B4M36UIR & BE4M36UIR

Task03 - Grid-based path planning

The main task is to implement a simple grid-based planning approach.

Deadline	27. October 2018, 23:59 PST
Points	3
Label in BRUTE	Task03
Files to submit	archive with `gridmap` directory and `gridplanner` directory
	Minimal content of the archive: `gridmap/GridMap.py` , `gridplanner/GridPlanner.py`
Resources	Task03 resource package

Assignment

Implement the following basic steps of the grid-based navigation pipeline:

1. In file `GridMap.py` implement the obstacle growing (method `grow_obstacles`) to take into account the robot embodiment
2. In file `Planner.py` implement a grid-based planning method (method `plan`) to find a path between the start and goal position
3. In file `Planner.py` implement the trajectory smoothing method (method `simplify_path`) which purpose is to provide smoother path for the path following controller that drives the real robot

Approach

1. **Obstacle growing** - can be achieved by different means, typically methods of mathematical morphology are being used. In particular, binary dilation of the obstacles for a predefined `distance` which is set by user to take into account the robot embodiment. Typically, half of the size of the being used as the `distance` . Further, methods based on distance transform can be also used to grow the obstacles or as a heuristic function for the planner to stay away from walls.
2. **Planning** - grid-based path planning takes the gridmap, starting position(`start`) and goal position(`goal`) on the input and provide a list of cell coordinates on the output. If the path is not found, the planner returns `None`
3. **Path simplification** - path simplification is usually done by excluding navigation points from the path, that are not necessary in a sense, that the robot does not have to visit them precisely. Typical approach to trajectory smoothing is to connect the neighboring segments one by one using straight-line segments (using Bresenham line algorithm) up to the point where the straight-line segment collide with an obstacle (grown obsttacle) and then follow with another straight-line segment.

Evaluation

The code can be evaluated using the following script

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

import sys
import math
import time
import numpy as np
import matplotlib.pyplot as plt

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

import Robot as rob
import GridMap as gmap
import GridPlanner as gplanner

PLOT_ENABLE = True

if __name__=="__main__":

    #define planning problems:
    #  map file
    #  map scale [m] (how big is a one map cell in comparison to the real world)
    #  start position [m]
    #  goal position [m]
    #  execute flag
    scenarios = [("maps/maze01.csv", 0.1, (8.5, 8.5), (1, 1), False),
                 ("maps/maze02.csv", 0.1, (8.5, 8.5), (5.6, 3), False),
                 ("maps/maze02.csv", 0.1, (8.5, 8.5), (5.8, 2.5), False),
                 ("maps/maze03.csv", 0.1, (19.5, 23.0), (6.2, 26.4), False),
                 ("maps/maze04.csv", 0.1, (51.6, 53.3), (19.2, 12.4), False),
                 ("maps/maze05.csv", 0.1, (78.8, 79.2), (7.0, 14.6), False),
                 ("maps/maze02.csv", 0.1, (8.5, 8.5), (5.6, 3), True)]

    #fetch individual scenarios
    for scenario in scenarios:
        mapfile = scenario[0] #the name of the map
        scale = scenario[1]
        start = scenario[2]   #start point
        goal = scenario[3]    #goal point
```

```
        execution_flag = scenario[4] #execute the trajectory in vrep

        #instantiate the map
        gridmap = gmap.GridMap()

        #load map from file
        gridmap.load_map(mapfile, 0.1)

        #plot the map with the start and goal positions
        if PLOT_ENABLE:
            gmap.plot_map(gridmap)
            gmap.plot_path(gridmap.world_to_map([start, goal]))
            plt.show()

            #show the free/occupied space
            gmap.plot_map(gridmap, clf=True, data='free')
            plt.show()

        #blow the obstacles to avoid collisions
        gridmap.grow_obstacles(0.4)

        #show the map after obstacle blowing
        if PLOT_ENABLE:
            gmap.plot_map(gridmap, clf=True, data='free')
            plt.show()

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

        if path == None:
            print("Destination unreachable")
            continue

        #show the planned path
        if PLOT_ENABLE:
            gmap.plot_map(gridmap)
            gmap.plot_path(path)
            plt.show()

        #simplify the path
        path_s = planner.simplify_path(gridmap, path)

        #show the simplified path
        if PLOT_ENABLE:
            gmap.plot_map(gridmap)
            gmap.plot_path(path)
            gmap.plot_path(path_s, color='blue')
            plt.show()

        if execution_flag:
            #instantiate the robot
            robot = rob.Robot()

            #execute the path
            for waypoint in path_s:
                #navigate the robot towards the target
                status1 = robot.goto(gridmap.map_to_world(waypoint))

                #check for the execution problems
                if not status1:
                    print("The robot has collided en route")
                    break
```

courses/b4m36uir/hw/task03.txt · Last modified: 2018/10/15 13:41 by cizekpe6