Lab 1 Report
Motion Planning for Mobile Robots

Name: Niharika Shrivastava
Matriculation No: A0254355A

Note: The code can be evaluated as

```
$ python evaluation.py \
      -g '../../../files/goals.json' \
      -m '../maps/aug_maps/' \
      -c '../../../files/Controls/'
```

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?**
   For DSDA, I have used the A* search algorithm.
   Let `rows = (world_height * resolution), cols = (world_width * resolution)`.
   - Since the state space is discretized, the entire map can be treated as a grid of size `[rows, cols]` with discrete points as integers ($\mathbb{Z}$). Therefore, there are only `(rows * cols)` possibilities for the robot's position. This makes the grid manageable for computation for all the maps provided and thus any forward search algorithm using graphs (BFS, DFS, A*, etc) can be used to get the optimal path.

   - However, in order to find the optimal path efficiently, i.e., reduce computation time by exploring fewer states to reach the goal at the lowest cost with the help of an intelligent cost estimate, the A* algorithm is used.

   - The search heuristic used is the **euclidean distance between the current state and the goal state**.

2. **For the CSDA model, how do you discretize the state space and the action space?**
   - **State space:** The FoodTurtle state is specified as `(x, y, θ)` where:
     - $x \in [x_{min}, x_{max}]$, $y \in [y_{min}, y_{max}]$, and $x, y \in \mathbb{Z}$.
     - $\theta \in \{0, 1, 2, 3, 4, 5, 6, 7\}$ where the eight elements of θ correspond to $\{0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°\}$ respectively.

   - **Action space:** FoodTurtle controls linear velocity v and angular velocity ω where:
     - $v \in \{0, 1\}$
     - $\omega \in \{-\pi, -3\pi/4, -\pi/2, -\pi/4, 0, \pi/4, \pi/2, 3\pi/4, \pi\}$

3. **For the DSPA MDP model, what reward function do you use? How do you solve the MDP?**
   A. For the DSPA MDP Model, the reward function is as follows:
      - `Reward(goal) = 10,`
      - `Reward(obstacle) = -10,`
      - `Reward(current_state, action, next_state) = -0.5`
        `∀ next_state = obstacle`
      - `Reward(free state) = 0`

   B. This path-planning problem can be formulated as a discrete MDP as follows:
      - `S` is a set of states and $s \in S$ where `s` represents the current position of the robot.
        $s = (x, y, \theta)$ such that $x \in [x_{min}, x_{max}]$, $y \in [y_{min}, y_{max}]$, and $x, y \in Z$.
        $\theta \in \{0, 1, 2, 3\}$ where the four elements of θ correspond to {0°, 90°, 180°, 270°} respectively.

      - `A` is a set of actions for the robot.
        `A = {FORWARD (1,0), LEFT (0, -1), RIGHT (0, 1), STAY(0, 0)}`.

      - $P(s, a, s') = p(s' | s, a)$ is a probabilistic state-transition function which accounts for action uncertainty by prescribing a probability distribution for the new state s' at time $t+1$ if the robot takes action a in state s at time t. This follows the dynamics given in the question.

      - `R(s)` is a reward function according to the specification in part A.

   I use Value Iteration for computing an optimal MDP policy and its value.
   $V_t(s) = R(s) + \gamma * \max_{(a \in A)}\{\sum_{(s' \in S)}\{P(s,a,s') * V_{t-1}(s')\}\}$, where γ is the discount factor.

4. **Describe any interesting lessons and insights that you have learned.**
   - The most significant lesson was that no matter how accurate the algorithm and action plan is, errors accumulate over time in a dynamic and stochastic environment leading to the agent never reaching the intended target in some cases (even if theoretically following the suggested action plan would've led the agent to the target).

   - In the case of CSDA, because of the accumulation of errors in the case of rotation, a rotation of -pi/4 is more accurate than a rotation of 7pi/4 even if both result in the same end orientation. This mistake cost me 2 days to debug.

   - In the case of DSPA, it is of utmost importance to design the rewards intelligently and handle all uncertain conditions, e.g., penalize future collision due to an action, incentivize the agent to move forward towards the goal and explore rather than stay put in the current

state, avoid paths extremely close to the obstacle so that the probability of collision due to wheel slippage is minimized.