Institute for Aerospace Studies
UNIVERSITY OF TORONTO

LABORATORY III

Path-Following Robot

ROB301 Introduction to Robotics
Fall 2020

1 Introduction

This is the third laboratory exercise of ROB301 Introduction to Robotics. The focus for the next two sessions is a simple path-following robot. We will investigate introductory control theory for mobile robotic applications. For a more in-depth analysis of control theory, refer to AER372 (next term) or ECE557 (next year, if you decide to take it). For a more in-depth analysis of mobile robotics and path-planning, refer to ROB521 (next year). This laboratory exercise will require the submission of a report.

2 Objective

The objective of this exercise is to introduce the foundational concepts of control theory and its practical use in mobile robotics using the Turtlebot 3 Waffle Pi, more specifically, its virtual cousin. You will be required

- *To implement a controller to execute path following in stages beginning with* (i) *a bang-bang controller, followed by* (ii) *a P controller, then* (iii) *a PI controller and finally* (iv) *a PID controller*
- *To design and test a controller to follow a closed-circuit race course.*

More on the race course shortly.

3 Lab Deliverables

To demonstrate functionality, each team is required to submit a one-page report. In the report, you need to describe how your robot completed all the tasks, and what was your approach. If you were unable to complete a task, explain why that was the case in your report.

You are free to work on §§5.1 to 5.4 during the first lab session, so you can get familiar with different controllers. Each team is required to submit one rosbag file for each task along with the report. Please refer to the bold text marked by 'Lab Deliverable' throughout the document for instructions on how to start rosbag recordings.

Section 5.5 is a competition on a customized race track. Please note that the second lab session will be mandatory to attend via Bb Collaborate, as the race will take place then. You will not be able to test your robot on the race track prior to the demo, but each team has two attempts to complete the race track and the best time will be recorded.

4 Equipment & Software

As introduced in Lab I and used in Lab II, the platform used for this lab is the TurtleBot 3 Waffle Pi and the simulation environment is based on Gazebo. You will find the starter code in:

`catkin_ws/src/rob301_simulation/nodes/lab03.py`.

As a quick reminder from Lab I, the following commands should be run in separate terminal windows sequentially: (1) run the ROS launch file for lab03, (2) start Gzweb and visualize the graphical interface on your local browser and (3) run your custom `lab03.py` node (you should keep these terminal windows open as they are continuously running the program):

`$ roslaunch rob301_simulation lab_03.launch` [Remote PC]

`$ cd ~ /gzweb && npm start -p {port number}` [Remote PC]

`http://100.92.1.X:808X` [Local PC browser]

`$ rosrun rob301_simulation lab03.py` [Remote PC]

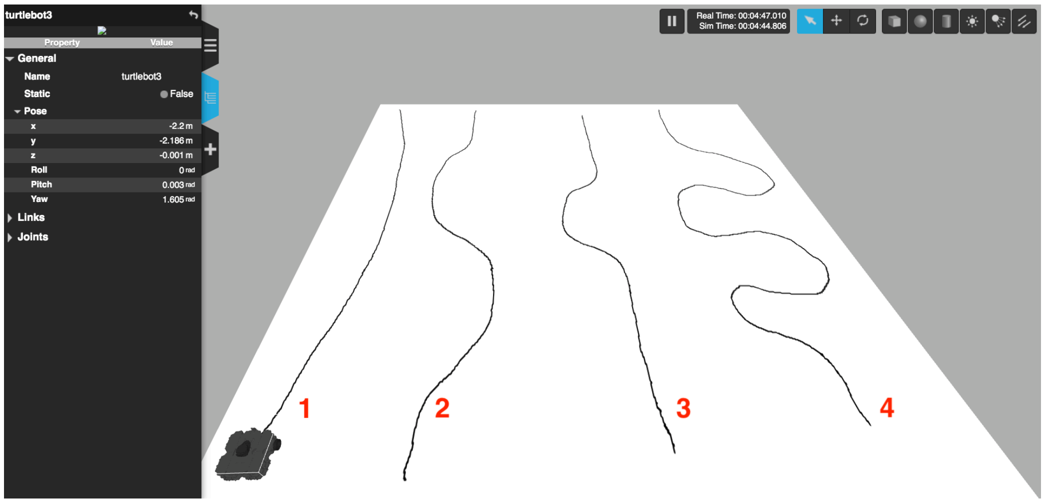
4.1 Gazebo Simulation World

For §§5.1 to 5.4, you will be testing your robot on 4 different practice tracks (Figure 1). **You can perform testings on all four tracks, but you are only required to demonstrate functionality on one of the track in rosbag recordings.** For §§5.1 to 5.3, you can record a successful run on the most difficult track completed by your controller, but you are required to submit a recording on Track 4 only for §5.4.

If you want to modify the initial pose of the robot on Gzweb, you can go to **'scene tree'** $\rightarrow$ **'Models'** $\rightarrow$ **'turtlebot3'** $\rightarrow$ **'General'** $\rightarrow$ **'Pose'**, and edit the values of the following parameters: x, y and Yaw. However, please don't change the values of z, Roll or Pitch. Figure 2 gives an example of defining the initial pose when starting on Track 1.

4.2 Motor Driving

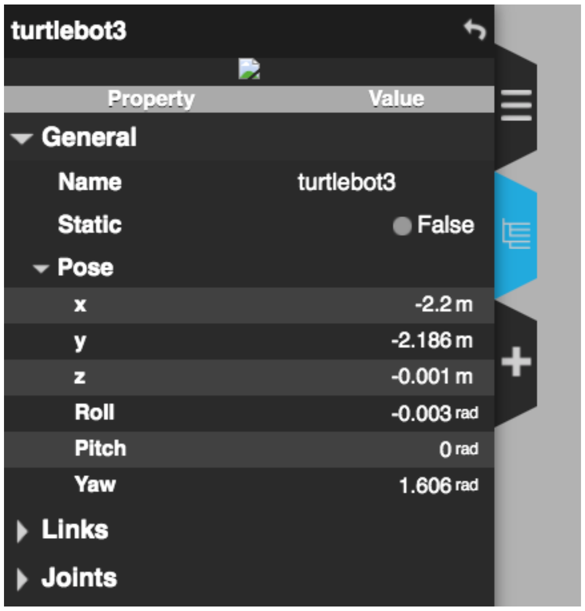
For motor driving command interface, please refer to Lab I `motor.py` file. The general idea is that you would want to modify the linear and angular velocity in the `Twist` message type and create a publisher node to send the message to the motor controller's subscriber node in the Turtlebot bringup package, as described in Lab I.



turtlebot3

Property	Value
General	
Name	turtlebot3
Static	● False
Pose	
x	-2.2 m
y	-2.186 m
z	-0.001 m
Roll	0 rad
Pitch	0.003 rad
Yaw	1.605 rad
Links	
Joints	

Figure 1: Practice Tracks



turtlebot3

Property	Value
General	
Name	turtlebot3
Static	● False
Pose	
x	-2.2 m
y	-2.186 m
z	-0.001 m
Roll	-0.003 rad
Pitch	0 rad
Yaw	1.606 rad
Links	
Joints	

Figure 2: Changing the Initial Pose

4.3 Color Sensing

The TurtleBot robot includes a Rasberry Pi camera. In this lab, the camera will be used as a sensor to distinguish the black tape from the background in the Gazebo simulation world; the robot should follow the black tape line by turning such that the line is in the center of the captured images. We have provided a ROS node that subscribes to the captured grayscale images and publishes the horizontal pixel location of the line to the ROS topic `/line_idx`.

The `line_detector.py` script imports the camera image as a 2D matrix of an arbitrary height h and fixed width of 640 pixels. Based on the color hex code, the color black will return the smallest value while white will return the largest. The script then averages the values of each column of the matrix to obtain a 1×640 array and returns the index of the maximum value in the array; since the tape is the darkest object in the image, the returned index should correspond with the horizontal pixel location of the tape and would be in the middle of the array (320) when the line is centered in the image. Your tracking error is the difference between the center pixel coordinate and the measured pixel coordinate in the current image frame; this error is to be used in your feedback controller.

You do not need to touch the `line_detector.py` file, as this file will be automatically executed while launching `lab_03.launch`. It will publish the location of the black tape with respect to the width of the camera view to `/line_idx`, which is an integer ranging from 0 to 640. You can view the messages that are published to the ROS topic by running the following command:

```
$ rostopic echo line_idx
```

Note that if the robot is directly on top of the black tape, `/line_idx` would output 320.

A Waffle Pi camera tutorial is referenced in §7 for those interested to know more about the Pi camera.

5 Assignment

Four different controllers applicable to the path-following—more specifically here, line-following—problem will be investigated. First, bang-bang control sets the basis of comparison for solution methods. Next, proportional, integral, and derivative gains are introduced to improve the tracking ability of the robot. A final closed-circuit race course will be used as a proving ground for the performance of the designed controller.

5.1 Bang-Bang Control

With no knowledge of control theory, bang-bang control is often the first method considered in stabilizing a system. As the name implies, the intent is to observe the environment for the actual state of the system and move toward the desired state with maximal control commands.

Pseudocode. The following is a general approach to the implementation of bang-bang control:

```
desired = set desired position
begin loop
    actual = measure actual position
    error = desired - actual
    if error < 0 then correction = positive
    elseif error > 0 then correction = negative
    else correction = zero
    drive + correction
end loop
```

Write a program to implement the pseudocode such that the robot simultaneously drives forward and follows the line. Test your robot on any of the practice tracks in the Gazebo simulation. The speed of the robot, in this and all deliverables, must not be less than $0.1\ \mathrm{m\,s^{-1}}$. Note the behavior of the robot and try to make it follow the track as smoothly as possible.

Lab Deliverable 1. **Before you move onto the next task, you are required to demonstrate functionality of your implementation of a bang-bang controller with a forward speed of at least 0.1 $\mathrm{m\,s^{-1}}$. Please record a successful run on the most difficult track completed by your controller using rosbag. You can record by running:**

```
$ rosbag record /cmd_vel /odom /line_idx
```

prior to running `lab03.py`**.**

5.2 Proportional Control

One of the most basic controllers is the proportional-gain controller. The intent of this controller is to observe the actual state of the system and move toward the desired state more smoothly than the bang-bang type. As the error increases, the control command has greater influence.

Pseudocode. The following is a general approach to the implementation of proportional-gain control:

```
desired = set desired position
$k_p$ = set proportional gain
begin loop
    actual = measure actual position
    error = desired - actual
    correction = ($k_p$ × error)
    drive + correction
end loop
```

The value of k_p can be determined in multiple ways. If a mathematical model of the plant is known, then a controller can be designed to stabilize it. Alternatively, when the plant model is not known, too complex or uncertain, an iterative approach to gain selection can be used. With an iterative approach, the proportional gain value is, at first, set very low and tested. It is then increased until the system becomes oscillatory with a constant amplitude. Half of this value of k_p is usually used as the final gain as in accordance with the Ziegler-Nichols method. The general effect of increasing the proportional gain is qualitatively listed in Table 1.

Table 1: Effect of increasing the proportional gain

Parameter	Rise Time	Overshoot	Settling Time	Steady-State Error
k_p	Decrease	Increase	Little Effect	Decrease

Rise time is typically how long it takes the system to go from 0 to 100% of the desired target; *overshoot* is the maximum amount, usually expressed as a percentage, the response exceeds its desired target; *settling time* is how long it takes the response to reach and stay within a specified band around the desired target; and *steady-state error* is the difference between the response and the desired target in the long term.

Lab Deliverable 2. Write a program to implement the pseudocode of a proportional controller. Before you move onto the next task, you are required to demonstrate your implementation of proportional control with a forward speed of at least 0.1 $\mathrm{m\,s^{-1}}$ by recording a successful run on the most difficult track completed by your controller using rosbag. In your report, describe the difference between proportional control and bang-bang control in terms of performance.

5.3 Proportional-Integral Control

To improve the tracking of the system to the desired state, you can add another element to the correction calculation called the integral gain, k_i. This parameter is special in that it gives the system memory. By accumulating the error over all timesteps, an integral gain will pull the response back to the target thus making the steady-state error zero.

Pseudocode. The following is a general approach to the implementation of proportional-integral-gain control:

```
desired = set desired position
integral = initialize to zero
$k_p$ = set proportional gain
$k_i$ = set integral gain
begin loop
    actual = measure actual position
    error = desired - actual
    integral = integral + error
    correction = ($k_p$ × error) + ($k_i$ × integral)
    drive + correction
end loop
```

The value of k_i can also be determined in multiple ways. Usually, the value of k_p is determined first when k_i is temporarily set to zero. Next, the integral gain is slowly increased until the steady-state error is adequately decreased. Too much k_i and the system will become unstable. (One may also follow the Ziegler-Nichols procedure here.) The typical effect of increasing the integral gain is qualitatively listed in Table 2.

Table 2: Effect of increasing the integral gain

Parameter	Rise Time	Overshoot	Settling Time	Steady-State Error
k_i	Decrease	Increase	Increase	Eliminate

Lab Deliverable 3. Write a program to implement the pseudocode of a PI controller. Before you move onto the next task, you are required to demonstrate your implementation of a PI controller with a forward speed of at least 0.1 $\mathrm{m\,s^{-1}}$ by recording a successful run on the most difficult track completed by your controller using rosbag.

5.4 Proportional-Integral-Derivative Control

The final improvement to system tracking is the addition of the derivative gain, k_d. This parameter is special in that it gives the system the ability to predict the future. By comparing the difference between the current error and the error of the previous timestep, a derivative gain will expose the rate of change of the error in order to damp the effect of the other two gains when nearing the target state.

Pseudocode. The following is a general approach to the implementation of proportional-integral-derivative-gain control:

```
desired = set desired position
integral = initialize to zero
derivative = initialize to zero
lasterror = initialize to zero
$k_p$ = set proportional gain
$k_i$ = set integral gain
$k_d$ = set derivative gain
begin loop
    actual = measure actual position
    error = desired - actual
    integral = integral + error
    derivative = error - lasterror
    correction = ($k_p$ × error) + ($k_i$ × integral) + ($k_d$ × derivative)
    drive + correction
    lasterror = error
end loop
```

Usually, the values of k_p and k_i are determined first as discussed previously. Next, k_d is slowly increased until there is no more improvement in tracking response. Too much k_d and the system will result in an excessive response. (Again, one may appeal to the Ziegler-Nichols method.) The typical effect of increasing the derivative gain is qualitatively listed in Table 3.

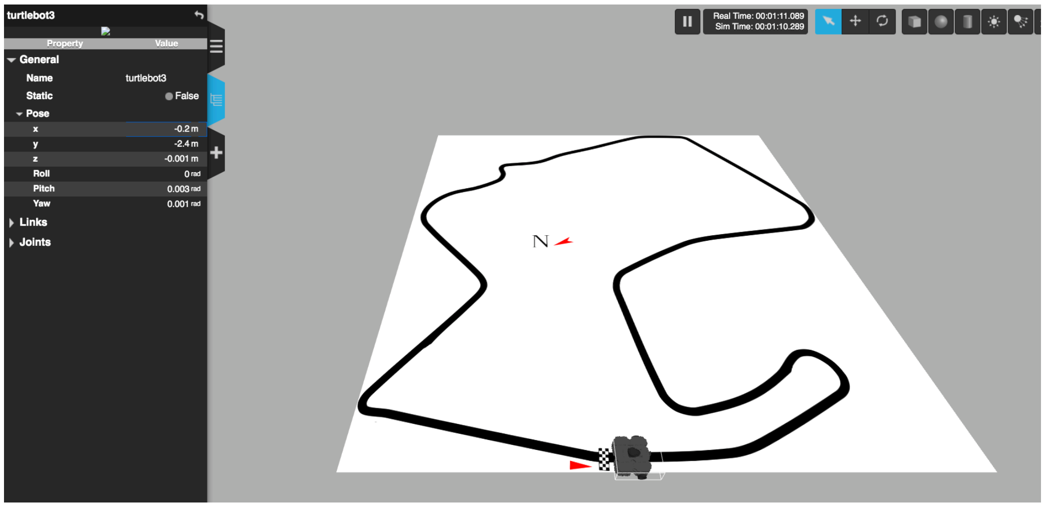
Table 3: Effect of increasing the derivative gain

Parameter	**Rise Time**	**Overshoot**	**Settling Time**	**Steady-State Error**
k_d	Little Effect	Decrease	Decrease	No Effect

Lab Deliverable 4. Write a program to implement the pseudocode of a PID controller. Before you move on to the next task, you are required to demonstrate functionality with a forward speed of at least 0.1 m s^{-1} by recording a successful run on Track 4 using rosbag. Compare the performance of the PID controller to the previous controllers and include a short discussion in your report.

5.5 Laguna NΨ Raceway

A closed-circuit course—the "Laguna NΨ" raceway—will be provided where you are to test the maximum performance of your controller. You are to design a controller to have the robot complete the circuit as quickly as possible. You will likely have to adjust the speed and rotation controller gains to handle the turns without losing tracking performance. When you are satisfied with your robot's performance, demonstrate the functionality to the TA who will record your best time. (A schedule will be provided so be ready when your team is called to race. The details of the demo will be announced soon.)


turtlebot3

Property	Value
General	
Name	turtlebot3
Static	False
Pose	
x	-0.2 m
y	-2.4 m
z	-0.001 m
Roll	0 rad
Pitch	0.003 rad
Yaw	0.001 rad
Links	
Joints	

Real Time: 00:01:11.089
Sim Time: 00:01:10.289

Figure 3: Laguna NΨ Gazebo Race Track

6 Concluding Remarks

We have studied in this exercise some fundamental introductory control concepts. You have experimented with simple applications and learned that it's not really about *where* you finish, just *that* you finish. Above all, you should have learned a little bit more about robot control on a real, ahem, virtual system.

7 Additional Resources

1. TurtleBot3 Waffle Pi Camera Tutorial — `http://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_raspi_cam/`
2. Official ROS Website — `https://www.ros.org/`
3. ROS Wiki — `http://wiki.ros.org/ROS/Introduction`
4. Useful Tutorials to Run Through from ROS Wiki — `http://wiki.ros.org/ROS/Tutorials`

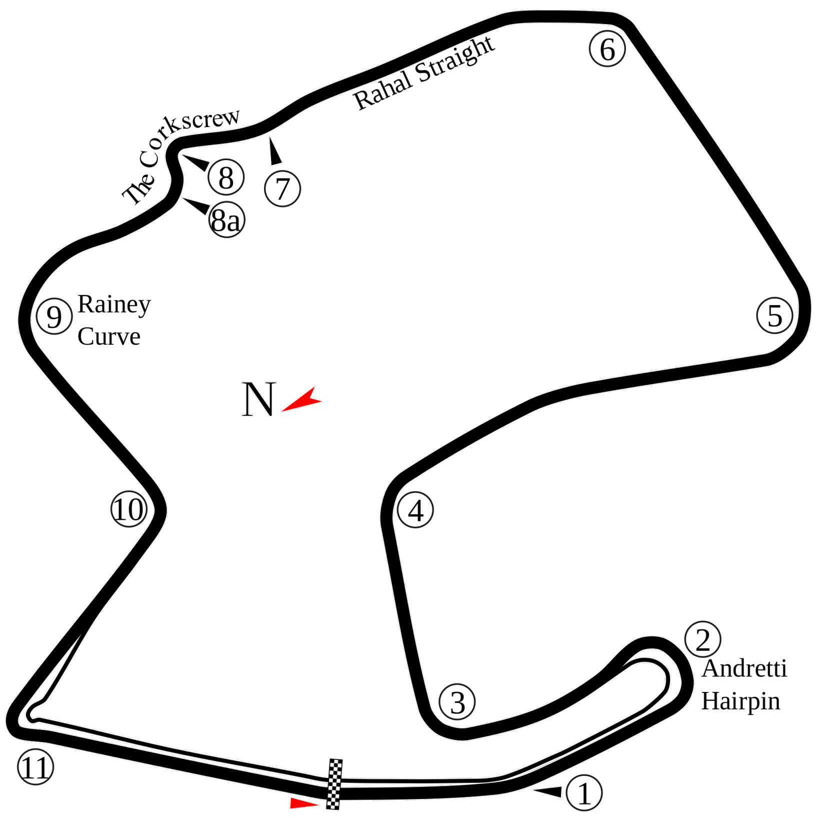


Figure 4: Laguna NΨ Raceway

5. ROS Robot Programming Textbook, written by the TurtleBot3 developers — `http://www.pishrobot.com/wp-content/uploads/2018/02/ROS-robot-programming-book-by-turtlebo3-developers-EN.pdf`