



PROJECT REPORT:
NIRYO MOBILE BASE



Sébastien Chrétien, Cong Chen,
Cédric Leveneur, Olivier Jombart,
Guillaume Halloy, Annemarie
Kokosy, Gilles Tagne

Summary:

Mechanical Part:.....	3
I] SolidWorks :.....	3
II] Engine block:	4
III] Elevator:	5
Electronic Part:	7
I] Batteries:	7
II] Shields :	7
III] Wiring:	8
Sensor Part:	9
I] Infrared sensors:	9
II] Ultrasonic sensors:	10
III] Limit switches:.....	11
IV] LED matrix:.....	13
V] LCD screen :.....	15
VI] RFID :	17
VII] Linear potentiometer:.....	18
Integration part:	20
I] Hardware Diagramm :	20
II] Software Diagramm :	20
III] Code :	21
Conclusion :	22
I] Summary of each function:	22
II] Summary of each sensor:.....	23

Mechanical Part:

] SolidWorks :

For the modeling part of the robot we chose to use SolidWorks. This allowed us to design each piece separately and then to assemble them with each other by applying certain constraints. We could also place each piece in a defined place in order to better organise the robot before realizing it (see Fig1). This allowed us to perform simple simulations of our system on the computer to see if all the parts were correctly designated and corresponded to our needs.

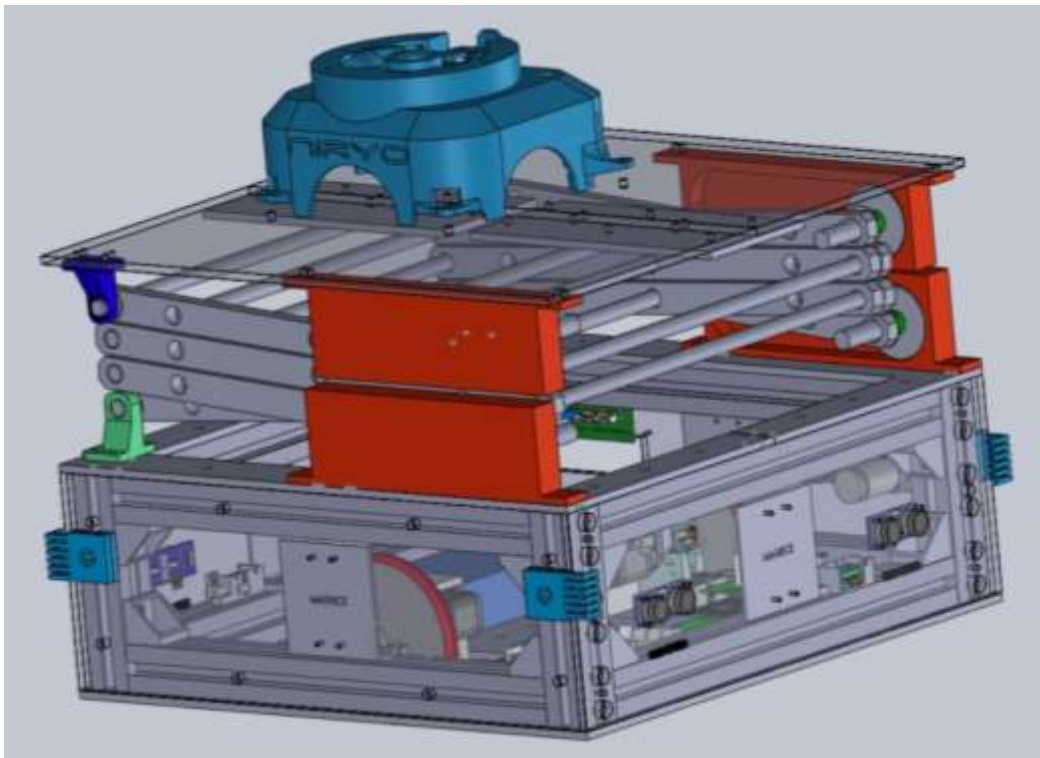


Figure 1: Robot modeling (screenshot on SolidWorks)

All the 3D printed parts as well as the laser cutout plans and the assembly of the various parts are present in the "Mechanical module" folder on Gitlab. The 3D printed parts on the robot are the latest versions that do not need touch-ups. In case of breakage just reprint them. The assembly and the plexiglass plates also correspond to the current modeling of the robot. However, we have noticed that the modeling of the Arduino Uno board and ultrasonic sensors do not correspond exactly to the real parts. Indeed when we place the holes on the modeling, the components do not fix themselves correctly. However in reality the fixing holes placed on the plexiglass plates are not aligned with those of the components. Therefore, we had to spike the plexi plate where the Arduino Uno board is attached but not for the ultrasonic sensors in view of the fact that they remained fixed without it.

II] Engine block:

For the motor part of the robot we chose to use the AX-12A motors from Dynamixel (see Fig2) because they have the technical characteristics (see Tab1) which is closest to the new model that will be released soon and will be used by the following. We then ordered wheels, hubs, D-drive axles and more at Robotshop. When receiving the orders, we were surprised to find that the axes did not fit properly with the rest of the parts, however, recommended for this type of axis. That's why we had to print in 3D hubs to be fixed on the motors as well as various supports for the other parts so that the axes could be used.



Figure 2: Dynamixel engine AX-12A

To fix the axles to the hubs of the engines, we first placed a screw in the PLA of the hub to take advantage of the fact that it was a D-axis. However after multiple uses, the threading the PLA hub eventually left and the screw supposed to hold the axis is raised. This resulted in the engine running in a vacuum. The solution for this problem was to drill the pins slightly where the screws only touched them. Thus the screw could hold the motor shaft and those regardless of the wear of the PLA hub.

To control the Dynamixel motors it is necessary to use a communication protocol of their own. As Niryo had already developed a code to use this language on Raspberry we used a Raspberry to transmit the orders of Arduino Due. Arduino and Raspberry communicate in CAN, via a SPI / CAN module between the two cards. We used the three libraries given by Niryo for that, they are in the folder of all our codes.

Feature	Value
Dimension (mm)	32x50x40
Weight (g)	54,6
Resolution	0,29°
Reduction ratio	254 : 1
Stall torque (Nm) at 12V / 1.5A	1,52
Speed without load (rpm) at 12V	59
Operational angle	0° - 300°
Supply voltage	9V – 12V
Max current	900mA
Operating temperature	-5°C – 70°C
Communication speed	7343bps – 1 Mbps

Table 1: Technical Characteristics of Dynamixel Engine AX-12A

III] Elevator:

For the elevator platform we were inspired by already existing systems, especially those present on the construction nacelles (see Fig3). We therefore opted for a system consisting of three crossed scissors pushed by a jack. The cylinder is fixed to the rest of the robot by an axis fixed on the plates of plexi located on the lower part of the robot. Similarly, a 3D printed part is the connection between an axis of the elevator system and a fixed axis at the end of the jack. We also printed four rails for the elevator wheels. In addition, we made several prototypes so that the rails no longer damage the wooden connecting rods and include the linear potentiometer used to determine the height of the elevator.

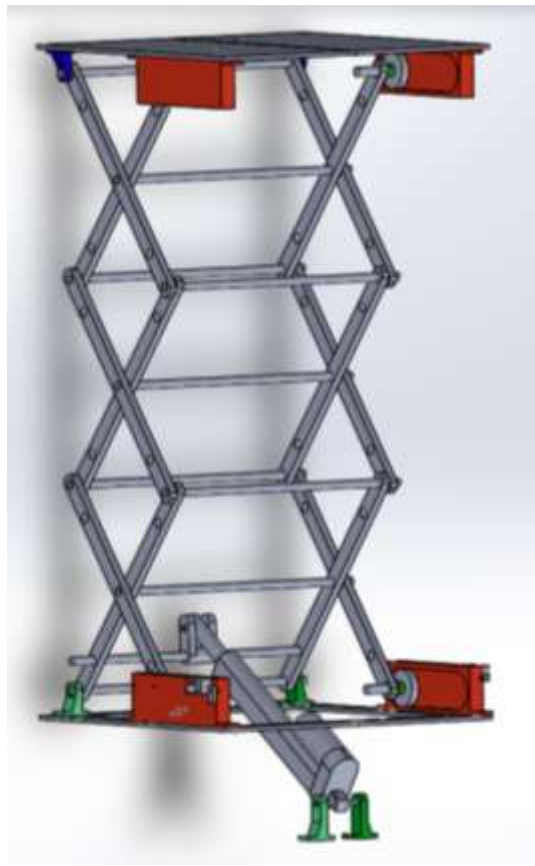


Figure 3: SolidWorks Elevator Modeling

In order to save time and money, we made the wooden connecting rods. In steel, they would have been more resistant and less cumbersome but that implies to make rods to measure which would have taken a lot of time and also a lot of money. Therefore, we can not lower the point of attachment of the cylinder because otherwise the forces exerted on the rods of the elevator will be too large thus causing the breakage of the latter. And that's why if you want the elevator to climb higher, you have to change the material of the rods so that they are more resistant.

The jack used is a jack (for features see Tab2) of twenty centimeters at rest, it is a translation cylinder that we use to activate the lifting part. The cylinder is controlled by a motor driver that takes two logic signals that indicate the direction of the cylinder and another input for the power to be transmitted to the cylinder. To do this we used three pins PWM cylinder: pins 32 and 33 for logic signals and pin 34 for power. When the cylinder is at rest the elevator is at a height of thirty-seven centimeters, when the cylinder is at the highest possible the elevator is at a height of eighty centimeters.

Feature	Value
Voltage (V)	12
Current (mA)	500
Speed (mm / s) without load	3-40
Load capacity (N)	100-1500
Race (cm)	12

Table 2: Technical characteristic of the jack (cylinder)

Electronic Part:

I] Batteries:

To satisfy the autonomy required by the specifications, we opted for lithium ion batteries (see Fig4) so as not to weigh down the robot. These batteries can deliver a voltage of 12.6V when charged and 10.6V when discharged. They also have a capacity of 20Ah. In the event of a short circuit, the batteries are able to be safe or when the current draw is high. That's why we had to use a second battery. Indeed, the cylinder requires a lot of current in addition to the rest of the components thus engaging the safety of the batteries. Subsequently it will be necessary to find a battery that can provide more power to the system to optimize this part.



Figure 4: photo of the robot battery

II] Shields :

We use three microcontrollers to control the robot: a Raspberry Pi 3, an Arduino Due and an Arduino Uno. Each of its microcontrollers has its own Shield to avoid any cable disconnection problem. For the Raspberry, we took the Shield developed for the Niryo One because it only served us to transmit commands to Dynamixel engines to advance the robot. We then made a Shield for the two Arduino whose diagrams are present on the Gitlab in the folder "PCB module". The one made for the Arduino Uno, only provides the necessary connections for power supply, I2C communication, linear potentiometer and RFID.

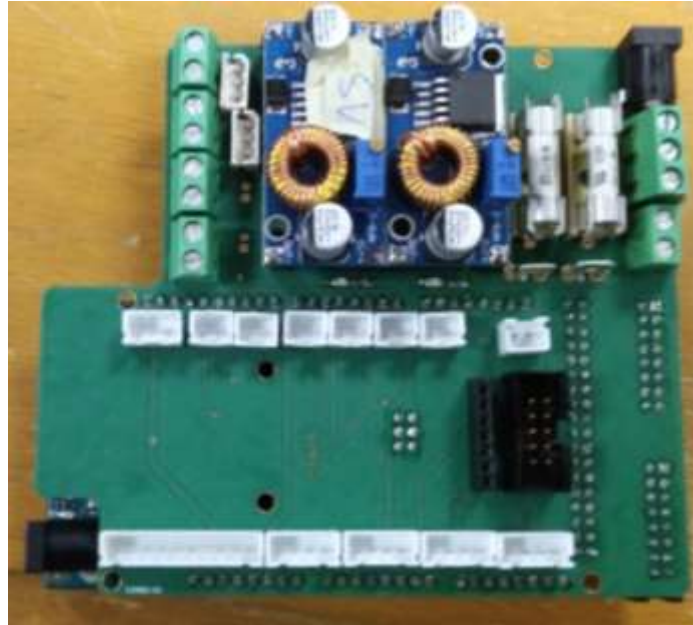


Figure 5: Arduino Due Shield Photo

On the Shield of the Arduino Due (see Fig5) we have all the connections with the sensors and the different HMIs. We also have connections to the cylinder driver, the Raspberry and the Arduino Uno. There is also the entire power supply board that is present on the Shield and that provides power (3.3V, 5V or 12V) to all components of the robot. The power supply board part is protected by two fuses supporting a maximum voltage of 250V and a maximum current of 10A. On the latter Shield, we had a problem feeding the Arduino board, which resulted in the Arduino Due microcontroller being burned. By taking current measurements at the Arduino power terminals, we noticed that the current supplied to the Arduino was about 300mA, five times more than usual, for a voltage of 12V. Not knowing which component of the Shield was problematic and to save time, we made a new Shield by having it varnished this time to prevent a cable from touching a track.

III] Wiring:

For the sake of clarity we have defined some conventions for the color and type of cable to use. We used red and black to symbolize diet and mass. All other colors will be used for data communication. The colors that can be encountered are: white, beige, purple, blue, green, yellow, orange, brown and gray. To connect the sensors and HMI on the Shields, we used keys to avoid making mistakes in the connections. Then for the cables of the battery we used pods which are difficult to remove to avoid that the batteries can be disconnected inadvertently. Finally for the supply of the other cards we used wire-to-card terminals so that the cables can not be removed by accident.

Sensor Part:

All sensors communicate in SPI with the Arduino board to which they are connected. The Arduino boards communicate in I2C, the Arduino Due card is considered as the master card (it receives the data sent by the Arduino Uno) and the Arduino Uno card is considered as the slave card (it sends the information to the Arduino Uno card). Arduino Due). The exchanges are made in the direction slave towards master, to simplify the communications there is not exchange data in the direction master towards slave board.

1] Infrared sensors:

To follow a black line on the ground we used ultrasonic sensors from Pololu, model QTR-8RC (see Fig6). The advantage of these sensors is that they are sold in the form of a bar of eight sensors. This saves us time because it is exactly what we need to achieve our line follower without making ourselves the sensor array. We have one of his bars at the front and another at the back of the robot. We use one or the other if the robot moves forward or backward. In terms of programming, these sensors already have a library of their own and allows us to use the sensors easily. We also found that the sensors could work with an acquisition time of 10ms between each measurement, which is more than enough compared to what we want to do.

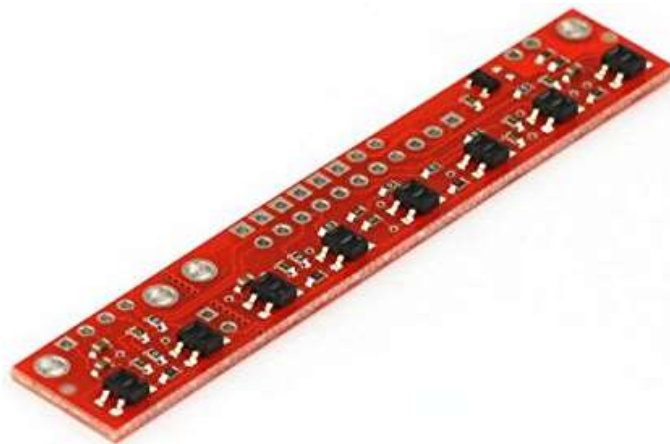


Figure 6: Infrared sensor array

We use these sensors with a latency of 10ms between each measurement five times and then there is a pause of about 250ms to allow other sensors to perform their measurements. To program these sensors we used the library "QTR_Sensor" that we found on online tutorials, the library is on the Gitlab in the folder "Libraries". We did not make any modifications on this library because it already allowed us to carry out the measurements that we wanted. This allowed us to obtain measurement values between 0 and 2500, it is a contrast measurement with 0 for something very clear and 2500 for something very dark. Of course it is possible to change this configuration but we found it more convenient and intuitive to keep this configuration.

In practice we rarely get a value more than 900 when the sensors are not on the black line, most often the values are around 250 for the laboratory floor. For the black line it depends on the lighting and also the state of cleanliness of the ground layout, usually we obtain values higher than 1500 for a line that has been posed for some time already and we have values that can reach 2000 for a recent layout. We were able to get this scale of value through different tests that we did at different

times, to check the impact of wear on the plot, and with different lighting to see the impact of the latter on the returned values by the sensors.

For the sensor connections we used pins of sixteen pins in two rows, including eight pins PWM for reading information. For the front sensors it is the connections shown in Fig7, recess for the rear sensors pins are different (from 46 to 53 instead of 22 to 29). Power and ground are not directly connected to the Arduino, the Shield also serves as a power board and converts the voltage provided by the battery to redistribute it to the sensors.

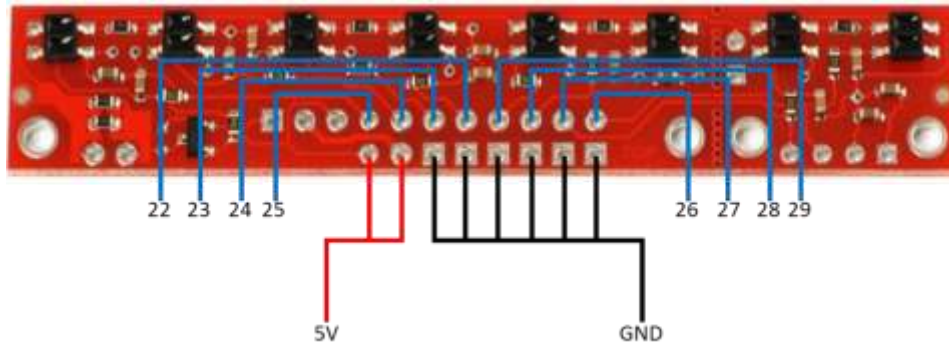


Figure 7: Hardware diagram of the front infrared sensor

II] Ultrasonic sensors:

For the detection of obstacles, we took ultrasonic sensors very used HC-SR04, marketed by Neuftech (see Fig8). There are two of these sensors on the front and back of the robot. We only use two of these sensors at once, those in front if the robot is moving forward and those in the back if the robot is moving backwards. We saw in the documentation that the sensors were able to operate at an acquisition time of 10ms, which is also much faster than we need. They are also easy to use: at first, it is allowed to emit a sound for a certain duration, greater than or equal to 10ms, then the echo returned by the possible obstacles is captured.



Figure 8: image of an ultrasonic sensor

We perform a measurement with two of its sensors every 100ms to allow the line follower and the RFID to make their own measurements. At first one of the sensors will emit for 10ms before we capture the echo of this emission, then the second sensor will do the same after a delay of 100ms. This delay makes it possible to make the echo of the first emission disappear so as not to disturb the

second. Then we use the value Convert the measured value to a distance to get a more intuitive value and easier to use. To do this we did not need to use a sensor-specific library.

In practice we made different attempts to define an equation that could accurately gives us the obstacle distance. So depending on the distance the robot will slow down as and when: from forty centimeters the speed will be equal to three quarters of the command, from thirty centimeters will be half, from twenty centimeters will be a quarter and finally at ten the robot stops. It was during his tests that we realized that sensors sometimes had trouble detecting cotton clothing when they were close but that they could also have some difficulty with distant metal surfaces. When this problem arises the sensors returns us a null value that can not be interpreted as a useful value.

For the connections we have chosen to use polarizers so as not to be able to reverse the connections and thus damage the components. The sensor uses two pins from the Arduino (see Fig. 9), one for the program and the other for the reception of the echo. In Fig 9 we can see the connections of the front and rear sensors. The power supply and ground are not directly connected to the Arduino, the Shield also serves as a power supply board is converted the voltage provided by the battery to redistribute it to the sensors.

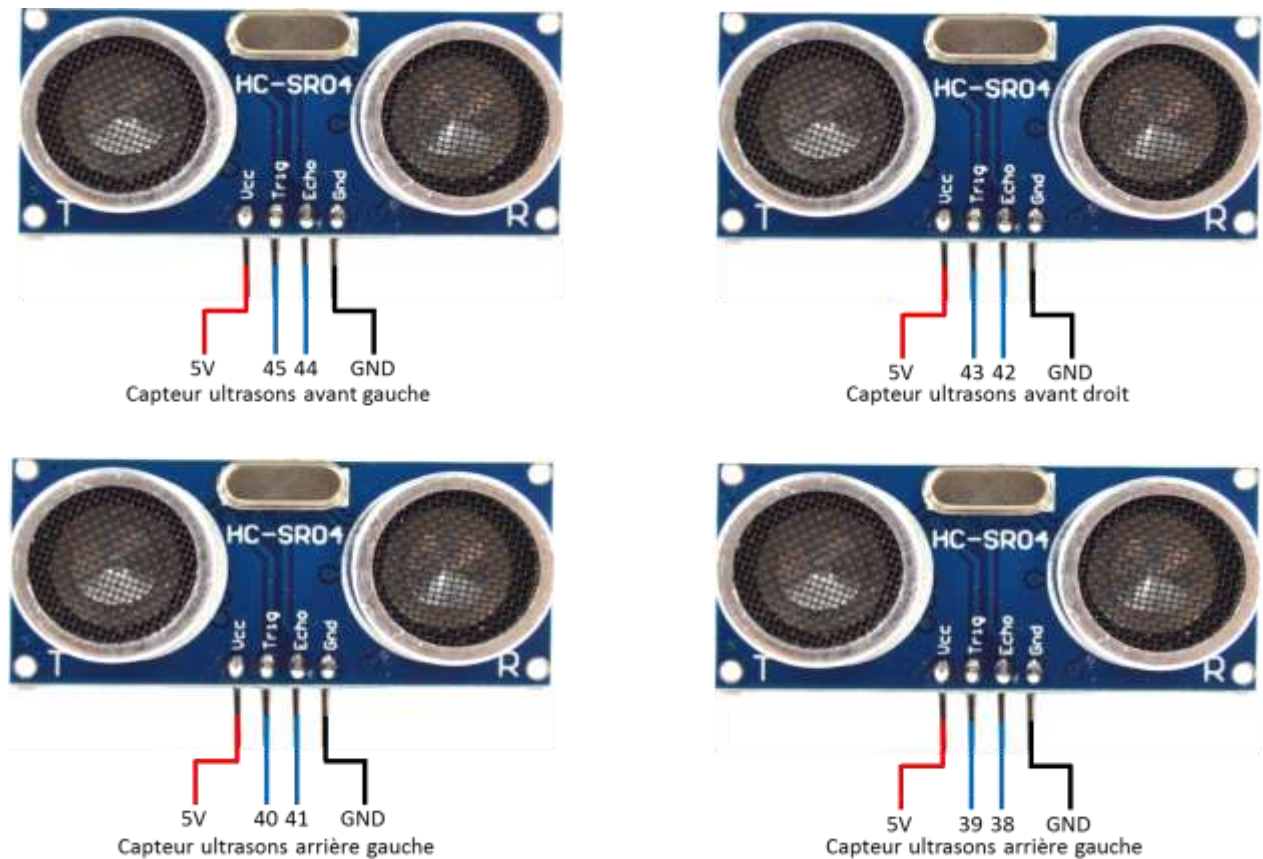


Figure 9: Hardware diagram of an ultrasonic sensor

III] Limit switches:

For the detection of obstacles we have chosen end-of-stroke sensors (see Fig10) from Honeywell. This is a switch that we have coupled with plexiglass plates that can move to realize a system of bumper

for the robot. Thus, each time the robot encounters a shock at the front or rear of the corresponding plexiglass plate will be pushed to engage the limit switches directly in contact with it.



Figure 10 : Limit switches

In practice the limit switches are similar to switches, therefore it is not necessary to use any library to be able to measure their state. The limit switch has only two possible states, a high state and a low state. We found that the limit switches were in the high state when not engaged and in the low state when they were engaged.

So we just had to place a resistor directly after the mass of the limit switches to be able to read their states. We used an OR logic gate for the four limit sensors for the front and rear bumper. This is not to use three pin Arduino unnecessarily, no matter where the shock comes if you have to stop if there is one, and it simplifies and reduces the code. Thus we use the pin 19 of the Arduino (see Fig 11) for the limit sensors relative to the bumper and we use pin 18 for the end of stroke sensor indicating the collapsed position of the elevator. The power supply is not directly connected to the Arduino, the Shield also serves as a power board that is converted to the voltage provided by the battery to redistribute it to the sensors.



IV] LED matrix:

For the remote man-machine interface part with the user, we used the RGD LED matrices from Adafruit (see Fig12). We have used four so the user can easily see what the robot is like no matter where it is. Despite a fairly high price, about 30 euros, they are very practical and easy to use. Hardware level, just connect three pins on our Shield: power, mass and control. Then the other three matrices connect directly to the matrix that precedes it. At the software level, the matrix-specific libraries manage the serial connection of matrices, which allows us to program the four matrices as if it were only one large matrix. The negative point of this type of display is that it is limited in the complexity of what it can display. As a result, the information returned remains simplistic and transmits only the operating state of the robot at a given moment: advance, recoil, elevator running, encounter a shock or obstacle and if the BAU is engaged



Figure 12 : LED matrix

In practice we have used three libraries, present in the directory "Librairies" present on the Gitlab, found on the internet to use its matrices: Adafruit_GFX, Adafruit_NeoPixel and Adafruit_NeoMatrix. Its libraries are also available in the Arduino Library Manager but the ones we use have been modified by third parties to make them easier to use and also add some additional functions. Thus we only have to program the arrays in the form of tables by using the intensity of blue, red and green via hexadecimal writing, example 0x0F0 for a green LED at a maximum intensity. Thus each LED of the matrix can be programmed independently so that we can display the message of our choice. The information displayed is visible in Tab3.

Then the communication between the four matrices 8X8 RGB and the Arduino Due is done thanks to a single communication pin. Indeed matrices being connected in series with each other the libraries treat them as a single matrix 16x16 RGB. The programming is done as if we had only one matrix. Then the matrices receive an address on this artificial network according to their order of connection, the matrix directly connected to the Arduino at address 1 that which is connected to it receives the address 2 and so on. Thus we only use pin 12 of the Arduino for dies (see Fig13). The power supply and ground are not directly connected to the Arduino, the Shield also serves as a power

supply board is converted the voltage provided by the battery to redistribute it to the first matrix that transmits it to others.

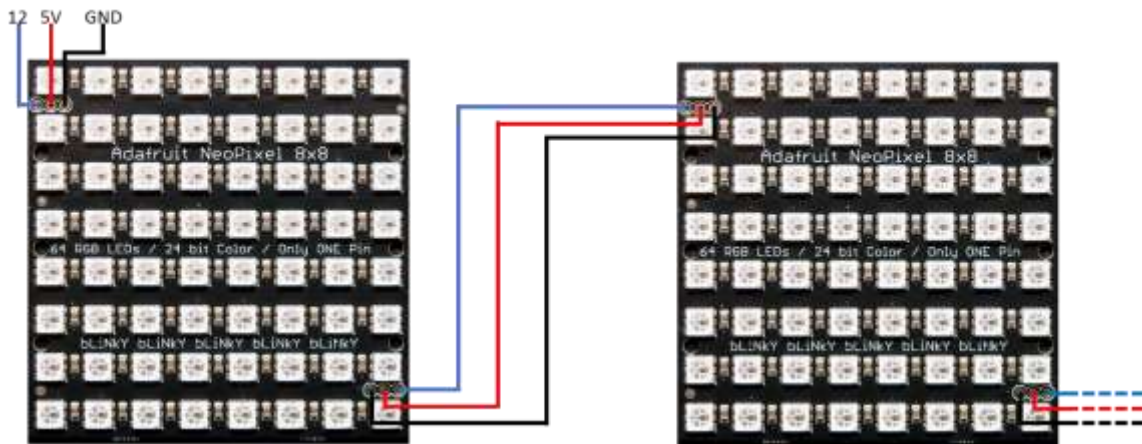
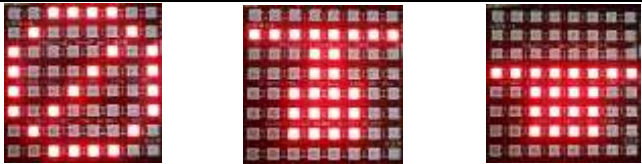



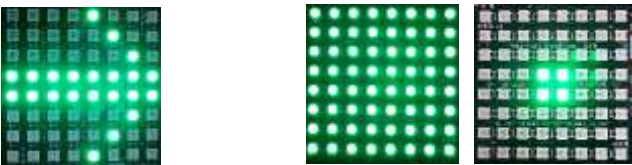



Figure 13: hardware diagram of LED arrays, here we only represent the first two

Robot status	When does it start?	LED matrices
BAU enabled	The user activates the emergency stop button	
Bumper enabled	The robot between collision with an obstacle	
Ultrasound enabled	The robot detects an obstacle	
Lost line	The color detected by the infrared sensors is not black	
The robot moves forward / backward	Infrared sensors detect color	 <p>On the sides to place at the front and back</p> <p>Announce the meaning</p>
The actuator activated	The RFID module detects a card	

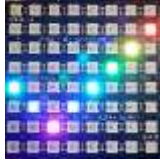



initialisation	Step that appears when launching the program		
		Well initialized	Bad initialization
unknown error	Error that is not save and not processed by the robot		

Table 3: message displayed on the matrices

V] LCD screen :

For more complex communications as well as to facilitate debugging, we used a Raystar LCD (see Fig 14). On this LCD screen, we can display messages explaining more precisely the cause of the error, the state of charge of the battery, what position is the robot, ... The negative point of this screen is that it must be ready enough to be able to discern what is written on it and it is therefore little used.

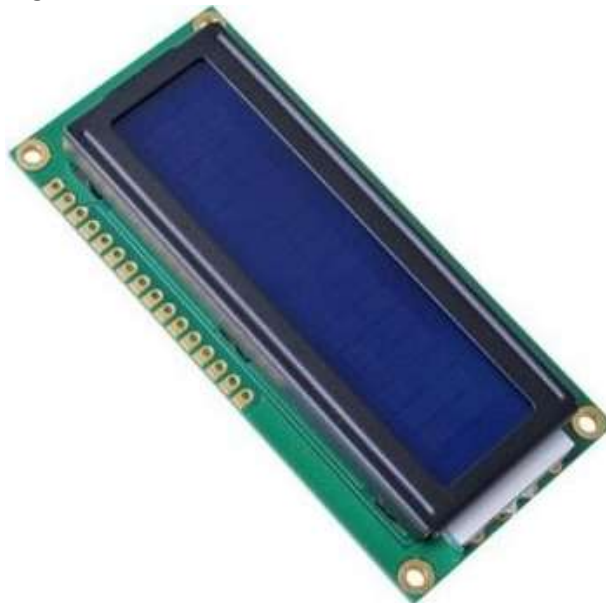


Figure 14: LCD screen

In practice we have used the "LiquidCrystal" library, version 1.0.7, available in the Arduino library manager. This is a two-line LCD that can each contain sixteen characters. This LCD screen is used to obtain more precise information on the state of the robot, in particular a more precise explanation of the errors (see Tab4). We can also use this screen for debugging but we preferred to use the Arduino monitor for this.

The library allows us to simply display a sentence on one of the two previously selected lines, taking into account that we can not display more than sixteen characters per line. We used six pins to be able to control this LCD screen (see Fig15): 2,3,4,5,6 and 30. We thus use four pins for the information display, two for each line, another pin to reset the screen and finally a last to select when to inform the screen that it will receive information. The potentiometer is used to adjust the brightness

of the screen, it must be adjusted manually. The power supply and ground are not directly connected to the Arduino, the Shield also serves as a power supply board is converted the voltage provided by the battery to redistribute it to the first matrix that transmits it to others.

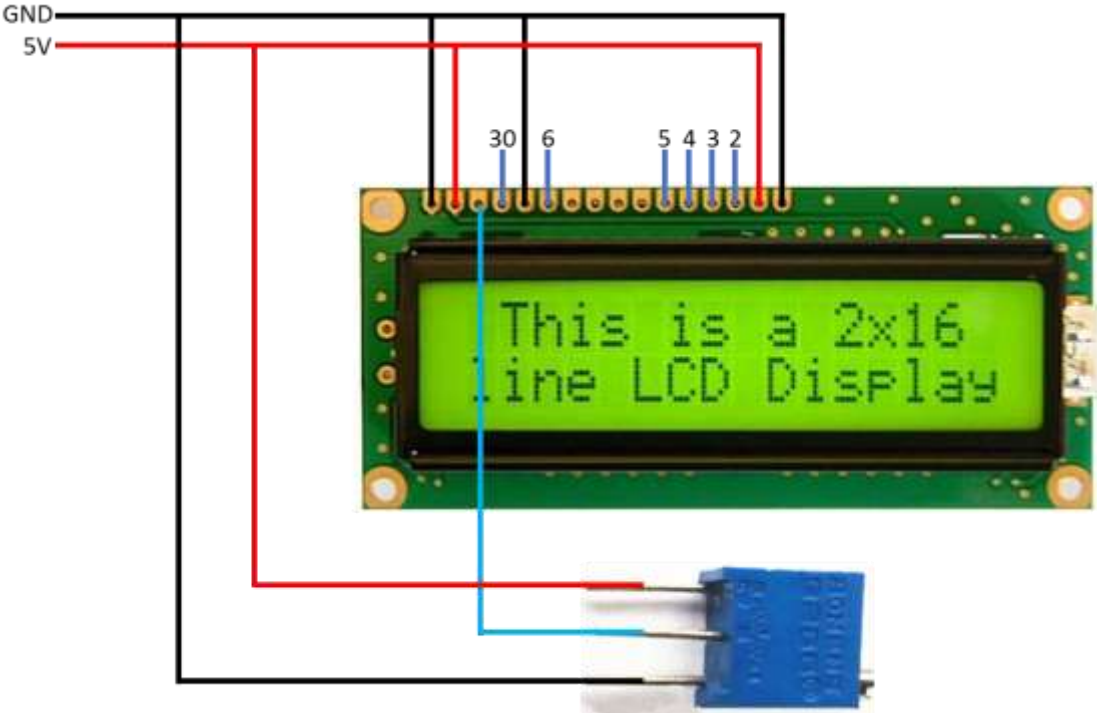


Figure 15: Hardware diagram of the LCD screen

Robot status	When does it start?	LCD screen display
BAU enabled	The user activates the emergency stop button	***ROBOT STOP*** BAU ENABLE
Bumper enabled	The robot between collision with an obstacle	***ROBOT STOP*** BUMPER(S) ENABLE
Ultrasound that detects an obstacle	The robot detects an obstacle	***ROBOT STOP*** OBSTACLE DETECTED
Lost line	The color detected by the infrared sensors is not black	
The robot moves forward / backward	Infrared sensors detect color	
The actuator activates	The RFID module detects a card	Displays the name of the detected map
initialization	Step that appears when launching the program	

Table 4: message displayed on the LCD screen

VI] RFID :

To stop the robot at a points of interest we chose to use RFID technology with the RC522 module from AZDelivery (see Fig16). This module can detect any badge or RFID card, so we could also work with the cards and badges provides the RFID module with our student card. As each card and badge has a different tag we were able to list them all and associate them with different properties. The advantage of this module is that it has a significant detection distance.



Figure 16: RFID kit image (the RC522 module, a badge and a card)

In practice we have used two libraries to use this module: the "Wire" library, version 1.0.0, found in the Arduino library manager and the "RFID" library that we found on the internet, available in the "Libraries" directory of the Gitlab. Thus we were able to recover the RFID tags from our cards in the same form as an IP address, example: 230.98.180.225.209. The RFID module is placed on the Arduino Uno because some pins that are needed by the module was already used for CAN communication with the Raspberry.

After different tests we have been able to note that the module manage to detect an RFID tag through a large surface: sheet of paper, plate of plexi, garment, hand, ... Moreover the maximum detection distance can go up to ten centimeter, not applicable between the module and the tag. We did our tests as well as the practice with cards and not with the badges because it often happens that the robot rolls inadvertently on the cards, which is not always possible with the badges which are thicker.

We try to detect the presence of an RFID tag every 10ms around so as not to miss the point of interest. Once a tag is detected the Arduino Uno sends it directly to the Arduino Due via an I2C connection. We use five pins from the Arduino Uno to operate the RFID module (see Fig. 17). The power supply and ground are not directly connected to the Arduino, the Shield also serves as a power supply card that is converted to the voltage provided by the battery to redistribute it to the first matrix that transmits it to the module.

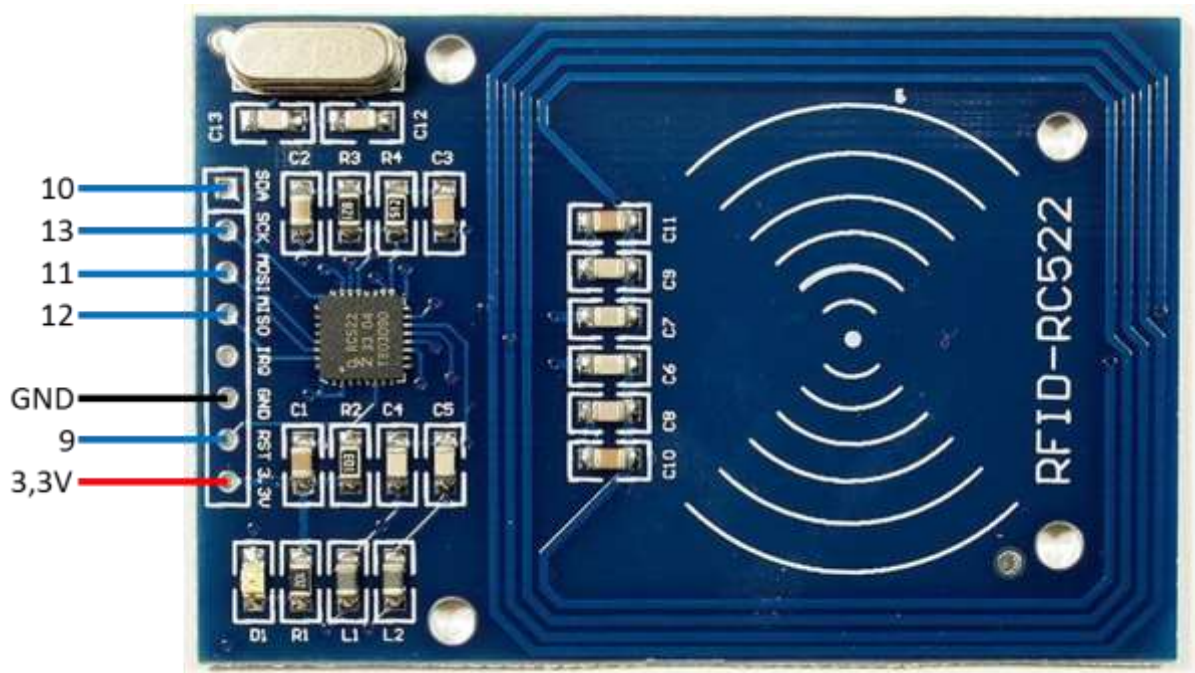


Figure 17: hardware diagram of the RFID module

VII] Linear potentiometer:

Finally we chose to use a linear potentiometer (see Fig18) to be able to measure the stroke of the elevator. We opted for a linear potentiometer because none of the axes had linear rotation during the activation of the cylinder. So we decided to measure the stroke of one of the wheels in its rail to deduce the height of the elevator. The link between the wheel and the linear potentiometer is provided by a piece of PLA which will land on the tip of the potentiometer at one end and frame the wheel of the other. We used a linear potentiometer from Alps with a 10cm stroke because we did not find bigger.



Figure 18: Image of the linear potentiometer

In practice we have not planned to use any library to use this sensor because it is a variable resistor. However it was necessary to place a resistance in the input of the potentiometer during our first test because when it is in its minimum position the resistance is null and the system is damaged if we do not put an additional resistance. The values returned by the potentiometer are between 0 and 3000 approximately, when we use the entire stroke of the potentiometer. This will allow a good approximation of the height of the elevator platform afterwards.

The linear potentiometer is plugged into the Arduino Uno because when designing the Shield for the Arduino Due we had not yet chosen which technology we would use to get the height of the board. We use a single analog pin of the Arduino Uno to be able to read the value of the sensor (see Fig19). The power supply and ground are not directly connected to the Arduino, the Shield also serves as a

power supply board that is converted from the voltage supplied by the battery to redistribute it to the first matrix that transmits it to the potentiometer.

Originally we plan to look at the value of the potentiometer every 100ms once an RFID tag has been detected. However, due to some problem with the Arduino Due Shield, we were late because we had to find the source of the problem, a component of the Shield, and change the Shield before the end of the project. Therefore we integrated the potentiometer hardware level, the cables are connected to the Arduino Uno Shield, but the measured value is not processed or sent to the Arduino Due.



Figure 19: hardware diagram of the linear potentiometer

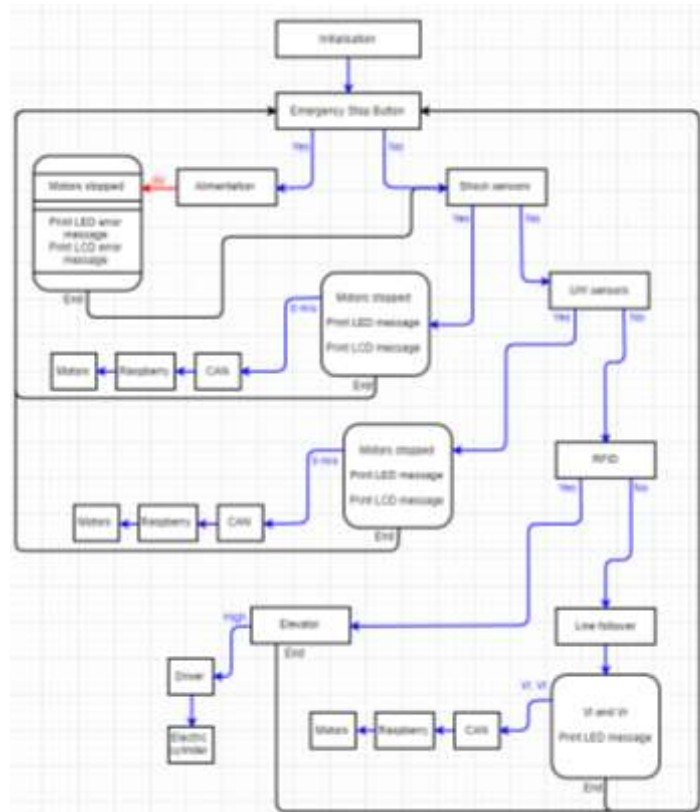


Figure 13 : diagramme software du robot

III] Code :

For the code part we integrated each separately created part as we could test them in order to anticipate the final integration of the sensors. Thus we realized several programs of integrations for the robot that one finds on the Gitlab. In each version in the "Sensor module" folder, there are more or less important changes to each new version. We have grouped the main integration codes in the "Useful code" folder in order to limit the number of code.

Conclusion :

I] Summary of each function:

Lift function:

Operational part:

- The cylinder is powerful enough to lift a load of 5 kg
- Finalized structure

Part to improve:

- Less solid wooden connecting rod than steel
- Do not go up to 1m high
- No position feedback currently

Communicate with the user:

Operational part:

- Distance and loan communication
- Several simple messages to convey the state of the robot
- debug interface

Part to improve:

- Find more messages to pass to LCD
- Find cheaper matrices

Autonomy:

Operational part:

- The battery is correctly feeding the whole robot
- Has a range greater than 4 hours

Part to improve:

- Need to use two batteries
- Long charging time

Line follower:

Operational part:

- High performance infrared sensor
- Can move forward and backward

Part to improve:

- Difficulties managing important turns

Detection in case of shock:

Operational part:

- Operational end-of-travel sensor
- Operational shock system
- The robot stops as soon as there is shock

Part to improve:

- Find a more optimal way to fix plexiglass plates

Obstacle detection:

Operational part:

- Detects obstacles
- Reduces speed before impact
- Stop before shock

Part to improve:

- Refine obstacle detection and slowdown

Stop at a specific place:

Operational part:

- RFID module correctly programmed
- Each place has a height in the program
- Raise the elevator at each detection

Part to improve:

- Pass the RFID module on the Arduino Due

II] Summary of each sensor:

Component	State	Recommendation
Infrared sensor	Integrated and operational	Find shorter sensor arrays
End of race sensor	Integrated and operational	
Ultrasonic sensor	Integrated and operational	Bring them a little closer to the ground to avoid low obstacles like steps
RFID module	Integrated and operational via Arduino Uno board and I2C bus	Toggle the module on the Arduino Due card to simplify communications
cylinder	Raises the elevator to about 70cm	Enslaving him in height
Engine	Integrated and operational	Subsequently it may be necessary to find more powerful engines if you want to move larger loads
LED matrix	There is an animation for each state of the robot	Trying to find cheaper matrices
LCD screen	There is an error message for each problem that the robot may encounter	Find him a place where he will be more visible
BAU	Integrated and operational	
Drums	Two batteries used to provide the necessary current for the robot	Find another battery that can deliver more power with faster charging
Linear potentiometer	Integrated hardware level but not software level	Integrate it at the hardware level to slave the cylinder into position