RRT*-guided robot learning for planning

Vidyaa Krishnan Nivash, Shivam Bhat

***Abstract*—Imitation learning is a type of learning policy where in an agent learns to perform a task by observing demonstrations of the task performed by an expert. In imitation learning policy, the agent learns a policy by imitating the expert's behavior. The goal is to learn a policy that can generalize to new situations beyond the demonstration set. It is useful when it is difficult or costly to design reward functions or when the expert's behavior is optimal or near-optimal. In this work we implement an imitation learning policy for a turtlebot in a maze environment. Initially RRT* is used to gather data, including local lidar observations. RRT* is a probabilistically complete algorithm used for path planning in robotics that aims to find an optimal path from the start to the goal while avoiding obstacles. Furthermore, an imitation learning policy is implemented to generate the next waypoint toward the goal using only the local lidar observations. Broadly the work consists of following three contributions:**

- **Setup maze environment with Turtlebot in Pybullet**
- **Run RRT* to gather data, including local lidar observations**
- **Implement an imitation learning policy to generate the next waypoint toward the goal using only the local lidar observations**

I. Milestone 1

In the first milestone we setup a U-maze environment with a turtlebot robot using gym in pybullet environment. The robot moves as per given control inputs. In the current initial setup the robot is implemented to move with a random policy.

A. U - Maze

We have extended the original PyBullet environments provided in pybullet_envs. Pybullet has a physics engine which can be connected to using graphical or non-graphical mode. URDF are taken from [1] and modified according to our environment.

Steps for creating the walls:

- Create ground using a plane urdf by calling loadURDF() function
- each wall (left, right, top, bottom) consist of a block in itself, which can be created using a box geometry with length, width and height specified.
- Parameters of loadURDF()[2]:
  - (x,y,z) - position in which the model has to be loaded
  - useFixedBase set to True, since the walls need not move according to physical interactions
  - URDF_USE_SELF_COLLISION - set to True in order to check the collision of the wall with itself
- changeVisualShape() is used to generate walls of different colors and textures
- Everytime the robot is reset to original position, walls are created and updated

B. Turtlebot

PyBullet is a physics simulation engine that can be used to simulate various robots, including the TurtleBot we use herein. Through PyBullet, developers can simulate the TurtleBot's movement, sensor data, and environment in a virtual setting. This allows for testing and development of the TurtleBot's algorithms and behaviors without the need for a physical robot. Additionally, PyBullet also allows for the creation of complex scenarios that may be otherwise difficult to recreate in the real world.

The TurtleBot robot can be represented in PyBullet as a set of rigid bodies and joints joined together by constraints.The robot's primary body can be visualized as a box-shaped structure with two wheels hinged to it.The motors that power the wheels can be modelled in PyBullet by applying rotational forces or torques to the wheels.The depth camera and 2D laser scanner sensors on the TurtleBot can be modelled using PyBullet's collision detection and raycasting features. Another way to simulate the gripper arm is to use a number of rigid bodies and joints. Furthermore, PyBullet has a variety of environmental models and objects that can be utilized to construct intricate scenarios for

The TurtleBot is equipped with various sensors, including a 2D laser scanner, lidar sensors, depth camera amongst other, which enable it to perform autonomous navigation and mapping tasks. In this implementation we will be using the lidar data for the purpose of path planning.

C. Maze.py

Main code where the episodes of robot and maze are created using gym.make() and run using env.reset(), step() and render()

D. Versions

- gym==0.17.2
- pybullet==2.6.4
- opencv-python==4.1.2.30

E. Code structure

The main driving code is present in the file 'maze.py'. After registering the various global parameters using the 'bullet_envs' init file the environment 'TurtlebotMazeEnv-v0' is initialized using gym.make(). The scene is then finally rendered using a random policy. '/src/bullet_envs/turtlebot/' contains all the assets required to render the robot and maze environments. '/src/bullet envs/utils.py' is used to write various utillity functions like those for seeding and rnadom noise generation.

F. Results

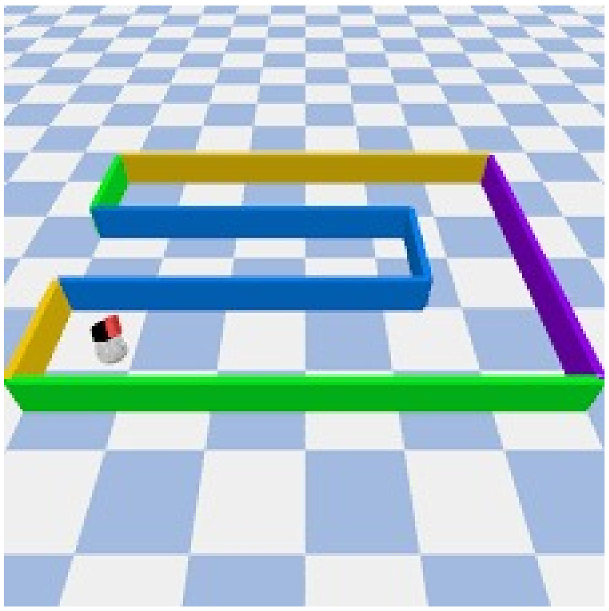

Fig. 1. Maze environment with turtlebot

References

[1] https://github.com/astrid-merckling/bullet_envs
[2] https://medium.com/@chand.shelvin/pybullet-getting-started-a068a0e3d492