B4M36UIR & BE4M36UIR

Task04 - Incremental Path Planning (D* Lite)

The main task is to implement the D* lite dynamic replanning algorithm.

Deadline	3. November 2018, 23:59 PST
Points	5
Label in BRUTE	Task04
Files to submit	archive with `robot` , `gridmap` and `gridplanner` directory
	Minimal content of the archive: `robot/Robot.py` , `gridmap/GridMapy.py` , `gridplanner/GridPlanner.py` , `gridplanner/DStarLitePlanner.py`

Resources Task04 resource package

Assignment

Implement the D* lite incremental path planning algorithm according to the description and pseudocode presented in the Lecture 4. Grid and Graph-based path planning methods.

In file `DStarLitePlanner.py` implement the 4-neighborhood D * lite algorithm. You may use the provided code and implement only the functions `calculate_key` , `compute_shortest_path` and `update_vertex` and complete the `plan` function according to the pseudocode described in Lecture 4. Grid and Graph-based path planning methods.

The implementation requirements are as follows:

1. The `DStarLitePlanner` class is inherited from the `GridPlanner` class and overrides the `plan(gridmap, start, goal)` method.
2. The class keeps its own representation of the environment in a form of map that is initialized to proper dimensions during the first call of the `plan` method based on the dimensions of the passed `gridmap` . The dimensions of the map are fixed and won't change.
3. The `DStarLitePlanner` class provides access to its environment representation through the `getRHSvalue` and `getGvalue` methods, that provide the respective values as an array that are used for proper annotation of the visualized grid map (if called without parameters).
4. The `plan` method is called whenever the path is obstructed, which forces an update of the occupancy grid map that is passed to the `plan` method as the `gridmap` parameter. The planner has to figure out what has been changed in the map and replan accordingly, i.e., it has to figure out the coordinates of the updated cells, update all the affected rhs and g values in its neighborhoods (4-neighborhood is used for the algorithm) and recompute the shortest path
5. The `calculate_key` function uses manhattan distance as the heuristics

Evaluation

The simplified evaluation script for sole testing of the `DStarLitePLanner.py` implementation is following

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys
import math
import time
import numpy as np
import matplotlib.pyplot as plt
from collections import deque

sys.path.append('gridmap')
sys.path.append('gridplanner')

import GridMap as gmap
import DStarLitePlanner as dplanner

def plot_d_star_map(gridmap, rhs, g):
    """
    method to plot the annotated d star lite map with rhs and g values
    """
    plt.clf()
    gmap.plot_map(gridmap)
    ax = plt.gca()

    for i in range(0, gridmap.width):
        for j in range(0, gridmap.height):
            ax.annotate(g[i][j], xy=(i-0.2, j-0.1), color="red")
            ax.annotate(rhs[i][j], xy=(i-0.2, j+0.3), color="blue")

if __name__=="__main__":

    #define planning problems:
    scenarios = [([(1, 0), (1, 1), (1, 2), (1, 3), (2, 3), (3, 3), (4, 3), (4, 2), (4, 1), (5, 1), (6, 1), (6, 2), (6, 3), (6, 4), (6
                 ([(1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (3, 9), (3, 8), (3, 7), (3, 6), (3, 5), (3
                 ([(1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (7, 1), (8, 1), (6, 1), (8, 2), (8, 3), (8, 4), (7, 4), (7, 5), (7, 6), (9

    start = (4, 4)
```

```
goal = (9, 5)

plt.ion()

#fetch individual scenarios
for scenario in scenarios:
    robot_position = start

    #instantiate the map
    gridmap = gmap.GridMap(10, 10, 1)

    #instantiate the d star lite planner
    planner = dplanner.DStarLitePlanner()

    #plan the route from start to goal
    path = planner.plan(gridmap, gridmap.world_to_map(robot_position), gridmap.world_to_map(goal))

    if path == None:
        print("Destination unreachable")
        break
    else:
        path = deque(path)

    #plot the resulting map and the path
    plot_d_star_map(gridmap, planner.getRHSvalues(), planner.getGvalues())
    gmap.plot_path(path)
    plt.pause(0.2)

    #locomote
    while robot_position != goal:
        #execute the step and update the map
        next_step = path.popleft()

        if next_step in scenario: #there is an obstacle in the way
            #update the gridmap - the cell is occupied
            gridmap.set_cell_p(next_step, 0.95)
            #replan
            path = planner.plan(gridmap, gridmap.world_to_map(robot_position), gridmap.world_to_map(goal))

            if path == None:
                print("Destination unreachable")
                break
            else:
                path = deque(path)

        else: #if not, take the next step
            #update the gridmap - the cell is free
            gridmap.set_cell_p(next_step, 0.05)

            #take the next step
            robot_position = next_step

        #plot the resulting map and the path
        plot_d_star_map(gridmap, planner.getRHSvalues(), planner.getGvalues())
        gmap.plot_path(path)
        plt.pause(0.2)
```