Week 7 Lab Notes

A. Closed-loop motion control of the Robot

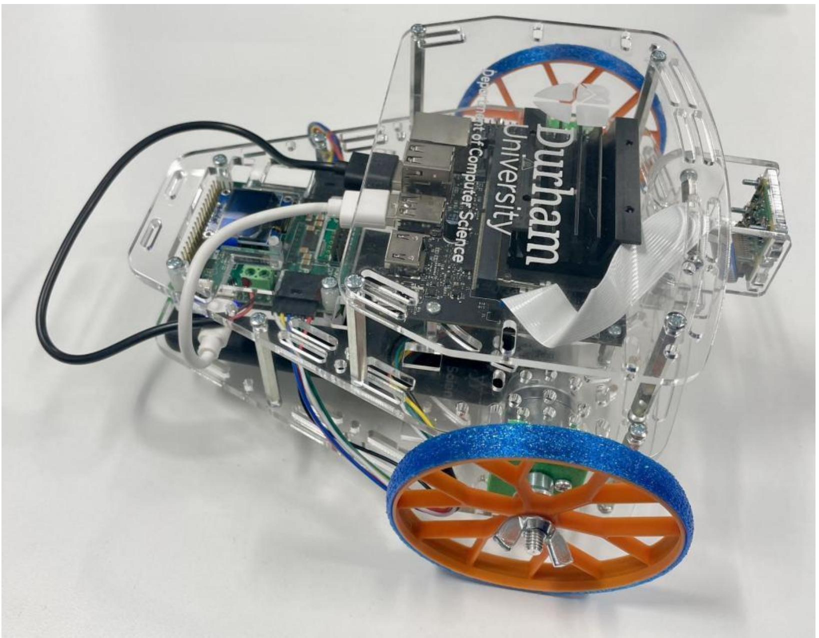
Objective

The purpose of this lab is to understand how to implement a closed-loop motion controller for mobile robot.

Learning outcomes

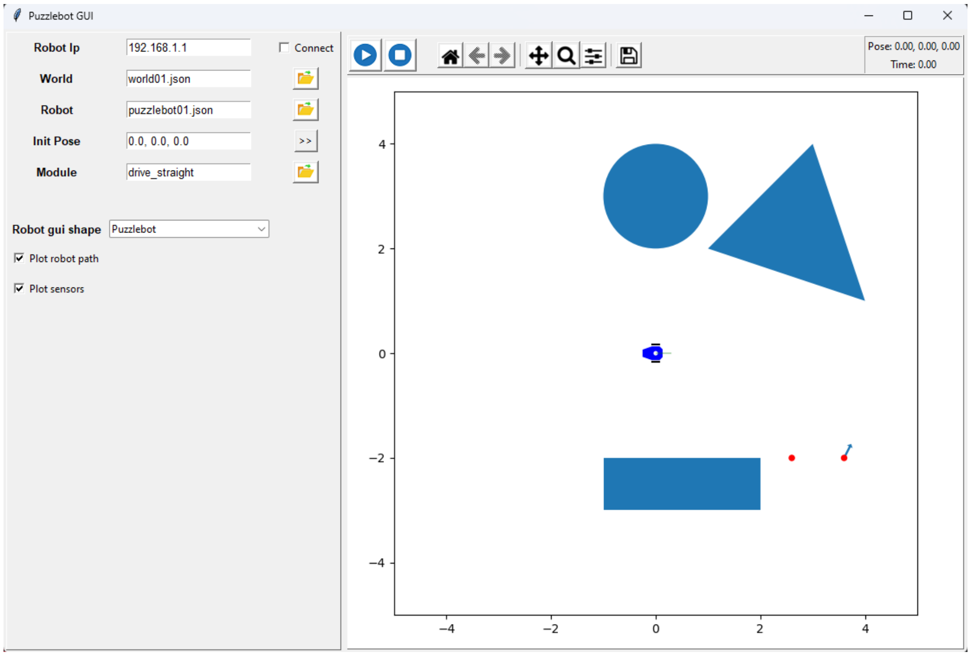
At the end of this exercise, you should be able to:

- Implement a Proportional Controller for a two-wheel mobile robot and let the robot reach a desired goal position.
- Assess the performance of the robot under the motion-based closed-loop control design.
- Observe and analyse the performance when the robot performing a continuous task.



Task 1: Preparation

- Download the ‘`driveToGoal.py`’ `file` from Ultra, and put it into the ‘`my_examples`’ folder.
- Open the GUI by running puzz_gui.py



Robot Ip 192.168.1.1 ☐ Connect

World world01.json

Robot puzzlebot01.json

Init Pose 0.0, 0.0, 0.0 >>

Module drive_straight

Robot gui shape Puzzlebot

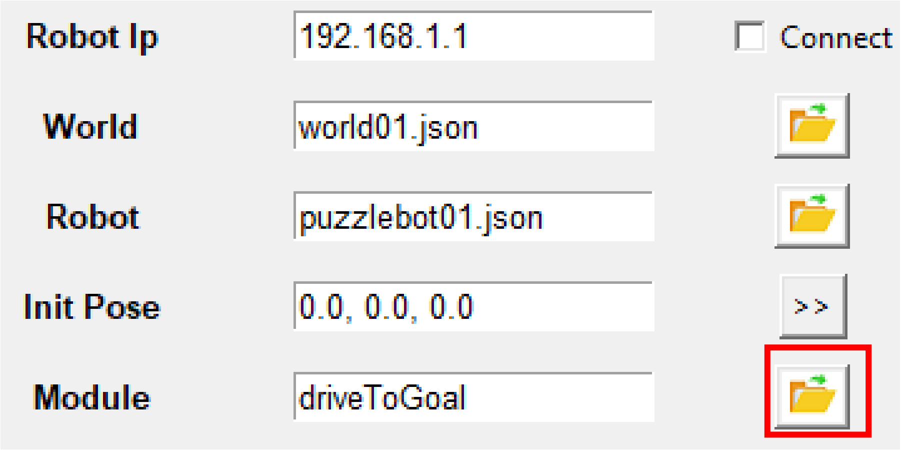
☑ Plot robot path

☑ Plot sensors

Pose: 0.00, 0.00, 0.00

Time: 0.00

- Select ‘**Module**’ with ‘`driveToGoal.py`’. Note, there is no control program at this file.



Robot Ip 192.168.1.1 ☐ Connect

World world01.json

Robot puzzlebot01.json

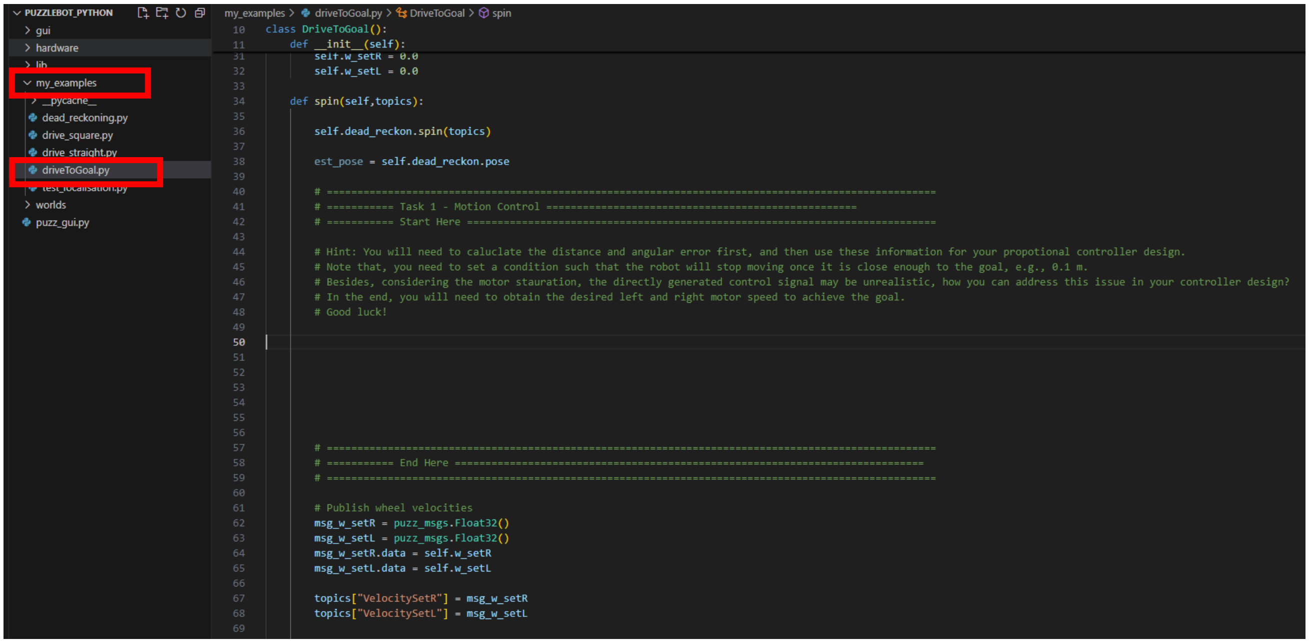
Init Pose 0.0, 0.0, 0.0 >>

Module driveToGoal

Task 2: Motion Control

Implement a control algorithm to move the robot from its current position to an arbitrary goal point within the boundaries of the simulated world, following the next steps:

- Open '`driveToGoal.py`' file.
- Set the goal point in line 16 and line 17, e.g., $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$.
- Write your code for Motion Control in the allocated section for this task



```
10  class DriveToGoal():
11      def __init__(self):
31          self.w_setR = 0.0
32          self.w_setL = 0.0
33
34      def spin(self,topics):
35
36          self.dead_reckon.spin(topics)
37
38          est_pose = self.dead_reckon.pose
39
40          # ====================================================================================================
41          # ========== Task 1 - Motion Control ===================================================
42          # ========== Start Here ============================================================================
43
44          # Hint: You will need to caluclate the distance and angular error first, and then use these information for your propotional controller design.
45          # Note that, you need to set a condition such that the robot will stop moving once it is close enough to the goal, e.g., 0.1 m.
46          # Besides, considering the motor stauration, the directly generated control signal may be unrealistic, how you can address this issue in your controller design?
47          # In the end, you will need to obtain the desired left and right motor speed to achieve the goal.
48          # Good luck!
49
50
51
52
53
54
55
56
57          # ====================================================================================================
58          # ========== End Here ============================================================================
59          # ====================================================================================================
60
61          # Publish wheel velocities
62          msg_w_setR = puzz_msgs.Float32()
63          msg_w_setL = puzz_msgs.Float32()
64          msg_w_setR.data = self.w_setR
65          msg_w_setL.data = self.w_setL
66
67          topics["VelocitySetR"] = msg_w_setR
68          topics["VelocitySetL"] = msg_w_setL
69
```

The following parameters are used in the program.

Parameter	Notation	Description
est_pose	$\boldsymbol{\mu}_k$	Robot pose mean (3x1) $[x\ y\ \theta]^T$ where $x[m]$, $y[m]$ and $\theta[rad]$
self.target_x self.target_y	$\mathbf{X}_g$	Goal point (2x1) $[x\ y]^T$
self.w_setR	ω_r	Right motor speed set point. $[rad/s]$
self.w_setL	ω_l	Left motor speed set point. $[rad/s\ ]$
self.w_max	ω_{max}	Maximum angular speed of both the right and left wheels $[rad/s]$.
self.R	r	Radius of the wheels (0.05 $[m]$)
self.L	l	Robot wheel base (1x1) (0.09$[m]$)
topics["IsDone"] = True	…	A command to stop the simulator.

Note that , `self.w_setR` and `self.w_setL` are the parameters to be updated by your code

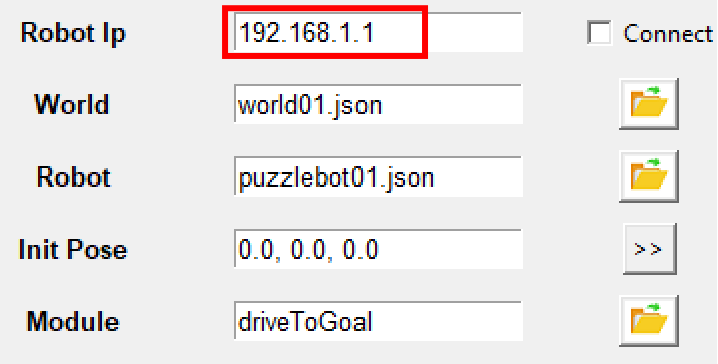
- Run your code using GUI by clicking the icon ▶ .
- Test your algorithm for different scenarios with different goal points.

Task 3: Real-Robot Experiments

Apply the motion control algorithm developed in Task 2 to a real robot, Puzzlebot.

You should follow the next steps to apply your control strategy to the robot:

- Check the IP address in the GUI and the robot IP address that is shown in the screen of the real robot. If they are different, change the IP address in the GUI to be the same as the one shown on the robot screen.



Robot Ip: 192.168.1.1 □ Connect

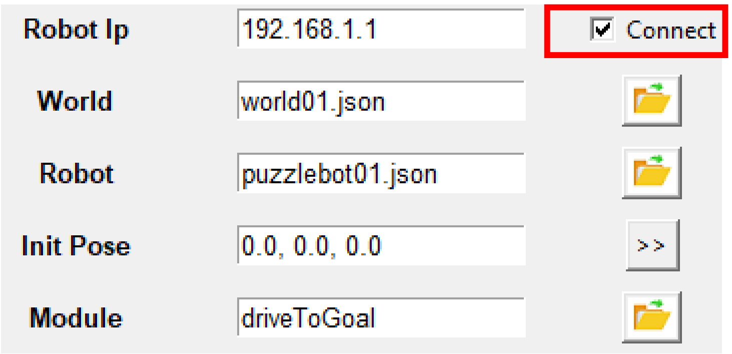
World: world01.json

Robot: puzzlebot01.json

Init Pose: 0.0, 0.0, 0.0

Module: driveToGoal

- Click the checkbox 'Connect'



Robot Ip: 192.168.1.1 ☑ Connect

World: world01.json

Robot: puzzlebot01.json

Init Pose: 0.0, 0.0, 0.0

Module: driveToGoal

- Run your code using GUI by clicking the icon ▶.

[End of Lab]