B4M36UIR Semester Project

Elizaveta Isianova

February 4, 2024

1 Introduction

The primary objective of the project was to develop a runnable Python script, Explorer.py, capable of connecting to the simulated robot in the CoppeliaSim simulator. This script is expected to enable autonomous navigation, collision avoidance, and the collection of metric maps of the environment. The success of the mobile (multi-)robotic exploration is measured by the coverage percentage of the explored area over time.

2 Implemented approaches

2.1 Mapping

The map of the environment is represented as a probabilistic occupancy grid map of a static size, which can be changed with parameters `GRIDMAP_WIDTH` and `GRIDMAP_HEIGHT` in `Explorer.py`. The map is updated by fusing the laser scan data sampled by the robot into the current state of the map.

2.2 Frontier detection and Planning

The `Explorer.py` contains three different exploration approaches for solving the frontier detection and planning tasks, which are described bellow.

2.2.1 Exploration type 1

Frontiers detection. Frontiers are selected as centroids of segmented free-edge cells, which are then divided into multiple clusters.
Planning. The multiple-representative free-edge cluster frontiers list is sorted by the distance from the current robot's position in the ascending order. The closest feasible frontier is selected as the next goal. If a path to this frontier can not be estimated, the next closest frontier is selected.
Tasks: f1 with f2 extension, p1.
How to run:

```
$ python Explorer.py 0 -p 1
```

2.2.2 Exploration type 2

Frontiers detection. First the frontiers are detected the same way as in the previos method. Then the mutual information of the frontiers is calculated. Thus each frontier is assigned a utility value based on the information which can be gained about occupancy from observing the environment from this frontier.
Planning. The frontier with the greatest utility value is selected as the next goal. If a path to this frontier can not be estimated, the frontier with next greatest utility value is selected.
Tasks: f3, p2.
How to run:

```
$ python Explorer.py 0 -p 2
```

2.2.3 Exploration type 3

Frontiers detection. Multiple-representative free-edge cluster frontiers detection, the same as in the first exploration method.
Planning. The goals are selected with as a solution of the TSP, which is obtained with the `LKH` solver.
Tasks: f1 with f2 extension, p3.
How to run:

```
$ python Explorer.py 0 -p 3
```

2.3 Multi robot exploration

2.3.1 Multi-robot without interprocess communication

The multi robot exploration is implemented in the script `MultiExplorer.py`. The centralized system utilizes the class `MultiExplorer` which performs mapping and path planning for both robots. Each of the two robots create an instance of the modified class `Explorer`, which functionality is now limited to trajectory following, continous collision detection and connection to CoppeliaSim API. The occupancy map of the environment is created in the instance of `MultiExplorer` by sequentially fusing data from the laser scanners of the both robots. The `MultiExplorer` class also provides goal selection for both robots based on the greedy approach, e.g. it selects the closest frontier to the given robot. If the newly selected frontier is already close to the goal of the other robot, this goal is discarded and another frontier is assigned. The exploration terminates when there are no feasible frontiers left.
Tasks: a1, a3.

How to run:

```
$ python MultiExplorer.py
```

2.3.2 Multi-robot with interprocess communication

Every instance of the robot navigation is launched via a new script launch, but the connection to CoppeliaSim API is only established from the first one. Inter process communication is implemented with TCP socket communication, where the first launched robot creates an instance of the `TCP_Server` class, and the next connected robot instantiates the `TCP_Client`. The robots share their fused occupancy grid maps, their current locations and current status values. To prevent collisions with each other the other robots' locations are added to their navigation map as an obstacle, but only after the laser scan data is infused to the map. Decision making is decentralized, each robot picks the goal frontier corresponding to the selected exploration method (2.2).
The mutexes are used to protect the data intergrity.
Task: a4.

How to run (in two terminal windows):

```
$ python Explorer.py 0 -m
```

```
$ python Explorer.py 1 -m
```

3 Features for robustness

3.1 Continous collision monitoring

While following the planned trajectory the robot might discover new obstacles on the way. To prevent collision, the current path is always monitored with the Bresenham's algorithm. In case of collision detection, the path is discarded and a new process of path planning is started.

3.2 Goal frontier presence monitoring (deprecated)

A method for continuously monitoring the presence of the selected frontier was implemented. The idea was to discard the currently used path, if the frontier is no longer present. But after some testing, this feature became deprecated due the caused oscillation of the exploration.

4 How to run the project

The project was developed on Ubuntu 22.04, with Python 3.10 in virtual conda environment. The required Python packages are listed in the `requirements.txt`. It is also possible to build the conda environment by running from the project root folder:

```
$ conda env create -f environment.yml

$ conda activate mobile-exploration
```

To run the project:

1. launch the Coppelia simulator (the used version is CoppeliaSim v4.6.0 (rev. 16)), select a scene,
2. go to root folder of the project repository and run

   ```
   $ python Explorer.py [-p {1,2,3}] [-m] ID
   ```

   with appropriate arguments:

`[-p {1,2,3}]`	defines the exploration type from (2.2), the default is method 1,
`[-m]`	the flag used for multi robot exploration with interprocess communication,
`ID`	defines the required positional argument of the robot's ID, which has to start from 0

   or run

   ```
   $ python MultiExplorer.py
   ```

   for launching multi robot exploration with centralized control.

5 GUI description

After the robot is initialised, its exploration process is visualised on two graphs. The first graph represents the occupancy grid map constructed from data received from the laser sensor. The second graph demostrates the same occupancy map, but with enlarged obstacles used for continous obstacle avoidance.

- blue dot - the current robot's position,
- red dots - the frontiers detected before path planning,
- red line - the planned path to the goal frontier,
- green crosses - points on the simplified path.

6 Implemented tasks

Task	Feature	Expected points
M1	Map static size	1
F1	Free-edge cluster frontiers	1
F2	Multiple-representative free-edge cluster frontiers	2
F3	Mutual information at frontiers	7
P1	Planning towards the closest frontier	1
P2	Planning with highest utility	2
P3	Planning considering visitation of all goals	4
A1	Multi-robot simple task-allocation	5
A3	Multi-robot without interprocess communication	0
A4	Multi-robot with interprocess communication	4
report		1
code		2
Total		30