Mobile Manipulator in a Hospital Environment - PDM Project

Group 31
Amin Berjaoui Tahmaz - 5610737
Edoardo Panichi - 5630444
Gioele Buriani - 5629888
Giovanni Corvi - 5608899

***Abstract*— This project deals with the development of a motion planner that generates a feasible, low-cost path for a differential drive robot with a 2 degree-of-freedom planar manipulator in a hospital environment. The robot starts from a specified position and will have to reach a randomized goal position by navigating different rooms and avoiding collision with obstacles. To develop a motion planner, a combination of handcrafted RRT and informed-RRT* algorithm is used. Later, a trajectory smoothing algorithm is applied in order to make the path compatible with the differential drive. Lastly, the movement of the robot both in terms of base and manipulator is controlled by a PID controller.**

I. INTRODUCTION

The aim of this project is to simulate a realistic path planning task that could hypothetically be applied in a real environment. Looking into possible applications for a path planning robot, the field of healthcare presents itself as a promising candidate. In particular, a robot that can navigate different rooms of a hospital environment can prove highly beneficial for the overall efficiency of the medical workers.

The chosen robot morphology is a mobile manipulator with a 'roomba-like' differential drive base and a manipulator mounted on top. This setup can reliably complete simple mechanical tasks that do not require extensive knowledge and complex models. Ideally, the robot will be able to pick up medical supplies with the manipulator from narrow compartments and carry them around the hospital for doctors and nurses to use. Moreover, as the supplies can be placed in different rooms, the motion planning will need to adapt and generate paths accordingly.

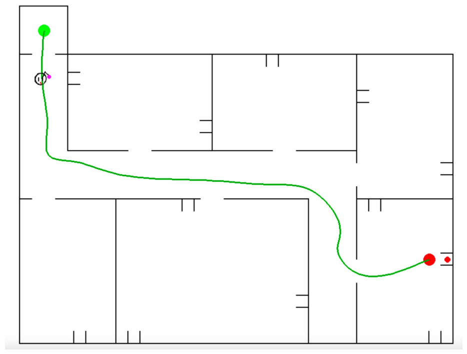

Fig. 1. A possible top view of the scenario.

The picture above shows a simple example of the robot navigating from its recharging station (top left) towards an object inside a package (bottom right). The path is generated using a combination of RRT, informed-RRT* and trajectory smoothing, which will be elaborated on in the subsequent sections.

II. ROBOT MODEL

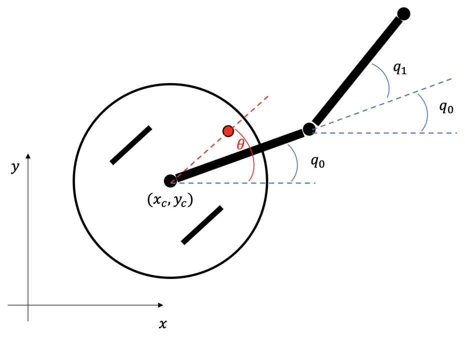

Fig. 2. Robot Model.

The kinematics is fully defined by the geometric expressions of the velocities.
Since the robot is a differential-drive mobile robot, each wheel produces an independent velocity. Under the assumption of pure rolling, it is possible to relate the linear and angular velocities according to the following formulas:

$$v_R = R\omega_R, \qquad v_L = R\omega_L$$

where R is the radius for the right and the left wheel. Notice that ω_R and ω_L will be the control variables for the base of the robot.

Taking into account the non-holonomic nature of the system, it is necessary to impose a non-slipping (zero lateral velocity) constraint. Consequently, $\dot{y} = 0$, expressed in body reference frame.

Using v_R and v_L, the forward and angular velocities of the entire robot, expressed with respect to the body reference system, can be derived (marked as C).

$$\dot{x} = v_c = \frac{v_R + v_L}{2}$$
$$\dot{\theta} = \frac{v_R - v_L}{2d}$$

When putting these equations together, the kinematics, expressed with respect to the *body reference system*, can be written as follows:

$$\begin{bmatrix} \dot{x}_c^B \\ \dot{y}_c^B \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} (v_R + v_L)/2 \\ 0 \\ (v_R - v_L)/2d \end{bmatrix}.$$

Considering that the robot moves in the plane *xy*, then rotates around the *z-axis*, we can exploit a rotation matrix to express the kinematics with reference to the *inertial reference frame*.

$$\begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} (v_R + v_L)/2 \\ 0 \\ (v_R - v_L)/2d \end{bmatrix}.$$

Regarding the acceleration of x_c, y_c and θ:

$$\begin{bmatrix} \ddot{x}_c \\ \ddot{y}_c \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{v}_c \cos\theta - v_c \omega \sin\theta \\ \dot{v}_c \sin\theta + v_c \omega \cos\theta \\ \dot{\omega} \end{bmatrix}.$$

The position of the two *d.o.f.* manipulator end-point can be described starting from the position of the mobile robot using the following equation:

$$\begin{bmatrix} x_e \\ y_e \end{bmatrix} = \begin{bmatrix} x_c + l\cos(q_0 + l\cos(q_0 + q_1 \\ y_c + l\sin(q_0 + l\sin(q_0 + q_1 \end{bmatrix}.$$

where q_0 is the orientation of the first link of the manipulator with respect to the inertial frame, and q_1 is the orientation of the second link with respect to the first link; l is the length of the links; the subscript e stands for *end-point*.

Since the output of the motion planner provides the reference q_0 and q_1 directly, then it is not necessary to use the Jacobian to control the manipulator. The controller can be applied directly on the error measured in the configuration space, i.e. $q_{desired} - q$.

Regarding the workspace, a 2D environment was chosen since the third dimension (in this case the height) does not contribute to solving the problem. Thus, it can be neglected. Consequently, the final workspace is $\mathbb{R}^2$.

Regarding the configuration space, the robot relies on both wheels for motion, and also has a robotic arm. The main body can be represented as a simple differential drive vehicle moving on a plane, and can therefore be described by three coordinates: x and y define the vehicle's center of mass's position on the plane, while θ describes its angular orientation. Furthermore, two additional coordinates q_0 and q_1 are needed to describe the angular position of the robot's manipulator. To sum up, there are five total coordinates: two for the position, and three for orientation. The corresponding configuration space can be represented as $\mathbb{R}^2 \times (S^1)^3$.

III. MOTION PLANNING

For the motion planner, the following steps have been followed in order to enhance the optimality of the algorithm.

First, a simple RRT motion planner was chosen due to tackle this problem. The single query is enough since the software has to compute the path between two points, rather than an entire graph of the environment. Furthermore, combinatorial optimization methods would be too computationally complex for a four-dimensional configuration space, as presented in the case of this mobile manipulator. The algorithm was developed by randomly choosing points on the map and checking whether they were parts of obstacles (walls) or not. If the chosen points are in free space, then the closest neighbor to the point is identified using a simple euclidean distance metric. To check for collisions between these two points, the line connecting them must not intersect the obstacle represented by a polygon. If these checks were positively passed, the initial node would be valid and an edge would be created. Both the node and the edge would then be added to a list, which will later be used to identify the sequence of edges and nodes leading from the start to the end. This sequence will then be defined as the path and saved accordingly. In the end, the algorithm was fast in exploring the whole environment and finding a feasible path, but was lacking in optimality.

For this reason, a more optimal solution was explored in the form of RRT*. The overall functioning was similar to RRT, but included some significant changes. Following the check that the newly sampled node was not part of an obstacle, RRT would directly identify the closest neighbor was and start the collision checking process. However, RRT* initially considers a circle around the new node. The new node takes into consideration all the nodes present in this circle, and uses Dijkstra to check for the node to connect to which provides the shortest path to the start. Following this step, all the other nodes within the circle go through the same process and attempt to connect with a node that would offer a shorter path compared to one currently achieved. Naturally, all the connections go through a collision checking procedure, just like RRT. Note that this procedure shortens the length of many possible paths, which leads to a decrease in the need for further approaching the optimal solution. For this reason, the radius of the neighboring circle mentioned above decreases dynamically with the number of iterations, in order to avoid unnecessarily slowing down the algorithm. For the final path definition, the algorithm functions the same as RRT. Overall, this algorithm was significantly slower with respect to RRT, but allowed the motion planner to achieve more optimal paths.

In order to reduce computation time while preserving the optimality of RRT*, informed-RRT* is introduced [1]. This algorithm functions the same as RRT*, with one exception: After finding a goal for the first time, the random points will be sampled within an ellipse containing the start and goal node, instead of sampling points throughout the entire map. In turn, this shifted the algorithm's attention away from the redundant points and more towards the area of interest.

The final solution that is implemented using a combination of these approaches: at first an RRT runs on the whole map in order to find the goal as soon as possible. As soon as the goal is found, the algorithm switches into an informed-RRT* in order to shorten the path as much as possible getting closer to the optimal solution. Depending on the level of optimization wanted, a limit on the number of random nodes used for optimization was set. Logically, the closer the path to the optimal one, the more time on average it would require to reach that result.

After planning a trajectory, the path needs to be smoothed to accommodate the non-holonomic motion of the differential drive, while simultaneously avoiding collision due to the joint angle configurations in the arm. The trajectory smoothing algorithm used in this project was inspired by the smoothing algorithm proposed by Noreen et al. [2]. The algorithm handles the desired path as an unconstrained optimization problem and uses B-Splines to connect the nodes of the 4-D configuration path.

In the case of the differential-drive, we only use the 2-D position coordinates to generate a smooth path. After generating a preliminary smooth trajectory, the algorithm loops over the points to check for collisions in the configuration space. If a collision is detected, then the algorithm will insert two midpoints: one midpoint between the two nodes containing the colliding spline (P_{i-1} and P_i), and another midpoint between the two nodes of the following spline (P_i and P_{i+1}). The collision evaluation with midpoint insertion is repeated iteratively until the path becomes collision-free.

Since the manipulator is mounted on the differential-drive, the location of the manipulator base needs to be taken into consideration. Thus, the smooth trajectory of the differential-drive robot is used as the position of the base. Then, the smooth trajectories going through the informed-RRT* configurations are connected by linearly sampling points between the successive joint angle configurations.

IV. RESULTS

As a simulation environment, Pygame [3] (which runs on Python) was chosen due to its simplicity and versatility.

Concerning the scenarios, the adopted procedure involved first picking a relatively complex scenario to design the planner. Later, as the planner becomes more robust, a randomization of the goal position was be performed by changing the end goal room, as well as the positions within them.

The overall functioning scheme for the motion planner is the following:

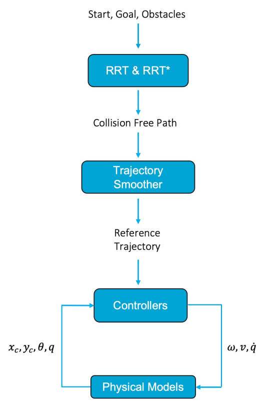

Fig. 3. Functioning scheme of the motion planner

Everything starts with the knowledge of the position of the start and goal points and, more importantly, the position of the obstacles (namely the walls of the hospital). With that knowledge, the RRT and informed-RRT* algorithm can be applied to find a collision-free path close to optimal. When the path is defined, the trajectory smoothing algorithm is run in order to make the path feasible for the non-holonomic base.

When the final trajectory is defined, the control loop starts. There are three controllers: a PID to control the angular velocity of the robot's base, a simple proportional (P) regulator is applied to the linear velocity and another PID controller to guide the manipulator.

The randomization of the goal obviously influences the duration of the algorithm since further away goals are, on average, more difficult to reach they should be and therefore require the algorithm to run longer. Depending on the obstacles' positions, however, the correlation can be challenged. This idea can be summed up in the following graph.

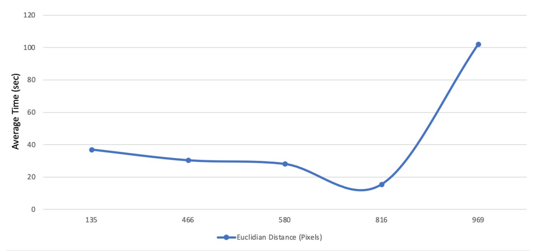

Fig. 4. Correlation between euclidean distance from start and time taken to reach each goal

Obviously, as can be verified from the graph, the correlation isn't linear since the presence of obstacles and walls in between sometimes makes the required optimal path much longer than the euclidean distance. Overall, however, it is expected that a further away requires more time to be found and reached compared to a closer one. The algorithm implemented in this project is able to cope with both closer and further obstacles without requiring excessive amounts of time.

The computation time is also strictly correlated with the level of optimization required. As mentioned before, the algorithm initially works as an RRT until the goal is found. Due to the randomness of the algorithm this can take very variable amounts of time, but overall 93% of the times finds a goal with maximum 1000 nodes. After that, the optimization part begins with the algorithm turning into an informed-RRT*.

The optimization part can be tuned by setting the number of nodes used for the procedure. The more these nodes are, the more optimal the final path is expected to be, but on the other hand the longer the computation time becomes. The following graph illustrates different levels of optimization against the number of nodes required for the process.

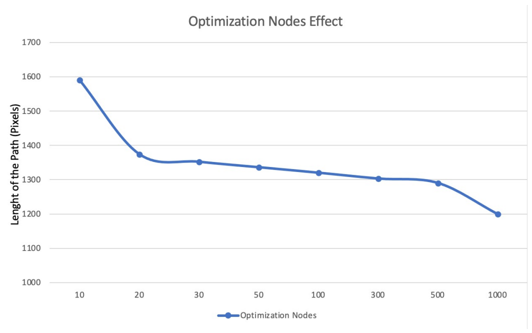

Fig. 5. Correlation between number of optimization nodes used and length of the final path (keeping a constant goal)

As can be seen from the graph, after around 20 optimization nodes the length of the final path steadily decreases with the increasing number of optimization nodes. This result was expected and can be used to tune the trade-off between computation time and optimization required. For a situation where time is crucial, a lower number of optimization nodes might be enough to get a decent trajectory. If instead time is not an issue and a solution close to optimal is required (maybe in terms of battery saving, for example) then the number of optimization nodes can be kept high in order to find the best possible path.

V. DISCUSSION

In the end, the final algorithm takes advantage of the benefits of both RRT and informed-RRT*.

In particular, the solution is complete since it does reach a solution, if any is present, in a finite amount of time. In particular, it is also probabilistic complete since the probability to find the solution increases with time approaching 1.

Regarding optimality, instead, RRT is not optimal since it only focuses on finding a feasible solution, rather than an optimal one. Informed-RRT*, instead, is a form of RRT* and is therefore asymptotically optimal. The cost of the path on average decreases with the computation time dedicated to the optimization.

As for the complexity, the space and time complexity for RRT* are given by $O(n)$ and $O(n\log(n))$, respectively. Concerning informed-RRT*, instead, mathematical analysis shows that the space and time complexity is the same as that of RRT* but the value of n is significantly reduced [4]. Overall, the time complexity of our algorithm should be much lower than an informed-RRT* with the same number of nodes since the first nodes are all laid out with an RRT algorithm that has a much lower time complexity with respect to RRT*.

From a more qualitative point of view, it can be said that the algorithm performed overall in a satisfying manner, whereby it consistently reaches the goal within a limited amount of time and with low path costs.

The arm planning, however, could benefit some improvements since it moves in a seemingly random or redundant manner when moving through configurations. For example, it will rotate in free space instead of staying still. Another approach can be to keep the arm folded on the robot and extend it just when it is time to reach inside the box, however, this approach would not be didactically particularly interesting. Therefore we opted for this ”more free” approach that can be modified just by imputing some requested arm configurations throughout the journey (e.g. move something around, hold something, ...).

Also, the planning algorithm itself has some limitations. First of all, it needs a map of the environment to plan a path and avoid obstacles. Secondly, the computation is made offline before starting the journey and therefore cannot cope with dynamic obstacles. These two limitations prevent this algorithm to work in a busy environment not previously known, but if the conditions are the ones expressed for this project, the motion planner would be theoretically able to reach any reachable point in the map.

One last notice can be done about the controller for the motion of the robot. In this case, the necessary movements were rather simple and therefore a PID controller for the wheels and one for the robotic arm was enough to deal with the problem. In more complex situations or if more movement precision and optimization are required, an MPC controller should definitely be more indicated.

Overall, though the performance is more than satisfying, with some margins of improvements for more specific and complex scenarios.

REFERENCES

[1] 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 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2014.
[2] I. Noreen, “Collision free smooth path for mobile robots in cluttered environment using an economical clamped cubic B-Spline,” Symmetry (Basel), vol. 12, no. 9, p. 1567, 2020.
[3] Pete Shinners, Pygame - Python Game Development, version 2.0.1. Available at https://www.pygame.org
[4] J. Nasir et al., “RRT*-SMART: A rapid convergence implementation of RRT,” Int. J. Adv. Robot. Syst., vol. 10, no. 7, p. 299, 2013.