, let us assume that $f(x)$ is the cost of an optimal path and there is a set of states $X_f \subseteq X$ that improve the current path with current solution cost c_{best}.

$$X_f = \{x \in X \mid f(x) < c_{best}\} \tag{2}$$

In general $f(\cdot)$ is unknown, but we can estimate an heuristic function $\hat{f}(\cdot)$. Similarly we can estimate a subset of poses $X_{\hat{f}}$, such that $X_{\hat{f}} \supseteq X_f$.

Informed-RRT* uses an ellipsoidal informed subset of states that may improve the current solution with current solution cost c_{best}.

$$X_{\hat{f}} = \{x \in X \mid \|x_{start} - x\|_2 + \|x_{goal} - x\|_2 \leqslant c_{best}\} \tag{3}$$

In the resulting ellipse, x_{start} and x_{goal} are the focal points. The value of major axis is c_{best} and the minor axis is $\sqrt{c_{best}^2 - c_{min}^2}$. Figure 2 shows the resulting ellipse.

The algorithm is similar to RRT*. In the same way it looks for optimal path by incrementally building a tree in state space. It differs from RRT* once a first solution is found, because it focuses the search on the part of the planning problem that can improve the solution. This is possible by sampling in ellipsoidal informed subset of states $X_{\hat{f}}$. In this way it reaches an optimal solution quicker than RRT*. Figure 6 and Figure 7 show the different behaviour of RRT* and Informed-RRT*

Sample-based algorithm requires a definition of State Space (or Configuration Space). State Space defines all possible configuration of states where the algorithm can sample and build exploration tree. The bounds of State Space are defined by dimension of the environment map. It is necessary to define the degree of freedom of State Space. In this work it is used a robot with differential drive that it rotates and moves forward in 2D workspace. As already discussed in previous section, the trajectory planner algorithm is used to define the optimal path from current to desired pose. After the robot follows each position in solution path using a local planner algorithm. This means that the orientation of the robot during the path is not relevant, except the final pose. For this reason the Configuration Space is described using R^2, where distance function is the L^2 norm.

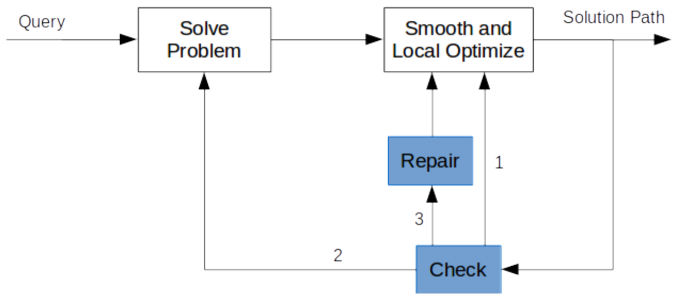


Figure 3. Main architecture of the proposed algorithm. After Check routine there are three possible cases: path valid (1); path invalid (2); path invalid but repairable (3).

III. Informed-RRT* based Trajectory Planning

The architecture of the path planning algorithm is depicted in Figure 3. The core is the Informed-RRT* algorithm described in the previous Section, that finds a near-optimal solution rapidly. The idea is that while the robot executes the current solution, the trajectory planner continues to check its validity and, if necessary, it computes a new one. In fact with dynamic environments, the path is valid for sure only in the moment it is computed.

Since we consider a dynamic environments, then X_{obs} and X_{free} change in time, so that, at each iteration k, at time s_k of the algorithm, we can define different sets $X_{obs}(k)$ and $X_{free}(k)$, such that $X_{free}(k) = X \setminus X_{obs}(k)$.

At every iteration, the sensors measurements allow to update $X_{obs}(k)$ and $X_{free}(k)$; then the Check routine is executed in order to validate the current path in the new environment. The algorithm computes a score $\chi(\sigma)$ over the current path, that is defined as

$$\chi(\sigma) = \frac{|\{V_{obs}, E_{obs} \in \sigma([s_k, 1])\}|}{|\{V, E \in \sigma\}|}, \tag{4}$$

where V and E are, respectively, the vertexes and the edges of the path σ, while V_{obs} and E_{obs} are the invalid vertexes and edges of the path not yet executed $\sigma([s_k, 1])$, i.e., edges that intersect $X_{obs}(k)$ and edges that belong to $X_{obs}(k)$; the $|\cdot|$ set operator represents the cardinality of a set.

Three different cases can occur:

1) The current path is valid. There is no intersection between the current path not yet executed $\sigma([s_k, 1])$ and the obstacles $X_{obs}(k)$, i.e.,
$$\chi(\sigma) = 0. \tag{5}$$
2) The path is invalid, and several states intersects $X_{obs}(k)$, i.e.,
$$\chi(\sigma) \geq \chi_{max}. \tag{6}$$
3) The path is invalid, but only few states intersects $X_{obs}(k)$, that is
$$0 < \chi(\sigma) < \chi_{max}. \tag{7}$$

Where χ_{max} is a given threshold.

In the first case the current path is valid and the robot can continue to follow it. In our approach we execute a short-cut and smoothing routine over the path. In this way the robot can follow a smoother trajectory.

In the second case the algorithm searches a new solution. The search space is completely changed and the old exploration tree is invalid. Given a new start pose x_{start}, the algorithm solves the new planning problem and returns a new solution path to execute. Of course, the new start pose is set as the current robot pose $x_{start,new} = \sigma(s_k)$.

In the third case the path is invalid only in few states. In this case it is useless to find a new solution, since we can simply repair the invalid states.

The path planner evaluates the occupation of the robot by determining the robot radius r_{robot}, in this way we are sure that the robot can navigate through obstacles. The value of r_{robot} is then inflated to 110% for safety reasons.

As already discussed, while the algorithm checks and returns the updated path, the robot follows the current solution. This means that it can follows an invalid path for brief time. In the case of normal crowded environments the trajectory planner frequency must run at least at 2 Hz (or higher frequency). With a lower frequency the path can be invalid too many times, and the executed path might move towards people.

In [17] authors study the frequency and velocity of walking people; in the worst case the mean value of walking velocity is 1.38 m/s. This means that with a planner frequency of 2 Hz a robot follows an invalid path for 500 ms, while people move 0.69 meters. Experimental tests described in Section IV showed that with normal human crowd there is no collision.

This new approach has some advantages. Computing new solution each time, a robot can change the trajectory, causing a discontinuous movement. With a check and repair routine, the main trajectory remains stable, because the algorithm adapts the path to the obstacles movement. The resulting robot motion is smoother. Preserving old path does not always guarantees the optimal solution, but the resulting robot behaviour is considerably improved.

A. Pseudo-code

In this Section a pseudo-code is presented to illustrate the proposed algorithm. Algorithm 1 describes the main routine that computes and evaluates the solution path. Algorithms 2 and 3 describe the repair routines, in case of invalid states and edges respectively.

As showed in Algorithm 1, when a new x_{goal} goal position is available, the Trajectory Planning routine starts. First of all, the exploration tree τ and the path σ are reset, in order to erase old data.

After that, the algorithm starts the main routine, until robot reaches x_{goal}.

The function *GoalReached()* returns the robot status while it follows the solution path. It returns the true value only if the robot reaches the goal position.

Algorithm 1 Trajectory Planning Informed-RRT* based

```
1: procedure DYNAMICPLANNING($x_{start}, x_{goal}$)
2:     $\tau \leftarrow reset()$
3:     $\sigma \leftarrow reset()$
4:     while $GoalReached() = False$ do
5:         $UpdateSearchSpace()$
6:         $\chi \leftarrow Check(\sigma)$
7:         if $\chi > \chi_{max}$ then
8:             $\tau \leftarrow reset()$
9:             $\tau \leftarrow InformedRRT^*(x_{start}, x_{goal})$
10:            $\sigma \leftarrow BestTrajectory(\tau)$
11:        else if $\chi > 0$ then
12:            $\sigma \leftarrow RepairStates(\sigma)$
13:            $\sigma \leftarrow RepairEdges(\sigma)$
14:        end if
15:        $\sigma \leftarrow LocalShortcut(\sigma)$
16:        $\sigma \leftarrow Smoother(\sigma)$
17:        $\sigma \leftarrow Interpolate(\sigma)$
18:        $ExecutePath(\sigma)$
19:    end while
20:    return
21: end procedure
```

Algorithm 2 RepairStates() routine

```
1: function REPAIRSTATES($\sigma$)
2:     for $i = 1$ to $i = InvalidState.size()$ do
3:         $x_n = InvalidState[i]$
4:         $\sigma \leftarrow Disconnect(x_n)$
5:         $n_{new} = NeighborSample(x_n, r_{max})$
6:         if $StateValid(x_{new})$ then
7:             $\sigma \leftarrow Reconnect(x_{new})$
8:         end if
9:     end for
10:    return $\sigma$
11: end function
```

Algorithm 3 RepairEdges() routine

```
1: function REPAIREDGES($\sigma$)
2:     for $i = 1$ to $i = InvalidEdge.size()$ do
3:         $E_n = InvalidEdge[i]$
4:         $SearchNearState(x_{n-1}, x_{n+1})$
5:         $\sigma \leftarrow EraseEdge(E_n)$
6:         $n_{new} = NeighborSample(x_{n-1}, r_{max})$
7:         if $StateValid(x_{new})$ then
8:             $\sigma \leftarrow Reconnect(x_{new})$
9:         end if
10:    end for
11:    return $\sigma$
12: end function
```

At every loop the State Space must be updated. The environment is modified by sensor measurements that set $X_{obs} \subseteq X$ and hence X_{free}. When algorithm starts, it loads a static map of environments. During execution sensors detect any type of obstacle, both static and dynamic and sensors measurement allow the map to be modified, and by consequence the State Space changes.

In line 6, *Check()* function verifies that the current solution path is valid and returns a ratio of invalid states, according to equation 4. The algorithm checks that each state and edge is in free space in the updated Configuration Space, then $x_n, E_n \in X_{free}$. Verification is carried out using a r_{robot} tolerance, where r_{robot} is the robot inflation radius. At this point, the function evaluates the $\chi(\sigma)$ value; after that, in lines from 7 to 14, the value from $\chi(\sigma)$ is evaluated according to equations (5), (6) and (7) and, accordingly, the appropriate strategy on the path is executed.

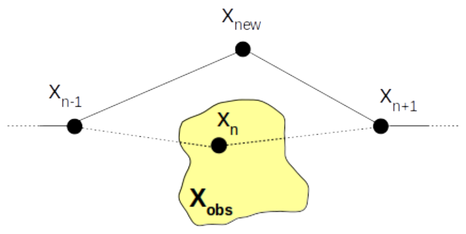
If path is invalid, the algorithm computes a new solution (lines from 8 to 10). Given x_{start} and x_{goal}, the Informed-RRT* algorithm computes a solution path in search space. Before computing new plan, the old exploration tree is reset; this is necessary, since checking and repairing all states in τ is inefficient. Informed-RRT* builds a new exploration tree τ. The solution path is the trajectory with the lower cost; *BestTrajectory()* function searches the minimum path σ in τ.

If the path is invalid but repairable, the algorithm repairs the invalid states and edges (lines 12, 13). As shown in Algorithms 2 and 3, there are two main simple cases: an invalid state or an invalid edge between states. Figure 4(a) describes the first case, where the algorithm searches a new state by sampling around the invalid state. New sample must belong to a restricted area around the old state with a radius r_{max}, that in the present context is set to 1 meter. If a new state is found, the procedure erases near edges $e_n = \{x_{n-1}, x_n\}$, $e_{n+1} = \{x_n, x_{n+1}\}$ connected to invalid state x_n, and connects x_{new} with other states, then $e_n = \{x_{n-1}, x_{new}\}$, $e_{n+1} = \{x_{new}, x_{n+1}\}$. Instead in Figure 4(b) there are valid states, but invalid edges. In this case there is no need to replace any state. The algorithm erases the edge $e_n = \{x_{n-1}, x_n\}$, and looks for a new state x_{new} and connects it to neighbours states. Then $e_n = \{x_{n-1}, x_{new}\}$, $e_{n+1} = \{x_{new}, x_n\}$. With successions of invalid states the procedure is the same as for a single state. In general, with a *Invalid Ratio* less than χ_{max}, the repair function works with success. If the function returns a failure, a new path must be computed, as in line 8, 9, 10.

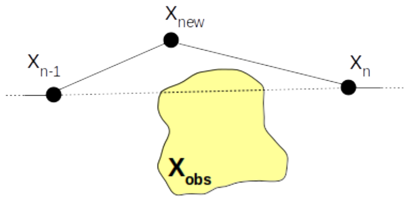
If the path is completely valid, no extra routines are executed on the path.

Finally, until robot reaches goal pose, in line 18 the resulting path σ is executed by external local planner.

In this work we set experimentally $\chi_{max} = 0.15$ that represents a balanced threshold. This value may change in different scenarios or with different map sizes. With higher threshold value, the algorithm may not successfully repair a path in the required time. However with lower ratio it is necessary to compute a new path, even if it is not necessary.



(a)



(b)

Figure 4. The main simple cases of repair procedure. In (a) there is an invalid state and it's need to replace it with new one. In (b) there's an invalid edge. It's need to add new state in order to avoid the X_{obs} area.

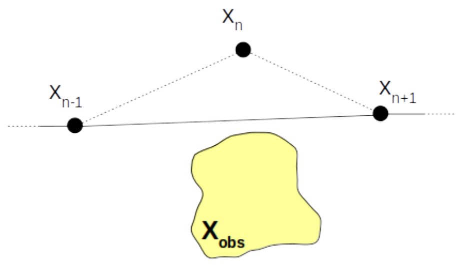


Figure 5. Example of short-cut procedure. x_{n-1} and x_{n+1} can be connected directly without connecting to x_n. The function erases x_n only if the resulting cost is lower.

Notice that, in the first cycle, the solution path σ is empty, and it is considered as invalid.

At the end the algorithm improves the path using three different routines: *LocalShortcut()*, *Smoother()* and *Interpolate()*.

LocalShortcut() provides a simplified solution. As shown in Figure 5, given a state x_n, if one can connect directly x_{n-1} and x_{n+1} with a minor cost, x_n can be eliminated. This procedure is dangerous if it compares states at long distance, because it can significantly change the solution path.

The *Smoother()* function smooths the path using a Spline function.

Finally the *Interpolate()* function adds new states in the path in order to guarantee to have states every 0.10 meters. This is useful when the robot executes the trajectory, since it follows each state in sequence. Using densely placed states, the robot follows the solution path more closely.

At the end of every loop the algorithm returns the solution path σ.

IV. Experimental Results

The algorithm presented in the previous Sections has been tested in simulation and in real case in crowded environments.

The algorithm has been implemented over the Open Motion Planning Library (OMPL) [18]. It contains implementations of many sample-based algorithms for path planning, in particular many RRTs-based planner. In addition there are many State Space included in the library, both in 2D and in 3D.

The main advantage of OMPL is modularity. OMPL code is divided in classes, where each class has a specific function to solve planning problem. Most important classes are *ProblemDefinition*, *Planner* and *SpaceInformation* and they are independent. For this reason, in the proposed framework it is possible to replace the *Planner* type algorithm with another one without modifying the external code. The same concept applies with the State Space type (in *ProblemDefinition* class). As already explained, we have implemented an Informed-RRT* based Planner that works over a Real Vector State Space R^2.

In order to work in 3D space, the entire map system must be changed.

The proposed solution has been implemented using the Robot Operating System (ROS) [19], that enables an easy deployment in simulation and on real robots.

The simulation and real tests are described below; in both cases a Turtlebot 2 robot is used.

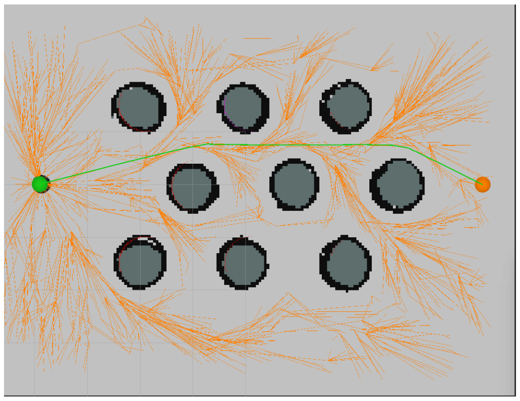
A. Simulation

To check and debug the developed solution, we firstly run it on a simulated environment. The first test was executed in a simulated static environments, in order to evaluate the behaviour of the trajectory planner. To simulate robot in a realistic scenario we used the Gazebo Simulator [20].

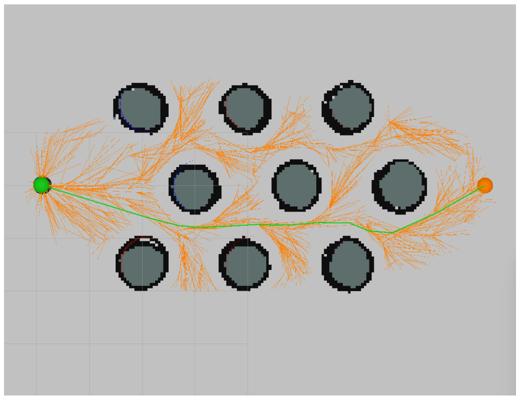
As described in the previous Section, the algorithm presented in this paper uses Informed-RRT* to search the solution path. In Figures 6 and 7 the use of Informed-RRT* and RRT* is compared. The Informed Space used by Informed-RRT*, as described in equation (3), is reduced in comparison with the RRT* that uses the entire search space. In Figure 6 the algorithm was able to compute a solution path in about $0.5s$. When there is no time constrain RRT* and Informed-RRT* find similar solution paths. In Figure 7 the solution time has been reduced to $0.1s$ and the behaviour of the algorithms changes. The best use of the search space permits Informed-RRT* to find a near optimal solution. RRT* finds a solution too, but it is not optimal.

Both RRT* and Informed-RRT* find first solution in a same way. Starting from it, in Figure 7 RRT* does not find any solution that improves the current cost c_{best}. However Informed-RRT* computes its informed space where it tries to find a better solution.

In this framework it is necessary to find the optimal solution path quickly. This is required because people moves continuously and, in worst cases, it is required to find new paths as quickly as possible. In Section III, we set the minimum planner frequency at $2Hz$; as a consequence, the solution time



(a)



(b)

Figure 6. An example where we simulate a planning problem. In both figures the start and goal points, as well as the statics environment, are the same; the solution time is 0.5 seconds for both methods. Figure (a) shows the RRT* solution, while in (b) the same solution obtained with Informed-RRT* is presented. The solution path is represented in green, the exploration tree in orange. Notice that Informed-RRT* uses Search-Space more efficiently, while RRT* continues to use the entire Space. After 0.5 seconds both algorithms find a near-optimal solution.

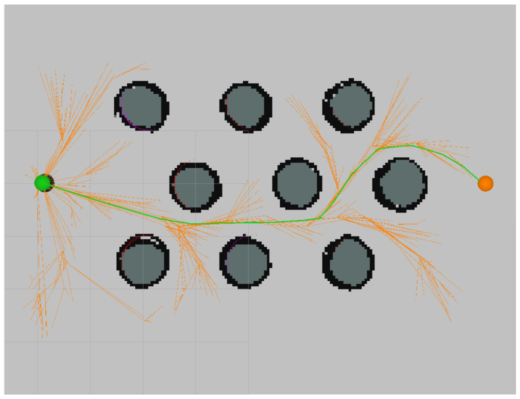
is limited to $0.5s$. Informed-RRT* satisfies this requirement also in a high-dimensional State Space.

B. Robot Setup and Real Experimentation

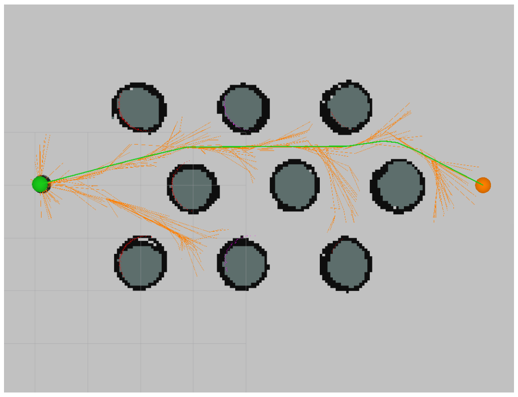
We have tested the algorithm in a real case scenario, with a real robot navigating in a dynamic and crowded environment.

In general the autonomous navigation architecture is a two-stage problem, composed by global planning and local planning. The global planner computes the entire path to reach a given target point. Once the path is computed, the local planner directly controls the robot in order to follow the path. In this paper, we focus on the trajectory planner computation. The local planner is implemented using Enhanced Vector Field Histogram (VFH+), a real-time motion planning algorithm for obstacle avoidance introduced by Borenstein et al. in [21]. This is an improved version of the original Vector Field Histogram described in [22] by the same authors.

Autonomous navigation requires a map, i.e., the Configuration Space, in order to known the environment and support the localization process. Localization is implemented using



(a)



(b)

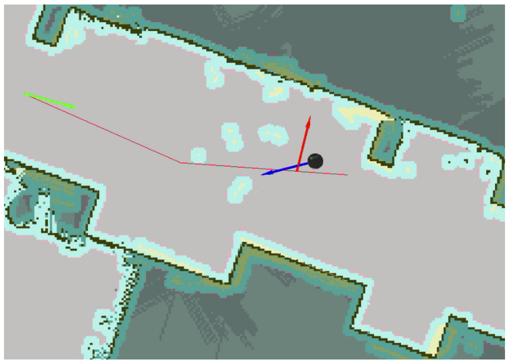
Figure 7. Example similar to Figure 6. The planning problem is the same, but the figures show the solutions after a reduced time of 0.1 seconds. In (a) RRT* finds a solution path that is not optimal. Instead in (b) Informed-RRT* finds a near-optimal solution. This example shows that Informed-RRT* uses efficiently its search space, finding near-optimal solutions in less time.

the Adaptive Monte-Carlo Localization (AMCL), proposed in [23]. Monte-Carlo localization approaches recursively estimate the posterior probability of the robot's pose using particle filters (sample-based implementation of Bayesian filters).

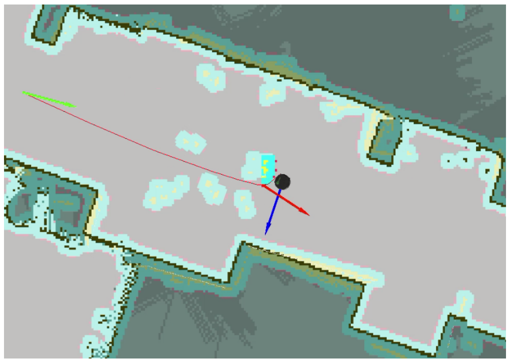
In the real test a Turtlebot 2 robot is used; it includes a differential drive Kobuki base robot, that includes a gyro to improve odometry and a bumper to detect possible collisions. In addition a Hokuyo Laser Range Scanner Sensor (URG-04LX-UG01) is used, that is able to detect obstacles at a maximum distance of $5.5m$.

The experiment was carried in our University corridors. Several parameters are necessary to the algorithm and they have been set finding the best configuration experimentally. The main algorithm loop routine should be able to run at a rate at least $2Hz$. With lower frequency the algorithm may fail to face the moving people, while with this frequency it guarantees always a valid and feasible path.

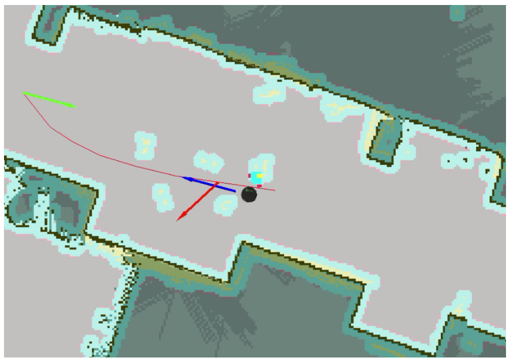
In the experiments shown in Figure 8, we test the algorithm in a mapped environment of about $35m \times 30m$. With the available hardware (CPU Intel CORE i7 with 2-core at 1.9 GHz) the main loop was able to run at a frequency of about $4Hz$. With higher frequency the algorithm may return an



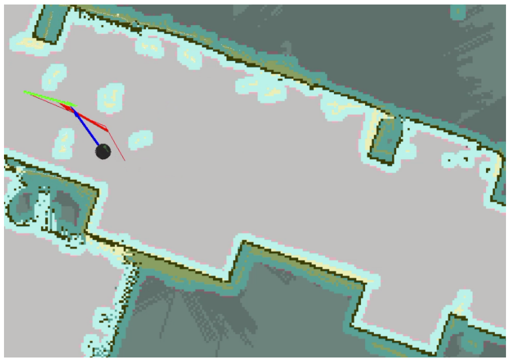
(a)



(b)



(c)



(d)

Figure 8. Navigation result in real environment. Map is known and it is updated by sensors that detect people and new obstacles. Note that laser detects people in about 5 meters. The algorithm continuously checks and improves or computes global path. It searches a gap between obstacles. The robot follows the path and finally reaches the goal point. The red line represents the current path. Green arrow the final pose, blue arrow the robot pose, while red arrow the pose gives to local planner.

incomplete or fragmented path, because there is less time to search for an optimal solution. This frequency has been possible due to Informed-RRT*, which optimizes its first solution in Informed space. In fact algorithms like RRT* search and optimize the solution in all the search space.

In this test the algorithm continuously returns the solution path, or tries to improve the previous one. In this way the robot always has a valid path to navigate in crowded environments. Following the path, the robot can reach the goal pose, avoiding collisions with people or other obstacles.

This approach requires a lower CPU usage, since the path planning algorithm is called fewer times; this feature is important, especially regarding the on-board computer power consumption.

V. Conclusions and Future Works

In this paper we propose a new method to perform path planning computation in crowded and dynamic environments, where dynamic obstacles frequently modify the environment and a new solution is often required.

While traditional algorithms always re-compute a new path, causing an indecision behavior and requiring many more computational resources, the new algorithm proposed is able to continuously evaluate and update the current solution path.

In this work, we demonstrated that a continuous check and repair routine provides a valid path through people. This approach solves the problem of indecision behavior and saves computational resources, since the path is rarely computed from scratch, but is continuously updated while the environment changes.

Another advantage of the proposed approach is its simplicity; in fact, it does not require moving obstacles to be tracked and can be executed with only on-board robot sensors, such as laser range scanners or 3D vision sensors.

Even if people obstruct the robot, the algorithm adjusts the solution path; in this way, the robot always has a valid path to follow and reached the goal position in 100% of the tests, avoiding collision with people and generic obstacles.

This framework could also be used in other applications; for example the same trajectory planning algorithm could be adapted to be used with robotic manipulators acting in a 3D environment.

Future works will include a simple prediction algorithm for dynamic obstacles able to evaluate their motion. In this way a better solution with reduced computational resources is expected, avoiding the computation of paths in those areas where we expect to find future obstacles.

Acknowledgment

This work was done at the Joint Open Lab CRAB and was supported by a fellowship from TIM.

References

[1] S. Rosa, L. O. Russo, and B. Bona, "Towards a ros-based autonomous cloud robotics platform for data center monitoring," in *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, Sept 2014, pp. 1–8.

[2] M. K. Ng, S. Primatesta, L. Giuliano, M. L. Lupetti, L. O. Russo, G. A. Farulla, M. Indaco, S. Rosa, C. Germak, and B. Bona, "A cloud robotics system for telepresence enabling mobility impaired people to enjoy the whole museum experience," in *10th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS), 2015*, April 2015, pp. 1–6.

[3] P. Trautman and A. Krause, "Unfreezing the robot: Navigation in dense, interacting crowds," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2010*, Oct 2010, pp. 797–803.

[4] P. Trautman, J. Ma, R. M. Murray, and A. Krause, "Robot navigation in dense human crowds: the case for cooperation," in *IEEE International Conference on Robotics and Automation (ICRA), 2013*, May 2013, pp. 2153–2160.

[5] P. Henry, C. Vollmer, B. Ferris, and D. Fox, "Learning to navigate through crowded environments," in *IEEE International Conference on Robotics and Automation (ICRA), 2010*, May 2010, pp. 981–986.

[6] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," in *TR 98-11*, Computer Science Dept., Iowa State University, October 1998.

[7] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *Proc. Robotics: Science and Systems (RSS)*, 2010.

[8] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt*," in *IEEE International Conference on Robotics and Automation (ICRA), 2011*, May 2011, pp. 1478–1483.

[9] M. Svenstrup, T. Bak, and H. J. Andersen, "Trajectory planning for robots in dynamic human environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2010*, Oct 2010, pp. 4293–4298.

[10] C. Fulgenzi, C. Tay, A. Spalanzani, and C. Laugier, "Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2008*, Sept 2008, pp. 1056–1062.

[11] S. C. Yun, S. Parasuraman, and V. Ganapathy, "Dynamic path planning algorithm in mobile robot navigation," in *IEEE Symposium on Industrial Electronics and Applications (ISIEA), 2011*, Sept 2011, pp. 364–369.

[12] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with rrts," in *IEEE International Conference on Robotics and Automation (ICRA), 2006*, May 2006, pp. 1243–1248.

[13] E. Yoshida, K. Yokoi, and P. Gergondet, "Online replanning for reactive robot motion: Practical aspects," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2010*, Oct 2010, pp. 5927–5933.

[14] E. Yoshida and F. Kanehiro, "Reactive robot motion using path replanning and deformation," in *IEEE International Conference on Robotics and Automation (ICRA), 2011*, May 2011, pp. 5456–5462.

[15] H. Lin and C. S. Yang, "2d-span resampling of bi-rrt in dynamic path planning," *International Journal of Automation and Smart Technology*, vol. 5, no. 1, 2015.

[16] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2014*, Sept 2014, pp. 2997–3004.

[17] P. Aikaterini and J. Tianjian, "Frequency and velocity of people walking," *The Structural Engineer*, vol. 83, no. 3, 2005.

[18] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.

[19] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[20] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2004*, vol. 3, Sept 2004, pp. 2149–2154 vol.3.

[21] I. Ulrich and J. Borenstein, "Vfh+: reliable obstacle avoidance for fast mobile robots," in *IEEE International Conference on Robotics and Automation, 1998*, vol. 2, May 1998, pp. 1572–1577 vol.2.

[22] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, Jun 1991.

[23] D. Fox, "Kld-sampling: Adaptive particle filters," in *Advances in neural information processing systems*, 2001, pp. 713–720.