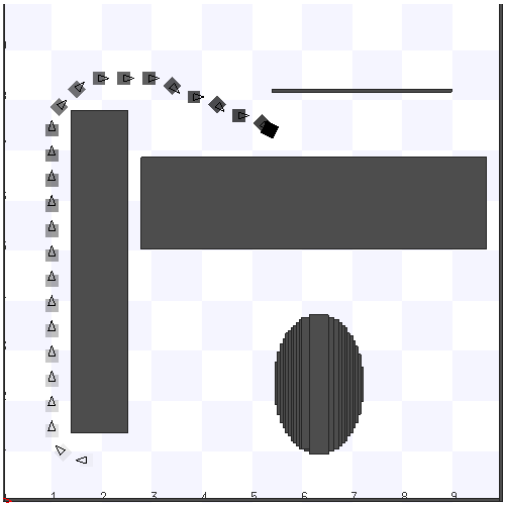
Assignment 3
Mobile Robot Planning

Complete by **TBA**

In this assignment, we will develop and implement simple planning algorithms for mobile robot navigation and test them in simulation.



Mobility is essential to intelligent robots. As a burgeoning roboticist, you plan to develop a food delivery robot to operate in the new SoC building under construction now. The robot picks up a delivery order at the cafe, navigates the building, and sends the order to the final destination. Over time, the robot may learn to perform other useful navigation tasks, *e.g.*, guiding a first-time visitor to the building. The system will be implemented on TurtleBot, a low-cost personal robot. Hence its name *FoodTurtle*.

For this assignment, assume that FoodTurtle has a map of the building and knows its exact location all the time. The objective is to reach the specified goal by following a collision-free path. You will develop and implement a motion planning system with

- *Input*: a map, start robot pose, and goal robot pose;
- *Output*: a sequence of robot actions that (i) avoid collision with obstacles marked in the map and (ii) minimize travel distance to the goal.

You will test your implementation in the ROS Stage simulator. The simulator simulates TurtleBot motion control as well as well motion execution uncertainty. So you cannot always assume that the computed actions are executed perfectly.

You will develop the motion planning system, using three increasingly more sophisticated and accurate models:

- discrete state space and deterministic actions (DSDA) ;
- continuous state space and deterministic actions (CSDA));

- discrete state space and probabilistic actions (DSPA).

Accurate models potentially improve the system performance, but they also require more sophisticated algorithms and greater computational resources. It is natural to ask whether there are models with continuous state space and probabilistic actions. The answer is yes, but they require complex mathematical and algorithmic tools beyond the scope of this class.

To measure system performance, we use the average *Success weighted by Path Length* (SPL) over N trials:

$$\sigma_{\text{SPL}} = \frac{1}{N} \sum_{i=1}^{N} S_i \frac{\ell_i}{p_i} \tag{1}$$

where S_i is a binary variable indicating whether the robot succeeds in reaching the goal without collision, ℓ_i is the length of the shortest path, and p_i is the length of the actual path that the robot traverses, in the ith trial. Intuitively, SPL is a combined measure of the success rate and the path length.

1 Discrete States and Deterministic Actions

To begin with, we use a much simplified robot motion model.

States

The TurtleBot state space is discretized as a grid of size $M \times N \times K$. A discrete state is represented as a tuple (x, y, θ), where $x, y \in \mathbb{Z}$ specify the horizontal and vertical positions of the robot in the grid and $\theta \in \{0, 90^\circ, 180^\circ, 270^\circ\}$ specifies the orientation.

Actions

There are four discrete actions: FORWARD, LEFT, RIGHT, STAY.

- FORWARD: move forward by 1 grid cell in the current orientation;
- LEFT: turn left by 90°;
- RIGHT: turn right by 90°;
- STAY: stay in place with no movement.

Motion Planning

Any forward search algorithm can solve the motion planning task here. For a complex environment, A* search likely provides the best performance. For A* search, it is important to choose the search heuristic carefully to find an optimal path efficiently.

TurtleBot motion controller executes the discrete actions. However, it may have small errors because of robot control imperfection and unknown environmental interactions. So, to avoid collision with obstacles as a result of control uncertainty, one way is to "inflate" the robot by a small amount to leave some margin of safety.

2 Continuous States and Deterministic Actions

Discrete states and actions are simplifying model assumptions. In reality, the robot state space and action space are continuous.

States

The state space is continuous. The robot pose is specified as (x, y, θ), with $x, y, \theta \in \mathbb{R}$ and $x \in [x_{\min}, x_{\max}], y \in [y_{\min}, y_{\max}], \theta \in [0, 2\pi)$.

Actions

Suppose that we control FoodTurtle by the linear velocity v and angular velocity ω, with $v \in [0, 1]$ and $\omega \in [-\pi, \pi)$. If the initial robot pose is (x, y, θ), applying (v, ω) for a small time duration Δt results in the new pose:

$$\begin{aligned} x' &= x + \frac{v}{\omega}\cos\theta - \frac{v}{w}\cos(\theta + \omega\Delta t) \\ y' &= y - \frac{v}{\omega}\sin\theta + \frac{v}{w}\sin(\theta + \omega\Delta t) \\ \theta' &= \theta + \omega\Delta t \end{aligned} \tag{2}$$

In our simulator, the default simulation frequency is 10Hz, which gives $\Delta t = 0.1$. More details can be found in the GitHub repository for this assignment.

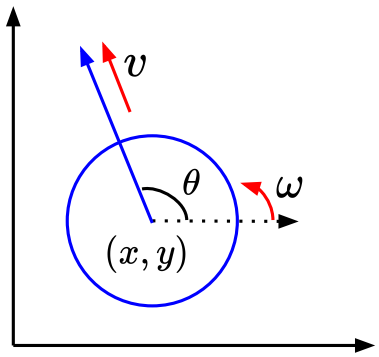


Figure 1: Continuous motion model of the robot. Blue circle and arrow denote the robot at (x, y) with orientation θ; red arrows denote its the linear velocity v and the angular velocity ω.

Motion Planning

The hybrid A* algorithm is well-suited for motion planning in the continuous state space with deterministic actions. In addition to the usual considerations for the A* algorithm, you also need to discretize the action space and the state space during the implementation.

3 Discrete States and Probabilistic Actions

A good model of uncertainty is critical to robust robot performance. For small amount of control uncertainty, we can "inflate" the robot to a safety margin. As uncertainty gets larger and more complex, it is impractical to provide a sufficiently large safety margin. More sophisticated tools, such as the Markov Decision Process (MDP), are required to model uncertain action consequences probabilistically.

States

While the continuous state space is more accurate, we use the MDP model, which assumes the simpler discrete state space, in order to keep computational cost moderate.

Actions

There are again four discrete actions: FORWARD, LEFT, RIGHT, STAY.

FORWARD models $(v, \omega) = (1, 0)$ for $\Delta t = 1$ second. However, the linear and the angular velocities are not achieved exactly because of wheel slippage.

Specifically, when the robot is at $s = (x, y, \theta)$ and want to take an action $a =$ FORWARD, it has to execute the action $(v, \omega) = (1, 0)$ for one second. 90% of the times, the robot can perform an accurate forward action i.e., $P(s_{forward} \mid s, a) = 0.9$. However, 5% of the times, the robot will falsely have an angular speed and a forward velocity of $\pi/2$, i.e., $p(s_{forward\_left} \mid s, a) = 0.05$; 5% of the times, robot will have an angular speed of $-\pi/2$, i.e., $p(s_{forward\_right} \mid s, a) = 0.05$. The states $s_{forward}$, $s_{forward\_left}$ and $s_{forward\_right}$ can be computed using the dynamic model Eq. 2 by setting the corresponding actions and let $\Delta t = 1$.

For simplicity, we assume the turning actions and stay action are deterministic, i.e., $p(s' \mid s, a) = 1$, where $s' = (x, y, \theta + \pi/2)$ and $a =$ LEFT. The same transition function applies to the RIGHT and STAY action.

4 Simulator Setup

To set up the simulator, install the following packages:

```
$ sudo apt install ros-kinetic-turtlebot-stage
$ sudo apt install ros-kinetic-joint-state-publisher-gui
```

After installation, clone the assignment GitHub repository and set up the shell PATH:

```
$ git clone git@github.com:AdaCompNUS/CS4278-CS5478_assignments.git
$ cd CS4278-CS5478_assignments
$ catkin_make
$ source devel/setup.bash
```

Now, you should be able to launch your ROS stage simulator

```
$ roslaunch planner turtlebot_in_stage.launch
```

5 Coding Template

The GitHub repository contains the skeletal code for you to get started. To implement the algorithms for the three models, fill in the missing code snippets marked with "FILL ME":

- Subscribe and store start and goal poses;
- Compute the shortest collision-free path or policy from the start to the goal pose.
- Publish the control commands found to the controller.

More details on the implementation are available in the GitHub repository.

6 Report

Answer the following questions briefly in your report.

- For the DSDA model, what search algorithm do you use and why? If the A* algorithm is used, what is the search heuristic?
- For the CSDA model, how do you discretize the state space and the action space?
- For the DSPA MDP model, what reward function do you use? How do you solve the MDP?

7 Grading Criteria

You will be given a set of different maps along with start-goal pairs for grading. Your grade consists of two components. The primary component is the SPL scores on the test maps. The secondary component is your report.

Submission