Homework Assignment 2:
Differential Drive Navigation

Overview

In this assignment, you will implement simple algorithms to direct your robot to a goal pose and avoid obstacles.

I have included my reference implementation of Homework 1 for you to build off of. You can also re-use your own code from Homework 1, if you prefer.

RobotLib/IO.py has been updated with a small bug fix. The new Sparki Firmware v3 gives the option of communicating over USB instead of Bluetooth – change the MySerial definition to be Serial instead of Serial1 at the top of the file and upload the firmware. You will need to provide the correct port to your Python script for USB instead of Bluetooth if you want to try that option.

Homework submission notes

Please mark your team number in the filenames of the files you submit – at least in the filename of the .zip file (if submitting a single .zip) or the .pdf write-up file (if submitting individual files).

Coding style requirements

I am adding some coding style requirements to help me grade your homework more easily. This is good programming practice as well.

Each function must be documented with its purpose, arguments, and outputs, like this:

```
def get_motor_speeds(lin_vel,ang_vel):
     """Calculates Sparki motor speeds from linear and
angular velocities.
     Arguments:
          lin_vel: linear velocity in cm/s
          ang_vel: angular velocity in rad/s
     Returns:
          Left motor speed, left motor direction, right
motor speed, and right motor direction
     """
     # calculations go here
```

```
return left_motor_speed, left_motor_dir, ...
```

Variables should have understandable names and comments documenting their purpose. Classes and significant pieces of code should be commented as well.

Include a README file which explains how to run your code.

Assignment components

1. Implement smooth navigation to a goal point using PID loops. (30 points)

(Create a copy of `template.py` called `pid_nav.py` for this part.)

i. When the mouse is clicked, set a goal location for the robot.

   Use the map-to-robot transformation matrix to project the mouse point to the robot's reference frame.

ii. At each time step, calculate the distance to the goal and the angle to the goal.

   If (x,y) is the goal point in the robot's reference frame, then you can use `math.atan2(y,x)` to calculate the angle to the goal.

iii. Calculate the current linear velocity and angular velocities as follows:

   linear velocity = K_linear * (distance to goal)
   angular velocity = K_angular * (angle to goal)

   This creates a P loop (PID loop with no I or D) for each velocity.

   You can choose K_linear and K_angular as you like.

   When the distance to the goal drops below a threshold, set both velocities to zero.

2. Implement "holonomic" navigation using only pure rotation and translation. (20 points)

(Create a copy of `template.py` called `holo_nav.py` for this part.)

As in the last part, you will set a goal location using the mouse and calculate the distance and angle to the goal.

Implement the following control law:

i. If the angular error is above a threshold, turn toward the goal point (without moving forward).
ii. If the angular error is below a threshold, but the distance is above a threshold, move forward.

3. Add obstacle avoidance to the PID control algorithm. (20 points)

(Create a copy of `pid_nav.py` called `bug.py` for this part.)

i. The provided `OccupancyMap` class maintains an occupancy map that can be used for simulated obstacle detection. The class can read an image from file to initialize the map.

   In simulation mode, your code will simulate the rangefinder reading according to the current transformation matrices and the occupancy map.

   Complete the `OccupancyMap.get_first_hit()` function. This function determines where the rangefinder would hit given the current sonar-to-map transformation. The algorithm is as follows:
   a. Test a range of distances in one centimeter increments, from 1 to the diagonal length of the map (the maximum possible distance).
   b. Use the sonar-to-map transformation to get the location of the possible hit point in the map.
   c. If the location is inside the map and the location is occupied, return that distance.
   d. If the loop completes without finding any hit points, return 0.

ii. At each update cycle, get the simulated or actual rangefinder distance (depending on whether you are in simulation mode or not). Check if the rangefinder distance is valid and below a threshold. If it is, override the angular velocity PID loop and set the angular velocity so that the robot turns left.

4. Evaluate and improve your robot navigation system. (30 points)

In this step, you will evaluate your system and present results in a write-up document. Submit the write-up as a Word doc or PDF.

i. Test different PID settings and thresholds for the PID navigation algorithm. Describe the interactions between the linear and angular PID coefficients and the distance-to-goal threshold.

ii. Test the obstacle avoidance algorithm in simulation and with the real robot (if working). How does it behave? Does it avoid obstacles and reach the

goal? Why or why not?

iii. Think of a way to improve the obstacle avoidance algorithm and implement and test it. Describe your proposed improvement and whether it succeeded.