B4M36UIR & BE4M36UIR

Task02b - Map building

The main task is to implement a function that will fuse the laser scan data into the occupancy grid map.

Deadline	27. October 2018, 23:59 PST
Points	2
Label in BRUTE	Task02b
Files to submit	archive with `GridMap.py` file
Resources	Task03 resource package
	Blocks V-REP scene

Assignment

In class `GridMap.py` implement the `fuse_laser_scan` function. The purpose of the function is to fuse new data into the occupancy grid map as the robot traverses the environment. The occupancy grid map is initialized to the size of the VREP scene (10×10 m). The laser scan measurements shall be fused to the grid using the Bayesian update described in Lab03 - Grid-based Path Planning.

The obstacle sensing is achieved using the simulated laser range finder through the `RobotHAL` interface through the `self.robot` object.

```
scan_x, scan_y = self.robot.get_laser_scan()
```

The laser scan is in relative coordinates w.r.t. the hexapod base where the x axis correspond to the heading of the robot and y axis is perpendicular to the robot heading.

The `fuse_laser_scan` function has a following prescription

```
def fuse_laser_scan(self, pose, scan_x, scan_y):
    """
    Method to fuse the laser scanner data into the map

    Parameters
    ----------
    pose: (float, float, float)
        pose of the robot (x, y, orientation)
    scan_x: list(float)
    scan_y: list(float)
        relative x and y coordinates of the points in laser scan w.r.t. the current robot pose
    """
```

Approach

The recommended approach for the occupancy map building using Bayesian approach is described in Lab03 - Grid-based Path Planning.

The `GridMap` represents a probabilistic representation of the word. In particular the variable "self.grid" holds the probabilities of individual states to be occupied `self.grid['p']` and the derived binary information about the passability of the given cell `self.grid['free']` . Access and update of the probabilities can be done using functions `self.get_cell_p(coord)` and `self.set_cell_p(coord)` that will also automatically update the passability information, when the probability changes.

Hence the correct approach to the fusion of the laser scan data into the occupancy grid map is as follows

1. Compensate for the heading of the robot by rotating the scanned points
2. Compensate for the position of the robot by offsetting the scanned points to the robot's coordinates
3. Raytrace individual scanned points (preferably using Bresenham line algorithm, e.g., `bresenham_line(start, goal)`) which will give you coordinates of the cells which occupancy probability should be updated
4. Update the occupancy grid using the Bayesian update and the simplified laser scan sensor model with $\epsilon = 1$, i.e., using the Bresenham line algorithm trace all the points lying between the position of the robot and the reflection point $(d \in [0, r))$ and update their probability (being free) and only the reflection point $(d = r)$ being occupied

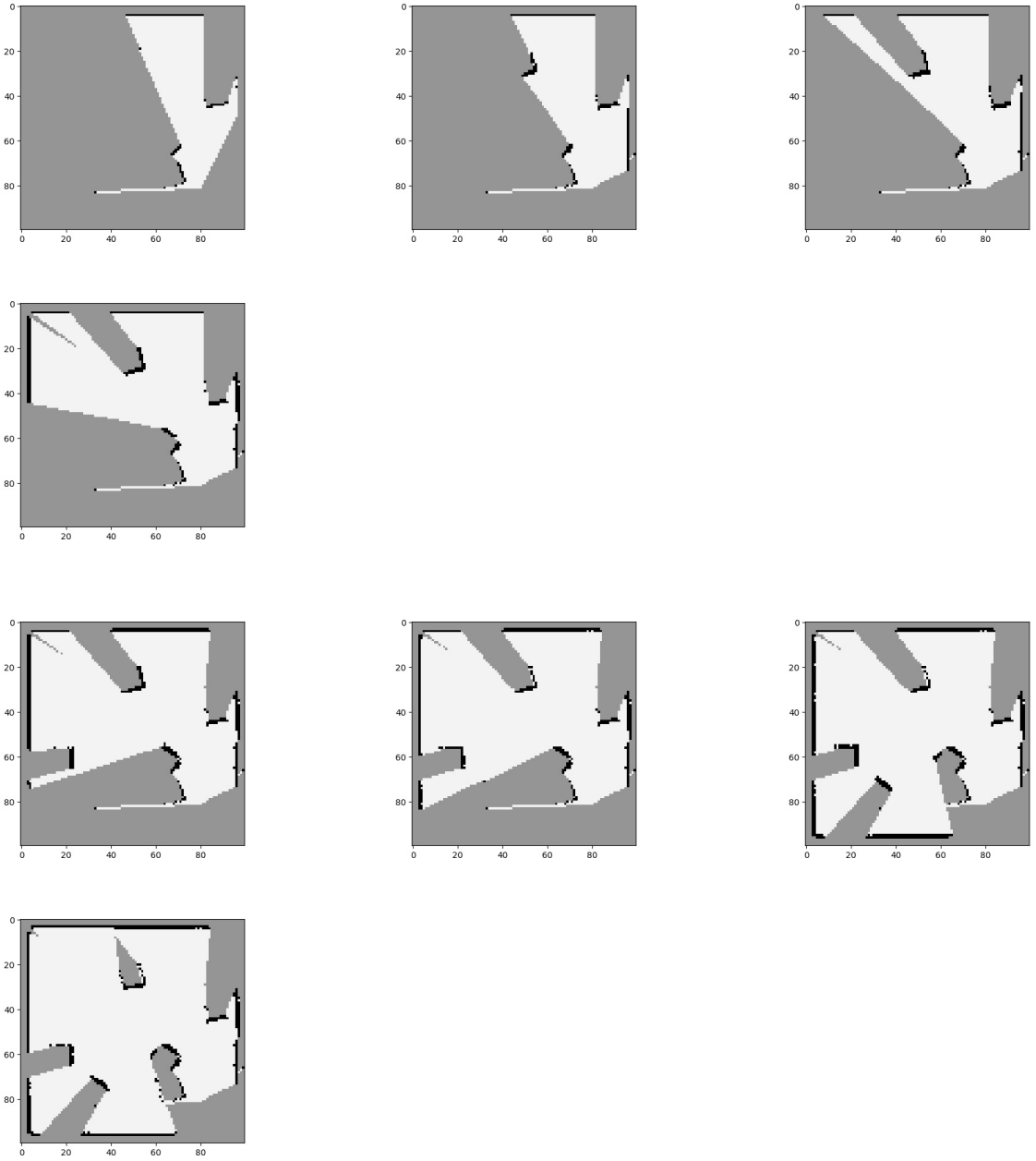
Note, when using the sensory model as it is described in Lab03 - Grid-based Path Planning, once the grid cell probability is set to 0, it stays 0, hence, all the nice features of the probabilistic mapping vanishes. Therefore it is recommended to use a different sensory model with:

$$S^z_{occupied} = \begin{cases} 0.9 & \text{for} \quad d = r \\ 0 & \text{otherwise} \end{cases},$$

$$S^z_{free} = \begin{cases} 0.9 & \text{for} \quad d \in [0, r) \\ 0 & \text{otherwise} \end{cases},$$

To further simplify and speed-up the verification of the `fuse_laser_scan` function, it is recommended to construct an evaluation dataset by recording the data necessary as the input of the function during a single simulated run and then only read a process these data.

Following figures visualize the possible sequence of map building given the following evaluation script.



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
import threading as thread
from collections import deque

sys.path.append('robot')
sys.path.append('gridmap')
import Robot as rob
import GridMap as gmap

class task2b_eval:
```

```
    def __init__(self):
        #instantiate the robot
        self.robot = rob.Robot()

        #instantiate the map
        self.gridmap = gmap.GridMap(100, 100, 0.1)

        #route
        self.route = deque([(6.0, 8.5, None), (4.0, 6.0, None), (5.0, 5.0, None), (1.5, 1.5, None)])
        self.route_lock = thread.Lock()

        #navigation thread stopping
        self.navigation_stop = False

    def fetch_navigation_point(self):
        """
        Method for getting the next navigation point

        Returns
        -------
        (float, float)
            Next navigation point
        """
        coord = None
        self.route_lock.acquire()
        try:
            coord = self.route.popleft()
        except IndexError:
            coord = None
        self.route_lock.release()
        return coord

    def navigation(self):
        """
        navigation function that executes the trajectory point-by-point
        """
        print("starting navigation")
        while not self.navigation_stop:

            pos = self.fetch_navigation_point()

            if pos == None:
                print("No further points to navigate")
                self.navigation_stop = True
                break

            print("Navigation point " + str(pos))
            #navigate the robot towards the target

            status1 = self.robot.goto(pos[0:2],pos[2])


    def eval(self):
        """
        Evaluation function
        """
        try:
            nav_t = thread.Thread(target=self.navigation)
        except:
            print("Error: unable to start navigation thread")
            sys.exit(1)
        nav_t.start()

        plt.ion()

        while not self.navigation_stop:
```

```
            #get the robot pose
            pose = self.robot.get_pose()

            #get the laser scan data
            scan_x, scan_y = self.robot.robot.get_laser_scan()

            #fuse the data into the map
            print("fusing new data to the occupancy grid map")
            self.gridmap.fuse_laser_scan(pose, scan_x, scan_y)

            #plot the gridmap
            gmap.plot_map(self.gridmap)
            plt.pause(1)

        nav_t.join()

if __name__ == "__main__":
    task = task2b_eval()
    task.eval()
```

courses/b4m36uir/hw/task02b.txt · Last modified: 2018/10/25 09:15 by cizekpe6