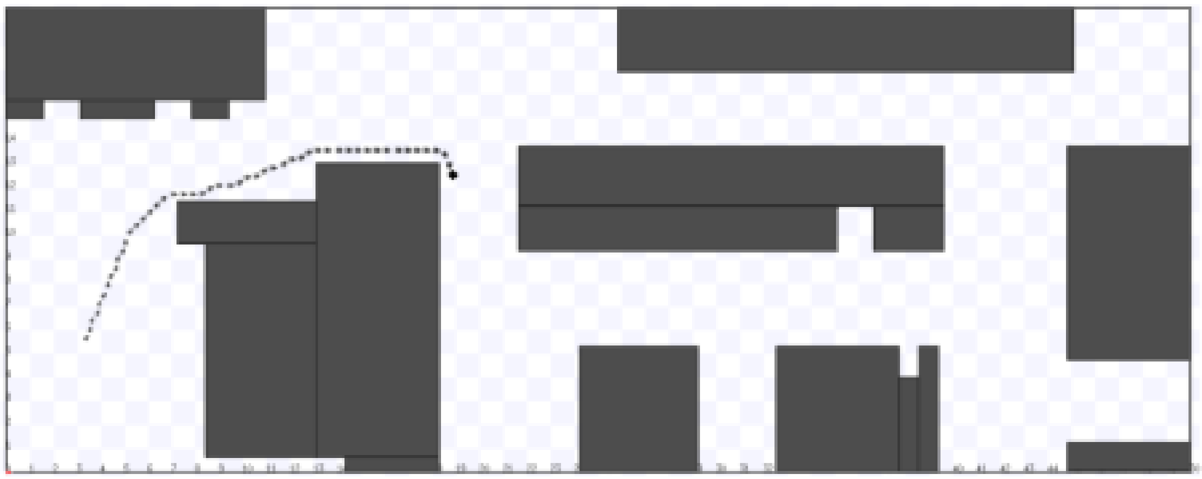
Lab 1

Motion Planning for Mobile Robots

Instructions

- Due on **2 March 2023 11:59pm**.
- Submit your work online to Canvas under `Assignments/Lab 1` as a single zip archive. `MatricNo-Lab1.zip`. See GitHub repository for detailed instructions.



Mobility is crucial for intelligent robots. As a budding robot engineer, you plan to develop a delivery robot that operates in our SoC buildings. The robot picks up a delivery order at a cafe, navigates the building, and delivers the order to its final destination. Over time, the robot may learn to perform other useful navigation tasks, e.g., guiding first-time visitors to the building. The system will be implemented on TurtleBot, a low-cost personal robot. Hence its name, FoodTurtle. In this lab, you will develop a motion planning system for FoodTurtle and test it in simulation.

Assume that FoodTurtle has a map of the building and always knows its exact location. It tries to reach a specified goal by following a collision-free path. We will design and implement motion planning algorithms with

- *Input*: the map and the robot's starting and target poses;
- *Output*: a sequence of robot actions that avoids collision with obstacles marked on the map and minimizes the robot's travel distance to the goal.

We will test the algorithms in the ROS Stage simulator. The simulator simulates TurtleBot motion with uncertainty. In other words, we cannot assume perfect robot control.

We will develop motion planning algorithms under three models with increasing power and complexity:

- Discrete states & deterministic actions (DSDA)
- Continuous states & deterministic actions (CSDA)
- Discrete states & probabilistic actions (DSPA)

One may wonder whether there are models with continuous states and probabilistic actions. The answer is yes, but they require more sophisticated mathematical and algorithmic tools, not fully covered in this class. In general, more powerful models potentially improve system performance, but they also demand more sophisticated planning algorithms and greater computational resources for planning.

To measure performance, we use the success-weighted path length (SPL) averaged over N trials:

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

where S_i is a binary variable indicating whether the robot successfully reaches the goal without collision, p_i is the length of the actual path traversed by the robot, and l_i is the length of the shortest path, all in the ith trial. Intuitively, SPL is a combined measure of success rate and path length.

Discrete States & Deterministic Actions (DSDA)

DSDA is a much simplified robot motion model.

States. FoodTurtle has position $(x,\ y)$ in the plane and orientation θ. The state space is discretized into a grid, with $x, y \in \mathbb{Z}$ and $\theta \in \{0,\ 1,\ 2,\ 3\}$ (the four elements of θ correspond to 0, 90, 180, and 270 degrees).

Actions. There are four discrete actions:

- FORWARD: move 1 grid cell forward in the current orientation.
- LEFT: turn left by 90°.
- RIGHT: turn right by 90°.
- STAY: stay in the current grid cell.

Hint. Any forward search algorithm can solve the motion planning task here. For complex environments, the A* algorithm likely provides the best performance. To find the optimal path efficiently, the A* algorithm requires a carefully designed heuristic.

While the TurtleBot motion controller performs discrete moves, it may have small errors as a result of unmodeled imperfections in robot control and environment interactions. To avoid collisions with obstacles due to uncertainty, one practical solution is to "inflate" the robot by a small amount to create a safety margin.

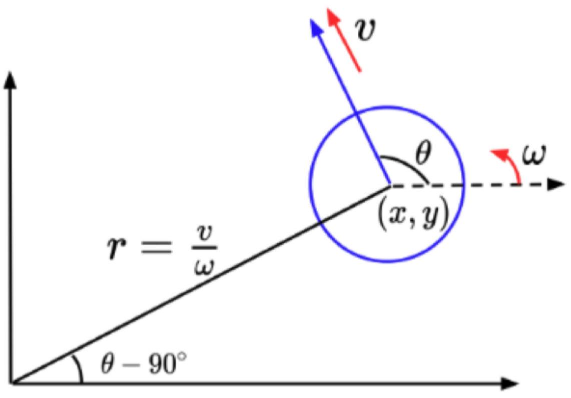
Continuous States & Deterministic Actions (CSDA)

In reality, most robots have continuous states and actions.

States. The FoodTurtle state is specified as $(x,\,y,\,\theta)$ with, $x\in[x_{\min},\,x_{\max}]$, $y\in[y_{\min},\,y_{\max}]$ and $\theta\in[0,\,2\pi)$.

Actions. FoodTurtle controls linear velocity v and angular velocity ω, with $v\in[0,\,1]$ and $\omega\in[-\pi,\,\pi]$. If the robot has initial state $(x,\,y,\,\theta)$, the new state after applying $(v,\,\omega)$ for time Δt is given by

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



In our simulator, the default simulation frequency is 10Hz, which gives $\Delta t = 0.1$s.

Hint. The hybrid A* algorithm is well-suited for motion planning in the continuous state space with deterministic actions.

Discrete States and Probabilistic Actions (DSPA)

A model of uncertainty is important for robust robot performance. For a small amount of action uncertainty, we can inflate the robot to create a safety margin. As the uncertainty becomes larger and more complex, it would be impractical to create a safety margin this way. More sophisticated tools, such as the Markov Decision Process (MDP), are needed to model uncertain actions.

States. While continuous states are more accurate, we use the simpler MDP model, which assumes discrete states.

Actions. There are four discrete probabilistic actions: FORWARD, LEFT, RIGHT, and STAY. These four actions control the linear and angular velocities $(v,\,\omega)$. However, the nominal linear and angular velocities cannot be fully achieved, because of wheel slippage and other factors. For the purpose of this lab, we use a much simplified model of uncertainty on actions.

Action	**Nominal** (v, ω)	**Actual** (v, ω)	**Probability**
FORWARD	$(v, \omega) = (1, 0)$	$(v, \omega) = (1, 0)$	0.9
		$(v, \omega) = (\pi/2, \pi/2)$	0.05
		$(v, \omega) = (\pi/2, -\pi/2)$	0.05
LEFT	$(v, \omega) = (0, \pi/2)$	$(v, \omega) = (0, \pi/2)$	1
RIGHT	$(v, \omega) = (0, -\pi/2)$	$(v, \omega) = (0, -\pi/2)$	1
STAY	$(v, \omega) = (0, 0)$	$(v, \omega) = (0, 0)$	1

Coding Template

The GitHub repository contains the skeleton code to get started. Briefly,

- subscribe and store the starting and target poses,
- compute the shortest collision-free path or an optimal closed-loop plan,
- publish the computed control commands to the controller.

To implement the algorithms for the three models, complete the missing code snippets labeled `# TODO: FILL ME!`. Details about the implementation can be found in the GitHub repository.

Report

Answer the following questions briefly (maximum 250 words for each question).

1. For the DSDA model, what search algorithm do you use and why? If the A* algorithm is used, what search heuristic do you use?
2. For the CSDA model, how do you discretize the state space and the action space?
3. For the DSPA MDP model, what reward function do you use? How do you solve the MDP?
4. Describe any interesting lessons and insights that you have learned.

Grading

For grading, we will provide a new dataset, which is 728402 from the dataset currently in GitHub repository. The performance, *i.e.*, the average SPL of your implementation on the grading dataset is the primary consideration for grading, with some additional considerations given to the report.