Mini Project

Steve Gillet
Student ID: 111232341

Course: Algorithmic Motion Planning – ASEN 5254-001 – Fall 2024
Professor: Morteza Lahijanian
Teaching Assistant: Yusif Razzaq

1 Scenario

A company has generously donated a small robot to CU Boulder's CAES to be used for delivering mail to faculty offices during non-business hours when the building is closed. I have been selected to lead the motion planning team for this robot. My team's task is to design a motion planning algorithm under the following assumptions:

- Full maps of the building floors are available.
- The indoor GPS system and local sensors are fully accurate, with no sensor uncertainty.

The robot is in the shape of a rectangle with length $L = 2$ and width $W = 1$ with second-order car dynamics:

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = \frac{v}{L}\tan\varphi, \quad \dot{v} = u_1, \quad \dot{\varphi} = u_2,$$

where x and y determine the position, θ is the orientation, v is the speed, and φ is the steering angle of the front wheels of the car. The control inputs u_1 and u_2 are the acceleration and steering turn rate, respectively. The physical constraints of the car are:

$$\theta \in [-\pi, \pi], \quad v \in \left[-\frac{1}{6}, \frac{1}{2}\right], \quad \varphi \in \left[-\frac{\pi}{6}, \frac{\pi}{6}\right], \quad u_1 \in \left[-\frac{1}{6}, \frac{1}{6}\right], \quad u_2 \in \left[-\frac{\pi}{6}, \frac{\pi}{6}\right].$$

The robot experiences bounded motion disturbances that are not explicitly modeled in the dynamics. To mitigate this, it is equipped with a closed-loop controller that ensures the robot can follow a nominal trajectory while maintaining its state within a ball of radius r around the trajectory.

Given an initial condition for the car and a destination set defined by $x_d \in [x_d^-, x_d^+]$, $y_d \in [y_d^-, y_d^+]$, $\theta_d \in [\theta_d^-, \theta_d^+]$, and $v_d \in [v_d^-, v_d^+]$, the motion planning algorithm must find a valid trajectory that:

- Respects all physical and control constraints,
- Ensures that, by following the trajectory using the provided controller, the robot can move from the initial state to a state within the destination set, and
- Avoids collisions with obstacles in the environment.

2 Motion Planning Space

I will use the state space to motion plan in. The state space will consist of the x and y position coordinates, velocity, orientation, and the steering angle of the robot. The x and y dimensions will be bounded by the dimensions of the maps of the building floors, the velocity will be bounded between -1/6 and 1/2, the orientation betweeen negative pi and pi, and the steering angle between negative pi/6 and pi/6. The obstacles will be mapped out in the x and y dimensions and I will add the radius r (defined in the scenario) to them in order to help compensate for the robots motion disturbances. Since the robot has kynodynamical constraints I have to motion plan in these contrained states in order to take the limitations into account and in order to simplify the motion planning problem (I can treat the robot as a point robot in this 5-dimensional space). Which essentially means I can offload some of the contraint checking to the bounds of the state space instead of having to plan in the 2-dimensional position space and constantly check that all of the states are in bounds.

3 Advantages and Disadvantages of Various Motion Planning Methods

There are many methods of motion planning to consider, different motion plan methods work best in different scenarios. I will consider 4 different methods and weight their pros and cons for this particular application in order to decide which will work best. The 4 under consideration are:

- Gradient Descent Planner with a Potential Function
- Wave-Front Planner
- Probabilistic Roadmap Planner
- Randomized Tree-Based Planner

3.1 Gradient Descent With Potential Function

Gradient Descent with a potential function is a fun (personal opinion) and optimal method for motion planning. Optimal in the sense that it can give you an optimal motion plan although it is not guaranteed. The difficulty with this method is that it generally suffers from the local minima problem (it gets stuck easily) and that I would have to carefully design the planner and its parameters. I would have to design potential functions for the various states and make sure they work and make sense. And I would have to tweak the parameters such as the strength of the potential functions to make sure that the motion plan works consistently for the particular environment. The distances and the number/size of the obstacles can effect how well the potential functions work, for example if the attractive potential function for the goal state is not strong enough and the repulsive functions for the obstacles are too strong then the planner might get stuck or not find a plan adequately.

3.2 Wave-Front Planner

This planner is actually the optimal planner in a fully discretized space. If the space is discretized finely enough then a motion plan (if one possibly exists) is guaranteed and it is also guaranteed to be optimal. The primary drawback and the reason it isn't used all the time is because discretizing a large, multi-dimensional space is computationally costly and often impossible. However, I think in this case we only have 5 dimensions and I think it could be a solid and plausible option depending on how big the building is and how finely we need to discretize it in order to get a good plan and avoid obstacles. I will lead with this planner until experiments prove it impractical since I like the opportunity this gives us for optimality. Plus we have big computers, we should be fine.

3.3 Probabilistic Roadmap Planner

PRM is actually also another very good choice for this scenario because it is computationally efficient and since it creates a roadmap of the space we can keep coming back and reusing the same roadmap and just changing the start and goal state. Since the robot is presumably going be delivering mail from and to different points in the building all the time it might be useful to be able to use the same roadmap. If we spent a lot of time refining the sampling we could get a really good roadmap too. Since it is sampling the states instead of discretizing them all it is much more efficient and quicker than the previous planners. However, it is not complete in the sense that it is not guaranteed to find a plan even if one exists. I don't think that risk is very big in this situation but it is something to consider. I think this is a solid second choice if Wave-Front fails.

3.4 Randomized Tree-Based Planner

A randomized tree-based planner like RRT is the gold standard for motion planning because it is fast, efficient, and probabilistically complete. It has the advantages of PRM except that it does not create a roadmap for the whole space, it just immediately cuts to trying to get from start to goal. This is a great planner overall and a great third place choice, but I think the other two are actually going to be better for this particular scenario.

4 Motion Planner

Using the Wave-Front planner method the pseudocode will look like this:

```
Discretize state space
Set value of obstacle cells to 1 and goal cell to 2
Set the rest of the cells to 0
$i \leftarrow 2$
while Start cell not reached do
    Set value of cells adjacent to i to i + 1
    $i \leftarrow i + 1$
end while
while $i \neq 2$ do
    $i \leftarrow i - 1$
    Sample random, valid control that gets you to cell i
end while
```

The algorithm starts by discretizing the state space into cells of valid state spaces. Then it gives values to all of the cells starting with 1 for the obstacles and 2 for the goal state. Then starting from the goal cell it assigns increasing incremental values to the adjacent cells until it reaches the start cell. Then when it reaches the start cell it begins sampling random controls that fall within the control bounds until it gets a valid control that takes it into a cell with the decrementing values from the start until it reaches goal.

5 Completeness

As long as I can discretize my state space finely enough the wavefront planner theory says that I am guaranteed completeness, I will find a valid path if one exists.

6 Optimality

The wavefront planner theory also states that I am guaranteed optimality though the random control method of propogating through my planner might alter that slightly, it still seems plausible that whatever I end up with will be very close to optimal. Optimal minus random control error. It makes sense that it would be near optimal since we are essentially discretizing and searching the entire space uniformly we will find the shortest path in our discretization. I can imagine that there would be a way to choose the controls more specifically so that it would optimally get you to the next state cell, I'm not sure how that would be done but if I wanted to ensure optimality I would look there.

7 Task Planning

The scenario is expanded in this way:

The robot is required to return to its charging station at the end of each day. However, due to heavy rain and roof leakage, puddles have formed on the building floor. While the robot can traverse through puddles without difficulty, it must dry off on a carpeted area before proceeding to the charging station.

I will address this expanded requirement using Linear Temporal Logic. I will use the letter p to represent the atomic proposition, 'visit puddle' representing the robot going through an area with a puddle. ca will represent 'visit carpeted area' and cs will represent 'visit charging station'. The LTL formula will look like:

F cs $\wedge G(p \rightarrow \text{ca})$

This means eventually go to the charging station and if you ever go to a puddle area you have to then go to a carpeted area. This logic applies to the basic case of taking one day at a time in any given day you must eventually go to the charger and you must always dry off. It also makes sense to do this:

GF cs $\wedge G(p \rightarrow \text{ca})$

Globally, finally go to the charging station if you are thinking about it in terms of over the span of all time it will always have to go to the charger in the future. Also if we wanted to take picking up and dropping off mail I would probably do something like pum for 'pick up mail' and d for 'drop off mail' and the resulting LTL formula would look like:

$GF(pum \rightarrow d) \wedge GFcs \wedge G(p \rightarrow ca)$

Globally, finally pick up mail then drop it off and globally, finally go to your charging station and globally go to a carpeted area if you go to a puddle.

8 Motion Planning With Expanded Requirements

In order to generate a motion plan while taking into account the new scenario with puddles and chargers I think I would break it up into multiple motion plans using the same basic motion plan before at the low level to take me from location to location and a higher-order, task-planning logic to deal with the overall plan. So that high order logic would be to pick up the mail at the beginning of the day, drop it off where it needs to go, if you go through a puddle, go to a carpeted area, then when everything is dropped off go to the charging station. In order to do this I would first make a motion plan from the charging station to the mail pick up point and if the plan takes me through any puddles then plan a motion plan from the puddle to the carpeted area and then carpeted area to mail pick up point and continue like that until the mail pick up point is reached. Then do the same from the mail pick up point to the first destination, then first to second, etc until the last drop off point is reached then do a motion plan from last drop off point to charging station.

So essentially the same pseudocode except that there will have to be a way to handle puddles:

```
1: Discretize state space
2: Set value of obstacle cells to 1 and goal cell to 2
3: Set the rest of the cells to 0
4: $i \leftarrow 2$
5: **while** Start cell not reached **do**
6:    Set value of cells adjacent to i to i + 1
7:    $i \leftarrow i + 1$
8: **end while**
9: **while** $i \neq 2$ **do**
10:    $i \leftarrow i - 1$
11:    Sample random, valid control that gets you to cell i
12:    **if** cell i is a puddle **then**
13:       Break out of code and restart with puddle cell as start cell
14:       Set carpeted areas as goal cells
15:       Inform higher-level logic to plan from carpeted cell to original goal cell
16:    **end if**
17: **end while**
```

Here is what the pseudocode for the high-level logic will look like:

```
pickedUpMail ← **false**
charged ← **false**
while true do
  if not charged then
    waveFrontPlanner(currentLocation, chargingStation)
    while puddleHit do
      waveFrontPlanner(currentLocation, chargingStation)
    end while
    charged ← true
  end if
  if not pickedUpMail then
    waveFrontPlanner(currentLocation, mailPickUpPoint)
    while puddleHit do
      waveFrontPlanner(currentLocation, mailPickUpPoint)
    end while
    pickedUpMail ← true
  else
    for each destination in destinations do
      waveFrontPlanner(currentLocation, destination)
      while puddleHit do
        waveFrontPlanner(currentLocation, destination)
      end while
      dropOffMail()
    end for
    charged ← false
    pickedUpMail ← false
  end if
end while
```

9 Completeness of Modified Planner

Because the lower-level planners are all guaranteed to be complete with respect to the grid I think it is up to the higher-level logic and the discretization as to whether the whole thing is complete or not. In my mind the high-level logic almost has to be complete. Always deliver the mail or always return to charging station kind of defines the requirements of completeness. For example, in order for the motion plan to be complete it has to generate a plan from the charging station to the mail pick up point. The higher-level logic dictates what the success criteria is (going from the charging station to the mail pick up point). So as long as the high-level logic includes all of the requirements then the whole plan will be complete with respect to the grid.

Depending on how fine we have to discretize the state space will dictate how complete we can be and until we can actually see the space and try it we won't know if you can discretize it or not. We also need to know what hardware we are working with and how much time we have. I am optimistic that we can discretize it finely enough and break the problem up into motion plans that are complete and optimal.