
# Digital control Lab

# Advanced digital control strategies for robot line tracker competition

Hugues Garnier
Floriane Collin

April 2022

# Contents

# Digital Control Lab

# Advanced digital control strategies for robot line tracker competition

*The work done in pairs during this lab will be noted in a report and marked. You are asked to follow the instructions given below to write your report.*

## Instructions for writing your laboratory report

A report is a scientific document. It must therefore respect a certain structure. Your report should be structured as follows:

- a general introduction specifying the objectives of the lab
- For each assignment:
  - ▷ a brief presentation of the expected outcomes
  - ▷ a reminder or presentation (if necessary) of the theoretical results
  - ▷ a description of the obtained results in graphical or numerical form
  - ▷ a short conclusion
- a general conclusion explaining what has been understood during the lab and any difficulties encountered.
- an appendix containing the Arduino program you have developed.

## Sending your report to the tutor

The report should be written in groups of two students and sent by email to your instructor in the form of a single pdf format file attached to the message. It must be sent before the deadline given during the lab session.
Please indicate the following "subject" for your email to send your report to the instructor: **Lab_DigitalControl_Name_Lab_Group"**.

## Files needed for the lab

- Download the zipped file **Lab3pi.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.

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, a very impressive video showing a racing competition in Japan:
`www.youtube.com/watch?v=nGJ3SQmLtJo`
We will try to reproduce this contest in your group by using a 3pi+ robot on a racing competition track which takes the form of the F1 Estoril track in Portugal.

This lab is structured into three main parts:

1. A test of a simple proportional (P) digital control example program ;
2. A first improvement by using a PD digital controller ;
3. Additional improvements of the line following control strategy.

## 1.2 Hardware and software required

### 1.2.1 Hardware required

Figure 1.1: Front view of the 3pi+ robot from Pololu

The required hardware for this section includes:

- a 3pi+ Robot from Pololu as shown in Figure 1.1. Two servo motors are used to drive the two wheels. A back small castor wheel helps to stabilize the robot. The 3pi+ robot has a set of 5 sensors mounted at the front end to identify the line. More details are given in the Appendix. The 3pi+ robot is designed to follow a track made of black electrical tape (about 15 mm wide) on a white surface. The two wheels, which are separated by a distance of about 8,5 cm, should remain centered on the black line ;
- 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 (the robot works also with Alkaline batteries).
- a battery charger (if necessary);
- a racing competition track which has the form of the F1 Estoril course in Portugal shown in Figure 1.2.


Figure 1.2: Estoril track used for the racing competition

Note that the Estoril track is much more challenging that the basic test track you used in a previous lab since it includes a difficult curved area (located in 9 and 10 in Figure 1.2) which requires a low speed to keep the robot on the track. The nominal speed of the robot should therefore be set to a small or moderate value for the robot to be able to do handle this curve and complete a full lap.

**On/off and user pushbuttons**

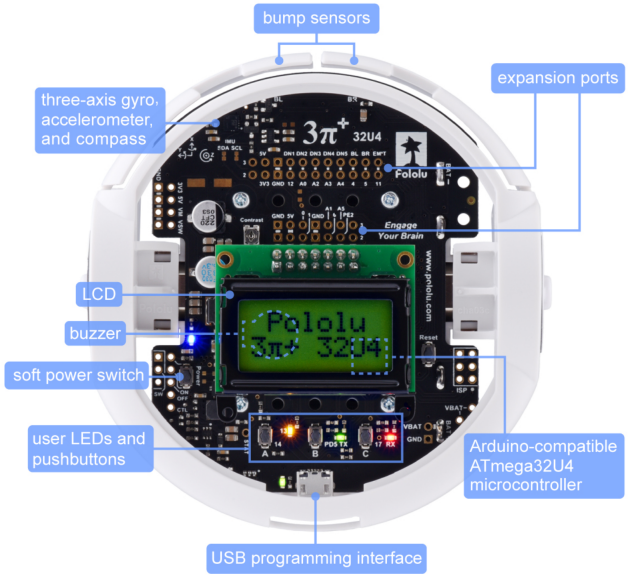


Figure 1.3: Top view of the 3pi+ robot showing its main components

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 1.3.
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`.
   Depending on your Arduino version, you might also have to do the following: from Arduino, go to the "Tools" menu, select "Manage Libraries...", then search for "3pi+32U4" and install the full library.

## 1.3 Simple digital proportional control

We will now test a feedback control by using an initial basic Proportional digital controller to make the 3pi+ robot track the black line of the 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 **Lab3pi+.ino** from the zipped file downloaded from the course website. Verify that the variable **nominalSpeed** 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 able to achieve one full lap without leaving the line in the sharp curved area.

10. Modify the proportional gain $K_p$ of the controller (note that $K_p$ should be even) so that the robot can complete one lap of the F1 test track in the shortest time possible. Once the robot is following the line with good accuracy, increase the **nominalSpeed** variable to 100 and 200 (25%, and 50% of its maximum value respectively) and see if it still is able to follow the line (this might not be the case for 200). 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-scheduling.

11. Another component of the digital P control to consider is the loop sampling period $T_s$. Speeding this parameter up or slowing it down can make a significant difference in the robot control performance. This can be set by uncommenting the line at the bottom of the code:
    `delay(100);` //Set the sampling frequency to 1/(100*10-3)=100 Hz ($T_s = 0.1$s)
    The `delay` variable is defined in ms. A value set to 100 will fix the sampling period to $T_s = 0.1$s. Execute the modified code for your best P controller tuning when the nominalSpeed is set to a given value and observe the performance of the control.

12. Modify the numerical value in the delay command and observe the impact of a slow/fast sampling time on the performance of the digital control.

13. What value for the sampling time of the control loop would you recommend here ?

Gather the performance of the different basic P control tests by completing the lines of Table 1.1.

| nominalSpeed | Algorithm | $K_p$ gain | $T_s$ sampling period | Completion time for one lap (sec) |
|---|---|---|---|---|
| 100 (25%) | P controller | | | |
| 200 (50%) | P controller | | | |
| ? ( ?%) | P controller | ? | ? | |

Table 1.1: Performance of the different tunings for the simple P digital control

To achieve better performance, there is a need to better understand how the control of the two motor speeds work.

## 1.4 Characteristics of the robot line tracking from a control system perspective

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.

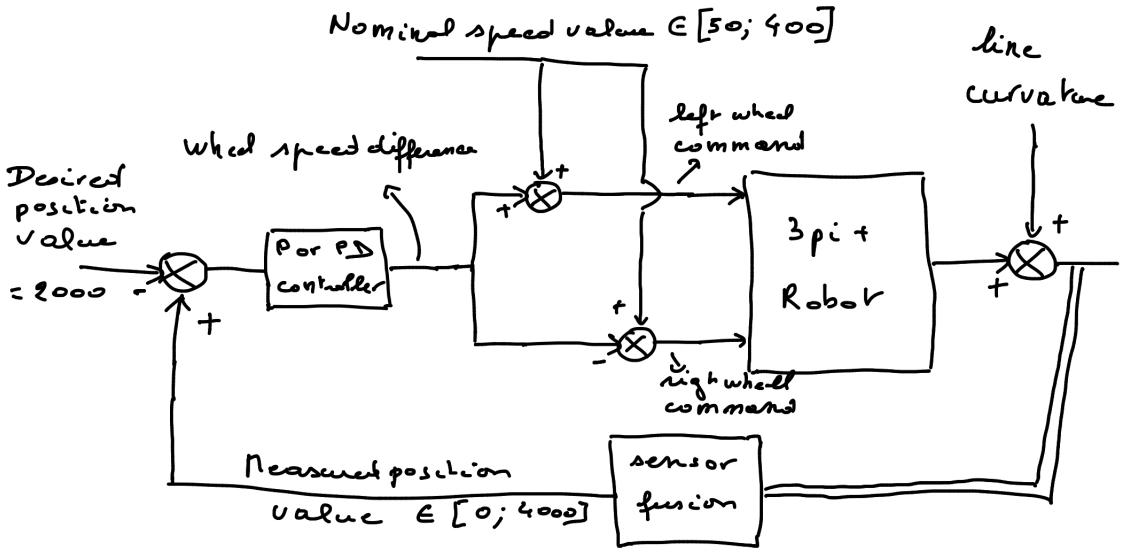


Figure 1.4: Closed-loop diagram for the line tracking robot control

By using Figure 1.4 along with the example program and comments given inside, determine:

1. the setpoint value for the line position;
2. the main disturbance for the control loop;
3. the variable computed from the controller;
4. the two variables used to steer the 3pi+ robot;
5. the role of the variable **nominalSpeed**.
6. Build a closed-loop block-diagram of the basic P digital controller for the robot line follower, showing the constant setpoint, the comparator, a block for the digital controller (simple P here), a first sampler, the zero-order hold, the robot, the disturbance, the main output, the sensor, the second sampler for the analog to digital converter.

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

- 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.
- If there is any discrepancy, line not being exactly straight, the 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).
- 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 1.5. However, the exertion of a constant control effort will cause the robot to overshoot the line.

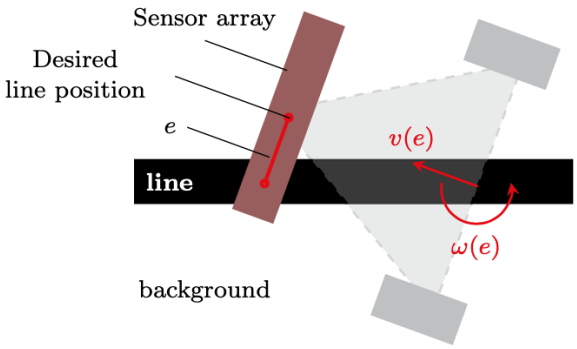


Figure 1.5: Open-loop test to illustrate the required control effort to steer the robot over the line

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 Proportional and Derivative (PD) controller, which is usually a good starting point for the design of the line follower for the robot.

## 1.5 Digital PD control 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.

Adding a derivative term in the controller will exploit how much the sensor array center has moved from the last time (its change speed). Adding this term to the controller will then make the robot to "predict" were the line will be in the next iteration which should result (if tuned correctly!) a huge decrease of oscillations.

### 1.5.1 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.

$$u(k) = k_p \times \varepsilon(k) + k_d \times \frac{\big(\varepsilon(k) - \varepsilon(k-1)\big)}{T_s} \tag{1.1}$$

or

$$u(k) = K_p \times \varepsilon(k) + K_d \times \big(\varepsilon(k) - \varepsilon(k-1)\big) \tag{1.2}$$

where $K_p = k_p$ and $K_d = \frac{k_d}{T_s}$; $T_s$ 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 (1.2) instead of the simple P control.
- Give the approximation method used to implement the derivative operation.
- Set the nominalSpeed variable to 100 (25% 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.
- 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 25% of the maximum value of the robot speed.
- Once the robot is following the line with good accuracy, increase the **nominalSpeed** variable to its maximum value such that it is able to follow the track. Note that the robot speed affects the PD 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 !

Tuning a PD or PID controller based on a trial and error methodology is tedious and can lead to poor or moderate performance. An alternative and well-know approach would consist in determining a mathematical model which serves at the basis of the PID controller tuning or a more advanced control strategy.

### 1.5.2 Performance summary of the different PD controllers

Gather the performance of the different controllers by completing Table 1.2.

| nominalSpeed | Algorithm | $K_p$ gain | $K_d$ gain | Completion time for one lap (sec) |
|---|---|---|---|---|
| 200 (50%) | PD controller | | | |
| ? ( ?%) | PD controller | | | |

Table 1.2: Performance of the PD digital controller

## 1.6 Addition of dynamic speed variation to improve performance

The 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. One interesting adaptive control approach is known as gain scheduling based control as shown in Figure 1.6.

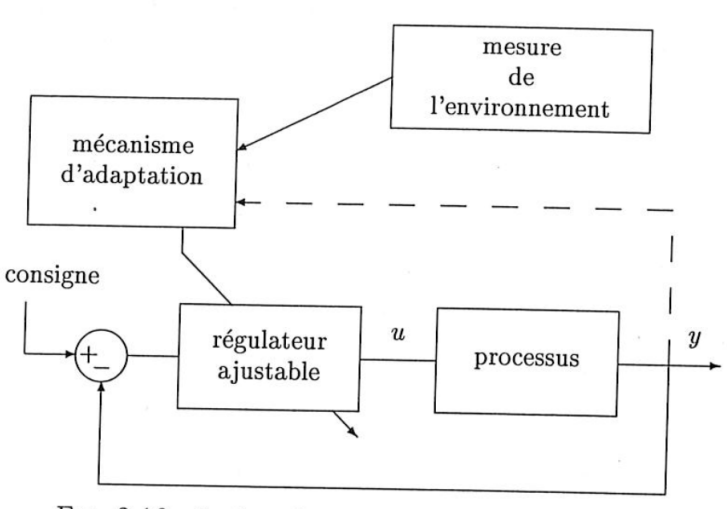


Figure 1.6: Gain scheduling based control strategy

#### 1.6.0.1 Dynamic speed variation via the addition of a speed reduction term

The proposed 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 (subscript "sr") term to be subtracted from the control equation, can be of the form:

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

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

Modify also the width of the moving window $N_{sr}$ (it can be set initially to 9) to compute the sum term if this is necessary. If correctly tuned, this speed reduction term addition should lead to improvements for this robot and racing competition track.

Report the best completion time (sec) for one lap in a table similar to Table 1.2

If f the proposed strategy is not successful, move on to the next suggested gear-shifting control strategy where another speed reduction control strategy is suggested.

#### 1.6.0.2 Dynamic speed variation via gain-scheduling of the PD controller

Gain scheduling is a common approach for improving the control of a nonlinear process. It involves the application of different controller tuning parameters as a process transitions from one operating range to another (to be understood in our situations from a change in the curvature of the track). In this way, the controller gains and the robot nominal speed can be more closely aligned with the sharp curve of the track.

For the control design, the following basics of gain scheduling is good to keep in mind:

- Operating range: modern controllers are sophisticated devices, and most are capable of performing advanced control functions such as gain scheduling. A lookup table is commonly used to register the different PID tuning parameters that correspond with a process typical operating ranges. A linear interpolation function is used applied so that the control strategy incrementally adjusts tuning parameters as the process steadily moves from one range of operation to the next. tools to automatically adjust The PID settings are usually adjusted based on error values or a moving window of the error.

- Testing requirements: to determine the controller?s lookup table, it?s necessary to conduct numerous tests at different ranges within a process?s design level of operation and to calculate the associated tuning parameters. Tests will produce values for the controller parameter and nominal speed values suitable for use with the controller.

- Common Practice: while gain scheduling involves a minimum of two sets of tuning parameters, more often than not three different sets are registered within the control system. That quantity is most frequently applied in industry, as it typically accommodates the nonlinear behavior of the process. That said, the ultimate goal of gain scheduling is to provide more precise control so the number of sets needed is subjective ? it depends on the application at hand.

Gain scheduling is an effective approach for overcoming the linear nature of the everyday PID and improving the control of nonlinear production processes. And although the PID may have its limits, approaches such as gain scheduling are a reason why PID control continues to play a dominant role in regulatory control across many industries.

Compute the accumulative errors $AE(k)$ over a moving window of a given width $N_{sr}$ (say 2 or 3 for example).

$$AE(k) = \sum_{i=0}^{N_{sr}} |\varepsilon(k-i)|$$

Then, based on thresholds on the accumulative errors, a set a limited ranges/intervals of operation (3 to 5 at max) can be defined with appropriate PD gains and nominal speed value.

It is advised to first start with two simple range of operations (if $AE(k)$ <Threshold then NominalSpeed=300,Kp= and Kd= else NominalSpeed=100,Kp= and Kd= ).
Good luck !

Gather the performance and tuning values of the best gain-scheduling controller in a Table similar to Table 1.2.

## 1.7 Summary and reflections

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

# Appendix

## The 3pi+ 32U4 robot in detail

### 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.

### The line sensor

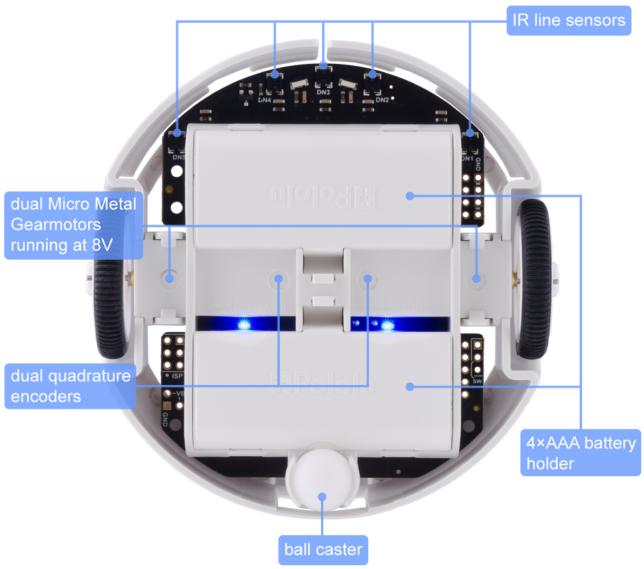


Figure 1.7: Underneath view of the 3pi+ 32U4 robot. The line sensor array are placed at the front.

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 1.7. 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 ±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.

## Troubleshouting with the 3pi+ robot

If your 3pi+ robot does not respond anymore, apply the following procedure to reset the robot. Your can also get further information at:
`https://www.pololu.com/docs/0J83/all10`

**The uploading-before-bootloader method**

The goal of the uploading-before-bootloader method is to select a non-existent serial port in the Arduino IDE and then make sure the Arduino IDE enters the uploading phase before the microcontroller goes into bootloader mode. This method has been tested on Arduino 1.0.5-r2 and 1.6.0. This method does not work on Arduino 1.5.6-r2 because that version of the IDE gives a fatal error message if the selected serial port is not present at the beginning of the uploading phase (e.g. “Board at COM7 is not available.”).

1. Connect the device to your computer via USB.
2. In the “Tools” menu, open the “Board” sub-menu, and select “Pololu A-Star 32U4”.
3. In the “Tools” menu, open the “Port” sub-menu, and check to see if any ports are selected. If the “Port” menu is grayed out or no ports in it are selected, that is good, and you can skip to step 6.
4. Reset the board twice to get the board into bootloader mode. While the board is in bootloader mode, quickly select the new serial port that corresponds to the bootloader in the “Port” menu.
5. After 8 seconds, the bootloader will exit and attempt to run the sketch again. Wait for the bootloader to exit. Verify that either the “Port” menu is grayed out or no ports in it are selected.
6. Click the Upload button. The Arduino IDE will compile your sketch and start uploading it.
7. As soon as the large status bar near the bottom of the IDE says “Uploading…”, reset the board twice to get into bootloader mode.

The Arduino IDE will stay in the uploading phase for 10 seconds, waiting for a new serial port to appear. Once the serial port of the bootloader appears, the Arduino IDE will connect to it and send programming commands.

# 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 |

| | | |
|---|---|---|
| time-delay | : | retard pur |
| time-invariant | : | invariant dans le temps |
| to steer | : | diriger |
| transient response | : | réponse transitoire |
| throttle | : | accélérateur |
| undamped | : | non amorti |
| undamped natural frequency | : | pulsation propre non amortie |
| underdamped | : | sous-amorti |
| yaw angle | : | angle de lacet |