Discrete Single-Robot Path Planning (SAPP)

Guillaume Sartoretti, Project 2 Proposal **Example**

Why:

- Many robotic deployment demand robots be able to plan efficient paths in complex, often unknown/dynamic environments, such as ground robots in warehouses/factories, or autonomous vehicles on the streets. In those cases, robots need to select the right move actions to reach their goal, while avoiding collisions with obstacles, that might be static or dynamic. In this project, we mostly focus on cases involving static obstacles, but the approach could easily be extended to dynamic ones (note from Guillaume: **in particular, you are not allowed to propose path planning with dynamic obstacles as your project of course, as this would be too close to this example proposal; any other similar project will also be refused**).
- To respect the robotic inspiration, we further consider that the robot will need to plan its move actions to reach its goal based on local sensing of the environment only, i.e., the robot does **no** have access to the global map of the environment (showing all obstacles along the way). Instead, the robot can only sense nearby obstacles in a small range around its position, and must learn to sequence move actions based on this partial knowledge.

Conventional Algorithms

- For single-agent path planning in discrete worlds with static obstacles, state-of-the-art planners are graph-based search algorithms, such as A* and Dijkstra (and variants of such algorithms such as Windowed A*, etc.). These planners pre-plan a (bounded-)optimal path for an agent before the agent even starts moving, which can sometimes slow down the robot's task (as it remains idle while the planning happens). Instead, I believe that allowing the agent to plan its path online might lead to better performances in many scenarios, by allowing the robot to start moving immediately upon receiving its goal, and select its route as it goes (instead of only starting to move once it has a full plan).

Problem Statement:

- Simple discrete grid world (N x M cells), with randomly placed obstacles. A single robot (occupying one cell) is tasked with getting to a goal cell (gx,gy) from its current position (x,y), as fast as possible (minimum number of move actions), while avoiding collisions with obstacles.
- The robot is only allowed a local observation range around its position, in a square field of view (say 11x11 cells). In this field of view, the agent can see obstacles/free space, and its goal if within range. Additionally, we provide the robot with a 2D unit vector always pointing to its goal, as well as the distance to its goal (in a straight line). Details below.

RL Cast:

- State space:
  - Three 11x11 (FOV-sizes) arrays showing: 1) obstacles around the agent (1 if obstacle, 0 if free space), 2) goal location if within FOV (1 if goal, 0 otherwise), and 3) agent's position (1 at the central position, this array is constant since the FOV moves with the agent).
  - 3 scalar values: the x- and y-components of a unit vector pointing from the agent to its goal, and the distance to the goal in a straight line.
- Action space: 5 actions -- move North/East/South/West, and stay in place.
- Reward structure: move action have a small penalty (say -0.1), staying in place is a larger penalty (say -0.5, to encourage exploration), collisions with obstacles an even larger penalty (say -1), and arriving on goal is a big positive reward (say +5).
- Neural Network Structure: CNN followed by some fully connected layers (MLP), and finally an LSTM (unsure that last part is useful, but why not). Scalar values are concatenated with the features extracted by the CNN, before being fed to the fully connected layers.

RL algorithms and envisioned results:

- To approach the problem, we will use the Advantage Actor-Critic algorithm, as it is an efficient policy gradient algorithm, which I also like it particularly (yes, that is a valid reason to propose to try an algorithm). However, note that for such a simple problem with discrete state and action spaces, we could also be looking into DQN approaches, or look into state-of-the-art single-agent algorithms such as PPO/Ape-X.
- We will compare the performance of our RL-based algorithm with these planners, in terms of path optimality (number of steps until the goal is reached), success rates (number of problems successfully solved by RL, vs number of problems solves by A* under a given time limit) as well as planning times.
- These tests will be performed in worlds of varying sizes and obstacles densities, and we will discuss these results and the advantages/downsides of either approaches for each case.