Homework 6 - Multilateration-based location estimation

-- Course: *Intelligent Robotics – Professor: Qi Hao*

Coding Homeworks. Your final submission should be a compressed package with extension .zip, which includes your codes and explanations (you need to know how to write the manuscript with Markdown or LATEX). Your code should be run step-by-step without any error. Real-time animation is also recommended.

Overview

Multilateration is the technique of using only distance measurements from stations to determine the location of a target, with no information about the direction from which this distance was measured. Given a coordinate and a measured distance, the set of all points for which the target could be located forms a circle; the coordinate is the circle center and the distance is the circle radius. Multilateration is a classic problem in navigation and target tracking, and also applicable to problems such as using time-domain reflectometry to determine locations of faults in modem connections. For example, a GPS receiver uses multilateration to determine its exact location from its measured distances from at least three satellites, each satellite is at the center of a sphere and where they all intersect is the position of the GPS receiver.
For detail of GPS localization, please refer to How GPS Receivers Work - Trilateration vs Triangulation.

Multilateration of a single target

To simplify the problem, here we only consider the Multilateration-based location in the two-dimensional coordinate frame.
We simulates an 802.11az network consisting of a station (STA) and multiple access points (APs). To estimate the position of a STA, the network requires a minimum of three APs. The example simulates a ranging measurement exchange for each STA-AP pair, then trilaterates the position of the STA by using these distance measurements.
 demo

With only one station (STA), there are infinitely many points on the circle where the targets could be located. With two stations whose distance circles intersect, the possible target location is reduced

down to two possible points. With three or more stations that intersect at a single point, the target location can be isolated to that point.

Mathematical Derivation

This example shows how to estimate the position of a station (STA) in a multipath environment by using a time-of-arrival-based (ToA-based) positioning algorithm defined in the IEEE® 802.11az™ Wi-Fi™ standard.

x^3

then estimates the two-dimensional position of a STA by using trilateration.

- white: the start position
- red: the goal position
- green: the obstacle
- black: ground
- obstacle reward: -10
- goal reward: 10
- other reward: -1
- over the bound: -5

Question1

Please simulate the value iteration algorithm to calculate the value for a given policy.

Question2

Please simulate the policy iteration algorithm to improve a policy under the given value function.

Question3

Please find the optimal path under a given grid map environment with reward using Markov Decision Process (MDP)

Coding instruction

Install the intelligent robotics simulator

```
git clone -b edu https://github.com/hanruihua/intelligent-robot-simulator.git
cd intelligent-robot-simulator
pip install -e .
```

Note1: Please confirm that this repository is under the *edu* branch. You can use **git branch** to check current branch. If it is not under the *edu* branch, you can use **git checkout edu** to change current branch to *edu* branch.

Note2: The pycharm reduces the functionality of Matplotlib, which may lead to the failure of saving the gif animation. You can follow this link to solve this problem

Note3: If you have installed this simulator, you can use *git pull* to fetch the code update.

Code for questions

There are seven files for these questions in the source folder, *question_run1.py*, *question_run2.py*, *question_run3.py*, *mdp.py*, *grid_map.py*, *map_matrix.npy*, and *reward_matrix.npy*

- *question_run1.py* is the main program you should run for question1
- *question_run2.py* is the main program you should run for question2
- *question_run3.py* is the main program you should run for question3
- **mdp.py** is the file to perform Markov Decision Process. You should complete the functions include value iteration and policy iteration in this file for question1 and question2 respectively.
- *grid_map.py* is the file that defines the class about the grid map for you to use.
- *map_matrix.npy* and *reward_matrix.npy* define the map and the reward in each grid.

You should complete the file **mdp.py** for question1 and question2, and *question_run3.py* for the question3 to show the simulation results (as follows). You can set the parameter *animation* = *True* in *question_run.py* to generate the animation such as the follows.