B4M36UIR & BE4M36UIR

Task02a - Reactive obstacle avoidance

The main task is to implement a function that will steer the robot towards a given goal while reactively avoiding obstacles.

Deadline	20. October 2018, 23:59 PST
Points	3
Label in BRUTE	Task02a
Files to submit	archive with `Robot.py` file and optionally `RobotConst.py` file
Resources	Task02a resource package
	Blocks V-REP scene

Assignment

In class `Robot.py` implement the `goto_reactive(coord)` function. The purpose of the function is to navigate the robot towards the goal given by coordinates $coord = (x_{goal}, y_{goal})$ while reactively avoiding collisions. The steering of the locomotion is achieved by the `goto_reactive` function by setting the differential steering command of the CPG locomotion controller (v_{left}, v_{right}). The function returns `True` when the robot is at the goal coordinates and `False` if it has collided with an obstacle en route. In addition to the Task01 - Open-loop locomotion control the function `goto_reactive` implements the reactive collision avoidance.

Information about the current position (x, y), orientation ϕ and collision state is provided by the `RobotHAL` interface through the `self.robot` object. The respective functions are

```
#get position of the robot as a tuple (float, float)
self.robot.get_robot_position()
#get orientation of the robot as float
self.robot.get_robot_orientation()
#get collision state of the robot as bool
self.robot.get_robot_collision()
```

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 `goto_reactive` function has a following prescription

```
def goto_reactive(self, coord):
    """
    Navigate the robot towards the target with reactive obstacle avoidance

    Parameters
    ----------
    coord: (float, float)
        coordinates of the robot goal

    Returns
    -------
    bool
        True if the destination has been reached, False if the robot has collided
    """
```

Approach

The recommended approach for the reactive obstacle avoidance uses simple AI cognitive model of Braitenberg vehicles described in Lab02 - Exteroceptive sensing, Mapping and Reactive-based Obstacle Avoidance.

The direct sensory-motor mapping can be achieved using the following continuous navigation function (pseudocode).

```
while not goal_reached:
    dphi = the difference between the current heading and the heading towards the target
    scan_left = closest obstacle to the left of the robot
    scan_right = closest obstacle to the right of the robot
    v_left = 1/scan_left*C_AVOID_SPEED - dphi*C_TURNING_SPEED + BASE_SPEED
    v_right = 1/scan_right*C_AVOID_SPEED + dphi*C_TURNING_SPEED + BASE_SPEED
```

Where `C_AVOID_SPEED` is a constant that defines the robot fear of the obstacle, `C_TURNING_SPEED` is a constant that defines the aggression with which the robot will turn towards the desired heading and `BASE_SPEED` is the default speed of the robot when it is heading directly towards the target.

Note, that in a physical world it is impossible to get to a precise specific coordinates, therefore it is sufficient to navigate "close enough". The sufficient distance should be comparable in size to the actual robot. In our case, this distance is given as the navigation parameter `DISTANCE_THLD = 0.1 #m` defined in the `RobotConst.py` file.

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
sys.path.append('robot')

import Robot as rob

DISTANCE_THLD = 0.15 #m

def check(pose_set, pose_real):
    """
    Function to check that the navigation towards the goal has been successfull

    Parameters
    ----------
    pose_set: (float, float, float)
        desired coordinates to reach (x,y,phi)
    pose_real: (float, float, float)
        real coordinates of the robot (x,y,phi)

    Returns
    -------
    bool
        True if the robot real position is in delta neighborhood of the desired position, False otherwise
    """
    ret = True
    (x1, y1, phi1) = pose_set
    (x2, y2, phi2) = pose_real
    dist = (x1 - x2)**2 + (y1-y2)**2
    #check the distance to the target
    if math.sqrt(dist) > DISTANCE_THLD:
        ret = False
    #check the final heading
    if not phi1 == None:
        dphi = phi1 - phi2
        dphi = (dphi + math.pi) % (2*math.pi) - math.pi
        if dphi > ORIENTATION_THLD:
            ret = False

    return ret

if __name__=="__main__":
    #navigation points
    route = [(1.5, 1.5, None), (5, 5, None)]

    #instantiate the robot
    robot = rob.Robot()

    #navigate
    for waypoint in route:
        pos_des = waypoint[0:2]

        print("Navigation point " + str(pos_des))

        #navigate the robot towards the target
        status1 = robot.goto_reactive(pos_des)

        #get robot real position
        pose_real = robot.get_pose()
        #check that the robot reach the destination
        status2 = check(waypoint, pose_real)

        #print the result
        print(status1, status2)
```

The expected output on `blocks.ttt` map is

```
Connected to remote API server
Robot ready
Navigation point (1.5, 1.5) None
True True
Navigation point (5, 5) None
True True
```

courses/b4m36uir/hw/task02a.txt · Last modified: 2018/10/19 13:11 by cizekpe6