Planning and Control

Submission Guidelines

Writeups must be submitted as a PDF via Gradescope. LaTeX is preferred, but other typesetting methods are acceptable. Code for the programming component must be submitted in a zip archive via Canvas. Plots generated as part of the programming component should be included in the writeup.

Collaborators

List the names of all collaborators and which questions you collaborated on.

1 Motion Planning: Search and Sampling (45 points)

In this section you will be required to implement different motion planning algorithms and study the parameters that govern their behaviors.

1.1 Code Overview

The starter code is written in Python and depends on NumPy and Matplotlib. This section gives a brief overview.

- `run.py` - Contains the main function. Note the command-line arguments that you can provide.
- `MapEnvironment.py` - Environment-Specific functions. Some of them have to be filled in by you.
- `map1/2.txt` - Maps that you will work with. The numbers denote the collision status of the cell (1 - collision, 0 - collision-free).
- `AStarPlanner.py` - A* Planner. Logic to be filled in by you.
- `RRTPlanner.py` - RRT planner. Logic to be filled in by you.
- `RRTStarPlanner.py` - RRT* planner. Logic to be filled in by you.
- `RRTTree.py` - Contains datastructure taht can be useful for your implementation of RRT and RRT*.

To run A* on map 2, you would run

```
$ python run.py -m map2.txt -p astar -s 321 148 -g 106 202
```

1.2 Environment Modelling

The planning consists of a 2D map. You have been provided with two maps `map1.txt` and `map2.txt`. You can use the former to *test* your implementation but report all results on the latter with start and goal to be [321, 148] and [106,202]. Note that environement-specific functions need to be filled in by you in `MapEnvironment.py` file, these functions can include distance computations or sampling and nearest neighbor retrieval. Once you complete implementing the environment-specific functions, you are ready to implement the planners.

Note that in case of search algorithms like A*, the environment is considered to be a discrete grid while in sampling-based techniques the environment is assumed to be continuous. However, in this case since the underlying world is given to be grid, you can snap any continuous sample points onto the grid.

1.3 A* Implementation (10 points)

You will be implementing the weighted version of A-star where the heuristic is weighted by a factor of ϵ. Setting $\epsilon = 1$ gives vanilla A*. For more details on the algorithm, refer to the reading material on the course website. The main algorithm is to be implemented in `AStarPlanner.py` file.

1. Use an 8-connected neighbourhood structure so that diagonal actions are also allowed. Each action has a cost equal to the length of the action i.e. cost of action $(dx, dy) = \sqrt{dx^2 + dy^2}$. Note that diagonal actions cost $\sqrt{2}$.
2. Use the Euclidean distance from the goal as the heuristic function.
3. Try out different values of epsilon to see how the behavior changes. Report the final cost of the path and the number of states expanded for $\epsilon = 1,\ 10,\ 20$.
4. Discuss the effect of ϵ on the solution quality.
5. Visualize the final path in each case and the states visited.

1.4 RRT and RRT* Implementation (25 points)

You will be implementing a Rapidly-Exploring Random Tree (RRT) for the same 2D world. For more details on the algorithm, refer to the reading material on the course website. Think of the practical considerations in implementating the RRT-style algorithms. The main algorithms are to be implemented in

`RRTPlanner.py` and `RRTStarPlanner.py` files. Note that since these methods are non-deterministic, you'd need to provide statistical results (averages over around 10 runs).

1. Bias the sampling to pick the goal with 5%, 20% probability. Report the performance (cost, time) and include figures showing the final state of the tree for both values.
2. For this assignment, you can assume the point robot to be able to move in arbitrarily any direction i.e. you can extend states via a straight line. You will implement two versions of the extend function:
   - the nearest neighbor tries to extend all the way till the sampled point.
   - the nearest neighbor tries to extend to the sampled point only by a step-size η. Pick a small η of your choice and mention it in your write-up.

   As before, report the performance (cost, time) and include a figure showing the final state of the tree for both values. Which strategy would you employ in practice?
3. You will also be implementing RRT* as part of this question. You can implement this on top of your RRT planner with consideration for rewiring the tree whenever necessary. Compare the performance of the two algorithms, i.e. RRT* against RRT planner (with your choice of planner parameters).

2 Trajectory Optimization (20 points)

2.1 Representation (5 points)

1. Although CHOMP and TrajOpt are cavariant to trajectory representation, their implementation requires a choice of parameterization for the trajectory. Mention two ways to represent trajectories.
2. For each type of representation, express $U_{\text{smooth}} = \int \frac{1}{2} \|\xi'(t)\|^2 dt$ in terms of the parameters involved.

2.2 Inner Product and Gradient Descent (15 points)

Consider the three trajectories in Fig. 1. Note that they have a waypoint parameterization of a single-DOF system with $T1 = (0,0,0,0,0,0)$, $T2 = (0,5,0,0,5,0)$ and $T3 = (0,5,10,10,5,0)$ indexed by timepoint.

1. Consider the Euclidean distance norm where $< \xi_1, \xi_2 >_E = \xi_1^T \xi_2$. Which trajectory is closer to T1: T2 or T3?

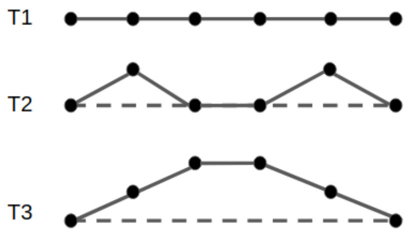

Figure 1:

2. If smoothness in deformation is weighed higher than closeness in position, which trajectory is closer to T1: T2 or T3?
   Hint: To determine this, start from $U_{smooth} = \int \frac{1}{2} \left\|\xi'(t)\right\|^2 dt$ *and define the inner product using the Hessian of* U_{smooth}*.*
3. Discuss your intuitive understanding of why the Euclidean metric does not capture the closeness in smoothness and how the Hessian overcomes the issue.

3 Task Planning (15 points)

It is essential for a symbolic planning problem to properly define the symbols, actions, the preconditions and effects appropriately to correctly solve the planning problem. Consider the Blocks World illustration in Fig. 2 as an example.

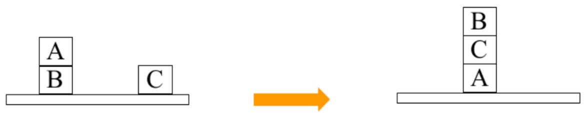

Figure 2: Blocks World

- **Symbols**: A, B, C, Table
- **Initial Condition**: On(A, B), On(B, Table), On(C, Table), Block(A), Block(B), Block(C), Clear(A), Clear(C)
- **Goal Condition**: On(B, C), On(C, A), On(A, Table)
- **Actions**:
  MoveToTable(b, x)
  Preconditions: On(b,x), Clear(b), Block(b), Block(x)
  Effects: On(b,Table), Clear(x), !On(b,x)

Move(b, x, y)
Preconditions: On(b,x), Clear(b), Clear(y), Block(b), Block(y), b != y
Effects: On(b,y), Clear(x), !On(b,x), !Clear(y)

In this homework, you are supposed to write the environment description files for two environments:

3.1 Blocks and Triangles World (5 points)

This environment is similar to the Blocks world problem discussed above. In addition to the blocks, this environment has triangles that can be moved in the exact same way as blocks with the exception that nothing can be put on top of them. A simple example of this environment with only three objects is shown below. Write a problem description file for an environment with 5 blocks (B0, B1, B2, B3, B4), 2 triangles (T0, T1) and a Table. The start and goal conditions are below:

- Start condition: B0 is on B1, B1 is on B4, B2 is on Table, B3 is on B2, B4 is on Table, T0 is on B0, and T1 is on B3.
- Goal condition: B0 is on B1, B1 is on B3, and T1 is on B0.

3.2 Fire Extinguisher Environment (10 points)

The goal of this problem is to have a pair of robots put out a fire. This domain has two robots 1) a quadcopter and 2) a mobile robot. The mobile robot can travel between locations. The quadcopter only moves between locations by landing on the mobile robot and having the mobile robot travel to the other location. The quadcopter can fly around a single location (cannot navigate between locations) if its battery level is High, but it won't be able to take off if its battery level is Low. Whenever the quadcopter is on the mobile robot, it can charge its battery by calling the charge action. The quadcopter has a tank that can be filled with water when the quadcopter is on the mobile robot at location W (where there is water). The fire is at location F. The W and F locations are far from each other. The quadcopter should fly around location F in order to pour water on the fire. The quadcopter needs to pour water on the fire three times in order to extinguish the fire. Every time the quadcopter pours water on the fire, its battery level becomes low and its water tank becomes empty (it should go back to W to fill its tank). The robots will each start at one of five different locations (A, B, C, D, E), which are far from W and F. The quadcopter cannot land on the ground.

- Start condition: The quadrotor is flying and at location B. The mobile robot is at location A. The quadrotor's water tank is empty.

- Goal condition: The fire is extinguished.

4 Control of a Cartpole (20 points)

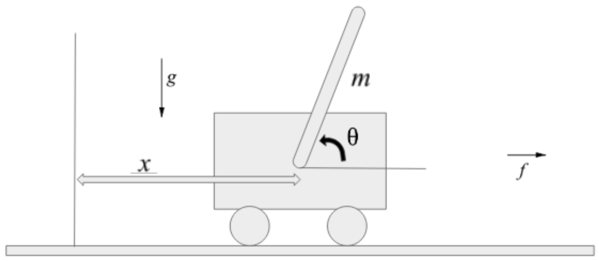

Figure 3: Cartpole

4.1 Modelling (5 points)

Control of a physical system requires the system to be modelled to determine the states to be controlled and the appropriate control inputs. Consider the inverted pendulum mounted on a cart as shown in Fig. 3. The pendulum has a uniform mass m and length l while the cart has a mass $3m$. Assume the surface to be frictionless. Gravity is as shown in the figure.

1. Assuming that the only control input that can be applied is the horizontal force f on the cart to control the position and velocities of the the pendulum and the cart, derive the statespace equation of the system.
2. Identify the points of equilibrium, if any, and if they exist, qualify their stability.

4.2 Control (15 points)

1. Linearize the system around $(x, \theta, x', \theta') = (0, \frac{\pi}{2}, 0, 0)$. Using the linearized model, determine if the system is stable and controllable?
2. If the system is unstable, design *a* stabilizing controller for it and discuss the choice and effect of the controller.
3. Using an LQR determine the optimal controller for the system for your choice of Q and R. Discuss the design decisions made. Assume $m = 1,\ l = 1,\ g = 10m/s^2$. Feel free to use available python packages to solve the Ricatti equation.