Abstract

In this project, we propose a motion planning framework based on ROS for an autonomous vehicle navigating through dynamic environments especially parking lot with dynamic obstacles. We modified and implemented several planning algorithms as the global planner, including $A^\star$, hybrid $A^\star$, and $D^\star$ lite, and Model Predictive Control (MPC) as the local planner. Combined with perception, the motion planning framework is tested in Gazebo simulation and the robot is capable of navigating through dynamic parking plot without collision. Our framework can also be integrated in the self-driving vehicle as one of the pipelines to fulfill planning task.

1 Introduction

With the increasing population and ever-growing urban centers over the world, the usage of private vehicles is fast-growing, especially in the urban areas, which makes car parking a problem. In the meantime, as a result of increasing living standards, a lot of the parking lots are designed to be fancy with complex interior layouts, which make the problem even worse. Although automatic parking is a common feature in a modern vehicle, it is not intelligent enough to move the vehicle from the entrance of a parking lot to the desired parking space. Thus, it is important and necessary to integrate a self-parking system that can automatically move the vehicle to the desired parking space once the vehicle enters the parking lot.

In autonomous driving, navigation and planning are generally acknowledged to be the most important techniques. Hence, in order to build an autonomous parking system, it is critical to come up with a planning system that is robust and efficient based on a parking lot environment.

There are several difficulties of doing planning for a vehicle in a parking lot environment, the primary one is the complex road conditions. Since there are usually many cars in a parking lot, the vehicle needs to expect interaction with other cars, which means the planning algorithm must treat those cars as dynamic obstacles and avoid them along the path. Another difficulty is the dynamic limitation of the vehicle, which means the steering radius of the vehicle model needs to be primarily considered. In this project, we propose a planning framework that can overcome these challenges and navigate the robot to the desired position without collision.

Section 2 describes the overall structure of the navigation stack. Section 3 explains how sensor inputs are processed, how the global and local costmap are updated, and how dynamic obstacles are controlled. The global planner is explained in section 4, and the local planner is introduced in section 5. The results are shown and discussed in section 6, and a brief summary is given in the final section.

2 Navigation Stack

On a conceptual level, the Navigation Stack is fairly straightforward. It receives data from sensor streams and generates velocity commands to send to a mobile base. However, when it comes to the finer points, things become a little more complicated because how each part works and how they coordinate with one another is critical for proper functionality.

We divide the navigation stack into 4 parts: sensors, costmaps, planners, and robot base. The following is the main procedure for a robot navigating to a target position. In the initialization step, the global map is loaded and the robot is located in the global map. When the goal position is set, the global planner takes over and plans a feasible and collision-free path based on an updated global map combined with the local map generated by sensors. Then the local planner optimizes a trajectory and generates control commands based on the global path. The control commands are then sent to the robot base, where they will be executed by some basic controllers. The structure of the navigation stack is shown in Figure 1.

In this project, we select basic $A^\star$ because of its optimality and completeness. Since the planner is required to quickly replan the path in an environment with dynamic obstacles, we choose $D^\star$ Lite. Considering the robot minimal steering radius and robot kinematic model, hybrid $A^\star$ is selected and Model Predictive Control(MPC) is chosen as the local planner.

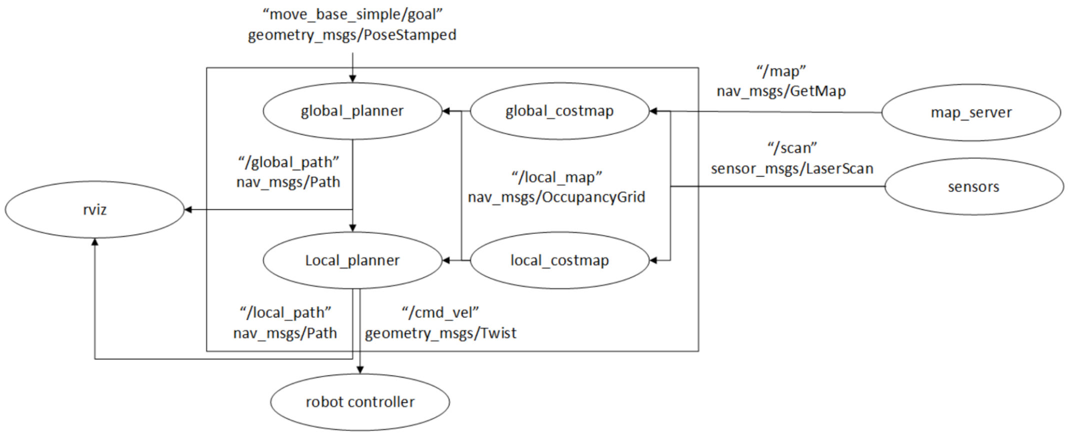


Figure 1: Structure of navigation stack.

3 Sensor Information

3.1 Costmaps and Laser

The global costmap comprises all of the static objects that cannot move in the environment. For example, the position of walls are contained in the global costmap but the cars are not because they can move. The local costmap is generated from a 2D laser scanner attached to the robot. Both of the costmaps are dilated to avoid collision. Every time the robot receives the local costmap, the global costmap will be updated. First the local costmap will be transformed from robot coordinate to the map coordiante. Then for each pixel of the local costmap, check if its value is different from the original global costmap. If they are different, then check the value of this pixel in the origin global costmap, if it is smaller than the obstacle threshold, then this pixel is updated in the global map.

Algorithm 1 Global costmap update

```
Input: original global costmap $GM$, current globla costmap $GM_t$.
Output: updated current global map $GM_t$
  Dilate local costmap $LM_t$.
  for i in range($local\_costmap\_size\_x$) do
      for j in range($local\_costmao\_size\_y$) do
          (i', j') ← transform (i, j) from robot coordinate to map coordinate.
          if $LM_t(i,j)! = GM(i',j')$ and $GM(i',j') < obstacle\_threshold$ then
              $GM_t(i',j') = LM_t(i,j)$
          end if
      end for
  end for
```

3.2 Dynamic obstacles

A simple strategy is used to control the dynamic obstacles. For each dynamic obstacle, we set a series of way points and interpolate between each two consecutive way points to generate the trajectory. The user can also set the speed of the dynamic obstacles. When an obstacle approaches the robot at a high speed, the robot may not be able to avoid it due to its limited speed and acceleration. In this case, the controller checks the distance between the obstacle and the robot; if it is less than a certain threshold, the obstacle will stop; if the distance is greater than the threshold, the obstacle will resume movement.

4 Global Planner

4.1 Basic $A^\star$

$A^\star$ is a typical global planning algorithm which leverages a heuristic function to guide the direction of the searching. A priority queue is maintained as the OpenList to store the nodes to be expanded. The state of the robot is represented by its position and orientation:

$$state \quad \mathbf{x} = [x, y, \theta]^T \tag{1}$$

By expanding the node from the top of the OpenList, the $A^\star$ planner continually explores the potential path to the goal position. The OpenList automatically sorts the unexpanded node by the ascending order of value function:

$$f(\mathbf{x}) = g(\mathbf{x}) + h(\mathbf{x}) \tag{2}$$

where $g(\mathbf{x})$ represents the cost from the start state to the current state, $h(\mathbf{s})$ represents the heuristic from the current state to the goal state. In this project the cost function and admissible heuristic are:

$$\begin{aligned}
&cost(\mathbf{x}, \mathbf{x}') = euclideanDist(\mathbf{x}, \mathbf{x}') + headingDist(\mathbf{x}, \mathbf{x}') \\
&euclideanDist(\mathbf{x}, \mathbf{x}') = \sqrt{(x - x')^2 + (y - y')^2} \\
&headingDist(\mathbf{x}, \mathbf{x}') = min(|\theta - \theta'|, 2\pi - |\theta - \theta'|) \\
&h(\mathbf{x}) = euclideanDist(\mathbf{x}, \mathbf{x}_{goal}) + headingDist(\mathbf{x}, \mathbf{x}_{goal})
\end{aligned} \tag{3}$$

where $\mathbf{x}'$ is the successor of $\mathbf{x}$.

4.2 Hybrid $A^\star$

Although $A^\star$ planner can find a path rapidly with the guidance of the heuristic function, it may not fit the real scenario due to the ignorance of the dynamic limitation on the vehicle. To solve this problem, Hybrid $A^\star$ combines the sampling method with the basic $A^\star$ algorithm to find the feasible path subjective to the steer radius of the vehicle. While the Hybrid $A^\star$ planner expanding a node, it just add the dynamically feasible successors to the OpenList instead of all of the neighbour successors of the current node. These feasible successors which are also known as the motion primitives, speciafically Reeds-Shepp Curves, are generated in the initialization step in the robot coordinate. We set some hyper-parameters *steer_radius*, *sample_num*, and *yaw_interval* to control the robot minimal steering radius, number of motion primitives, and the resolution of the robot heading respectively. During the planning process, the motion primitives are transformed from the robot coordinate to the map coordinate before performing the basic $A^\star$ search.

In this way, the Hybrid $A^\star$ planner can be applied to the planning task for vehicles that have different dynamic model besides the differential drive model. However, since the searching space of the Hybrid $A^\star$ planner has much higher dimensions than that of the basic $A^\star$ planner, it will take much more time for it to search towards the goal position.

4.3 $D^\star$ Lite

Since both of the basic $A^\star$ and the hybrid $A^\star$ search the path from the current robot state to the goal state repeatedly after every movement of the robot, the planner may compute through certain area in the global map many times, which is time-consuming and not necessary. Even though the g-value of some states in the global map changes from time to time as the robot moving and updating the global costmap, the g-value of most of the states stay unchanged. $D^\star$ Lite provides a suitable method to reuse the previous computed nodes throughout every replanning step until the robot reaches the goal position. Similarly with basic $A^\star$, $D^\star$ Lite also mantain a OpenList to store the nodes. Differently, the OpenList of the $D^\star$ Lite sorts the nodes by the ascending order of the key value:

$$\mathbf{k} = [(min(g(\mathbf{x}), rhs(\mathbf{x})) + h(\mathbf{x}_{start}, \mathbf{x})); min(g(\mathbf{x}), rhs(\mathbf{x}))] \tag{4}$$

where $g(\mathbf{x})$ represents the cost from the current state to the goal state, $rhs(\mathbf{x})$ represents the one step ahead value of $g(\mathbf{x})$, $h(\mathbf{x}_{start}, \mathbf{x})$ represents the heuristic from the current state to the start state.

At the beginning of planning, the $D^\star$ Lite planner first search backward from the goal position to the start position until it finds a path. Later, as the global costmap being updated, the planner updates the values of the nodes that change from traversable to not traversable or vise versa dut to dynamic obstacles. During replanning, the $D^\star$ Lite planner will start from the changed nodes and then the nodes around them. By recomputing the $g(\mathbf{x})$ and $rhs(\mathbf{x})$ of the effected nodes, those inconsistent nodes whose $g(\mathbf{x})$ is not equal to $rhs(\mathbf{x})$ will be added to the OpenList. Finally, the planner reconstructs the searching space and returns a new path to the local planner.

5 MPC Local Planner

When the global planner find a feasible path to the goal, the control commands are required so that the robot can navigate to the target. Assume that the global path $\mathbf{X}^g = \{\mathbf{x}^g_0, \mathbf{x}^g_1, \ldots, \mathbf{x}^g_M\}$ generated by global planner is obtained. Three hypter-parameters are required to compute the reference path $\mathbf{X}' = \{\mathbf{x}'_0, \mathbf{x}'_1, \ldots, \mathbf{x}'_N\}$: simulation time sim_tim, time interval dt, and horozon range $horizon\_range$. The reference path is computed by $\mathbf{x}'_i = \mathbf{x}^g_{min(i \times horizon\_range, M)}$ where $i = 0, 1, \ldots, N$ and $N = sim\_time/dt$. The local obstacles $\mathbf{X}^{obs} = \{\mathbf{x}^{obs}_0, \mathbf{x}^{obs}_1, \ldots, \mathbf{x}^{obs}_M\}$, where $\mathbf{x}^{obs} = [x, y]^T$, is obtained by the following process. For each $\mathbf{x}'_i$, $\mathbf{x}^{obs}_i$ is the position of the nearest obstacle if the distance between $\mathbf{x}'_i$ and the nearest obstacle is smaller than min_dist, which is another hyper-parameter.

The optimized local path $\mathbf{X} = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and the control commands $\mathbf{U} = \{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{N-1}\}$, where $\mathbf{u} = [v, w]^T$ and v is the linear velocity and w is the angular velocity, can be calculated by solving this cost function:

$$\begin{aligned}
\min_{U,S} \; & \sum_{t=0}^{N} (\mathbf{x}_t - \mathbf{x}'_t)^T \mathbf{R} (\mathbf{x}_t - \mathbf{x}'_t) + \sum_{i=0}^{N-1} \mathbf{u}_i^T \mathbf{Q} \mathbf{u}_i \\
\text{s.t. } & \mathbf{x}_{i+1} = \mathbf{f}(\mathbf{x}_i, \mathbf{u}_i) \\
& min\_vel \leq v \leq max\_vel \\
& min\_rot\_vel \leq w \leq max\_rot\_vel \\
& min\_vel\_acc \leq a_v \leq max\_vel\_acc \\
& min\_rot\_acc \leq a_w \leq max\_rot\_acc \\
& euclideanDist(\mathbf{X}, \mathbf{X}^{obs}) \geq min\_dist
\end{aligned} \tag{5}$$

Where $\mathbf{R}$ and $\mathbf{Q}$ are weight matrices and $\mathbf{f}$ is the robot dynamic model, which we use the differential drive model in this project. The local path is desired to be as close to the reference path as possible while still keeping a safe distance from the obstacles. It is also feasible when the robot's dynamic model, velocity, and acceleration limits are taken into account. The second term in the cost function penalizes energy consumption. It may not be required in simulation, but it is critical in practice.

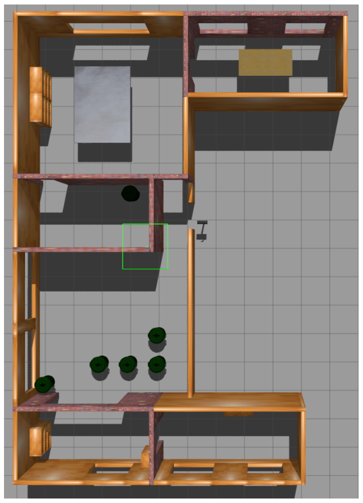
6 Experiments and Results

The navigation stack is tested in both indoor and outdoor environments, which is shown in Figure 2. We use gmapping ros package to build the 2D grid map. The map size is 384×384 and 608×608 respectively with resolution of 0.05.

We tested all the global planners in both environments. The video is available here. An example of the generated path is shown in Figure 3. The paths generated by basic $A^\star$ and $D^\star$ Lite are nearly identical, with a few minor differences, while the path generated by hybrid $A^\star$ has more curves because the robot motion primitives are considered.

During the experiment, basic $A^\star$ and $D^\star$ Lite planner and the MPC local planner can run at a rate of 20 Hz. However, the hybrid $A^\star$ planner cannot run in real-time because the search space is much larger considering the robot heading and motion primitives.

Although the robot was able to navigate to the goal in the experiments, our proposed methods still have a lot of room for improvement. One enhancement is to speed up the hybrid A^{star} planner. Another possible enhancement is to replace the differential drive model with a more advanced model that can represent the dynamic model of the actual car, such as the bicycle model or the Ackerman steering model.


(a) Indoor(House)

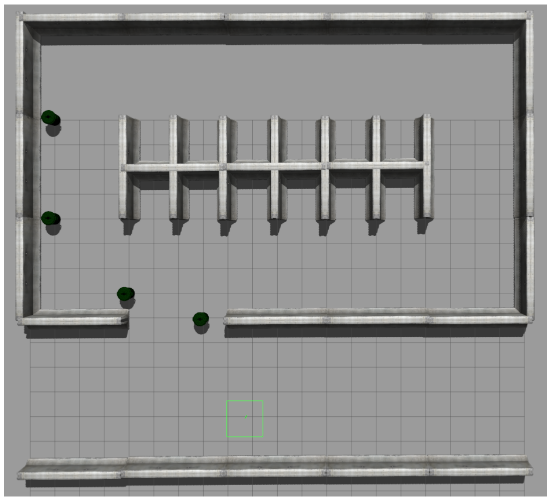
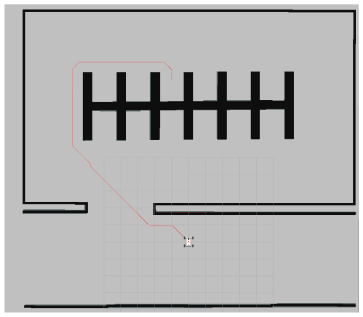
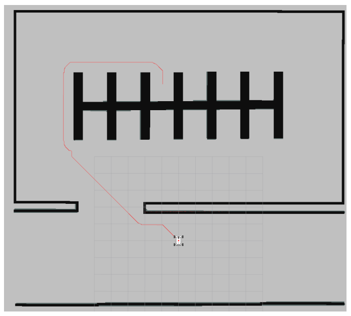
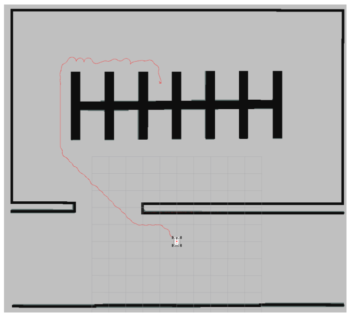
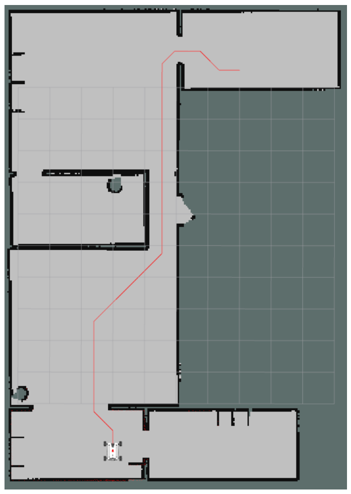

(b) Outdoor(Parking lot)

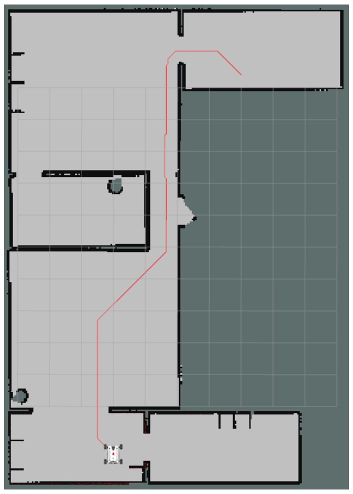
Figure 2: Environments.


(a) Basic $A^\star$ parking lot


(b) $D^\star$ Lite parking lot


(c) Hybrid $A^\star$ parking lot


(d) Basic $A^\star$ house


(e) $D^\star$ Lite house

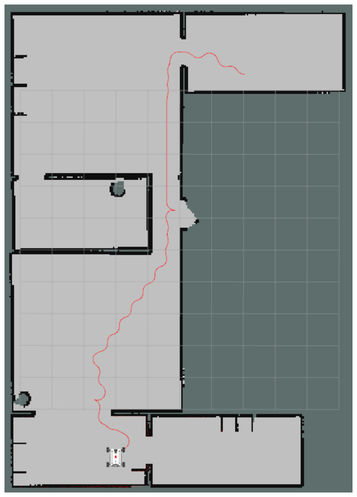

(f) Hybrid $A^\star$ house

Figure 3: Planned paths

7 Conclusion

In conclusion, the problem of navigation in a dynamic environment has been largely solved. Our proposed self-driving system can navigate the robot through a dynamic environment with a few obstacles. When dynamic obstacles obstruct the previously generated path, the system can quickly replan to avoid collision. When there is no viable path to the goal, the robot can come to a halt and wait for the dynamic obstacles to move. The robot resumes movement once a viable path is found.