POLYTECH NANCY

# Lab n° 3

# Line tracking control strategies for mobile robots

Hugues Garnier
Mayank Jha

January 2022

# Contents

# Lab 3

# Line tracking control strategies for mobile robots

## Files needed for the lab

- Download the zipped file **Lab3.zip** from the course website and unzip it on your disk.

## 1.1 Objectives and layout of the Lab

Line tracking has become the most convenient and reliable technique by which autonomous mobile robots can navigate in a controlled, usually indoor, environment. The path of the robot is demarcated with a distinguishable line or track, which the robot uses to navigate.

Traditionally, the approach to line tracking is usually considered in the digital domain where the outputs from line sensors are given to a microcontroller which is programmed to vary the motor speeds depending on where the robot is positioned with respect to the centre of the line.

In the industry, vehicles are often required to carry products from one manufacturing plant to another which are usually in different buildings or separate blocks. Conventionally, carts or trucks were used with human drivers. Unreliability and inefficiency in this part of the assembly line formed the weakest link. Solution based on automated guided vehicle following a line, instead of laying railway tracks which can be more costly, are nowadays available as shown in Figure 1.1.


Figure 1.1: Example of automated guided vehicle in the industry

You can also see the automated guided vehicle in action by watching the short video available at: `www.youtube.com/watch?v=WIlS3vNSuQ4`

Automated guided vehicles following a magnetic line have become of common use at Tesla

Mega factories as explained by Elon Musk:
`youtu.be/mr9kK0_7x08&t=6m43s?`

Many line following competitions have been organized by clubs of robot building enthusiasts over the last two decades. The goal of these racing competitions is usually to build an autonomous robot that does a certain number of laps of a test track the fastest. Watch, for example, the video that shows six 3pi robot running simultaneously on the same line-race course. Each robot was programmed independently, but as we can see, the results were remarkably consistent. The last one on the line wins!
`www.youtube.com/watch?v=fl0CJhPiEfYt=76s`
We will try to reproduce this contest in your group by using the same type of robots.

Two other interesting videos showing racing competitions at Universities in England and Japan are available from these links:
`www.youtube.com/watch?v=0dc1oUQ7OOY`
`www.youtube.com/watch?v=nGJ3SQmLtJo`

This lab is structured into three main parts:

1. A first use and test of a PID control example program for line tracking with a 3pi+ robot from Pololu;
2. A racing competition which will require from you a fine tuning of the PID controller so that the 3pi+ robot can complete one lap of the test track in the shortest time possible;
3. Additional improvements of the line following control strategy for handling crossroad or obstacle detection.

## 1.2 Test of a simple proportional control for line robot tracking

### 1.2.1 Hardware required

Figure 1.2: Front view of the 3pi+ robot from Pololu

The required hardware for this section includes:

- a 3pi+ Robot from Pololu as shown in Figure 1.2;
- a USB A to Micro-B cable to connect the robot to your computer for programming and debugging. The USB connection can be used to transmit and receive data from the computer and program the board over USB;
- four rechargeable AAA NiMH batteries;
- a battery charger;
- a basic track for initial test and a racing competition track which has the form of the F1 Estoril course in Portugal shown in Figure 1.3.

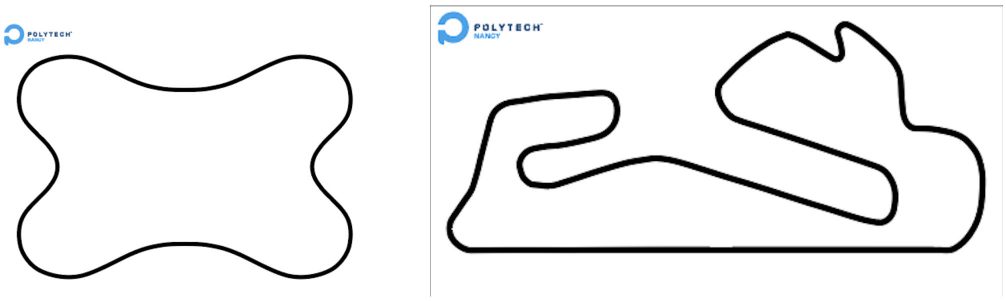

Figure 1.3: Tracks used for initial basic tests and for the racing competition

#### 1.2.1.1 Version of the 3pi+ 32U4 robot

The 3pi+ 32U4 robot is a high-performance, user-programmable robot that measures just 9.7 cm in diameter. The brain of the robot is an integrated, USB-enabled, Arduino-compatible ATmega32U4 microcontroller from Atmel, clocked by a precision 16 MHz crystal oscillator. The 3pi+'s ATmega32U4 comes preloaded with the same Arduino-compatible USB bootloader

as the A-Star 32U4, which allows it to be easily programmed using the Arduino IDE. Three different versions of the assembled 3pi+ 32U4 robots exist. They are equipped with different motors which will impact the maximum speed of the robot.
From the user guide available at: `www.pololu.com/docs/0J83`

- Determine the version of your 3pi+ 32U4 robot: standard, turtle or hyper edition;
- Find the maximum speed in cm/s for your 3pi+ 32U4 robot.

#### 1.2.1.2 On/off and user pushbuttons

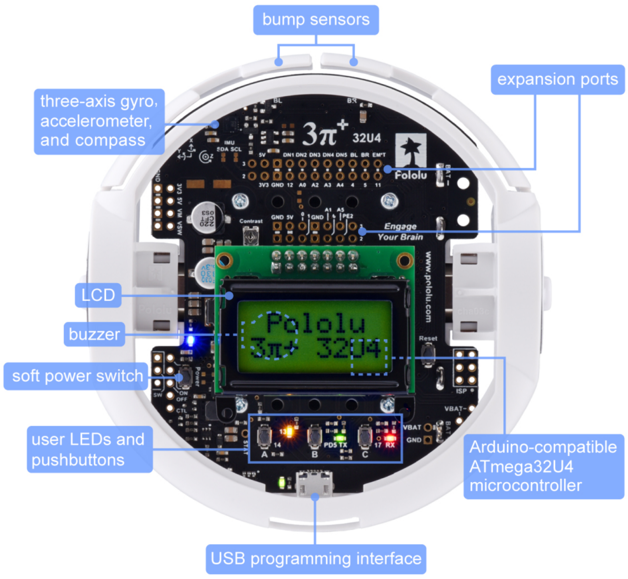


Figure 1.4: Top view of the 3pi+ robot showing its main components

The 3pi+ 32U4 control board has five pushbuttons: a power button on the left, a reset button on the right, and three user pushbuttons, labeled A, B, and C, located at the rear close to USB programming interface as shown in Figure 1.4.
Pushbutton B will be mainly used for calibrating and running the 3pi+ robot. At the bottom left side of the LCD display, there is the soft power button to switch on or switch off the 3pi+ robot. On the right side of the LCD display, there is the reset button that can be used to reset the board.

### 1.2.2 Operating system and software required

#### 1.2.2.1 Operating System required

The 3pi+ 32U4 robot can be programmed from a computer using any operating system that supports the Arduino environment. This includes Microsoft Windows 10, 8.1, 8, 7, Vista, XP (with Service Pack 3), Linux, and Mac OS X.

#### 1.2.2.2 Software required

To get started with your 3pi+ robot, follow the instructions given below. They are more detailed in Section **6. Programming the 3pi+ 32U4** of the 3pi+ robot user guide which can be downloaded from `www.pololu.com/docs/0J83`.

1. If necessary, download and install on your PC the Arduino compiler from `www.arduino.cc/en/Main/Software`.

2. To help interface with all the on-board hardware on the 3pi+ 32U4, Pololu has provided the 3pi+32U4 library. If not already installed, install the 3pi+ 32U4 Arduino library by following carefully all the instructions given in `www.pololu.com/docs/0J83/6.2`.

### 1.2.3 Test of a basic control in open loop

We will first test how to control in open loop the 3pi+ robot by setting manually the speed command of both right and left wheels. Follow the instructions below:

1. From the Arduino IDE, open the "File" menu, select "New", then enter the code below:

```
♯include <Pololu3piPlus32U4.h>
♯include <PololuMenu.h>

using namespace Pololu3piPlus32U4;

Motors motors;

void setup()
{
delay(5000); // wait for 5 seconds before starting the controller
}
void loop()
{int16_t rightSpeed = 50; // Set a nominal 50 speed command to both
//wheels
int16_t leftSpeed = 50; // The sign determines if the robot goes forward
// or reverse.
motors.setSpeeds(leftSpeed,rightSpeed);
}
```

2. Connect the 3pi+ robot to your PC through the USB cable.

3. In the "Tools" menu, select "Board" menu, then "Pololu A-Star 32U4" entry.

4. In the "Tools" menu, select "Port" menu, select the port for the device.

5. Press the "Upload" button to compile the sketch and upload it to the device. If everything goes correctly, you will see the message "Done uploading" appear near the bottom of the window.

6. Disconnect the 3pi+ robot and switch it on by pushing the power button of the robot, on the left of the LCD display.

7. Place the 3pi+ robot over the straight black line of the Estoril racing track. The robot should start at a fairly low speed. You might notice it is not driving in a straight line. That is the unfortunate consequence to real life: the same command to each motor might actually be driving them at slightly different speeds or may be one wheel is making better contact with the ground. Anyway, you can see that the robot is doing pretty much exactly what you thought it would do.

8. Catch the robot and switch it off.

9. Modify to left motor speed to the nominal value 50 + 10 = 60 and the right speed to the nominal value 50 - 10 = 40. Compile the program and upload it to the 3pi+ robot. Repeat the open loop control test and observe if the robot is now turning left or right ? Does it make sense ? From this simple open loop test, you should have understood how you can steer the robot direction by adjusting the speed difference between the wheels. The next goal will be to test a simple P control strategy to set automatically the speed difference command (manually chosen arbitrarily to $2 \times 10$ above).

### 1.2.4 Test of an initial simple P controller in closed loop

We will now test a closed loop control by using an initial basic Proportional controller to make the 3pi+ robot track the black line of the basic test track. The robot is expected to track the line faithfully without any problems. The robot should not leave the line at any instant and the movement of the robot should have minimum overshoot and all the movements should be smooth.

Note that for this simple P control strategy you will need to select a nominal speed at which the robot will travel. If the selected nominal speed is too high, the robot will not be able to stay on the track.

Follow the instructions below:

1. From the Arduino IDE, open the "File" menu, open the program named **Lab3.ino** from the zipped file downloaded from the course website. Verify that the variable **nominal-Speed** is set to 100;

2. Connect the 3pi+ robot to your PC through the USB cable.

3. In the "Tools" menu, select "Board" menu, then "Pololu A-Star 32U4" entry.

4. In the "Tools" menu, select "Port" menu, select the port for the device.

5. Press the "Upload" button to compile the sketch and upload it to the device. If everything goes correctly, you will see the message "Done uploading" displayed at the right bottom of the window.

6. Disconnect the 3pi+ robot and switch it on pushing the power button on the left side of the LCD display.

7. Place the 3pi+ robot over the black line of the basic test track and press the B-button. The robot will make two turns to start the automatic sensor calibration step by rotating in place to sweep the sensors over the line.

8. Prepare your smartphone to measure the time it takes for the 3pi+ robot to complete one lap. When you are ready to start the time, press again on the B-button and let the robot follow the line. Stop the time when the robot has complete a full lap, if it can make it ! Record the time that will constitute the reference time to be improved.
   You should observe that the simple P controller works reasonably well for this relatively slow nominal speed.

9. Place now the 3pi+ robot over the black line of the racing competition track et test if the robot is able to complete one full lap.
   If this is not the case, try to reduce the nominal speed and test if the robot is bale to achieve one full lap without leaving the line in the sharp curved area. To achieve better performance, here is a need to better understand how the control of the two motor speeds work. We will use the basic test track in the remaining of this section before exploring more sophisticated control strategies in the next section.

10. From the example program and comments given inside, determine the main command to be controlled and the two variables used to steer the 3pi+ robot.

11. Give the control equation for both left and right motor speeds.

12. Determine the setpoint value for the line position.

13. Determine the role of the variable **nominalSpeed**.

14. Build a block-diagram of the simple P controller for the robot line follower, showing a block for the actuator, the robot, the sensor and the controller.

15. Modify the proportional gain of the controller so that the robot can complete one lap of the basic test track in the shortest time possible. Once the robot is following the line with good accuracy, increase the **nominalSpeed** variable to 200 and 400 (50%, and 100% of its maximum value respectively) and see if it still is able to follow the line (this might not be the case for 300 or 400). Note that the robot nominal speed affects the P controller tuning and will require its retuning as the **nominalSpeed** changes. This is known as gain-sheduling.

Gather the performance of the different P controllers by completing the first three lines of Table 1.1.

## 1.3 Advanced control strategies for the robot line tracker competition

The final objective is to make the 3pi+ mobile robot move forward at the fastest speed so that it can complete one lap of the racing track in the shortest time possible.
The performance criterion that will be used is the time taken to complete one lap of the test track. It is assumed that, for a constant motor speed setting, the PID controller tuning which achieves the minimum mean square error (MSE) to do one lap of the test track in the shortest time, is the best. This is justified because when the error is more, the correction will naturally be more, which causes the speed of the robot to be reduced.

To achieve better performance, there is a need to better know the main components of the 3pi+ robot but also to examine the characteristics of the line following from a control system perspective. This should help for the tuning the different terms of the PID controller.

### 1.3.1 The 3pi+ 32U4 robot in detail

#### 1.3.1.1 The line sensor

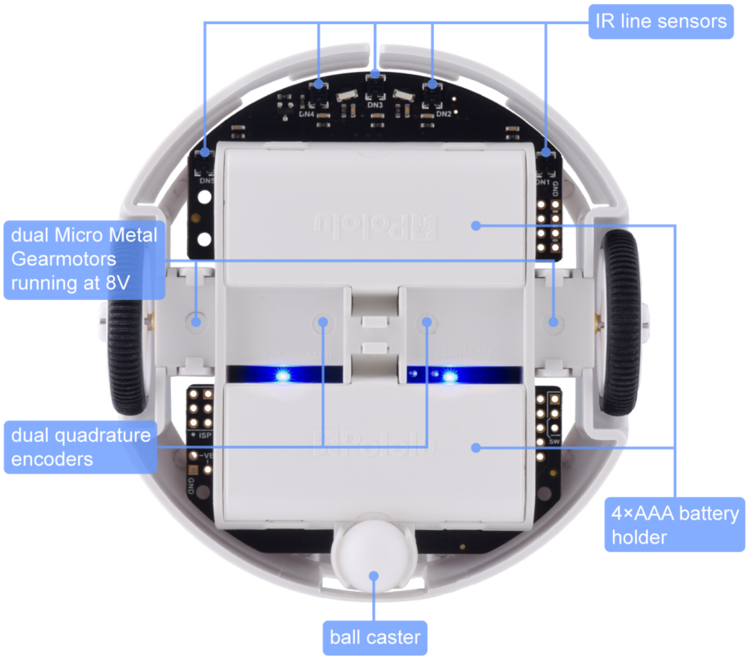


Figure 1.5: Underneath view of the 3pi+ 32U4 robot. The line sensor array are placed at the front.

The 3pi+ 32U4 line sensor array is a separate board that attaches to the main underneath board, placed in front of the robot as shown in Figure 1.5. The board features five line sensors connected to the microcontroller. The five line sensors face downward and can help the 3pi+ robot distinguish a non-reflective (black) line on a reflective (white) surface. Each reflectance sensor consists of a down-facing analog infrared (IR) emitter LED (Light-Emitting Diode) paired with a phototransistor that can detect reflected IR light from the LED. Three of the five sensors are placed in the middle of the board at gaps of about 10 mm and the last two are situated at the far end of both sides such that the total sensor coverage is about 60 mm.

The line IR sensors are just an indication of surface reflectivity. The sensor value is lower when over a white surface and higher when over a dark surface. The actual values are heavily dependent on the external lighting conditions even though sensor array is shielded. Further, there is appreciable difference between the readings of various sensors due to various factors like placement, manufacturing differences, etc.
To get usable values from all the sensors, the sensor values are normalized for environmental conditions and sensor differences and the normalization values are stored in Data Flash of the microcontroller. This enables the normalization data to be used multiple times. To normalize the sensor values, the robot is placed on the track and rotated such that all the sensors pass over the black and white surfaces. The maximum and minimum readings for all the sensors are noted down and stored in non-volatile memory (Data Flash) of the microcontroller.

The `lineSensors.readLine(lineSensorValues)` function returns a value between 0 and (number_of_sensors) - 1)×1000. As the 3pi+ 32U4 robot has an array of 5 sensors, the reading will be 0-4000. 0 represents the left most sensor and each increment of 1000 after that represents another sensor. 0 = first sensor, 1000 = second sensor and there are ranges in between as well. 500 means the line is between the first and second sensors, 1200 means it's between the second and third sensors but closer to the second, etc. Note that if the reading is 2000, it means that that the robot is centered on the line.

This output of the IR line sensor array is fed to the microcontroller which calculates the error term between the sensor value and the line position setpoint. The most convenient and reliable way to compute the error term is by dividing the weighted average of the sensor readings by the average value and normalizing the values. The formula used to compute the error $\varepsilon(k)$ at time-instant $k$ is given as follows:

$$\varepsilon(k) = 1000 \times \frac{\sum_{i=0}^{4} i \times L_i(k)}{\sum_{i=0}^{4} L_i(k)} - 2000$$

In the equation, $L_i(k)$ refers to the output of the $i$-th line sensor in the array. Here, 1000 is the normalizing factor which is used to ensure that the contribution of the error term from each sensor is normalized to a value of 1000. The subtraction by 2000 is to ensure that zero error occurs when the robot is centered on the line.
The sensors are numbered from the left to the right. An error value of zero refers to the robot being exactly on the center of the line. A positive error means that the robot has deviated to the left and a negative error value means that the robot has deviated to the right. The error can have a maximum value of $\pm 2000$ which corresponds to maximum deviation.
The main advantage of this approach is that the five sensor readings are replaced with a single error term which can be fed to the control algorithm to compute the motor speeds such that the error term becomes zero. Further, the error value becomes independent of the line width and so the robot can tackle lines of different thicknesses without needing any change in code.

#### 1.3.1.2 The actuators

The robot is driven by two micro metal gearmotors with extended motor shafts, coupled to wheels with differential drive. It is battery powered, with the motors supplied with regulated power supply to ensure consistent motor power at all operating scenarios. The robot can reach a maximum speed of about 1.5 cm per second.

To move the robot forward, both motors are rotated in the forward direction. To make the robot turn to the left or to the right, the speed of one motor is reduced while the speed of the second motor is increased by the same value. The amount of turn increases as the speed difference increases. Maximum amount of turn is achieved when one motor is turned in a backward direction at maximum speed, the other in the forward direction at maximum speed. This results in maximum speed difference and the robot just spins in place.

#### 1.3.1.3 Characteristics of the line following from a control system perspective

When the robot is placed on a straight track, it was observed that the robot was able to stay centered over the line. This proves that the steady state error is zero for a straight line. When the robot is placed on a curved track, it was seen that the robot overshoots the line centre by a greater amount in the direction opposite to the direction of curvature of the track. Further, it was observed that, as the curvature increases, the difference in overshoots increases. This proves that a steady state error exists when the track is curved.
For the closed-loop system, the setpoint is always constant and set to 2000 so that the robot

stays on the centre of line. However the track does not remain straight and so the line curvature should be considered as an external disturbance for the control system. Now, it can be concluded that, when the system has zero or negligible disturbance (analogous to a step input for a simple unity feedback system), the steady state error (position error) is zero. When a significant disturbance exists (analogous to a ramp or parabolic input for a simple unity feedback system), the steady state error (velocity and acceleration errors) are significant values. From the above discussion, it can be concluded that there are multiple (at least one) integrators in the feedforward path of the system.

The system characteristics can be better understood by considering an open-loop test on the line tracking system:

1. Assume that the robot is placed exactly on the centre of a straight line and exactly in the direction of the line. If no control effort is exerted (both motors run at equal speed), the robot moves forward with zero error. This is the only special condition where zero control effort is needed.

2. If there is any discrepancy, line not being exactly straight, robot not being perfectly aligned or the existence of speed difference between the motors, control effort is needed (speed of one of the motors must be reduced while the speed of the second motor must be increased).

3. Now, assume that the robot is away from the centre of the line by some distance. In order to reduce the error, a control effort must be given as illustrated in Figure 1.6. However, the exertion of a constant control effort will cause the robot to overshoot the line.

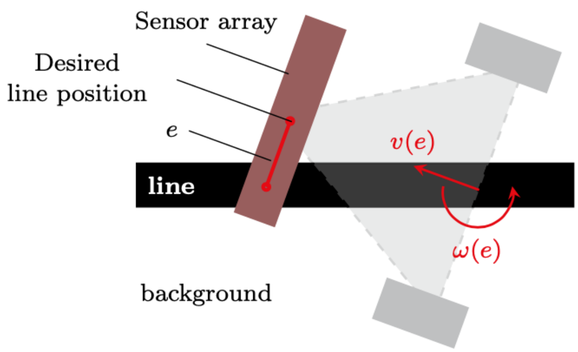


Figure 1.6: Open-loop test to illustrate the required control effort to steer the robot over the line.

From the above considerations, it can be understood that the loop is integrating (i.e.) control effort is being integrated by the system. The open loop response of the system thus resembles that of a servo position control system. Thus, the control techniques used for servo control may be applicable for the line tracking robot. One of the most popular control technique for servo position control takes the form of a PD controller, which is usually a good starting point for the design of the line follower for the robot.

### 1.3.2 Finding best values of the PD controller

The continuous-time PD controller equation in parallel form is recalled below

$$u(t) = k_p \times \varepsilon(t) + k_d \times \frac{d\varepsilon(t)}{dt}$$

where $k_p$ and $k_d$ refer to proportional and derivative gain constants respectively.

For digital implementation, the time-derivative of the error term can be approximated by using the backward Euler method.

$$u(k) = K_p \times \varepsilon(k) + K_d \times \big(\varepsilon(k) - \varepsilon(k-1)\big) \tag{1.1}$$

where $K_p = k_p$ and $K_d = \dfrac{k_d}{T_s}$; $T_s$ being the sampling period. $u(k)$ denotes the command value at time-instant $t = kT_s$.
Note that the sampling time, $T_s$, is simply eliminated from the control equation because it is embedded in the new derivative gain $K_d$.

Finding the best values of the PD controller, i.e., $K_p$ and $K_d$ values, is now the most important challenge for you. There is unfortunately no simple way to get a model of the robot which could serve as the basis to tune the PD controller gains. The latter will therefore be tuned in this lab by a trial and error method only. These gain values are expected to be different for every group of students and for the nominalSpeed variable setting, which determines the speed at which the robot is running.

Using a "trial and error" methodology to set the gains of a PD controller has some advantages (no need to determine a model, ...). The bad side of it is that you must re-compile the program each time that you change the gains.

You might want to try this empirical method for the tuning of the PD gain controller:

- Modify the initial program to implement a PD control given in (1.1) instead of the simple P control.
- Set the nominalSpeed variable to 200 (50% of its maximum value).
- Set the $K_d$ gain to 0 and tune the $K_p$ term alone first. If the robot cannot navigate a turn or reacts too slowly, increase its value. If the robot seems to react fast leading to a zigzagging behavior, decrease the value. Once the robot responses reasonably, record the time that will constitute the best time for the P controller at the nominalSpeed setting of 50% of the maximum value of the robot speed.
- Then set your $K_d$ gain value to 10 times of the $K_p$ value and check if the robot can navigate in a better way than with the simple P controller. Increase the derivative gain to decrease the overshoot, decrease it if the robot become too jerky or is not able to remain on the line.
- Once the PD controller tuning make the 3pi+ robot follow the line with good accuracy, record the time that will constitute the best time for the PD controller at 50% of the maximum value of the robot speed.

An excellent guide to tune your PID controller gain for robot line following is available from this link:
`robotresearchlab.com/2019/02/16/pid-line-follower-tuning/`

- Once the robot is following the line with good accuracy, increase the **nominalSpeed** variable to 300, 350 and 400 (75%, 87.5%and 100% of its maximum value respectively) and see if it still is able to follow the line. Note that the robot speed affects the PID controller and will require retuning as the nominalSpeed changes.

- It is worth noting that the tuning of the PD controller gain might not be possible for the maximum speed setting without additional modification of the control algorithm so that the speed of the robot is dynamically changed depending on curvature of the track, as we do when we drive a car !

### 1.3.3 Performance summary of the different controllers on the basic test track

Gather the performance of the different controllers by completing Table 1.1.

| nominalSpeed | Algorithm | $K_p$ gain | $K_d$ gain | Completion time for one lap (sec) |
|---|---|---|---|---|
| 100 (25%) | P controller | | / | |
| 200 (50%) | P controller | | / | |
| 200 (50%) | PD controller | | | |
| 300 (75%) | PD controller | | | |
| 350 (87.5%) | PD controller | | | |
| 400 (100%) | PD controller | | | |

Table 1.1: Performance of the various P and PD controller-based algorithms

### 1.3.4 Performance test of the basic PD controller on the racing competition track

From your initial experience obtained from your test with the basic track, tune the PD control the best you can so that your robot can traverse the Estoril track as fast as possible. Gather the performance and tuning value of the best PD controller in a Table similar to Table 1.1.

Note that the Estoril track is much more challenging that the basic track since it includes a difficult curved area which requires a low speed to keep the robot on the track. The nominal speed of the robot should therefore be set to small or moderate value for the robot to be able to do handle this curve and complete a full lap.

You will explore some possible improvements to get better line tracking performance of your robot in the next section.

### 1.3.5 More sophisticated control strategies to improve performance

The final goal of this section is to develop a more sophisticated gear-shifting control strategy for your robot to dynamically adjust its speed while traversing the track to travel as fast as possible on less challenging portions of the track (in the long straight line for example) while slowing down in difficult curved areas to maintain control.

#### 1.3.5.1 Addition of an integral action in the controller

The first modification would be to add an integral term to your PD controller where only the last ten error values are summed up instead of adding all the previous values. The modified controller equation takes then the form:

$$u(k) = K_p \times \varepsilon(k) + K_i \times \sum_{i=0}^{9} \varepsilon(k-i) + K_d \times \big(\varepsilon(k) - \varepsilon(k-1)\big)$$

where $K_i = k_i \times T_s$ is the integral gain of the discretized version of the PID controller in parallel form by the backward Euler method. Set $K_i$ to a very small value and then gradually increase it until overshoot is acceptable for curved paths and turns. Modify the width of the moving window (set to 10 above) to compute the integral term if this is necessary. The addition of the integral action might not lead to improvements for this robot and test track.

#### 1.3.5.2 Addition of dynamic speed variation

Another improvement consists in adding a speed reduction term in the control equation of both motor speeds so that the robot can navigate at a higher speed in the straight path and slows down slightly at turns so that stability is maintained. For example, the speed reduction term to be subtracted from the control equation, can be of the form:

$$SR(k) = K_{sr} \times \sum_{i=0}^{9} |\varepsilon(k-i)|$$

where $K_{sr}$ should be tuned from a small value and gradually increased until overshoot is acceptable for curved paths and turns.

If the proposed strategy is not successful, move on to the next subsection where another more advanced cascade control strategy is suggested.

#### 1.3.5.3 Adaptive speed variation via cascade control

Download the 2017 MSc report from Cleveland State University `https://engagedscholarship.csuohio.edu/cgi/viewcontent.cgi?article=2024context=etdarchive`

Go through Chapter 5 and derive a mathematical model of the 3pi+ mobile robot. Use your mathematical model for the tuning of your most sophisticated robot line tracking strategy. You are also free to implement some of the control strategies discussed in the MSc report. The inner-outer loop design, commonly known as a cascade control, discussed page 41 in the MSc report could constitute an interesting approach.

## 1.4 Robot line following and crossroad/obstacle detection

According to the time, you will more likely not have the possibility to test the two extensions suggested below. Explain in your report the different investigations suggested below that you have tested.

### 1.4.1 Control algorithm for robot line following and crossroad detection

Extend the line following algorithm with crossroad detection and path selection algorithm. Once the mobile system reaches the crossroad, it should stop. Based on the command (continue forward, left, right, backward) the mobile robot should take the appropriate path. Implement an algorithm for selection of four different paths in a crossroad.
Use a black rubber band to implement the crossroad in the straight line of the test track.

Fig. 1.7 shows how the IR sensors can be used for line tracking (front sensors) and for crossroad detection (side sensors).

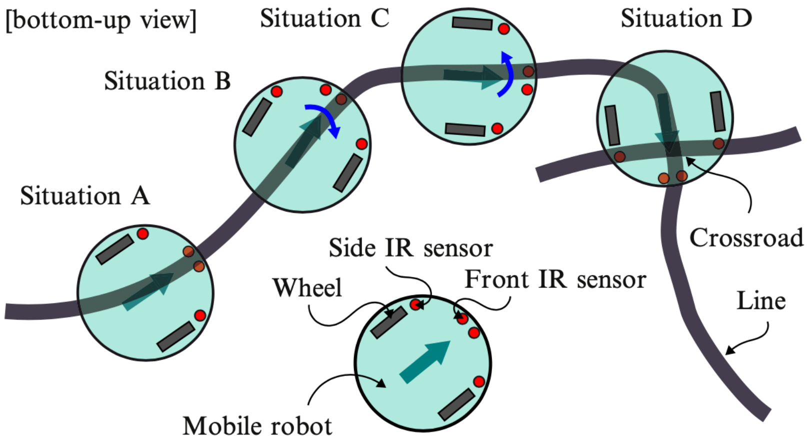


Figure 1.7: Line-following and crossroad detection approach based on IR sensors

### 1.4.2 Control algorithm for robot line following and obstacle detection

Extend the line following algorithm in adding an obstacle detection module such that the robot would stop, spin and go in reverse if an obstacle is detected by IR proximity sensors, in a range of 10 cm for example.

Develop a control algorithm so that once the obstacle is detected, the robot can spin and go in reverse to follow the line in the opposite direction.

As the 3pi+ robot is not equipped by any IR proximity sensors, you will use the Zumo robot, as shown in Figure 1.8 for testing and implementing your obstacle detection algorithm.

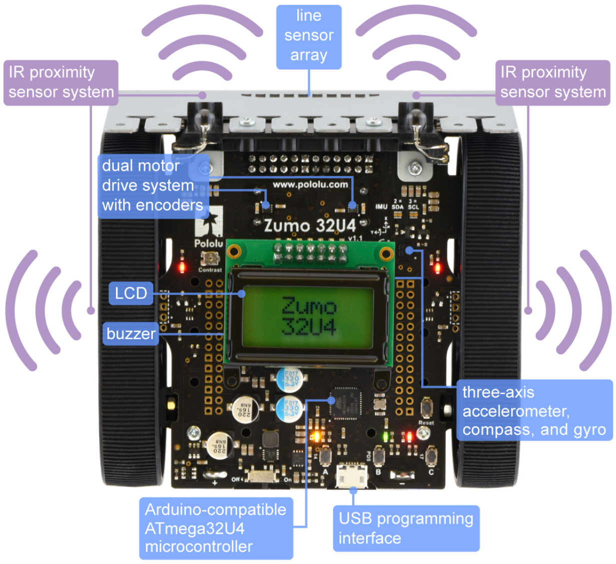


Figure 1.8: Zumo 32U4 robot from Pololu and its main components

The required hardware for this section includes:

- a Zumo 32U4 Robot from Pololu as shown in Figure 1.8;
- a USB A to Micro-B cable to connect the robot to your computer for programming and debugging. The USB connection can be used to transmit and receive data from the computer and program the board over USB;
- Four rechargeable AA NiMH batteries;
- a test track.

The Zumo 32U4 robot is a complete, ready-to-program robot platform. The brain of the robot is an integrated, USB-enabled, Arduino-compatible ATmega32U4 microcontroller from Atmel, clocked by a precision 16 MHz crystal oscillator. The Zumo's ATmega32U4 comes preloaded with the same Arduino-compatible USB bootloader as the A-Star 32U4, which allows it to be easily programmed using the Arduino IDE. A choice of three motor gear ratios offer different combinations of torque and speed. Different versions of the assembled Zumo 32U4 robots exist. They can be identified with a sticker on the underside of the main board, visible inside the battery compartment of the Zumo without batteries installed. The color of the sticker indicates the gear ratio of the robot's motors:

- Green: 50:1 HP
- Blue: 75:1 HP
- Red: 100:1 HP

Determine the version of your Zumo 32U4 robot from the color of the sticker.
Find the maximum speed in cm/s for your Zumo 32U4 robot by searching this information in the user guide available at: `www.pololu.com/docs/0J63`.

#### 1.4.2.1 On/off and user pushbuttons

The Zumo 32U4 has three user pushbuttons, labeled A, B, and C, located along the rear edge of the main board close to the USB programming interface as shown in Figure 1.8.
Pushbutton A will be only used for running the Zumo robot. On the left of the USB programming interface, there is a sliding button to switch on or switch off the Zumo robot.

#### 1.4.2.2 Operating System required

The Zumo 32U4 robot can be programmed from a computer using any operating system that supports the Arduino environment. This includes Microsoft Windows 10, 8.1, 8, 7, Vista, XP (with Service Pack 3), Linux, and Mac OS X.

#### 1.4.2.3 Software required

To get started with your Zumo robot, follow the instructions given below. They are more detailed in Section **5. Programming the Zumo 32U4** of the Zumo robot user guide which can be downloaded from `www.pololu.com/docs/0J63`.

1. If necessary, download and install on your PC the Arduino compiler from `www.arduino.cc/en/Main/Software`.

2. To help interface with all the on-board hardware on the Zumo 32U4, Pololu has provided the Zumo32U4 library. Follow carefully all the instructions given in `www.pololu.com/docs/0J63/5.2` to install the Zumo 32U4 Arduino library.
   Depending on your Arduino version, you can also try the following: from Arduino, go to the "Sketch" menu, select "Include Library", then "Manage Libraries...", then search for "Zumo32U4" and install the library.

3. Several example sketches are available that show how to use the Zumo robot with the library. You can access the example codes available from the Arduino IDE by opening the "File" menu, selecting "Examples", and then selecting "Zumo32U4". If you cannot find these examples, the library was probably installed incorrectly and you should retry the installation instructions above. Another option is to download and copy the folders including the code examples for Pololu's Arduino-based Zumo 32U4 robot from `github.com/pololu/zumo-32u4-arduino-library/tree/master/examples`.

For help in getting your computer set up with the Zumo robot, you can also see this video: `youtu.be/L6iX8ZJ6nNo`.

## 1.5 Summary and reflections

Summarize and reflect on what you have seen and learned during this lab.

# English to French glossary

| | | |
|---|---|---|
| bandwidth | : | bande passante |
| castor | : | roulette |
| crane | : | grue |
| closed-loop system | : | système bouclé |
| cut-off frequency | : | fréquence (ou pulsation) de coupure |
| damped frequency | : | pulsation amortie |
| damping ratio | : | coefficient d'amortissement |
| drag | : | traînée |
| feedback | : | contre-réaction |
| feedback system | : | système à contre-réaction |
| hoisting device | : | dispositif de levage |
| impulse response | : | réponse impulsionnelle |
| integral wind-up | : | emballement (de l'action) intégral |
| input | : | entrée |
| gain | : | gain |
| heading angle | : | angle de cap |
| linear time-invariant (LTI) | : | linéaire invariant dans le temps |
| motor shaft | : | arbre moteur |
| output | : | sortie |
| overdamped | : | sur-amorti |
| overshoot | : | dépassement |
| rise time | : | temps de montée |
| road grade | : | inclinaison de la route |
| robot arm joint | : | articulation d'un bras de robot |
| root locus | : | lieu des racines |
| setpoint | : | consigne |
| settling time | : | temps de réponse |
| steady-state gain | : | gain statique |
| steady-state response | : | réponse en régime permanent |
| steering | : | direction |
| step response | : | réponse indicielle |
| stream | : | courant |
| yaw angle | : | angle de lacet |

| | | |
|---|---|---|
| time-delay | : | retard pur |
| time-invariant | : | invariant dans le temps |
| transient response | : | réponse transitoire |
| throttle | : | accélérateur |
| undamped | : | non amorti |
| undamped natural frequency | : | pulsation propre non amortie |
| underdamped | : | sous-amorti |