Line Follower Robot

Group 2, Robot project

Abstract

The purpose of the project is to create a line follower robot. This report will provide a design, model, Simulink simulation, construction for the line follower robot and it deals with the development of the Sensing and Control system to allow the robot to follow the line and to avoid obstacles, using infrared sensors to detect the line and distance sensors to detect obstacles.

|  |  |
| --- | --- |
| **Supervisor:** | Henrik [Møller](https://www.campusnet.dtu.dk/cnnet/participants/showperson.aspx?id=45644&elementid=462277) |

|  |  |
| --- | --- |
| Student name | Student number |
| Abed Shoka | s137378 |
| Usama Kokaly | s137231 |
| Riccardo Miccini | s137345 |
| Ali Al-Moathin | s127635 |

Table of Contents

[1. Introduction 3](#_Toc408504446)

[1.1 Problem formulation 3](#_Toc408504447)

[1.2 Requirements 3](#_Toc408504448)

[1.3 Milestone plan 3](#_Toc408504449)

[1.4 Risk management 4](#_Toc408504450)

[1.4.1 Hardware part 4](#_Toc408504451)

[1.4.2 Software part 5](#_Toc408504452)

[1.4.3 Schedule part 5](#_Toc408504453)

[1.5 Resources 6](#_Toc408504454)

[1.5.1 Hardware 6](#_Toc408504455)

[1.5.2 Software 6](#_Toc408504456)

[2. Problem solution 7](#_Toc408504457)

[2.1 Scope of the project 7](#_Toc408504458)

[2.1.1 Extra feature 8](#_Toc408504459)

[2.2 Electronic part 8](#_Toc408504460)

[2.2.1 Sensor board 9](#_Toc408504461)

[2.2.2 DC Motor 10](#_Toc408504462)

[2.2.3 H-Bridge board 13](#_Toc408504463)

[2.2.4 MCU board 15](#_Toc408504464)

[2.2.5 XBee 16](#_Toc408504465)

[2.2.6 Distance sensor 17](#_Toc408504466)

[2.3 Software part 17](#_Toc408504467)

[2.3.1 MCU software 17](#_Toc408504468)

[2.3.2 Java software 20](#_Toc408504469)

[2.4 Mechanical part 23](#_Toc408504470)

[2.5 Final design 24](#_Toc408504471)

[3. Control system 26](#_Toc408504472)

[3.1 Introduction 26](#_Toc408504473)

[3.2 Feedback control 26](#_Toc408504474)

[3.3 PID controller 27](#_Toc408504475)

[3.3.1 PI controller 28](#_Toc408504476)

[3.4 Simulink loop and simulation 28](#_Toc408504477)

[3.4.1 Simulation 30](#_Toc408504478)

[4. Conclusion 30](#_Toc408504479)

[4.1 Product assessment 31](#_Toc408504480)

[4.1.1 Product specification 31](#_Toc408504481)

[4.2 Process assessment 31](#_Toc408504482)

[5. Bibliography 32](#_Toc408504483)

[6. Appendix 33](#_Toc408504484)

[6.1 Gantt chart 33](#_Toc408504485)

[6.2 Electronic schematics 34](#_Toc408504486)

[6.2.1 Sensors boards schematics 34](#_Toc408504487)

[6.2.2 H-bridge board schematic 34](#_Toc408504488)

[6.2.3 MCU board 35](#_Toc408504489)

[6.3 PCB design 36](#_Toc408504490)

[6.3.1 Sensors boards schematics 36](#_Toc408504491)

[6.3.2 H-bridge board schematic 37](#_Toc408504492)

[6.3.3 MCU board 37](#_Toc408504493)

[6.4 Code 38](#_Toc408504494)

[6.4.1 MCU 38](#_Toc408504495)

[6.4.2 Java 50](#_Toc408504496)

[6.5 Control system 66](#_Toc408504497)

[6.5.1 Control system block diagram 66](#_Toc408504498)

# Introduction

This report aims to describe the process of designing, implementing, and testing a fully functional line follower robot, given a set of given requirements and guidelines.

## Problem formulation

A line follower robot is to be analyzed, designed, implemented, constructed, tested and documented. The robot must be able to follow a black tape shaping a path, avoid obstacles in order to reach its final destination and perform some specific action at certain points which will be included in the requirements part. An extra feature should be implemented and delivered along with the robot.

## Requirements

* The robot must find a path made of 5 cm wide black tape.
* The robot must cross the line and follow it onwards, either from the left or the right side.
* The robot must find the path despite eventual gaps.
* On the first perpendicular strap, the robot must make a half circle with a diameter of 50 cm and get on the track again.
* On the second strap, the robot must speed up
* On the third strap, the robot must stop for 3 seconds and then continue forward.
* The robot must detect and coast a wall until it reaches the final stop mark.

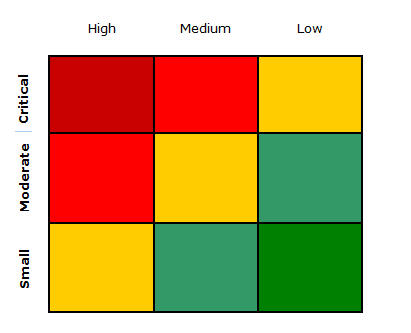
## Milestone plan

The project has been divided into separate modules, which could be carried out independently. Each module, or block, has then been designed, implemented, and tested, according to the time plan

* Software part
* Java program
* Microcontroller program
* Electronic part
* IR sensors board
* DC motors handling
* H-bridge board
* MCU board
* X-Bee communication
* Mechanical part
* Testing and tuning
* Report

## Risk management

The risks in this project may concern three different aspects: the hardware, the software and the schedule. For each part a list of possible risks has been compiled, together with an assessment of their likelihood and the impact they would have on the system. The table below shows a risk map which helps to quickly evaluate to what extent each specific situation would affect the system



### Hardware part

|  |  |  |  |
| --- | --- | --- | --- |
| Risk | Probability | Impact | How to relieve the risk |
| PCB manufacturing errors | Medium | Critical | Should have an overall idea of the role of each PCB, how they should communicate together and how many pins will be reserved for each PCB, in order to avoid extensive versioning |
| Components damage | Medium | Moderate | Test and measure each component before use |
| Connections | Low | Small | Test the hardware with ad hoc software test bench to see if it’s working properly |
| Physical disconnection between the different modules | Low | Critical | Ensuring that the cables and connections are stable before executing the system, can help to reduce the risk. |
| Power overload | Low | Moderate | 1. Add decoupling capacitors on power rails  2. Avoid short circuiting the components |

### Software part

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Risk | Probability | | | Impact | How to relieve the risk | |
| Pc crash, electricity cut off, system crashes (operating system, java runtime) | | Medium | Moderate | | | Pack up power supply  Test the software on more than one PCs |
| Wireless communication issues | | Medium | Critical | | | Ensure that there is no other device using the USB port.  Add a delay between the data transmission to make sure that the java program can handle these data in good time and not overload. |

### Schedule part

|  |  |  |  |
| --- | --- | --- | --- |
| Risk | Probability | Impact | How to relieve the risk |
| Project deadline | Low | Critical | Try to finish one or two weeks before the deadline |
| Group mates commitment | Medium | Critical | To warn them, have consistent meetings |
| Work load between the group members | Low | Critical | To split the work evenly and focusing on everyone’s capabilities. |
| Dropouts | Medium | Critical | Have to say to others in a good time before dropping out |
| Lack of knowledge among the group in matters concerning Control Theory | Low | Medium | 1. Follow the courses 2. Seek assistance from the teacher |

## Resources

### Hardware

* Laser cutter for chassis production
* Laboratory equipment (power supplies, oscilloscopes, multimeters)
* Acid etching machine and UV light chamber for PCB development
* Battery charger and 20 pieces of AA batteries
* Hot glue gun, heavy duty glue, hand tools
* X-Bee modules (XB24-Z7WIT-004)
* Sharp 2D120X distance sensor

### Software

* Blender
* Adobe Illustrator
* Mat lab
* Eagle
* NetBeans
* Sublime Text

# Problem solution

In order to build such a robot some steps have been made and concluded as follow:

* Group formulation consisting of four students.
* Brainstorming the ideas and deciding the best way to construct the robot.
* Deciding what the extra feature was going to be.
* Splitting the different tasks amongst the group members.
* Making a time plan.

The project has been divided into three main stages (Electronic design, Chassis design, Software implementation), each one consisting of different parts and tasks, as per milestone plan.

## Scope of the project

The way in which the different aspects of the robot have been planned can be seen in the block diagram below:

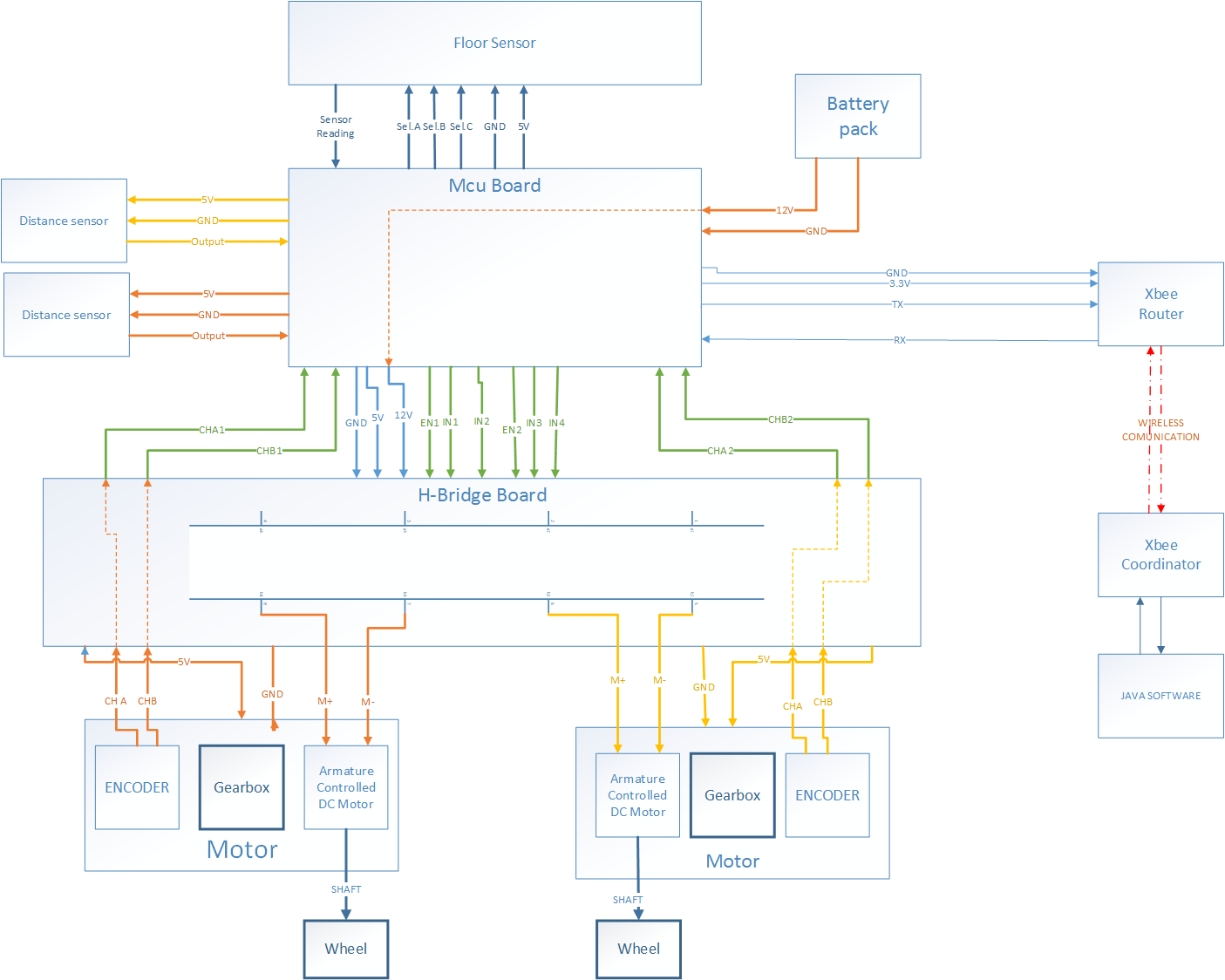


Figure 1 block diagram shows the different parts in this project

As it can be seen in the block diagram, the system is composed by the “Robot” and the “Laptop”, interchanging data through a wireless communication.

The selected microcontroller for our robot is the ATmega328P. The power given to the DC motors will be provided by the MCU through the H-Bridge board, based upon the designed control system, which uses the output from the 8 sensors placed by the lower surface of the robot. The control system and all the other functionalities of the robot (except for the wireless communication) are implemented in C++, taking advantage of several development tools (Wiring/Arduino libraries and AVR libc libraries). The main loop parameters and operating values will be displayed in a JAVA interface through a wireless communication provided by the X-Bee modules.

In addition, as part of the extra feature described below, the robot can be controlled and guided through the Java software which will send the instructions, once again, through the X-Bee modules.

### Extra feature

The main idea of the additional feature was to develop a program that can help us to debug the performance of our robot via wireless. This has been successfully achieved by implementing a JAVA application running on the “laptop” side and using X-Bee modules.

More details about the device, the program and its features, will be thoroughly explained in the subchapters below.

## Electronic part

The Electronic part consists of several electronic boards, which have their own role. These are the following:

* 2x sensor boards (each one bearing 4 sensors)
* H-Bridge board
* MCU board

The first twos provide all the data required by the processing apparatus (the MCU board), which uses such information to compute the most correct outputs for the motors.

### Sensor board

The sensor board can be seen as the first source of information, by which the robot is able to know whether it is on the track. It is composed of two sub-boards (due to board sizes limitations in the software used), equipped with 4 reflective IR sensors.

Each sensor consists of an infrared emitting diode (IR transmitter) and an NPN silicon photo transistor (IR receiver) mounted side by side in a black plastic housing. When a sensor is over the black tape, IR waves are continuously absorbed by the black surface and the lack of reflected waves turns the output high. However, IR waves are reflected by the lighter floor surface, turning the output low.

|  |  |
| --- | --- |
| The IR transmitter is similar to an LED. An operating voltage of 5 V has been chosen, and in order to provide about 25 mA, a 150 Ω resistor is placed in series. A 10K resistor is placed between VCC and the IR receiver. The output voltage is read across the collector and emitter of the IR receiver. | Figure 2 Sensor Schematic |

The use of 8 sensors means that 8 microcontroller analog inputs would be required: in order to decrease such number, an analog multiplexer has been employed. In this way the number of reserved pins for the sensors dropped to 4, with 3 of them being multipurpose digital ones. The channel select inputs in the multiplexer determines which one of the analog inputs is to be expected in the output. When the enable pin is high, all analog switches are turned off (however, this pin is hard-wired to ground, as such condition is never desired). The table below shows how the multiplexer works.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Inputs** | | | | **Output Sensor** |
| **Enable** | **Select** | | |
| **C** | **B** | **A** |
| L | L | L | L | 1 |
| L | L | L | H | 2 |
| L | L | H | L | 3 |
| L | L | H | H | 4 |
| L | H | L | L | 5 |
| L | H | L | H | 6 |
| L | H | H | L | 7 |
| L | H | H | H | 8 |
| H | X | X | X | None |

Table 1 Functional Table, X=don’t care

### DC Motor

The DC-motor is an essential part of the robot. The model used in this project is the Faulhaber 2233-012S, which is a 12 V armature-controlled motor. It is very important to gain control over the motor by understanding the datasheet that comes with it, as it provides all the parameters needed by the simulation and control system implementation. A total of two motors have been used, one for each steering wheel.

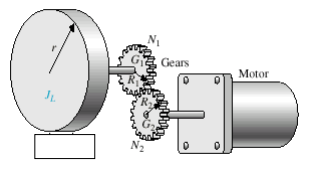


Figure 3 motor gears

The supplied motors came with a built-in gearbox with a reduction ratio of 17.2, in order to provide more torque. The cost of such high torque through a gear-system is a reduction in rotational speed.

The DC-motor includes an encoder composed of two channels which are used to determine position, velocity and direction of the DC-motor. Each encoder channel yields 15 pulses per motor round, giving 15·× 17.2 = 258 pulses per shaft round. The two channel signals are phase shifted at about 90°, giving the opportunity to determine the rotational direction of the motor by looking at one signal and then checking if the other is high or low. Although the secondary channels have been routed to microcontroller inputs, the aforementioned feature has not been used.

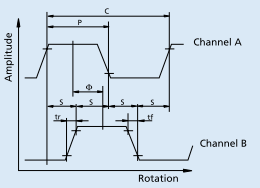


Figure 4 encoder pulses output

#### PWM

Since the motors are fed with their nominal voltage (12 V), PWM technology is used to obtain a range of different speeds. PWM stands for Pulse Width Modulation, and is a technique that uses a square wave with fixed frequency and varying duty cycle (the ratio between the on-time and the time period) in order to generate signals that can mimic analog values, whose value can be seen as the carrier (the pulsating signal) high voltage times the duty cycle. This method is used in several fields ranging from voltage regulation, audio amplification, and data transmission. PWM capabilities are embedded in the chosen microcontroller, which allows for up to 6 channels. Each channel has a resolution of 8 bits, meaning that 256 (from 0 to 255) different levels can be obtained. The carrier frequency is internally generated using the built-in timers, and can be changed by tweaking the prescaler registers (TCCR0B and TCCR2B).

#### Transfer function of the DC motor

The transfer function for the DC-motor is needed for making simulations of the whole system in order to predict its behavior. All the parameters has been taken from the datasheet and the weight of the robot has been found by weighting the robot on a digital scale and is fund to be 1.565 Kg.

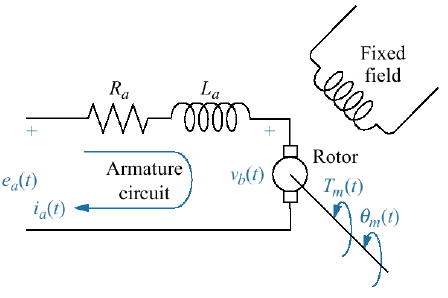


Figure 5 DC motor schematic

The following parameters have been obtained from the motor’s datasheet:

The weight of the robot is = 1.565 Kg

The wheel radius is = 3 cm

The transfer function for the robot with the load is:

Kmotor = 70.23

τ = 132 ms

#### DC motor step response

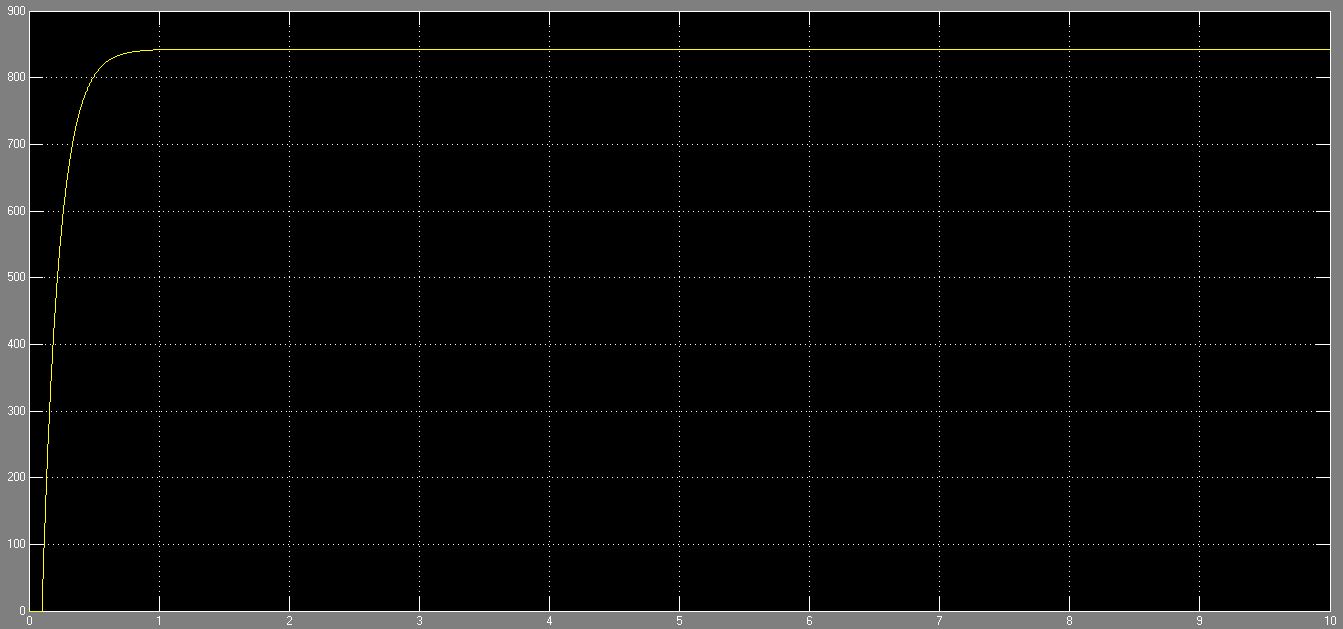


Figure 6 Robots transfer function step response

### H-Bridge board

Since the microcontroller alone is not capable of delivering the amount of power needed by the motors, an H-bridge driver has been employed. The device used is the L298 dual full-bridge driver, which comes in an integrated monolithic circuit. The general schematic of an H-bridge can be seen in the picture below (it can be seen that the name comes from its appearance):

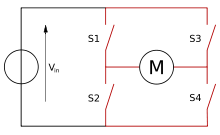


Figure 7 H-Bridge switch schematic

The IC is composed of a series of transistors acting as switches (simplified as S1, S2, S3 and S4 in the picture above), that can provide different paths for the current flow across the motor terminals.

The overall circuit board also includes “flyback” diodes, used to protect against spikes generated by the motor inductive load. It also provides routing for the encoder terminal, in order to optimize the number of interconnection amongst the different apparatuses.

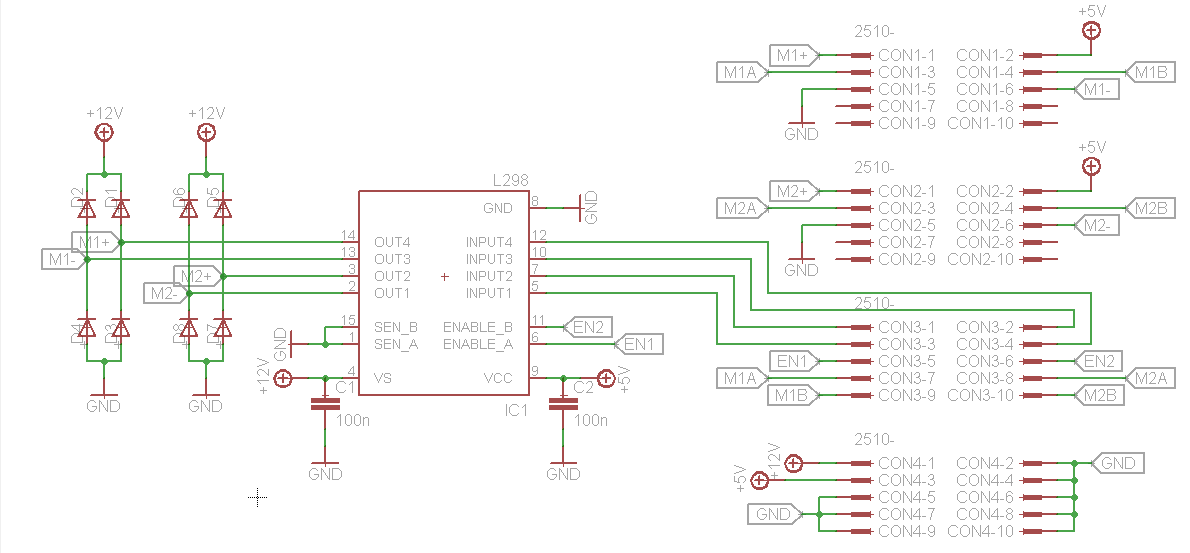


Figure 8 L298 H-Bridge schematic

The motor terminals are marked as Mx+ and Mx-and the enable signals are marked as ENx. The connectors shown in the schematics serve the following purposes (from top to bottom): connection from motor 1, connection from motor 2, connection to the MCU board, power supplies.

The following table sums up how the motor behaves according to its stimuli (input 1, input 2, enable):

|  |  |  |  |
| --- | --- | --- | --- |
| Enable | Input 1 | Input 2 | Operation |
| 0 | X | X | inertial free running |
| 1 | 0 | 1 | motor moves backward |
| 1 | 1 | 0 | motor moves forward |
| 1 | same value | | Robot stops |

Table 2 The behaviour of the motor using the H-Bridge

### MCU board

The MCU board is the main component of the robot, as it provides the computation needed to drive the motors, using the data from the sensors. Its most important component is the microcontroller.

Despite the staggering selection of microcontrollers that could have been used to fulfil the project requirements, it has been agreed to use ATmega328P because of its fine specifications. The key cons are the following:

* low power consumption
* high clock speed (16 MHz, up to 20)
* 32 KB flash memory for program storing
* 1 KB EEPROM memory for user-defined data
* 2 KB SRAM memory for temporary, runtime variables
* Highly compatible with several developing platforms (Arduino, Wiring, AVR libc…)
* 2 external interrupts, 3 timer interrupts
* Built-in USART, ADC (analog-to-digital conversion, with 10 bits resolution) and PWM capabilities

One of the main drawbacks is the scarcity of general-purpose I/O pins, which has been a major constraint in the PCB design. Besides that, the use of a 16 MHz clock frequency could lead to issues in the stability of the power rails, because at high frequencies most electrolytic capacitors would build up inductive/resistive properties. This issue has been addressed by adding an extra decoupling capacitor across the microcontroller supply.

Apart from the microcontroller circuitry, the MCU board generates, supplies, and routes the operating voltages used by the different apparatuses:

* 3.3 V used by the XBee module
* 5 V used by the MCU, the motor encoders, the H-bridge logic, and the sensors
* 12 V used by the H-bridge power side

The first two voltages are obtained through two different kinds of voltage regulators, whereas the latter comes directly from the batteries (although a 0.7 V drop occurs, due to the power socket protection diode).

Apart from the aforementioned features, the rest of the circuitry in the MCU board consists of:

* reset button for the microcontroller
* programming port
* LEDs for the “heartbeat” and power
* supply switch

A detailed schematic in full A3 format can be found in the appendix.

### XBee

The XBee is a radio module that provides point-to-point wireless communication between two or more nodes. It is based upon the IEEE 802.15.4 protocol and works within the ISM 2.4GHz frequency band. It features very low power consumption (40 mA @ 3.3 V) and can transmit within a range of 30 m.

In order to establish a communication between the robot and the laptop two modules are needed, one working as coordinator and the other one as router. The XBee has been set to transmit data at 9600bps, in order to obtain the maximum reliability.

None of the extra functionalities of the modules have been used (star-mode broadcasting, packet handling, sleep mode, built-in I/O and ADC, flow control, just to name a few) since it would have been beyond the scope and purpose of the project.

The following setup has been adopted:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| M:\Miccio\Dropbox\Camera Uploads\2015-01-07 21.35.31.jpg | This XBee module has been configured in order to work as router, and it is connected to the robot.  Only 4 pins are needed, and they are connected to MCU board in the following way:   |  |  |  | | --- | --- | --- | | **X-BEE** | | **MCU board** | | **Pin number** | **Pin name** | **Pin Name** | | 1 | VCC | 3.3 V | | 2 | TX | RX | | 3 | RX | TX | | 10 | GND | GND | |
| M:\Miccio\Dropbox\Camera Uploads\2015-01-07 21.34.25.jpg | The coordinator module is connected to an adapter which acts as a virtual serial interface and facilitates the connection with the laptop. |

### Distance sensor

In order to avoid hitting obstacles (mainly walls) when the robot is on its way to the final stage, two IR SHARP 2d120x distance measuring sensors with integrated signal processing and analog voltage output have been used.

These distance sensors operate at 5 V and the closer the obstacle, the higher the analog output voltage will be.

The sensors have been placed on the front and left side of the robot, facing outwards, and are directly wired to the main MCU board.

## Software part

### MCU software

The MCU firmware provides the robot with all its functionalities and computation capabilities.

The software has been developed in C++ in order to take advantage of several syntactic “sugars” (strings handling, Boolean data type…), although it features an imperative-oriented paradigm rather than an object-oriented one.

The code makes intensive usage of several libraries coming from the Wiring and Arduino platforms, in particular for what concerns the EEPROM management and the Serial communication (both part of the extra features set). Shortcuts for ports and pins initialization, querying, and handling have also been adopted.

In order to be able to be able to upload the software through a 2-pin serial connection, a bootloader has employed. The bootloader takes up approximately 2 KB of flash memory, and allows the robot to be programmed through a USB-to-TTL conversion interface, commonly known as “FTDI cable”.

The software overall structure can be summarized in these parts:

* Variables and functions declaration
* Initialization
* Repeating part
  + Sensors reading
  + Motors control system
  + Remote control listener
  + Path management
  + Debug transmission routine
* Motor speed assessment routine

#### Sensors reading

The sensors are read one at a time through the multiplexer interface described in the previous chapters. Since the sensors are connected to one of the ADC pins, their values are normalized in a range from 0 (tape full detection) to 10 (floor full detection) using fine-tuned, individual thresholds. The full set of 8 values is then used to calculate 2 parameters: one is simply given by the sum of the sensors values, and the other one is instead the weighed sum. The weights used are 8 numbers, which are additive inverse in pairs and ever-increasing (ex: -8 -4 -2 -1 1 2 4 8).

These two values will be later used by the path handling system. The input for the control system is however calculated as:

This algorithm is found to have a linear output range – compared to simply taking the weighted sum – even when the line is detected by more than 2 sensors.

#### Motors control system

This part of the software provides a discrete implementation of the control system described in chapter 3 of this report. It makes use of single-precision floating point data for the computations, and its iteration is clocked internally in order to have a constant loop frequency (which can be changed remotely).

#### Remote control listener

This part monitors the serial port for incoming data. If a command is present, a dispatcher performs the required function or changes the required parameter. The commands are read using a specific protocol, composed of a source (computer), a destination (left motor, right motor, sensors…), a parameter, and a new value. This routine is optional and can be disabled during the setup.

#### Path management

Since the path involves dealing with a series of extra actions, a piece of code responsible for its handling is needed. The path is divided into several stages, each triggered by a set of conditions (mostly given by the two sensors parameters described above, and the outputs from the distance sensors). Once a stage is entered, a set of actions is performed. Such stage is then said to be cleared, and gets removed from the dispatcher.

#### Debug transmission routine

Just like the control system, this part of the code is clocked in order to provide a steady and well-timed set of data to use in the computer interface. However, its frequency is much lower, as transmitting data over the UART is a time-demanding task (over 1.2 ms according to our measurements) and running it too often would compromise the consistency and stability of the control system.

#### Motor speed assessment routine

This piece of the software is in charge of spooling new readings of each motor speed. It gets executed every time the external hardware interrupts (INT0 and INT1) linked to the motors encoders are called (namely, when they receive a logic high value) and measures the time interval between the current interval and the previous one, each one representing one encoder line. Since the number of encoder lines/interrupt calls for each wheel revolution is known, the control system can easily use the time interval to calculate the wheel speed. On top of that, an averaging filter is added, in order to get smoother readings.

### Java software

The main aim of this piece of software is to provide a debugging tool for the robot, in order to assess issues and fine-tune its parameters without the hassle of re-updating new code every time. In fact, since the program can communicate wirelessly with the robot, there is no need to disassemble it, saving time and avoiding accidental damages that may have otherwise occurred. A screenshot of the software when the robot is on standby mode can be seen in the picture below:

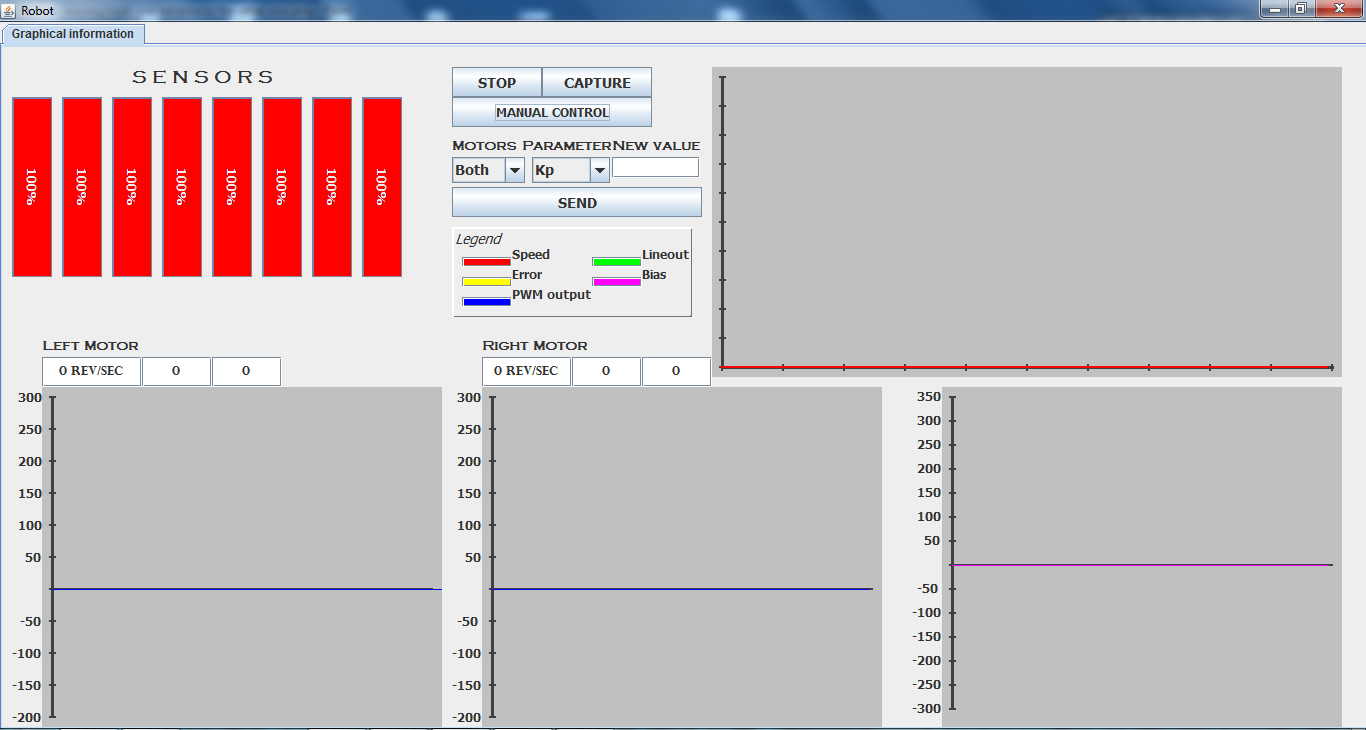


Figure 9 JAVA software screenshot - robot in standby

The following table explains in detail the different parts of the software.

|  |  |
| --- | --- |
| center sensors | The activity of the 8 floor sensors is represented in those bars. Colour feedback is also provided. The picture shows the robot being perfectly alligned. |
| graphic lines  graphlinescentered | This graph uses the sensors information to provide a time-based representation. Each line represent a sensor, shaded according to its position (darkest for the centremost pair, and lightest for the furthermost one). Low level indicates full line detection (as in second picture). |
| options | The START/STOP initializes or terminates the communication with the robot.  The CAPTURE button takes a screenshot (useful for future comparisons).  The other tools will be explained in the following subchapters. |
|  | This is the legend for the following graphs.   * Speed, Error and PWM output are shown in the two motors graphs * Lineout and Bias are shown in the third graph. |
| constant speed | These graphs show some of the main control system signals against time, for each motor (and loop “branch”).   * Red line: shows the speed in RPM (revolutions per minute) * Yellow line: it is the error used by the P-controller and it is affected by the reference and the bias point. * Blue line: it represents the PWM that is fed to the motors, in a range from 0 to 255. |
|  | This graph represents the common information between the two motors:   * Purple line: it is the output from the sensors * Black line: is the output of the I-controller, consisting of the integral sum multiplied by the constant Ki. It takes into account the lineout and a motors synchronization factor. |

#### Parameters fast tweaking

The graphs can help to evaluate how the control system parameters affect the behavior of the robot, allowing for more pondered and effective adjustments.

The table below summarizes a typical workflow:

|  |  |
| --- | --- |
| parametersjava | 1. Chose the destination of the adjustment (left/right motor, shared constants, or sensors) 2. Chose the parameter 3. Type the new value 4. Push send 5. The new value will be sent to the robot and permanently stored in the EEPROM |
| parametersparameters | The parameter list varies according to the chosen destination. The picture aside shows the options for both motors (shared parameters):   * Kp: Proportional constant * Ki: Integral constant * Kw: Lineout weighting factor * Ks: Motor synchronization factor * Fs: Loop frequency * Avg: number of samples considered in the averaging filter for the speed readings * Ref: average robot speed measured in RPM   If “sensors” is chosen, it is possible to change the weights, by typing 4 numbers separated by a space.  If “left motor” or “right motor” is chosen, the Kp constant will be the only parameter listed. |

#### Manual control

The software allows controlling the robot manually. This feature can be activated by clicking on the “MANUAL CONTROL” button.

The command set is the following:

|  |  |
| --- | --- |
| Command | Function |
| ↑ | Move forward |
| ↓ | Move backward |
| ← | Turn left |
| → | Turn right |
| Lowercase ‘s’ | Stop |
| Lowercase ‘i’ | Print current parameters to terminal |
| Plus (+) | Speed up |
| Minus (-) | Slow down |

## Mechanical part

Despite a number of different robots has been seen and introduced by the teacher, they all resemble each other. From here came the idea to make something different and unique.

Our robot chassis design consists of 18 different parts, each part is made of 3.5 mm thick extruded acrylic plastic except for the base plate, which is 5 mm thick. A draft of the robot has been designed using Blender and then arranged into separate pieces. Connection points and seams have been included and after several cardboard prototypes, a final version has been cut using a laser cutter. The design was carefully made to fit each electronic board, distance sensor, motors, wheels and even the battery case in a specific place.

The pictures below show a 3D rendering of the design and an overview of all the pieces spread out:

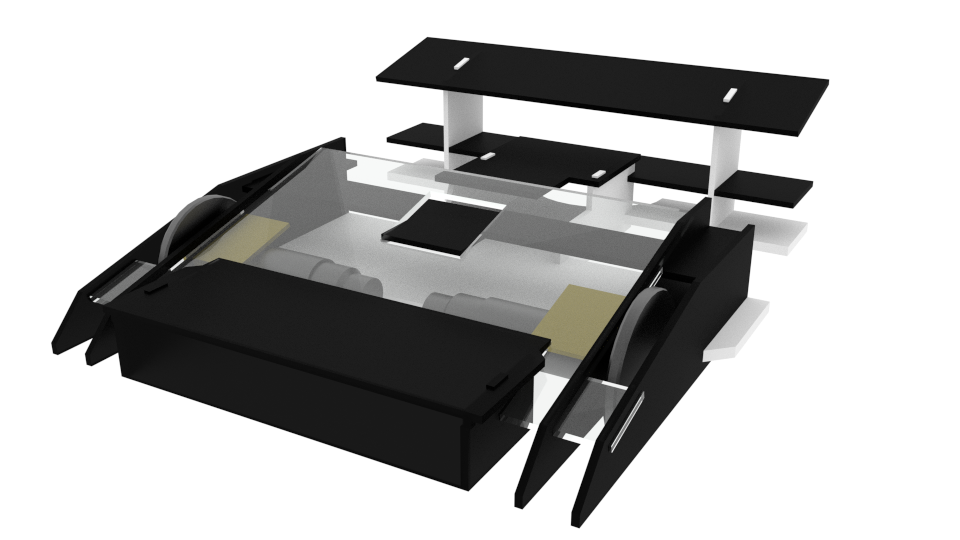


Figure 10 A 3D rendering of the design

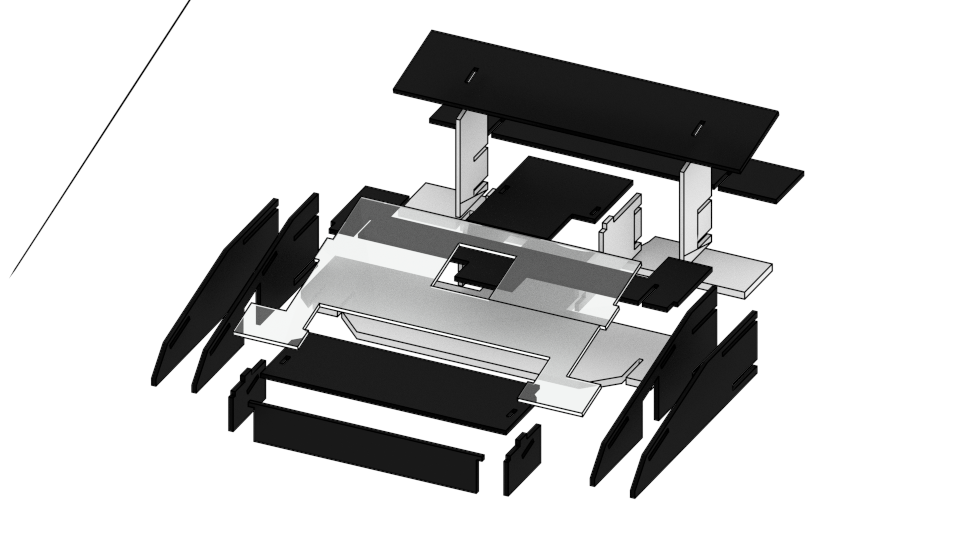


Figure 11 An exploded view of the different pieces

## Final design

After having designed, simulated, and tested all the blocks, a final rig has been assembled. Several other components have been included, in order to address the issues that arose and increase the overall performance of the robot.

Decoupling SMD capacitors have been added across the power rail in the MCU board. This helps smoothing out any possible spike on the power rail, allowing the circuit to run more reliably.

A larger heat sink had to be employed in order cope with the amount of heat generated by the 7805 voltage regulator. In fact, after running the robot for some time it has been found that the old, tinier heat sink couldn’t cope with the dissipation demand caused by the high voltage leap (7 V).

In order to provide a more reliable and safe communication between the MCU and the X-Bee, the transmission line has been scaled down to 3.3 V level logic with a voltage divider. The receiving line does not require any regulation as its 3.3 V level logic is still above the threshold for CMOS-based circuitry.

The following diagram shows the adopted design in its completeness, giving particular emphasis to the physical connection (each plug/cable is coloured differently) between all the boards and apparatuses.

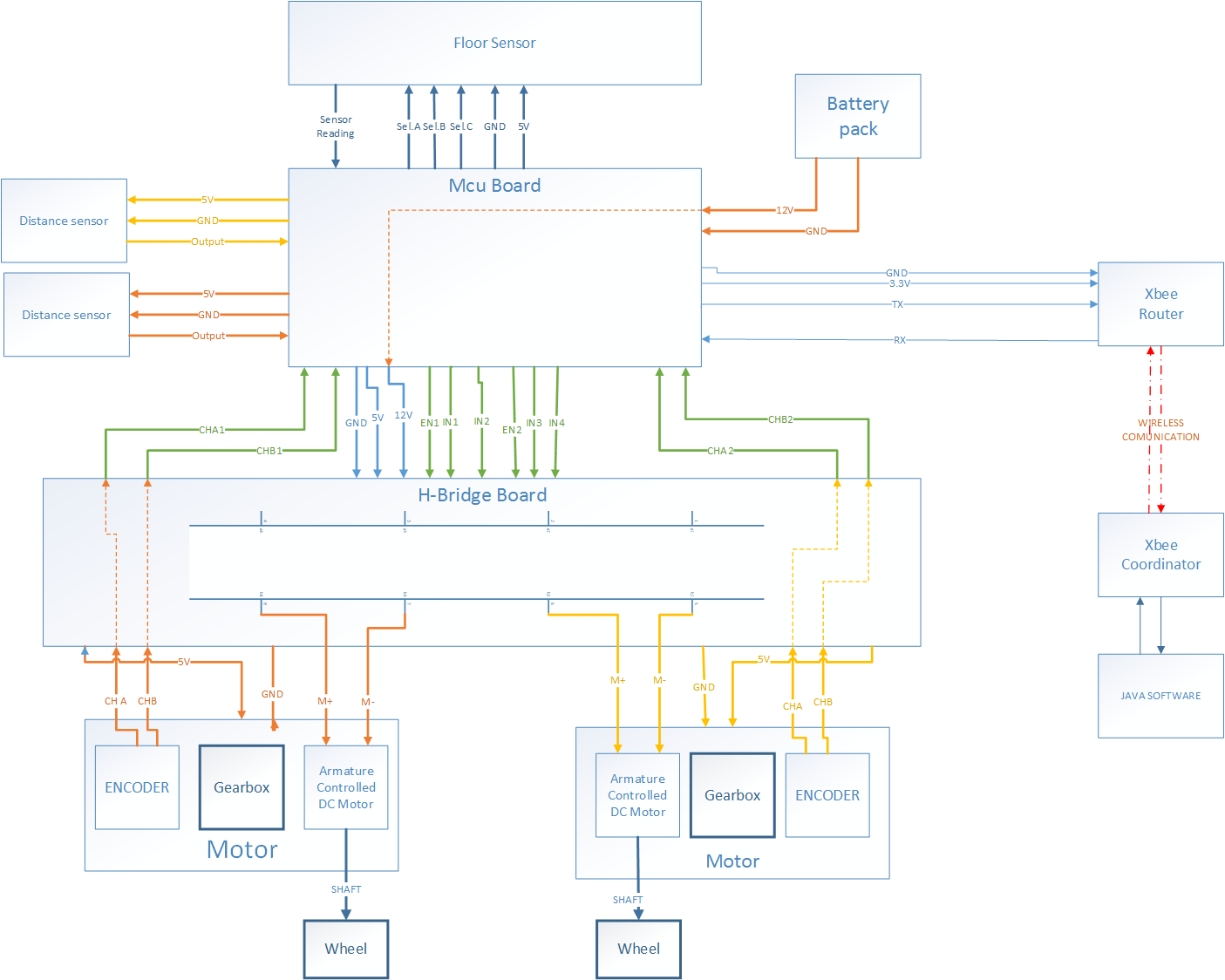


Figure 12 overall design

# Control system

## Introduction

To describe what a PID controller is and why it has been used, we will start by describing more generally what a controller is. An example of open loop system is the following:

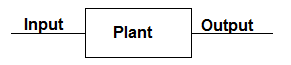


Figure 13 Open loop block diagram

In the picture above, an input acts on a plant or the system to be controlled and an output signal is generated. Often in these open loop scenarios the performance of the system is not good enough to meet the requirements, as an external factor could affect the system and diverge it from its operating point.

For instance, if the robot has to move from one point to another, the open loop commanding could achieve the purpose by sacrificing some accuracy. However, unpredictable events like dirt on one wheel or a slight difference between the motor speeds might interfere with the task. In both cases the robot won’t stop at the designated spot because it has no way of compensating for these errors. A more reliable control can therefore be achieved by providing feedback.

## Feedback control

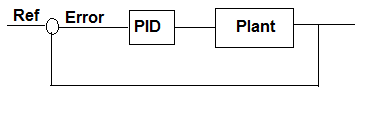


Figure 14 Close loop block diagram

Feedback control means that the output of the plant/system is measured and fed back, allowing the system to make adjustments accordingly. A reference signal representing the desired value - or the ultimate goal - is compared with the measured value: their difference is the error. Such error signal is used to compute a new input for the system.

The main goal of a feedback-based control system is to minimize the error in the shortest time possible. Other requirements in terms of maximum oscillation or settling time may be expressed.

In the robot case, the measured quantity could be its current position whereas the reference represents its desired one. Minimizing the error would mean having the robot at the desired location at any given time.

There are many types of controllers and one of these types is the PID controller.

## PID controller

The PID controller is a simple, efficient and effective in a wide range of applications.

PID stands for proportional, integral derivative. In fact, each of these terms describe how the error signal is processed: the results are then summed and sent to the plant. Its general form is the following:

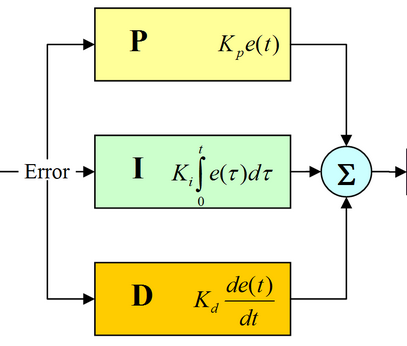


Figure 15 PID controller block diagram

In the proportional path the error signal is multiplied by a constant Kp, in the integral path the error is integrated and then multiplied by Ki, and in the derivative path it is differentiated and then multiplied by Kd.

The three paths are then summed together to produce the controller output, which is fed to the system. The three K constants are called gains and represent the influence that each path bears over the final result, and they have to be adjusted and tuned in order to meet the given requirements.

In order to simplify the design end tuning process, some of the terms can be skipped.

After some research and recommendations from the teacher, it has been agreed to equip the robot with a PI controller.

### PI controller

A PI controller is a special case of the PID controller in which the derivative (D) of the error is not used.

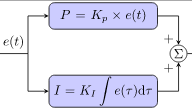


Figure 16 PI controller block diagram

The integral term eliminates the residual steady-state error and substantially decreases settling time in comparison with a proportional-only controller. However, since the integral term is responding to accumulated errors from the past, it can cause the present value to overshoot the set point.

The PI controller has been implemented in the MCU, and the constants have been manually tuned with the help of the JAVA interface.

## Simulink loop and simulation

Simulink is a data flow graphical programming language tool in Matlab, and it has been used to model and simulate our control system.

Our loop model is as follow (a more readable version can be found in the appendix):

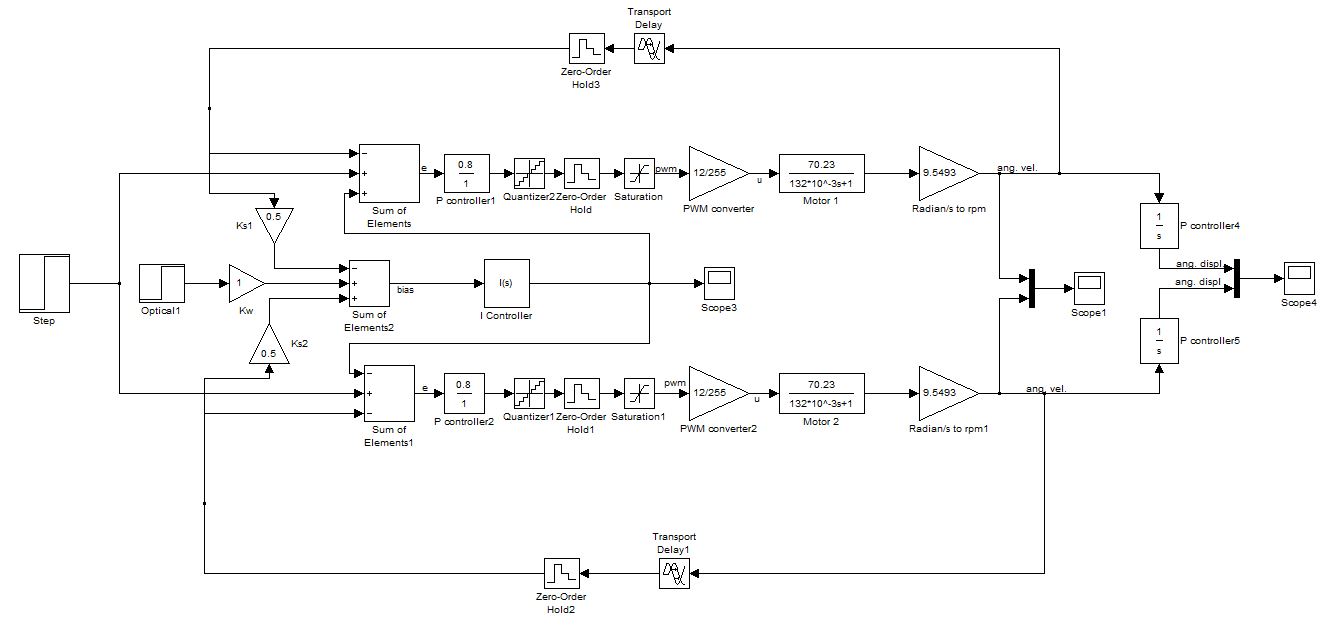


Figure 17 Robot close loop block diagram

The block diagram is symmetrical, and each side branch represents a motor.

The first block is a step-function generator, which provides the reference speed measured in RPM.

It is followed by a summation point, which takes into account the difference between the reference speed and the actual speed (on the feedback branch), and a bias point given by the I-controller (middle branch).

The output of the summation is fed into a P-controller, to which the nonlinearities – used to simulate the limitations of the discrete behaviour of the microcontroller - are then applied. These are

* Quantizer: applies a stair-step function to the input in order to quantize a smooth signal into a series of discrete values. Its output is calculated as: , where y is the output, u is the input, and q is the quantization interval, set to be 1.
* Zero-order hold: it takes an input and holds its value to the output for a specified sampling time.
* Saturation: limits the range of the input to a specific one, which has been set to be from 0 to 255 (PWM range for 8 bits of resolution).

It follows next a gain block which converts the signal in its PWM range into a voltage value suitable for the motor. This block roughly mimics the behaviour of the PWM modulation provided by the microcontroller.

The aforementioned voltage output is then provided to the transfer function of the motor (for a more thorough explanation of how it has been obtained, refer to the previous chapters).

Since the output of the transfer function is expressed in rad/s, a gain block is added, in order to convert it to RPM.

The feedback branch is composed of a zero-order hold and a transport delay block, which delays the signal going through it by a specific time. The latter one is used to simulate the latency of the microcontroller speed readings.

The middle branch provides the signal to the I-controller, and it includes:

* A step generator block called “optical”, simulating the readings from the floor sensors, multiplied by its weighting constant Kw
* The difference between the speeds of each motor, multiplied by a scaling factor Ks

The sum of these signals is then fed into the I-controller, which provides a biasing point for the P-controller calculation. When the “optical” block is off, the bias adjusts the changes in speed of each wheel, helping the robot to go straight.

### Simulation

This is a simulink simulation of the robot control system where “Ref” is the input and the angular velocity (“Speed”) of each motor is the output.

The simulation results don’t quite match the behaviour of the robot in the real environment, due to several reasons:

* Since the motors used for the project are fairly old and have been previously used, it is unlikely that the constants mentioned in their datasheet are still valid
* The load friction has been approximated
* The inductance value has been neglected

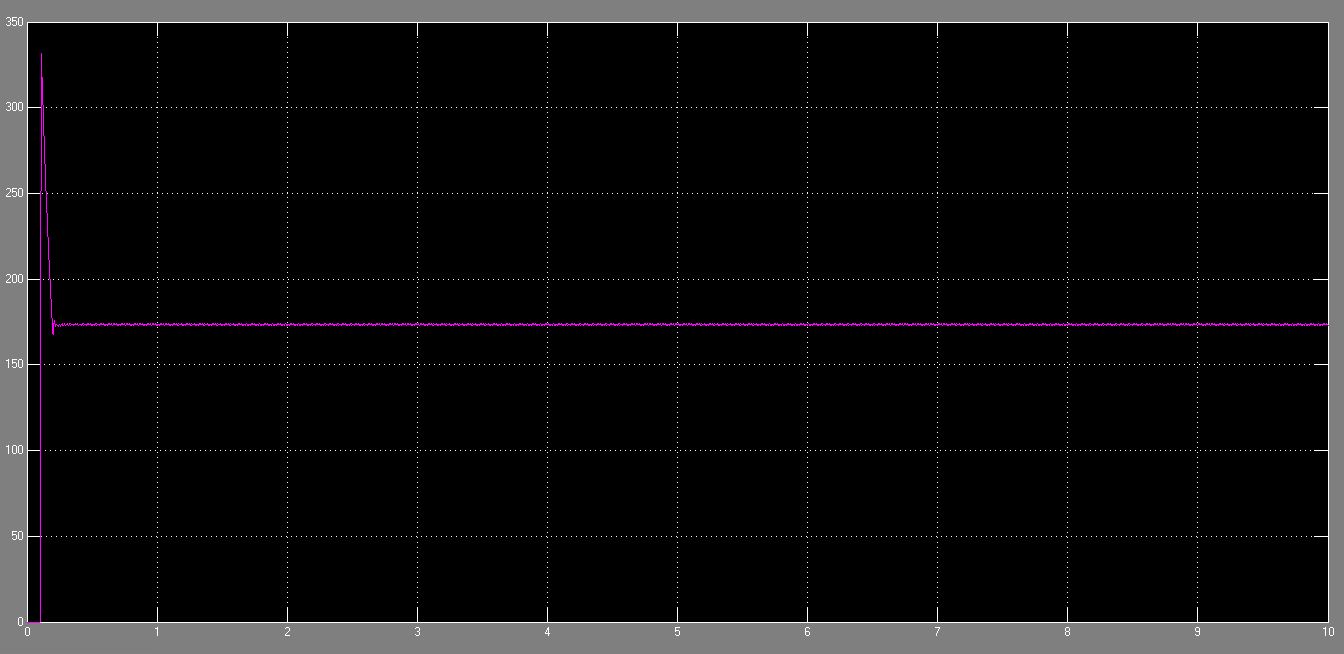


Figure 18 Simulink simulation plot

# Conclusion

Several skills have been gained from the realization of this project. Working with such a big project in a limited time span and manufacturing a viable product was a real challenge, especially for what concerns the usage of new components for the first time (such as the DC motor circuitry and IR sensors). The way the workload has been managed– assigning the tasks according to the area of competence of each member, but continuously sharing ideas and suggestions– allowed us to carry out the project within the given time while gaining precious experience and knowledge.

## Product assessment

The final product fulfils all the functional requirements, and provides a fairly useful set of additional features.

Further improvements may consist in manufacturing printed circuit boards with SMD components, which would effectively increase the reliability of the board, minimizing possible chances of malfunctioning caused by short circuits, bad connections, or common board handling. Providing more sensors for the line detection would also improve the performances, especially if they were laid down in two or several rows. Having more rows of sensors would in fact help to get a better overview of both the line position and orientation, which can’t be accomplished with only one row. Experimenting with different battery types and technologies may have also helped the performances, due to faster recharging cycles - which would have benefited the developing process - and higher current disposal.

### Product specification

* sizes: 30 cm long × 26.5 cm wide × 9.5 cm high
* weight: 1.65 Kg
* operating voltage: 15 V – 10 V
* current consumption: 350 mA (standing still), 500 mA (moving), 800 mA (highest measured peak)
* control system sampling frequency: 100 Hz (up to ≈250 Hz)
* average speed: 120 rpm

## Process assessment

Working according to a time schedule proved to be tougher than expected, mainly because of several major issues that unpredictably occurred throughout the development process.

The poor battery specs is one on such issues, as they were slow to charge and quite unreliable, since they couldn’t respond properly to the high current demand.

The sturdiness of the acrylic plastic used was also another major problem, as it was very prone to accidental fractures and damages, which would often set us back several days in the schedule.

Managing to establish a reliable connection between the two X-Bee modules has also revealed to be a rather time consuming task, mostly due to our lack of knowledge in that area.

# Bibliography

* DC motor notes and datasheet(s)
* IR sensor notes and datasheet
* ATmega328 datasheet
* FTDI cable datasheet
* X-Bee Series 2 datasheet
* Wikipedia entries for: PID-controller, H-Bridge, Rotary Encoder
* Control theory notes

# Appendix

## Gantt chart

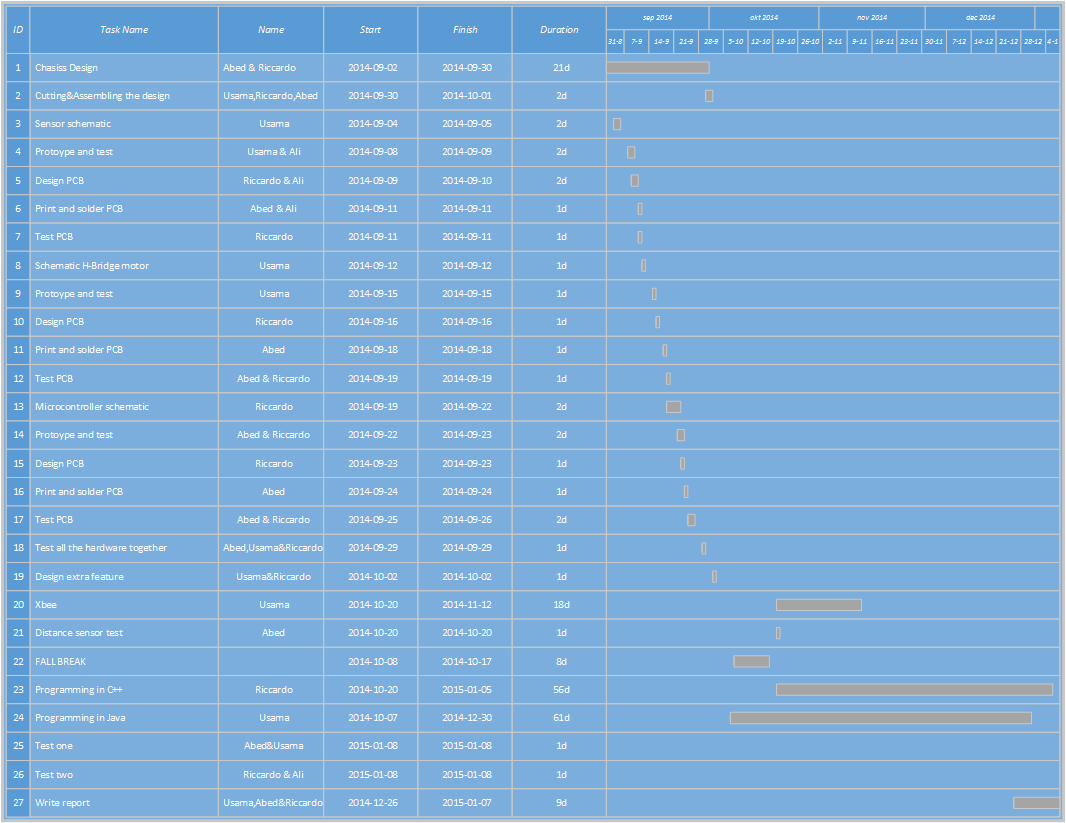


Table 3 Robot project time table

## Electronic schematics

### Sensors boards schematics

|  |
| --- |
|  |
| Figure 19 IR sensor boards schematic |

### H-bridge board schematic

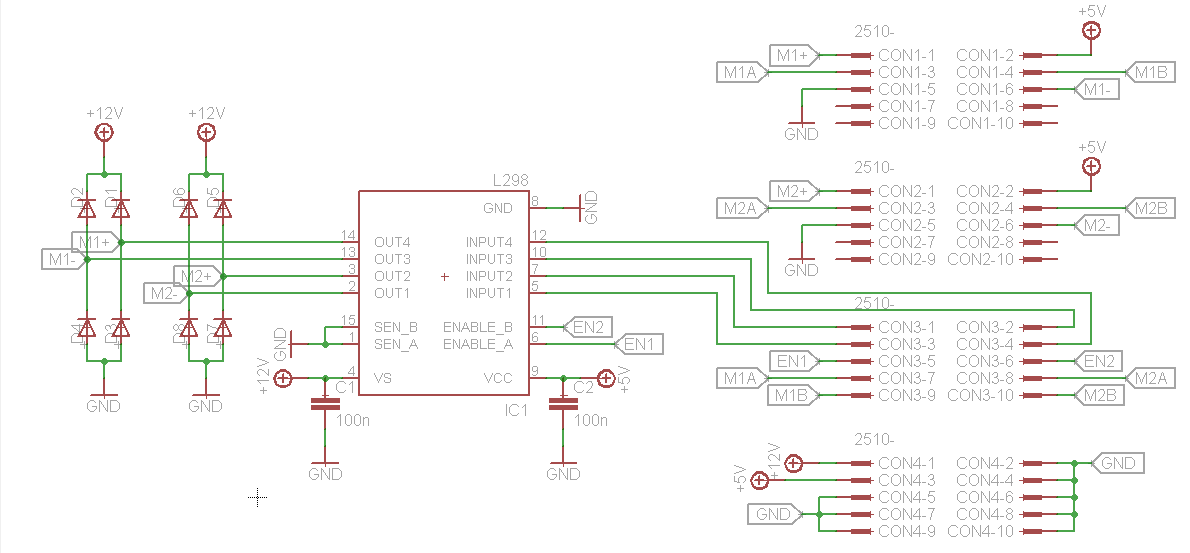
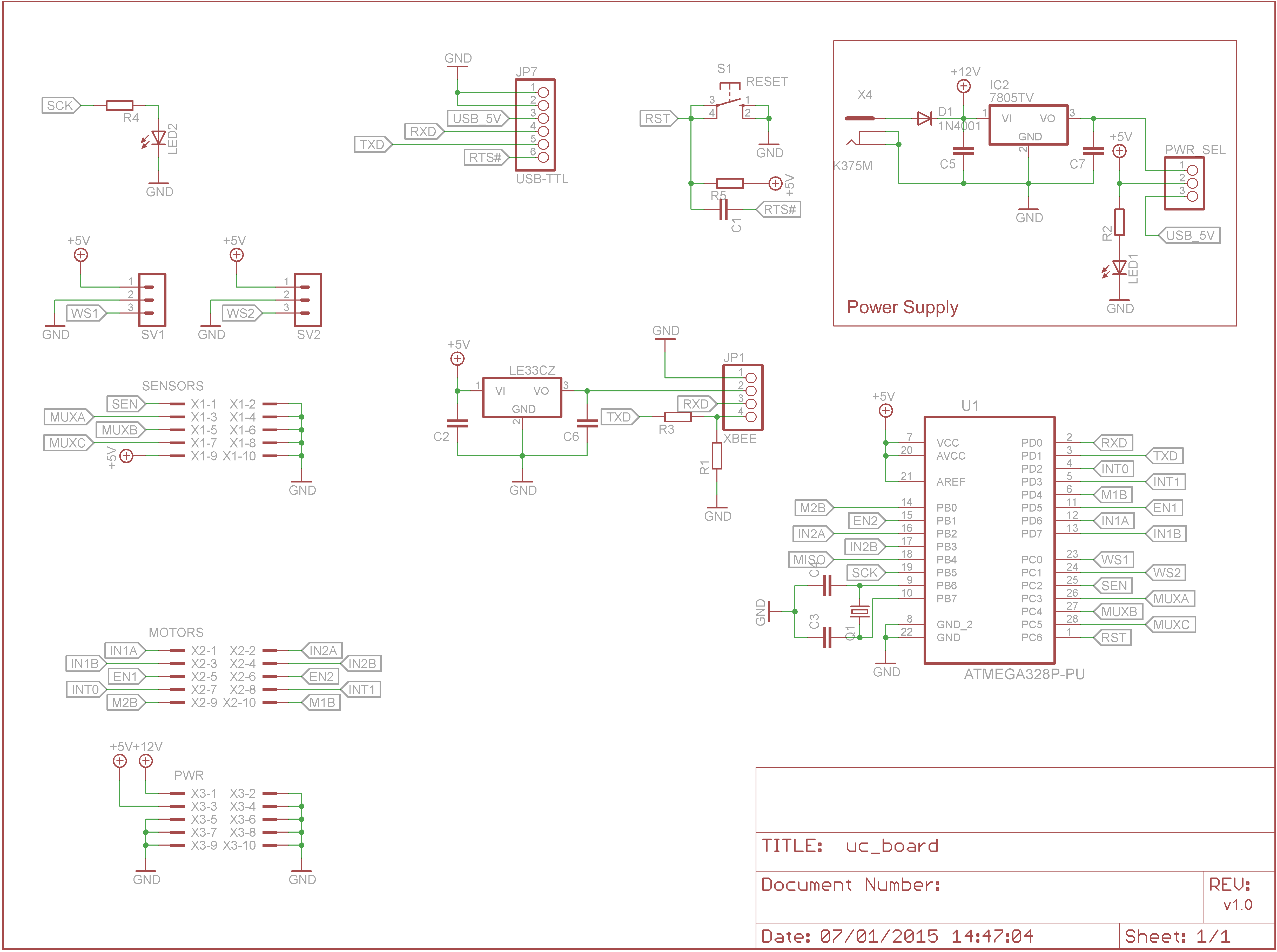


Figure 20 H-Bridge board schematic

### MCU board



## PCB design

### Sensors boards schematics

|  |
| --- |
| M:\Miccio\Dropbox\control theory and project\ROBOT REPORT BITCHES\Walak\robot schematics - pictures plz plz plz\sensor n2 bottom.png |
| M:\Miccio\Dropbox\control theory and project\ROBOT REPORT BITCHES\Walak\robot schematics - pictures plz plz plz\sensor2.png  M:\Miccio\Dropbox\control theory and project\ROBOT REPORT BITCHES\Walak\robot schematics - pictures plz plz plz\sensor3.png  Figure 21 IR sensor boards PCB |

### H-bridge board schematic

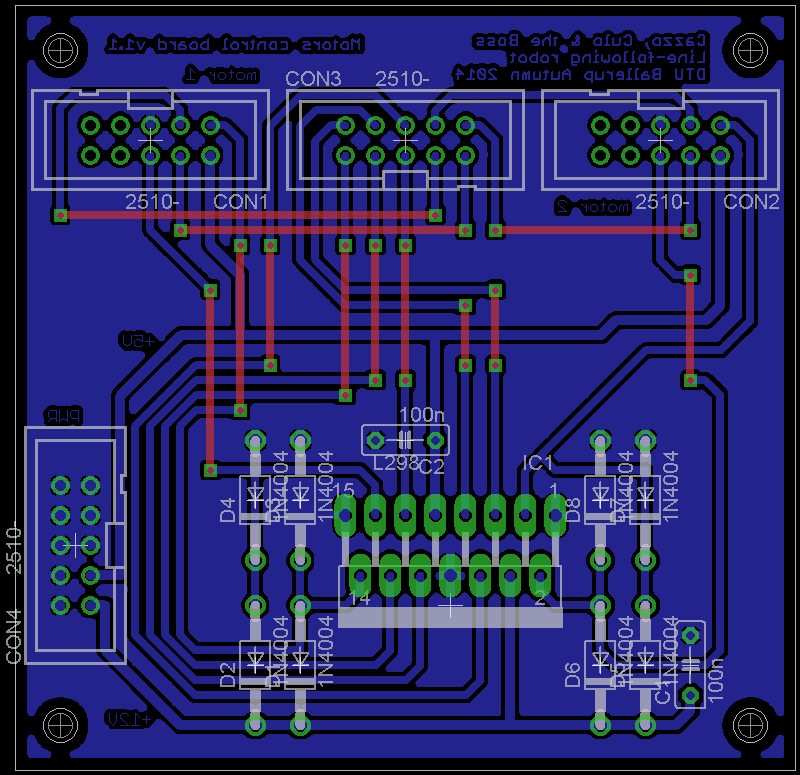
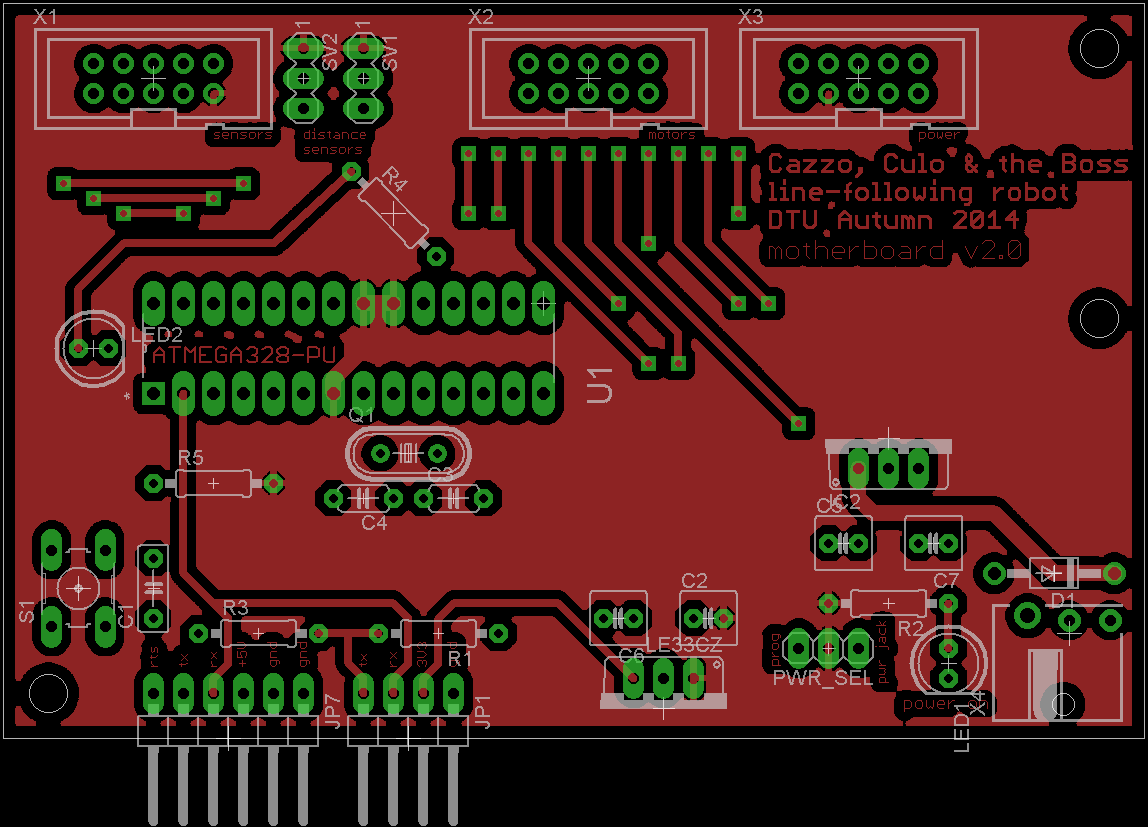


Figure 22 H-Bridge board PCB

### MCU board



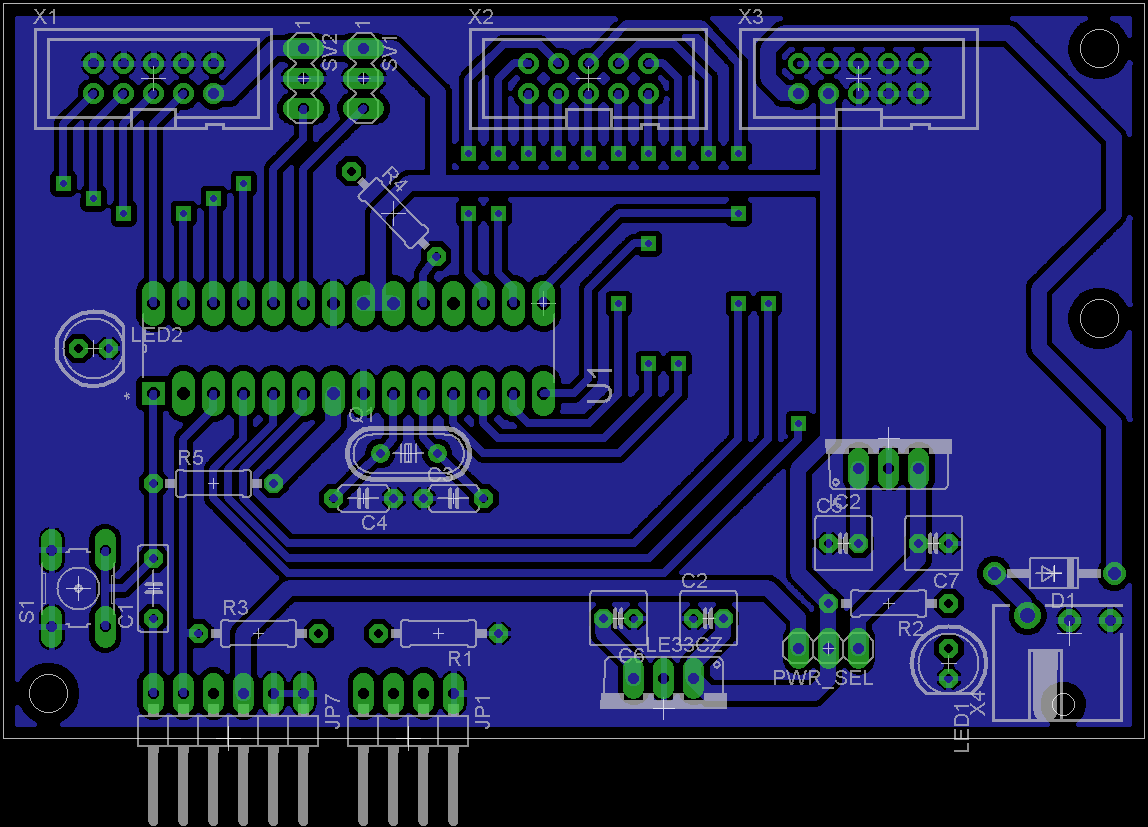


Figure 23 MCU board PCB

## Code

### MCU

#include "EEPROM.h"

**template** <**class** T> boolean inRange (T val, T min, T max) {

**return** ((min <= val) && (val <= max));

}

//debugging settings

boolean DEBUG\_MOTORS = **true**;

boolean DEBUG\_SENSORS = **false**;

boolean DEBUG\_SENSORS\_THRES = **false**;

boolean REMOTE\_CONTROL = **true**;

boolean SENSORS\_READING = **true**;

boolean LINE\_FOLLOWING = **true**;

boolean PATH = LINE\_FOLLOWING && **true**;

boolean XBEE = **false**;

boolean START\_STILL = **false**;

**unsigned** **int** DebugFreq = 4;

**unsigned** **int** steeringDuration = 50;

**unsigned** **int** steeringAmount = 100;

//Loop settings

**unsigned** **int** initialRef = 250;

**float** Kw = 0.75; // line sensors weight

**float** Ks = 0.5; // wheels synchronization

**float** Ki = 0.45; // constant for integral term

**float** Kp[2] = {0.77, 0.77}; // constant for proportional term

**const** **unsigned** **int** avgMax = 50;

**unsigned** **int** avgNum = 10; // smoothing factor

**unsigned** **int** Fs = 100; // sampling frequency (Hz)

//debugging variables

**const** **int** LedPin = 13;

String msg = "";

**unsigned** **long** currentDebugTime, previousDebugTime;

boolean manualSteering = **false**;

boolean manualStill = **false**;

**unsigned** **int** steeringCount = 0;

**enum** Direction {FORWARD, BACKWARD};

//path variables

boolean stage0 = **false**;

boolean stage1 = **false**;

boolean stage2 = **false**;

boolean stage3 = **false**;

boolean stage4 = **false**;

boolean stage5 = **false**;

//Floor sensors pins setup

**const** **int** MuxOut = A2;

**const** **int** MuxPin1 = A3;

**const** **int** MuxPin2 = A4;

**const** **int** MuxPin3 = A5;

//Motor 1 pins setup

**const** **int** Enable1 = 5;

**const** **int** InputA1 = 6;

**const** **int** InputB1 = 7;

**const** **int** EncoderA1 = 3;

**const** **int** EncoderB1 = 4;

//Motor 2 pins setup

**const** **int** Enable2 = 9;

**const** **int** InputA2 = 10;

**const** **int** InputB2 = 11;

**const** **int** EncoderA2 = 2;

**const** **int** EncoderB2 = 8;

//sensor variables

**const** **int** inputsNr = 8; //number of sensors

**int** sensors[inputsNr] = {0, 0, 0, 0, 0, 0, 0, 0}; //sensors values

**int** sensorsPos[inputsNr] = {4, 5, 6, 7, 0, 1, 2, 3}; //sensor order

**int** sensorsWeight[inputsNr] = {-7, -5, -3, -1, 1, 3, 5, 7}; //sensor weight

**int** sensorsMinThresh[inputsNr] = {725, 627, 586, 607, 586, 727, 719, 600};

**int** sensorsMaxThresh[inputsNr] = {986, 974, 968, 974, 968, 983, 981, 966};

**int** sensorsSum = 0;

**int** sensorsSumWeighed = 0;

//sensors thresholds

**int** minThresh = 1023/2;

**int** maxThresh = 1023/2;

//Motor Constants

**const** **int** gearRatio = 172; // 17.2:1

**const** **int** linesPerRev = 15; // 16 for 21B encoders

//interrupt variables (used to read tachometer)

**volatile** **unsigned** **long** TPeriod1, TPeriod2; // time periods

**volatile** **unsigned** **long** Time1, Time2; // running time

**volatile** **unsigned** **int** avgCount1, avgCount2;

**volatile** **unsigned** **long** avgTot1, avgTot2;

**volatile** **unsigned** **long** avgVals[2][avgMax];

//loop variables (used to control the motors)

**unsigned** **long** currentTime, previousTime, deltaTime;

**float** Ref; // reference for motor speed (RPM)

**float** Speed1, Speed2; // wheels speed (RPM)

**float** Error1, Error2, Error3; // difference between reference and measured value (RPM)

**float** Integral; // integral sum

**float** Bias; // speed bias given by the I-controller

**float** LineOut; // output from the sensors

**unsigned** **int** CSignal1, CSignal2; // power to the motor

**void** quickStop() {

//Motor 1

digitalWrite(InputA1, LOW);

digitalWrite(InputB1, LOW);

digitalWrite(Enable1, HIGH);

//Motor 2

digitalWrite(InputA2, LOW);

digitalWrite(InputB2, LOW);

digitalWrite(Enable2, HIGH);

}

**void** quickStop(**int** n) {

**switch** (n) {

**case** 1:

digitalWrite(InputA1, LOW);

digitalWrite(InputB1, LOW);

digitalWrite(Enable1, HIGH);

**break**;

**case** 2:

digitalWrite(InputA2, LOW);

digitalWrite(InputB2, LOW);

digitalWrite(Enable2, HIGH);

**break**;

**default**:

**break**;

}

}

**void** setDirection(**int** motor, Direction a) {

**switch** (motor) {

**case** 0:

**switch** (a) {

**case** FORWARD:

//Motor 1 direction

digitalWrite(InputA1, HIGH);

digitalWrite(InputB1, LOW);

//Motor 2 direction

digitalWrite(InputA2, HIGH);

digitalWrite(InputB2, LOW);

**break**;

**case** BACKWARD:

//Motor 1 direction

digitalWrite(InputA1, LOW);

digitalWrite(InputB1, HIGH);

//Motor 2 direction

digitalWrite(InputA2, LOW);

digitalWrite(InputB2, HIGH);

**break**;

}

**break**;

**case** 1:

**switch** (a) {

**case** FORWARD:

//Motor 1 direction

digitalWrite(InputA1, HIGH);

digitalWrite(InputB1, LOW);

**break**;

**case** BACKWARD:

//Motor 1 direction

digitalWrite(InputA1, LOW);

digitalWrite(InputB1, HIGH);

**break**;

}

**break**;

**case** 2:

**switch** (a) {

**case** FORWARD:

//Motor 2 direction

digitalWrite(InputA2, HIGH);

digitalWrite(InputB2, LOW);

**break**;

**case** BACKWARD:

//Motor 2 direction

digitalWrite(InputA2, LOW);

digitalWrite(InputB2, HIGH);

**break**;

}

**break**;

**default**:

**break**;

}

}

**void** loadK() {

**if**(EEPROM.read(0)==0) {

**if**(DEBUG\_MOTORS || DEBUG\_SENSORS || REMOTE\_CONTROL) {

Serial.println("Overriding constants...");

}

EEPROM.write(0,max(initialRef,50)-50);

EEPROM.write(1,map(Kw\*1000,0,2000,0,255));

EEPROM.write(2,map(Ks\*1000,0,2000,0,255));

EEPROM.write(3,map(Ki\*1000,0,2000,0,255));

EEPROM.write(4,map(Kp[0]\*1000,0,2000,0,255));

EEPROM.write(5,map(Kp[1]\*1000,0,2000,0,255));

EEPROM.write(6,min(avgNum,avgMax));

EEPROM.write(7,min(sensorsWeight[inputsNr/2],255));

EEPROM.write(8,min(sensorsWeight[inputsNr/2+1],255));

EEPROM.write(9,min(sensorsWeight[inputsNr/2+2],255));

EEPROM.write(10,min(sensorsWeight[inputsNr/2+3],255));

}

initialRef = EEPROM.read(0)+50;

Kw = ((**float**)map(EEPROM.read(1),0,255,0,2000))/1000;

Ks = ((**float**)map(EEPROM.read(2),0,255,0,2000))/1000;

Ki = ((**float**)map(EEPROM.read(3),0,255,0,2000))/1000;

Kp[0] = ((**float**)map(EEPROM.read(4),0,255,0,2000))/1000;

Kp[1] = ((**float**)map(EEPROM.read(5),0,255,0,2000))/1000;

avgNum = min(EEPROM.read(6),avgMax);

**for**(**int** i=0; i<inputsNr/2; i++){

**int** a = EEPROM.read(7+i);

sensorsWeight[inputsNr/2+i] = a;

sensorsWeight[inputsNr/2-i-1] = -a;

}

**if**(DEBUG\_MOTORS || DEBUG\_SENSORS || REMOTE\_CONTROL) {

msg = "Memory raw dump:";

msg += "\nKw\t\t\t";

msg += EEPROM.read(1);

msg += "\nKs\t\t\t";

msg += EEPROM.read(2);

msg += "\nKi\t\t\t";

msg += EEPROM.read(3);

msg += "\nKp (left)\t";

msg += EEPROM.read(4);

msg += "\nKp (right)\t";

msg += EEPROM.read(5);

msg += "\nInit. Speed\t";

msg += EEPROM.read(0);

msg += "\n# Avg.\t\t";

msg += EEPROM.read(6);

msg += "\nSensor wt.\t";

**for**(**int** i=0; i<inputsNr/2; i++){

**int** a = EEPROM.read(7+i);

msg += a;

msg += " ";

}

msg += "\n\nLoop constants:";

msg += "\nKw\t\t\t";

msg += (**int**)(Kw\*1000);

msg += "\nKs\t\t\t";

msg += (**int**)(Ks\*1000);

msg += "\nKi\t\t\t";

msg += (**int**)(Ki\*1000);

msg += "\nKp (left)\t";

msg += (**int**)(Kp[0]\*1000);

msg += "\nKp (right)\t";

msg += (**int**)(Kp[1]\*1000);

msg += "\nInit. Speed\t";

msg += (**int**)initialRef;

msg += "rpm\n# Avg.\t\t";

msg += (**int**)avgNum;

msg += "smp\nSensor wt.\t";

**for**(**int** i=0; i<inputsNr; i++){

msg += sensorsWeight[i];

msg += " ";

}

Serial.println(msg);

}

}

/\* SETUP \*/

**void** setup() {

//initialize serial only if needed

**if**(DEBUG\_MOTORS || DEBUG\_SENSORS || REMOTE\_CONTROL) {

// serial comm setup

Serial.begin(9600);

//initial message

Serial.println("## HELLO! ##\n\n");

**if**(XBEE) {

delay(15000);

}

}

//parameters setup

loadK();

Ref = initialRef;

CSignal1 = 10;

CSignal2 = 10;

manualStill = START\_STILL;

delay(5000);

// multiplexer pins

pinMode(MuxOut, INPUT);

pinMode(MuxPin1, OUTPUT);

pinMode(MuxPin2, OUTPUT);

pinMode(MuxPin3, OUTPUT);

//Motor 1

pinMode(Enable1, OUTPUT);

pinMode(InputA1, OUTPUT);

pinMode(InputB1, OUTPUT);

pinMode(EncoderA1, INPUT);

pinMode(EncoderB1, INPUT);

//Motor 2

pinMode(Enable2, OUTPUT);

pinMode(InputA2, OUTPUT);

pinMode(InputB2, OUTPUT);

pinMode(EncoderA2, INPUT);

pinMode(EncoderB2, INPUT);

//Led pin

pinMode(LedPin, OUTPUT);

digitalWrite(Enable1, LOW);

digitalWrite(Enable2, LOW);

setDirection(0, FORWARD);

//variables initialization

TPeriod1 = 0;

TPeriod2 = 0;

Time1 = 0;

Time2 = 0;

avgCount1 = 0;

avgCount2 = 0;

avgTot1 = 0;

avgTot2 = 0;

**for**(**int** i=0 ; i<avgMax ; i++) {

avgVals[0][i] = 0;

avgVals[1][i] = 0;

}

currentTime = 0;

previousTime = 0;

//loop vars setup

Integral = 0;

Error1 = 0;

Error2 = 0;

Error3 = 0;

LineOut = 0;

Bias = 0;

//interrups for TPeriod reading

//called on rising edge for encoders channel A

attachInterrupt(1, motor1, RISING);

attachInterrupt(0, motor2, RISING);

//turn on motors

analogWrite(Enable1, CSignal1);

analogWrite(Enable2, CSignal2);

delay(1000);

//increases PWM freqs

//TCCR0B = TCCR0B & 0b11111000 | 0x01;

//TCCR1B = TCCR1B & 0b11111000 | 0x01;

}

/\* LOOP \*/

**void** loop() {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* motors loop \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//update current time

currentTime = micros();

//runs loop with fixed time interval

**if**((currentTime-previousTime) > (1000000UL/Fs)) {

//find the loop time period (debug)

deltaTime = currentTime-previousTime;

//update loop timer

previousTime = currentTime;

//toggle led

digitalWrite(LedPin, !digitalRead(LedPin));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* floor sensor \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**if**(SENSORS\_READING) {

//reset the sensors output to the loop

LineOut = 0;

sensorsSum = 0;

sensorsSumWeighed = 0;

//cycle through the sensors

**for**(**int** i=0; i<inputsNr; i++) {

//set the multiplexer to read one of the sensors

digitalWrite(MuxPin1, bitRead(i,0));

digitalWrite(MuxPin2, bitRead(i,1));

digitalWrite(MuxPin3, bitRead(i,2));

//read the value

**int** reading = analogRead(MuxOut);

//save and scale the value(s) using the mapping vector

**int** realPos = sensorsPos[i];

sensors[realPos] = constrain(map(reading, sensorsMinThresh[i], sensorsMaxThresh[i], 10, 0), 0, 10);

//calculate sum of sensors with scaled value

sensorsSumWeighed += sensors[realPos]\*sensorsWeight[i];

//calculate sum of sensors (tells how many sensors are above the line)

sensorsSum += sensors[realPos];

//wait for better ADC accuracy

delayMicroseconds(5);

}

**if**(LINE\_FOLLOWING) {

//calculate real weighed sensors output

LineOut = sensorsSumWeighed/(80 - sensorsSum + 1)\*10;

}

**else** {

LineOut = 0;

}

}

//calculate speed

Speed1 = (1000000 \* 10 \* 60 \* 1.0) / (TPeriod1 \* gearRatio \* linesPerRev);

Speed2 = (1000000 \* 10 \* 60 \* 1.0) / (TPeriod2 \* gearRatio \* linesPerRev);

**if**(manualSteering) {

//if remote cotrols for steering are sent, ignore loop I-controller for a limited time

steeringCount++;

//after steeringDuration\*2, disable the manual steering and reanable loop control

**if**(steeringCount>=(steeringDuration\*2)) {

manualSteering = **false**;

steeringCount = 0;

}

//reset bias and keep running for steeringDuration

**if**(steeringCount>=(steeringDuration)) {

Bias = 0;

}

} **else** {

//calculations for the I-controller

Error3 = (LineOut\*Kw)+((Speed1-Speed2)\*Ks);

Integral += Error3;

Bias = Integral\*Ki;

}

//calculations for the P-controller(s)

Error1 = (Ref-Speed1-Bias);

Error2 = (Ref-Speed2+Bias);

//calculation of the outputs

CSignal1 = constrain((Error1\*Kp[0]), 0, 255);

CSignal2 = constrain((Error2\*Kp[1]), 0, 255);

**if**(manualStill) {

//if manual still mode, power to the motor is 0

quickStop();

} **else** **if**(Error1>=0 && Error2>=0) {

//update motor power

setDirection(0,FORWARD);

analogWrite(Enable1, CSignal1);

analogWrite(Enable2, CSignal2);

} **else** **if**(Error1<0 && Error2>=0) {

quickStop(1);

setDirection(2,FORWARD);

analogWrite(Enable2, CSignal2);

} **else** **if**(Error1>=0 && Error2<0) {

setDirection(1,FORWARD);

analogWrite(Enable1, CSignal1);

quickStop(2);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* remote control \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**if**(REMOTE\_CONTROL) {

**char** val = 0;

**int** dest = 0;

**int** param = 0;

**int** value = 0;

**if** (Serial.available()) {

Integral = 0;

Bias = 0;

LINE\_FOLLOWING = **false**;

val = Serial.read();

**switch**(val) {

**case** 'c': //change setting

dest = Serial.parseInt();

param = Serial.parseInt();

value = Serial.parseInt();

//parameters setting

**switch**(param) {

**case** 0:

**if**(dest==0) {

Kp[0] = value/1000.0;

Kp[1] = value/1000.0;

EEPROM.write(4,map(value,0,2000,0,255));

EEPROM.write(5,map(value,0,2000,0,255));

} **else** {

Kp[dest-1] = value/1000.0;

EEPROM.write(4+dest-1,map(value,0,2000,0,255));

Serial.print(EEPROM.read(4+dest-1));

Serial.println(" written in EEPROM.");

}

Serial.print(EEPROM.read(4));

Serial.println(" written in EEPROM (twice).");

**break**;

**case** 1:

**if**(dest==0) {

Ki = value/1000.0;

EEPROM.write(3,map(value,0,2000,0,255));

Serial.print(EEPROM.read(3));

Serial.println(" written in EEPROM.");

}

**break**;

**case** 2:

**if**(dest==0) {

Kw = value/1000.0;

EEPROM.write(1,map(value,0,2000,0,255));

Serial.print(EEPROM.read(1));

Serial.println(" written in EEPROM.");

}

**break**;

**case** 3:

**if**(dest==0) {

Ks = value/1000.0;

EEPROM.write(2,map(value,0,2000,0,255));

Serial.print(EEPROM.read(2));

Serial.println(" written in EEPROM.");

}

**break**;

**case** 4:

**if**(dest==0) {

Fs = value;

}

**break**;

**case** 5:

**if**(dest==0 && value<=avgMax) {

avgNum = value;

EEPROM.write(6,value);

Serial.print(EEPROM.read(6));

Serial.println(" written in EEPROM.");

}

**break**;

**case** 6:

**if**(dest==0) {

initialRef = value;

Ref = initialRef;

EEPROM.write(0,max(initialRef,50)-50);

Serial.print(EEPROM.read(0));

Serial.println(" written in EEPROM.");

}

**break**;

**case** 10:

**case** 11:

**case** 12:

**case** 13:

**if**(dest==3) {

sensorsWeight[inputsNr/2+(param-10)] = value;

sensorsWeight[inputsNr/2-(param-10)-1] = -value;

EEPROM.write(7+param-10,value);

Serial.print(EEPROM.read(7+param-10));

Serial.println(" written in EEPROM.");

}

**break**;

}

**break**;

**case** 'p': //increase speed

Ref \*= 1.2;

**break**;

**case** 'm': //decrease speed

Ref \*= 0.8;

**break**;

**case** 'l': //turn left

manualSteering = **true**;

Bias = -(steeringAmount/Ki);

**break**;

**case** 'r': //turn right

manualSteering = **true**;

Bias = (steeringAmount/Ki);

**break**;

**case** 'f': //go forward

setDirection(0, FORWARD);

manualSteering = **false**;

manualStill = **false**;

Ref = initialRef;

DEBUG\_MOTORS = **true**;

**break**;

**case** 's': //break

quickStop();

manualStill = **true**;

DEBUG\_MOTORS = **false**;

**break**;

**case** 'b': //go backward

setDirection(0, BACKWARD);

manualSteering = **false**;

manualStill = **false**;

Ref = initialRef;

**break**;

**case** 'i': //info

msg = "Loop constants:";

msg += "\nKw\t\t\t";

msg += (**int**)(Kw);

msg += ".";

msg += (**int**)((Kw-(**int**)Kw)\*10);

msg += (**int**)((Kw\*10-(**int**)(Kw\*10))\*10);

msg += (**int**)((Kw\*100-(**int**)(Kw\*100))\*10);

msg += "\nKs\t\t\t";

msg += (**int**)(Ks);

msg += ".";

msg += (**int**)((Ks-(**int**)Ks)\*10);

msg += (**int**)((Ks\*10-(**int**)(Ks\*10))\*10);

msg += (**int**)((Ks\*100-(**int**)(Ks\*100))\*10);

msg += "\nKi\t\t\t";

msg += (**int**)(Ki);

msg += ".";

msg += (**int**)((Ki-(**int**)Ki)\*10);

msg += (**int**)((Ki\*10-(**int**)(Ki\*10))\*10);

msg += (**int**)((Ki\*100-(**int**)(Ki\*100))\*10);

msg += "\nKp (left)\t";

msg += (**int**)(Kp[0]);

msg += ".";

msg += (**int**)((Kp[0]-(**int**)Kp[0])\*10);

msg += (**int**)((Kp[0]\*10-(**int**)(Kp[0]\*10))\*10);

msg += (**int**)((Kp[0]\*100-(**int**)(Kp[0]\*100))\*10);

msg += "\nKp (right)\t";

msg += (**int**)(Kp[1]);

msg += ".";

msg += (**int**)((Kp[1]-(**int**)Kp[1])\*10);

msg += (**int**)((Kp[1]\*10-(**int**)(Kp[1]\*10))\*10);

msg += (**int**)((Kp[1]\*100-(**int**)(Kp[1]\*100))\*10);

msg += "\nInit. Speed\t";

msg += (**int**)initialRef;

msg += " rpm\n# Avg.\t\t";

msg += (**int**)avgNum;

msg += " smp\nSensor wt.\t";

**for**(**int** i=0; i<inputsNr; i++){

msg += sensorsWeight[i];

msg += " ";

}

Serial.println(msg);

**break**;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* path handling \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**if**(PATH) {

// stage 0: outside the path

**if**(!stage0) {

**if**(!stage1 && inRange(sensorsSumWeighed,-10,10) && (sensorsSum>=70)) {

Serial.println("stage 0 entered!");

LINE\_FOLLOWING = **false**;

stage0 = **true**;

Serial.println("stage 0 cleared!");

}

}

// stage 1: entering the path

**else** **if**(!stage1) {

**if**(stage0 && !stage2 && inRange(sensorsSumWeighed,-10,10) && (sensorsSum<=10)) {

Serial.println("stage 1 entered!");

//turn

quickStop();

setDirection(1, BACKWARD);

analogWrite(Enable1, 255);

setDirection(2, FORWARD);

analogWrite(Enable2, 255);

delay(300);

//restore

quickStop();

Integral = 0;

delay(500);

LINE\_FOLLOWING = **true**;

setDirection(0, FORWARD);

stage1 = **true**;

Serial.println("stage 1 cleared!");

}

}

// stage 2: line gap

**else** **if**(!stage2) {

**if**(stage1 && !stage3 && inRange(sensorsSumWeighed,-10,10) && (sensorsSum>=70)) {

Serial.println("stage 2 entered!");

//turn

quickStop();

setDirection(1, FORWARD);

analogWrite(Enable1, 255);

setDirection(2, BACKWARD);

analogWrite(Enable2, 255);

delay(300);

//straight

quickStop();

setDirection(1, FORWARD);

analogWrite(Enable1, 255);

setDirection(2, FORWARD);

analogWrite(Enable2, 255);

delay(150);

//turn

quickStop();

setDirection(1, BACKWARD);

analogWrite(Enable1, 255);

setDirection(2, FORWARD);

analogWrite(Enable2, 255);

delay(300);

//restore

quickStop();

Integral = 0;

delay(500);

LINE\_FOLLOWING = **true**;

setDirection(0, FORWARD);

stage2 = **true**;

Serial.println("stage 2 cleared!");

}

}

// stage 3: 20cm-wide semicircle

**else** **if**(!stage3) {

**if**(stage2 && !stage4 && inRange(sensorsSumWeighed,-10,10) && (sensorsSum<=10)) {

Serial.println("stage 3 entered!");

//turn

quickStop();

setDirection(1, BACKWARD);

analogWrite(Enable1, 255);

setDirection(2, FORWARD);

analogWrite(Enable2, 255);

delay(300);

//circle

setDirection(0, BACKWARD);

analogWrite(Enable1, 100);

analogWrite(Enable2, 255);

delay(1500);

//turn

quickStop();

setDirection(1, FORWARD);

analogWrite(Enable1, 255);

setDirection(2, BACKWARD);

analogWrite(Enable2, 255);

delay(300);

//restore

quickStop();

Integral = 0;

delay(500);

LINE\_FOLLOWING = **true**;

setDirection(0, FORWARD);

stage3 = **true**;

Serial.println("stage 3 cleared!");

}

}

// stage 4: speed increment

**else** **if**(!stage4) {

**if**(stage3 && (sensorsSumWeighed>=80) && inRange(sensorsSum,20,40)) {

Serial.println("stage 4 entered!");

Ref \*= 1.5;

stage4 = **true**;

Serial.println("stage 4 cleared!");

}

}

// stage 5: stop

**else** **if**(!stage5) {

**if**(stage4 && !stage4 && inRange(sensorsSumWeighed,-10,10) && (sensorsSum<=10)) {

Serial.println("stage 5 entered!");

//turn

quickStop();

setDirection(1, BACKWARD);

analogWrite(Enable1, 255);

setDirection(2, FORWARD);

analogWrite(Enable2, 255);

delay(300);

//restore

quickStop();

Integral = 0;

delay(500);

LINE\_FOLLOWING = **true**;

setDirection(0, FORWARD);

stage5 = **true**;

Serial.println("stage 5 cleared!");

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* DEBUG DATA \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**if**(DEBUG\_MOTORS || DEBUG\_SENSORS || DEBUG\_SENSORS\_THRES) {

//update current time

currentDebugTime = micros();

//runs loop with fixed time interval

**if**((currentDebugTime-previousDebugTime) > (1000000UL / DebugFreq)) {

//update debug communication timer

previousDebugTime = currentDebugTime;

//construct the message in a single string

msg = "";

//gather sensors info

**if**(DEBUG\_SENSORS) {

**for**(**int** i=0; i<inputsNr; i++){

**int** realPos = sensorsPos[i];

msg += "s ";

msg += realPos;

msg += " ";

msg += sensors[realPos];

msg += "\n";

}

}

//gather sensors tuning info

**if**(DEBUG\_SENSORS\_THRES) {

**for**(**int** i=0; i<inputsNr; i++){

**int** realPos = sensorsPos[i];

msg += sensors[realPos];

msg += "\t";

}

msg += "\n";

}

//gathers motors/loop info

**if**(DEBUG\_MOTORS) {

msg += "m 1 ";

msg += (**int**)Speed1;

msg += " ";

msg += (**int**)Error1;

msg += " ";

msg += (**int**)CSignal1;

msg += "\nm 2 ";

msg += (**int**)Speed2;

msg += " ";

msg += (**int**)Error2;

msg += " ";

msg += (**int**)CSignal2;

msg += "\nm 0 ";

msg += (**int**)LineOut;

msg += " ";

msg += (**int**)Bias;

msg += "\n";

}

//send the string

Serial.print(msg);

//debug debugging :D

msg = "loop freq: ";

msg += (**int**)(1000000UL/deltaTime);

msg += " Hz\t time taken: ";

msg += (micros()-currentDebugTime);

msg += " us\n";

Serial.print(msg);

}

}

}

//motor1 interrupt

**void** motor1() {

//subtract the last reading

avgTot1 -= avgVals[0][avgCount1];

//obtain new reading

avgVals[0][avgCount1] = abs(micros()-Time1);

//update time counter

Time1 = micros();

//add the reading to the total

avgTot1 += avgVals[0][avgCount1];

//advance to the next position in the array

avgCount1++;

//reset counter

**if**(avgCount1 >= avgNum) avgCount1 = 0;

//calculate the average time period

TPeriod1 = avgTot1 / avgNum;

}

//motor2 interrupt

**void** motor2() {

avgTot2 -= avgVals[1][avgCount2];

avgVals[1][avgCount2] = abs(micros()-Time2);

Time2 = micros();

avgTot2 += avgVals[1][avgCount2];

avgCount2++;

**if**(avgCount2 >= avgNum) avgCount2 = 0;

TPeriod2 = avgTot2 / avgNum;

}

### Java

The code is divided in 10 classes, and they are ordered by relevance.

#### Robot class

**package** robot;

**import** javax.swing.BoxLayout;

**import** javax.swing.JFrame;

**import** static robot.Interface.sensorpanel;

**import** static robot.Interface.motorpanel;

**import** static robot.Interface.motorpanel1;

**import** static robot.Interface.comunpanel;

**public** **class** Robot {

**public** **static** void main(**String**[] args) {

JFrame frame = **new** JFrame("Robot");

frame.add(**new** Interface());

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.pack();

frame.setVisible(**true**);

sensorpanel.setLayout(**new** BoxLayout(sensorpanel, BoxLayout.Y\_AXIS));

motorpanel.setLayout(**new** BoxLayout(motorpanel, BoxLayout.Y\_AXIS));

motorpanel1.setLayout(**new** BoxLayout(motorpanel1, BoxLayout.Y\_AXIS));

comunpanel.setLayout(**new** BoxLayout(comunpanel, BoxLayout.Y\_AXIS));

robot.Interface.sensorpanel.add(sg);

}

}

* + - 1. **Interface class**

**package** robot;

**import** robot.SerialClass.**\***;

**import** java.awt.AWTException;

**import** java.awt.Robot.**\***;

**import** java.io.**\***;

**import** java.util.logging.Level;

**import** java.util.logging.Logger;

**public** **class** Interface **extends** javax.swing.JPanel {

**public** **static** int[] s1 = **new** int[122];

**public** **static** int[] s2 = **new** int[122];

**public** **static** int[] s3 = **new** int[122];

**public** **static** int[] s4 = **new** int[122];

**public** **static** int[] s5 = **new** int[122];

**public** **static** int[] s6 = **new** int[122];

**public** **static** int[] s7 = **new** int[122];

**public** **static** int[] s8 = **new** int[122];

**public** **static** int[] speedleft = **new** int[95];

**public** **static** int[] errleft = **new** int[95];

**public** **static** int[] csignalleft = **new** int[95];

**public** **static** int[] speedright = **new** int[95];

**public** **static** int[] errright = **new** int[95];

**public** **static** int[] csignalright = **new** int[95];

**public** **static** int[] bias = **new** int[95];

**public** **static** int[] lineout = **new** int[95];

**static** int activationpath = 0;

int sens = 0;

int reader = 0;

int contgraph = 0;

int parameter;

int motor;

SerialClass obj = **new** SerialClass();

**public** Interface() {

initComponents();

**for** (int i = 0; i < 122; i++) {

s1[i] = 0;

s2[i] = 0;

s3[i] = 0;

s4[i] = 0;

s5[i] = 0;

s6[i] = 0;

s7[i] = 0;

s8[i] = 0;

}

**for** (int i = 0; i < 95; i++) {

speedleft[i] = 0;

errleft[i] = 0;

csignalleft[i] = 0;

speedright[i] = 0;

errright[i] = 0;

csignalright[i] = 0;

bias[i] = 0;

lineout[i] = 0;

}

}

**private** void buttonreaderActionPerformed(java.awt.event.**ActionEvent** evt) {

reader++;

**if** (reader == 1) {

buttonreader.setText("STOP");

**try** {

obj.initialize();

} **catch** (**Exception** e) {

}

}

**if** (reader == 2) {

buttonreader.setText("Start");

reader = 0;

obj.close();

}

}

**private** void buttoncaptureActionPerformed(java.awt.event.**ActionEvent** evt) {

**try** {

robot.Capture.capturar();

} **catch** (**AWTException** ex) {

Logger.getLogger(Interface.class.getName()).log(Level.SEVERE, **null**, ex);

} **catch** (**IOException** ex) {

Logger.getLogger(Interface.class.getName()).log(Level.SEVERE, **null**, ex);

}

}

**private** void jButton1KeyPressed(java.awt.event.**KeyEvent** evt) {

int keyCode = evt.getKeyCode();

**switch** (keyCode) {

**case** java.awt.event.**KeyEvent**.VK\_UP:

// handle forwards

robot.SerialClass.writeData("f");

**break**;

**case** java.awt.event.**KeyEvent**.VK\_DOWN:

// handle backwards

robot.SerialClass.writeData("b");

**break**;

**case** java.awt.event.**KeyEvent**.VK\_LEFT:

// handle left

robot.SerialClass.writeData("l");

**break**;

**case** java.awt.event.**KeyEvent**.VK\_RIGHT:

// handle right

robot.SerialClass.writeData("r");

**break**;

**case** java.awt.event.**KeyEvent**.VK\_S:

// handle stop

robot.SerialClass.writeData("s");

**break**;

**case** java.awt.event.**KeyEvent**.VK\_ADD:

// handle increase speed

robot.SerialClass.writeData("p");

**break**;

**case** java.awt.event.**KeyEvent**.VK\_MINUS:

// handle decrease speed

robot.SerialClass.writeData("m");

**break**;

**case** java.awt.event.**KeyEvent**.VK\_I:

// handle information constants

robot.SerialClass.writeData("i");

**break**;

}

}

**private** void sendActionPerformed(java.awt.event.**ActionEvent** evt) {

double x = 0;

int result = 0;

int y = 0;

y = parameter;

**if** (motor == 3) {

**String**[] tokens = newvalue.getText().trim().split(" ");

**if** (tokens.length == 4) {

**for** (int i = 0; i < 4; i++) {

**try** {

robot.SerialClass.writeData("c 3 " + (10 + i) + " " + **Math**.abs(**Integer**.parseInt(tokens[i])));

} **catch** (**Exception** e) {

**System**.out.println("Not a valid command");

}

}

}

} **else** **if** (y >= 4) {

x = **Double**.parseDouble(newvalue.getText());

result = (int) x;

**try** {

robot.SerialClass.writeData("c " + motor + " " + parameter + " " + result);

} **catch** (**Exception** e) {

**System**.out.println("Not a valid command");

}

} **else** {

x = **Double**.parseDouble(newvalue.getText());

x = x \* 1000;

result = (int) x;

**try** {

robot.SerialClass.writeData("c " + motor + " " + parameter + " " + result);

} **catch** (**Exception** e) {

**System**.out.println("Not a valid command");

}

}

}

**private** void newvalueActionPerformed(java.awt.event.**ActionEvent** evt) {

}

**private** void parametersActionPerformed(java.awt.event.**ActionEvent** evt) {

parameter = parameters.getSelectedIndex();

}

**private** void electionActionPerformed(java.awt.event.**ActionEvent** evt) {

motor = election.getSelectedIndex();

**if** (motor == 1 || motor == 2) {

parameter = 0;

parameters.setModel(**new** javax.swing.DefaultComboBoxModel(**new** **String**[]{"Kp"}));

} **else** **if** (motor == 0) {

parameter = parameters.getSelectedIndex();

parameters.setModel(**new** javax.swing.DefaultComboBoxModel(**new** **String**[]{"Kp", "Ki", "Kw", "Ks", "Fs", "Avg", "Ref"}));

} **else** **if** (motor == 3) {

parameter = 0;

parameters.setModel(**new** javax.swing.DefaultComboBoxModel(**new** **String**[]{"Weights"}));

}

}

**private** void textleftmotorActionPerformed(java.awt.event.**ActionEvent** evt) {

// TODO add your handling code here:

}

**private** void buttonclearActionPerformed(java.awt.event.**ActionEvent** evt) {

**for** (int i = 0; i < 95; i++) {

bias[i] = 0;

lineout[i] = 0;

speedleft[i] = 0;

errleft[i] = 0;

csignalleft[i] = 0;

speedright[i] = 0;

errright[i] = 0;

csignalright[i] = 0;

}

**for** (int i = 0; i < 122; i++) {

s1[i] = 0;

s2[i] = 0;

s3[i] = 0;

s4[i] = 0;

s5[i] = 0;

s6[i] = 0;

s7[i] = 0;

s8[i] = 0;

}

}

**private** void infobuttonActionPerformed(java.awt.event.**ActionEvent** evt) {

robot.SerialClass.writeData("i");

}

**private** int parsetoInt(double x) {

**throw** **new** UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

**private** void delay(int i) {

**throw** **new** UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

}

* + - 1. **SerialClass class**

**package** robot;

**import** Graficas.CommunValues;

**import** Graficas.Sensorgraph;

**import** Graficas.Motorgraphleft;

**import** Graficas.Motorgraphright;

**import** java.io.BufferedReader;

**import** java.io.InputStreamReader;

**import** java.io.OutputStream;

**import** gnu.io.CommPortIdentifier;

**import** gnu.io.SerialPort;

**import** gnu.io.SerialPortEvent;

**import** gnu.io.SerialPortEventListener;

**import** java.util.Enumeration;

**import** robot.Interface.**\***;

**import** java.awt.event.ComponentListener;

**public** **class** SerialClass **implements** SerialPortEventListener {

**public** **static** **String**[] valuesSensor;

**public** **static** **String**[] valuesMotor;

**public** SerialPort serialPort;

/\*\*

**\*** The port we're normally going to use.

     \*/

**private** **static** **final** **String** PORT\_NAMES[] = {"COM22"}; //19 //12

**public** **static** **BufferedReader** input;

**public** **static** **OutputStream** output;

/\*\*

**\*** Milliseconds to block while waiting for port open

     \*/

**public** **static** **final** int TIME\_OUT = 2000;

/\*\*

**\*** Default bits per second for COM port.

     \*/

**public** **static** **final** int DATA\_RATE = 9600;

**public** void initialize() {

CommPortIdentifier portId = **null**;

**Enumeration** portEnum = CommPortIdentifier.getPortIdentifiers();

//First, Find an instance of serial port as set in PORT\_NAMES.

**while** (portEnum.hasMoreElements()) {

CommPortIdentifier currPortId = (CommPortIdentifier) portEnum.nextElement();

**for** (**String** portName : PORT\_NAMES) {

**if** (currPortId.getName().equals(portName)) {

portId = currPortId;

**break**;

}

}

}

**if** (portId == **null**) {

**System**.out.println("Could not find COM port.");

**return**;

}

**try** {

// open serial port, and use class name for the appName.

serialPort = (SerialPort) portId.open(**this**.getClass().getName(),

TIME\_OUT);

// set port parameters

serialPort.setSerialPortParams(DATA\_RATE,

SerialPort.DATABITS\_8,

SerialPort.STOPBITS\_1,

SerialPort.PARITY\_NONE);

// open the streams

input = **new** **BufferedReader**(**new** **InputStreamReader**(serialPort.getInputStream()));

output = serialPort.getOutputStream();

char ch = 1;

output.write(ch);

// add event listeners

serialPort.addEventListener(**this**);

serialPort.notifyOnDataAvailable(**true**);

} **catch** (**Exception** e) {

**System**.err.println(e.toString());

}

}

**public** **synchronized** void close() {

**if** (serialPort != **null**) {

serialPort.removeEventListener();

serialPort.close();

}

}

**public** **synchronized** void serialEvent(SerialPortEvent oEvent) {

**if** (oEvent.getEventType() == SerialPortEvent.DATA\_AVAILABLE) {

**try** {

char firstLetter;

**String** inputLine = input.readLine();

**System**.out.println(inputLine);

**if** (!inputLine.equals(""));

{

firstLetter = inputLine.charAt(0);

**if** (firstLetter == 's') {

valuesSensor = inputLine.split(" ");

robot.Sensorbars.bars();

Sensorgraph sensorsGraph = **new** Sensorgraph();

sensorsGraph.setBackground(**Color**.LIGHT\_GRAY);

robot.Interface.sensorpanel.removeAll();

robot.Interface.sensorpanel.add(sensorsGraph);

robot.Interface.sensorpanel.revalidate();

}

**if** (firstLetter == 'm') {

valuesMotor = inputLine.split(" ");

robot.MotorSpeed.bars();

Motorgraphleft mleft = **new** Motorgraphleft();

Motorgraphright mright = **new** Motorgraphright();

CommunValues cvalues = **new** CommunValues();

mleft.setBackground(**Color**.LIGHT\_GRAY);

mright.setBackground(**Color**.LIGHT\_GRAY);

cvalues.setBackground(**Color**.LIGHT\_GRAY);

robot.Interface.motorpanel.removeAll();

robot.Interface.motorpanel.add(mleft);

robot.Interface.motorpanel.revalidate();

robot.Interface.motorpanel1.removeAll();

robot.Interface.motorpanel1.add(mright);

robot.Interface.motorpanel1.revalidate();

robot.Interface.comunpanel.removeAll();

robot.Interface.comunpanel.add(cvalues);

robot.Interface.comunpanel.revalidate();

}

}

} **catch** (**Exception** e) {

**System**.err.println(e.toString());

}

}

}

**public** **static** **synchronized** void writeData(**String** data) {

**System**.out.println("Sent: " + data);

**try** {

output.write(data.getBytes());

} **catch** (**Exception** e) {

// System.out.println("could not write to port");

}

}

**public** **static** void main(**String**[] args) **throws** **Exception** {

SerialClass main = **new** SerialClass();

main.initialize();

**Thread** t = **new** Thread() {

**public** void run() {

//the following line will keep this app alive for 1000 seconds,

//waiting for events to occur and responding to them (printing incoming messages to console).

**try** {

**Thread**.sleep(1500);

writeData("2");

} **catch** (**InterruptedException** ie) {

}

}

};

t.start();

**System**.out.println("Started");

}

}

* + - 1. **MotorSpeed class**

**package** robot;

**public** **class** MotorSpeed {

**static** void bars() {

int electedmotor = **Integer**.parseInt(robot.SerialClass.valuesMotor[1]);

//MEASURING SPEED FOR LEFT/RIGHT MOTOR

**String** speed = (robot.SerialClass.valuesMotor[2]);

**String** err = (robot.SerialClass.valuesMotor[3]);

**String** pwm = (robot.SerialClass.valuesMotor[4]);

//LEFT MOTOR

**if** (electedmotor == 2) {

robot.Interface.textleftmotor.setText(speed + " rev/sec");

robot.Interface.textlefterr.setText(err);

robot.Interface.textleftpwm.setText(pwm);

}

//RIGHT MOTOR

**if** (electedmotor == 1) {

robot.Interface.textrightmotor.setText(speed + " rev/sec");

robot.Interface.textrighterr.setText(err);

robot.Interface.textrightpwm.setText(pwm);

}

}

}

* + - 1. **Sensorbars class**

**package** robot;

**import** java.awt.Color;

**public** **class** Sensorbars {

**static** void bars() {

int n;

int x;

double d;

int electedbar = **Integer**.parseInt(robot.SerialClass.valuesSensor[1]);

n = **Integer**.parseInt(robot.SerialClass.valuesSensor[2]);

d = n \* 25;

x = (int) d;

**Color** c = **new** **Color**(255, (x), 0);

n = 10 - n;

n = n \* 10;

**if** (electedbar == 0) {

robot.Interface.barsensor1.setValue(n);

robot.Interface.barsensor1.setForeground(c);

}

**if** (electedbar == 1) {

robot.Interface.barsensor2.setValue(n);

robot.Interface.barsensor2.setForeground(c);

}

**if** (electedbar == 2) {

robot.Interface.barsensor3.setValue(n);

robot.Interface.barsensor3.setForeground(c);

}

**if** (electedbar == 3) {

robot.Interface.barsensor4.setValue(n);

robot.Interface.barsensor4.setForeground(c);

}

**if** (electedbar == 4) {

robot.Interface.barsensor5.setValue(n);

robot.Interface.barsensor5.setForeground(c);

}

**if** (electedbar == 5) {

robot.Interface.barsensor6.setValue(n);

robot.Interface.barsensor6.setForeground(c);

}

**if** (electedbar == 6) {

robot.Interface.barsensor7.setValue(n);

robot.Interface.barsensor7.setForeground(c);

}

**if** (electedbar == 7) {

robot.Interface.barsensor8.setValue(n);

robot.Interface.barsensor8.setForeground(c);

}

}

}

* + - 1. **Sensorgraph class**

**package** Graficas;

**import** java.awt.BasicStroke;

**import** java.awt.Color;

**import** java.awt.Graphics;

**import** java.awt.Graphics2D;

**import** javax.swing.JPanel;

**import** robot.Interface.**\***;

**public** **class** Sensorgraph **extends** JPanel {

**final** int xsize = 630;

**final** int ysize = 280;

**public** void drawAxes(**Graphics** g) {

//AXIS PROPERTIES

Graphics2D axis = (Graphics2D) g;

axis.setColor(**Color**.DARK\_GRAY);

axis.setStroke(**new** BasicStroke(3.0f));

axis.drawLine(10, ysize - 10, xsize - 10, ysize - 10);

axis.drawLine(10, 10, 10, ysize - 10);

//MARKERS PROPERTIES

Graphics2D markers = (Graphics2D) g;

markers.setColor(**Color**.DARK\_GRAY);

markers.setStroke(**new** BasicStroke(2.0f));

// Draw markers on x-axis

**for** (int i = 10; i <= 630; i = i + 61) {

markers.drawLine(i, ysize - 12, i, ysize - 7);

}

// Draw markers on y-axis

**for** (int i = 10; i < 260; i = i + 26) {

markers.drawLine(8, i, 13, i);

}

}

**public** void datasensor(**Graphics** g) {

**if** (robot.SerialClass.valuesSensor.length == 3) {

int ysize = 280;

int xsize = 630;

int elected = **Integer**.parseInt(robot.SerialClass.valuesSensor[1]);

int n = **Integer**.parseInt(robot.SerialClass.valuesSensor[2]) \* 30; //I just wrote this 30 instead of "n=n\*30" and im afraid this will damage the perform of the code.

int i = 0;

int j = 10;

int ss1, ss11, ss2, ss22, ss3, ss33, ss4, ss44, ss5, ss55, ss6, ss66, ss7, ss77, ss8, ss88;

**for** (i = 0; i < 121; i++) {

robot.Interface.s1[i] = robot.Interface.s1[i + 1];

robot.Interface.s2[i] = robot.Interface.s2[i + 1];

robot.Interface.s3[i] = robot.Interface.s3[i + 1];

robot.Interface.s4[i] = robot.Interface.s4[i + 1];

robot.Interface.s5[i] = robot.Interface.s5[i + 1];

robot.Interface.s6[i] = robot.Interface.s6[i + 1];

robot.Interface.s7[i] = robot.Interface.s7[i + 1];

robot.Interface.s8[i] = robot.Interface.s8[i + 1];

}

**switch** (elected) {

**case** 0:

robot.Interface.s1[121] = n;

**break**;

**case** 1:

robot.Interface.s2[121] = n;

**break**;

**case** 2:

robot.Interface.s3[121] = n;

**break**;

**case** 3:

robot.Interface.s4[121] = n;

**break**;

**case** 4:

robot.Interface.s5[121] = n;

**break**;

**case** 5:

robot.Interface.s6[121] = n;

**break**;

**case** 6:

robot.Interface.s7[121] = n;

**break**;

**case** 7:

robot.Interface.s8[121] = n;

**break**;

}

//1ST,2ND,7TH,8TH SENSOR

Graphics2D data1278 = (Graphics2D) g;

data1278.setColor(**Color**.YELLOW);

data1278.setStroke(**new** BasicStroke(2.0f));

//3RD,6TH SENSOR

Graphics2D data36 = (Graphics2D) g;

data36.setColor(**Color**.ORANGE);

data36.setStroke(**new** BasicStroke(2.0f));

//4TH,5TH SENSOR

Graphics2D data45 = (Graphics2D) g;

data45.setColor(**Color**.RED);

data45.setStroke(**new** BasicStroke(2.0f));

//DRAWING

**for** (i = 0; i <= 120; i++) {

ss1 = (int) **Math**.round(robot.Interface.s1[i] \* 0.8667);

ss11 = (int) **Math**.round(robot.Interface.s1[i + 1] \* 0.8667);

ss2 = (int) **Math**.round(robot.Interface.s2[i] \* 0.8667);

ss22 = (int) **Math**.round(robot.Interface.s2[i + 1] \* 0.8667);

ss3 = (int) **Math**.round(robot.Interface.s3[i] \* 0.8667);

ss33 = (int) **Math**.round(robot.Interface.s3[i + 1] \* 0.8667);

ss4 = (int) **Math**.round(robot.Interface.s4[i] \* 0.8667);

ss44 = (int) **Math**.round(robot.Interface.s4[i + 1] \* 0.8667);

ss5 = (int) **Math**.round(robot.Interface.s5[i] \* 0.8667);

ss55 = (int) **Math**.round(robot.Interface.s5[i + 1] \* 0.8667);

ss6 = (int) **Math**.round(robot.Interface.s6[i] \* 0.8667);

ss66 = (int) **Math**.round(robot.Interface.s6[i + 1] \* 0.8667);

ss7 = (int) **Math**.round(robot.Interface.s7[i] \* 0.8667);

ss77 = (int) **Math**.round(robot.Interface.s7[i + 1] \* 0.8667);

ss8 = (int) **Math**.round(robot.Interface.s8[i] \* 0.8667);

ss88 = (int) **Math**.round(robot.Interface.s8[i + 1] \* 0.8667);

data1278.setColor(**Color**.YELLOW);

data1278.drawLine(j, ysize - 10 - ss1, j + 5, ysize - 10 - ss11);

data1278.drawLine(j, ysize - 10 - ss2, j + 5, ysize - 10 - ss22);

data1278.drawLine(j, ysize - 10 - ss7, j + 5, ysize - 10 - ss77);

data1278.drawLine(j, ysize - 10 - ss8, j + 5, ysize - 10 - ss88);

data36.setColor(**Color**.ORANGE);

data36.drawLine(j, ysize - 10 - ss3, j + 5, ysize - 10 - ss33);

data36.drawLine(j, ysize - 10 - ss6, j + 5, ysize - 10 - ss66);

data45.setColor(**Color**.RED);

data45.drawLine(j, ysize - 10 - ss4, j + 5, ysize - 10 - ss44);

data45.drawLine(j, ysize - 10 - ss5, j + 5, ysize - 10 - ss55);

j = j + 5;

}

}

}

@Override

**public** void paintComponent(**Graphics** g) {

**super**.paintComponent(g);

drawAxes(g);

datasensor(g);

}

}

* + - 1. **Motorgraphright class**

**package** Graficas;

**import** java.awt.BasicStroke;

**import** java.awt.Color;

**import** java.awt.Graphics;

**import** java.awt.Graphics2D;

**import** javax.swing.JPanel;

**public** **class** Motorgraphright **extends** JPanel {

int xsize = 400;

int ysize = 340;

int middleaxis = 202;

**public** void drawAxes(**Graphics** g) {

//AXIS PROPERTIES

Graphics2D axis = (Graphics2D) g;

axis.setColor(**Color**.DARK\_GRAY);

axis.setStroke(**new** BasicStroke(1.5f));

axis.drawLine(10, middleaxis, xsize - 10, middleaxis);

axis.setStroke(**new** BasicStroke(3.0f));

axis.drawLine(10, 10, 10, 328);

//MARKERS PROPERTIES

Graphics2D markers = (Graphics2D) g;

markers.setColor(**Color**.DARK\_GRAY);

markers.setStroke(**new** BasicStroke(2.0f));

// Draw markers on y-axis

**for** (int i = 10; i < ysize; i = i + 32) {

markers.drawLine(8, i, 13, i);

}

}

**public** void datamotor(**Graphics** g) {

int elected;

int speed;

int err;

int csignal;

int speed1;

int speed2;

int error1;

int error2;

int cs1;

int cs2;

**try** {

elected = **Integer**.parseInt(robot.SerialClass.valuesMotor[1]);

speed = **Integer**.parseInt(robot.SerialClass.valuesMotor[2]);

err = **Integer**.parseInt(robot.SerialClass.valuesMotor[3]);

csignal = **Integer**.parseInt(robot.SerialClass.valuesMotor[4]);

} **catch** (**Exception** e) {

elected = 0;

speed = 0;

err = 0;

csignal = 0;

}

int i = 0;

int j = 10;

**for** (i = 0; i < 94; i++) {

robot.Interface.speedright[i] = (robot.Interface.speedright[i + 1]);

robot.Interface.errright[i] = robot.Interface.errright[i + 1];

robot.Interface.csignalright[i] = robot.Interface.csignalright[i + 1];

}

**if** (elected == 1) {

robot.Interface.speedright[94] = **Math**.abs(speed);

robot.Interface.errright[94] = err;

robot.Interface.csignalright[94] = csignal;

}

//CREATING GRAPHICS

Graphics2D gspeed = (Graphics2D) g;

Graphics2D gerr = (Graphics2D) g;

Graphics2D gcsignal = (Graphics2D) g;

gspeed.setStroke(**new** BasicStroke(2.0f));

gerr.setStroke(**new** BasicStroke(1.0f));

gcsignal.setStroke(**new** BasicStroke(1.0f));

//DRAWING

**for** (i = 0; i <= 93; i++) {

speed1 = (int) **Math**.round(robot.Interface.speedright[i] \* 0.64);

speed2 = (int) **Math**.round(robot.Interface.speedright[i + 1] \* 0.64);

error1 = (int) **Math**.round(robot.Interface.errright[i] \* 0.64);

error2 = (int) **Math**.round(robot.Interface.errright[i + 1] \* 0.64);

cs1 = (int) **Math**.round(robot.Interface.csignalright[i] \* 0.64);

cs2 = (int) **Math**.round(robot.Interface.csignalright[i + 1] \* 0.64);

gspeed.setColor(**Color**.RED);

gspeed.setStroke(**new** BasicStroke(2.0f));

gspeed.drawLine(j, middleaxis - speed1, j + 4, middleaxis - speed2);

gerr.setColor(**Color**.YELLOW);

gerr.setStroke(**new** BasicStroke(1.0f));

gerr.drawLine(j, middleaxis - error1, j + 4, middleaxis - error2);

gcsignal.setColor(**Color**.BLUE);

gcsignal.drawLine(j, middleaxis - cs1, j + 4, middleaxis - cs2);

j = j + 4;

}

}

@Override

**public** void paintComponent(**Graphics** g) {

**super**.paintComponent(g);

drawAxes(g);

datamotor(g);

}

}

* + - 1. **Motorgraphleft class**

**package** Graficas;

**import** java.awt.BasicStroke;

**import** java.awt.Color;

**import** java.awt.Graphics;

**import** java.awt.Graphics2D;

**import** javax.swing.JPanel;

**public** **class** Motorgraphleft **extends** JPanel {

int xsize = 400;

int ysize = 340;

int middleaxis = 202;

int refn = 4;

**public** void drawAxes(**Graphics** g) {

//AXIS PROPERTIES

Graphics2D axis = (Graphics2D) g;

axis.setColor(**Color**.DARK\_GRAY);

axis.setStroke(**new** BasicStroke(1.5f));

axis.drawLine(10, middleaxis, xsize - 10, middleaxis);

axis.setStroke(**new** BasicStroke(3.0f));

axis.drawLine(10, 10, 10, 328);

//MARKERS PROPERTIES

Graphics2D markers = (Graphics2D) g;

markers.setColor(**Color**.DARK\_GRAY);

markers.setStroke(**new** BasicStroke(2.0f));

// Draw markers on y-axis

**for** (int i = 10; i < ysize; i = i + 32) {

markers.drawLine(8, i, 13, i);

}

}

**public** void datamotor(**Graphics** g) {

int elected;

int speed;

int err;

int csignal;

int speed1;

int speed2;

int error1;

int error2;

int cs1;

int cs2;

**try** {

elected = **Integer**.parseInt(robot.SerialClass.valuesMotor[1]);

speed = **Integer**.parseInt(robot.SerialClass.valuesMotor[2]);

err = **Integer**.parseInt(robot.SerialClass.valuesMotor[3]);

csignal = **Integer**.parseInt(robot.SerialClass.valuesMotor[4]);

} **catch** (**Exception** e) {

elected = 0;

speed = 0;

err = 0;

csignal = 0;

}

int i = 0;

int j = 10;

**for** (i = 0; i < 94; i++) {

robot.Interface.speedleft[i] = robot.Interface.speedleft[i + 1];

robot.Interface.errleft[i] = robot.Interface.errleft[i + 1];

robot.Interface.csignalleft[i] = robot.Interface.csignalleft[i + 1];

}

**if** (elected == 2) {

robot.Interface.speedleft[94] = **Math**.abs(speed);

robot.Interface.errleft[94] = err;

robot.Interface.csignalleft[94] = csignal;

}

//CREATING GRAPHICS

Graphics2D gspeed = (Graphics2D) g;

Graphics2D gerr = (Graphics2D) g;

Graphics2D gcsignal = (Graphics2D) g;

gspeed.setStroke(**new** BasicStroke(2.0f));

gerr.setStroke(**new** BasicStroke(1.0f));

gcsignal.setStroke(**new** BasicStroke(1.0f));

//DRAWING

**for** (i = 0; i <= 93; i++) {

speed1 = (int) **Math**.round(robot.Interface.speedleft[i] \* 0.64);

speed2 = (int) **Math**.round(robot.Interface.speedleft[i + 1] \* 0.64);

error1 = (int) **Math**.round(robot.Interface.errleft[i] \* 0.64);

error2 = (int) **Math**.round(robot.Interface.errleft[i + 1] \* 0.64);

cs1 = (int) **Math**.round(robot.Interface.csignalleft[i] \* 0.64);

cs2 = (int) **Math**.round(robot.Interface.csignalleft[i + 1] \* 0.64);

gspeed.setColor(**Color**.RED);

gspeed.setStroke(**new** BasicStroke(2.0f));

gspeed.drawLine(j, middleaxis - speed1, j + refn, middleaxis - speed2);

gerr.setColor(**Color**.YELLOW);

gerr.setStroke(**new** BasicStroke(1.0f));

gerr.drawLine(j, middleaxis - error1, j + refn, middleaxis - error2);

gcsignal.setColor(**Color**.BLUE);

gcsignal.drawLine(j, middleaxis - cs1, j + refn, middleaxis - cs2);

j = j + refn;

}

}

@Override

**public** void paintComponent(**Graphics** g) {

**super**.paintComponent(g);

drawAxes(g);

datamotor(g);

}

}

* + - 1. **CommunValues class**

**package** Graficas;

**import** java.awt.BasicStroke;

**import** java.awt.Color;

**import** java.awt.Graphics;

**import** java.awt.Graphics2D;

**import** javax.swing.JPanel;

**public** **class** CommunValues **extends** JPanel {

int xsize = 400;

int ysize = 340;

int middleaxis = 170;

int refn = 4;

**public** void drawAxes(**Graphics** g) {

//AXIS PROPERTIES

Graphics2D axis = (Graphics2D) g;

axis.setColor(**Color**.DARK\_GRAY);

axis.setStroke(**new** BasicStroke(1.5f));

axis.drawLine(10, middleaxis, xsize - 10, middleaxis);

axis.setStroke(**new** BasicStroke(3.0f));

axis.drawLine(10, 10, 10, 328);

//MARKERS PROPERTIES

Graphics2D markers = (Graphics2D) g;

markers.setColor(**Color**.DARK\_GRAY);

markers.setStroke(**new** BasicStroke(2.0f));

// Draw markers on y-axis

**for** (int i = 10; i < ysize; i = i + 32) {

markers.drawLine(8, i, 13, i);

}

}

**public** void datamotor(**Graphics** g) {

int elected;

int lineout;

int bias;

int bias1;

int bias2;

int lineout1;

int lineout2;

**try** {

//Reads value from the Serial

elected = **Integer**.parseInt(robot.SerialClass.valuesMotor[1]);

lineout = **Integer**.parseInt(robot.SerialClass.valuesMotor[2]) / 2;

bias = **Integer**.parseInt(robot.SerialClass.valuesMotor[3]) / 2;

} **catch** (**Exception** e) {

bias = 0;

lineout = 0;

elected = 0;

}

int i = 0;

int j = 10;

**for** (i = 0; i < 94; i++) {

robot.Interface.bias[i] = robot.Interface.bias[i + 1];

robot.Interface.lineout[i] = robot.Interface.lineout[i + 1];

}

**if** (elected == 0) {

robot.Interface.bias[94] = bias;

robot.Interface.lineout[94] = lineout;

}

//CREATING GRAPHICS

Graphics2D gbias = (Graphics2D) g;

Graphics2D glineout = (Graphics2D) g;

gbias.setStroke(**new** BasicStroke(1.0f));

glineout.setStroke(**new** BasicStroke(1.0f));

//DRAWING

**for** (i = 0; i <= 93; i++) {

bias1 = (int) **Math**.round(robot.Interface.bias[i] \* 2.9);

bias2 = (int) **Math**.round(robot.Interface.bias[i + 1] \* 2.9);

lineout1 = (int) **Math**.round(robot.Interface.lineout[i] \* 2.9);

lineout2 = (int) **Math**.round(robot.Interface.lineout[i + 1] \* 2.9);

robot.Interface.textbias.setText(**Integer**.toString(robot.Interface.bias[i] \* 2));

robot.Interface.textlineout.setText(**Integer**.toString(robot.Interface.lineout[i] \* 2));

gbias.setColor(**Color**.black);

gbias.drawLine(j, middleaxis - bias1, j + 4, middleaxis - bias2);

glineout.setColor(**Color**.MAGENTA);

glineout.drawLine(j, middleaxis - lineout1, j + 4, middleaxis - lineout2);

j = j + 4;

}

}

@Override

**public** void paintComponent(**Graphics** g) {

**super**.paintComponent(g);

drawAxes(g);

datamotor(g);

}

}

* + - 1. **Capture class**

**package** robot;

**import** java.awt.AWTException;

**import** java.awt.Rectangle;

**import** java.awt.Toolkit;

**import** java.awt.image.BufferedImage;

**import** java.io.File;

**import** java.io.IOException;

**import** java.util.Date;

**import** javax.imageio.ImageIO;

**public** **class** Capture {

**static** void capturar() **throws** **AWTException**, **IOException** {

**Date** day = **new** **Date**();

**String** fileName = **String**.format("%tY%<tm%<td%<tH%<tM%<tS.png", day);

BufferedImage image = **new** java.awt.Robot().createScreenCapture(**new** **Rectangle**(**Toolkit**.getDefaultToolkit().getScreenSize()));

ImageIO.write(image, "png", **new** **File**(fileName));

}

}

## Control system

### Control system block diagram



Figure 24 control system block diagram