T2i Interaction Lab

Documentation

by

Ernesto Lozano Calvo & Johannes Karlsson

INDEX

1. Project overview

The main goal to pursue in this project is to create (two) autonomous robots that can move independently in a room, while dealing with obstacles on the way, e.g. humans, objects, etc. Future work in the area will involve further coordinated maneuvers, concerning both human-robot or robot-robot interaction levels.

Structurally, each individual robot looks as follows (Figure 1):

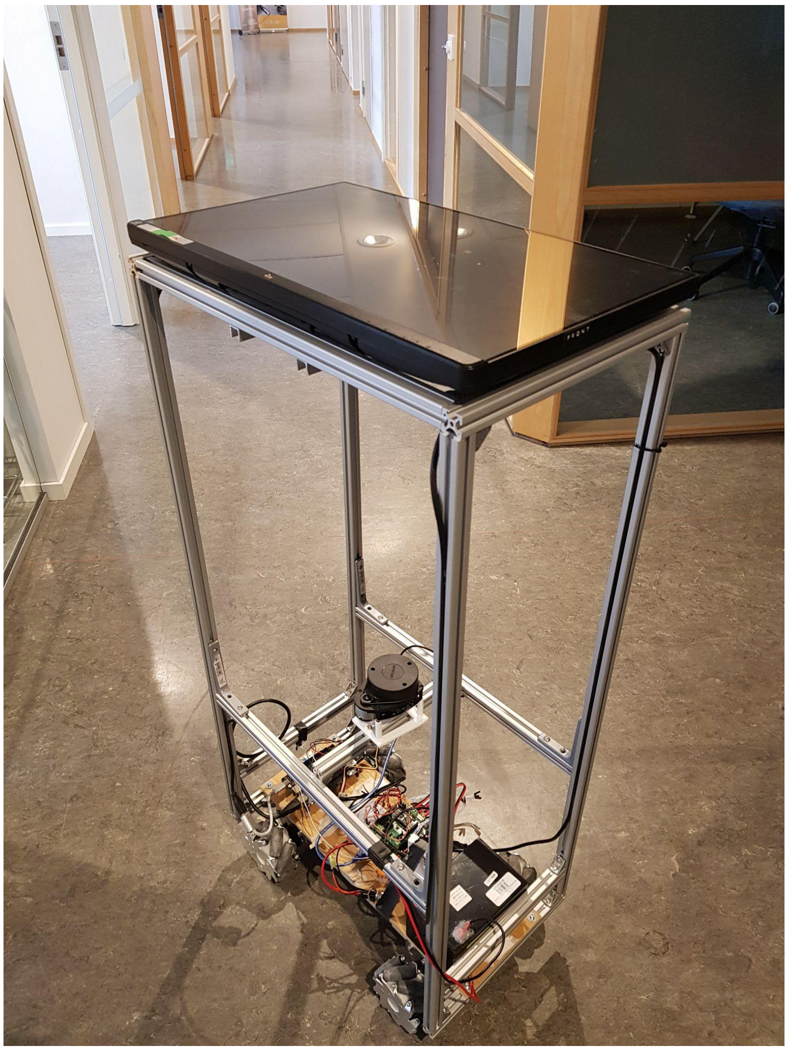


Figure 1: Robot

It visually pops out the existence of a touch-screen built at the top. The latter can be considered the main element for the user to interact with.

In parallel, more complex hardware is found. These range from the microcontroller itself, to actuators (motors) and numerous sensors, further detailed below.
The role played by sensors is fundamental in a project of this nature. Acting as the robots' "eyes", they allow the perception of environment events; this is the reason why most of the focus in the remaining material is centered around them.

1.1. Block diagram

From an implementation point of view, two very differentiated parts are distinguished to ensure software-hardware compatibility and further coding: an Arduino-based part and a Windows-based (Python-programming) one.
Aware of this division, detailed on incoming sections, the robot's block diagram is shown below (Figure 2). Its visualization is useful to look at the different building blocks as a whole.

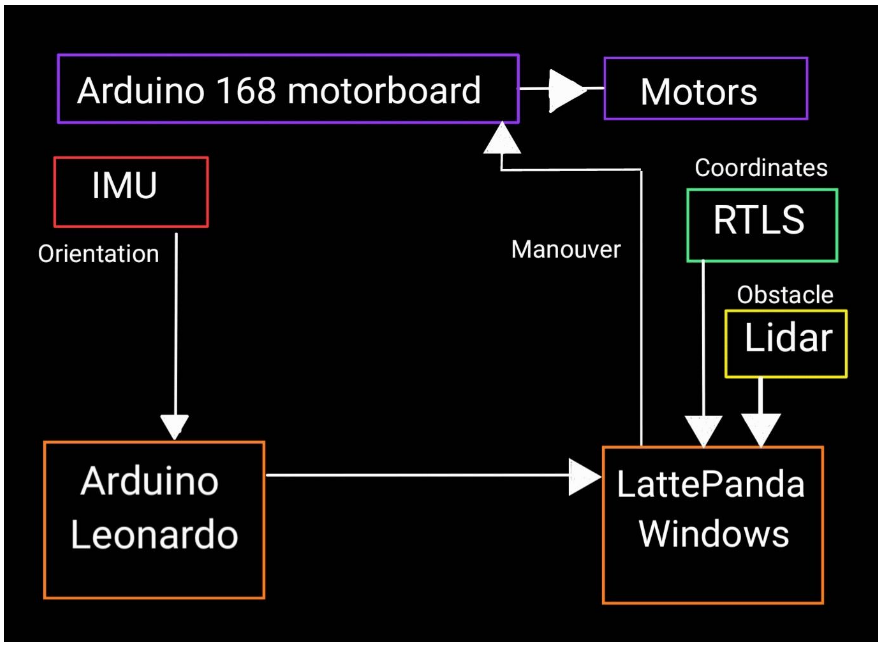


Figure 2: Block diagram

Starting with the "brain" of the robot, the microcontroller processes all the data coming from the rest of the sensors and components, and is in charge of the decision-making. The one of use, called Lattepanda, has Windows 10 as operating system and counts with an Arduino Leonardo in-built, see the *Components* chapter for more details.

The robots move using omnidirectional wheels. Their motion is controlled by an Arduino Duemilanove, and an extension board Arduino 168 motorboard. The communication is directly established with the microcontroller.

To determine its orientation (heading) in the room, an Inertial Measurement Unit (IMU) is used. The IMU is connected to the Arduino part of the microcontroller (Arduino Leonardo).

To locate the robot in a coordinate system (room's map), a Real Time Location System (RTLS) is used. Its output corresponds to the real-time position in an area delimited by the RTLS sensors' placement. The anchor beacons that construct the map corners are just powered with batteries, while the one integrated in the bot is connected to the Windows part of the microcontroller via USB.

Finally, a LiDAR is employed for obstacle detection. By scanning 360º objects with a laser beam, the positions of obstacles placed around the bot are detected to be processed in the decision-making algorithm. The communication is done via USB directly to the Windows part.

1.2. General instructions of use

1.2.1. Getting started

To power on the Lattepanda (Windows side), the power button should be pressed until the blue blinking LED becomes stationary powered on. The power button is the one inside the red circle in the figure below:

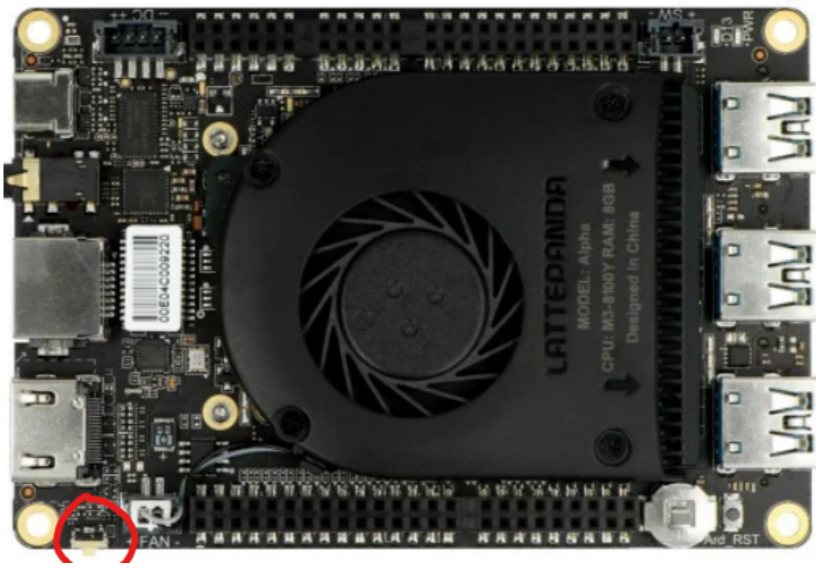


Figure 3: Lattepanda board

To be able to work on the computer, there is a possibility to connect another computer to the Lattepanda via TeamViewer. The two bots named Alpha and Beta have different TeamViewer IDs which are displayed below:

Table 1: Teamviewer info

Bot	TeamViewer ID	Password
Alpha	770810136	mecanum123
Beta	392007124	mecanum123

Once connected, in the desktop one can find the main programs of use to program and run the robots.
Visual Studio Code is the IDE picked for the software development of the Windows part.
The classical Arduino IDE does not allow running several Arduino programs independently. Both the motors and the IMU make use of Arduino code, but they are using two completely different boards. Because of this reason, the use of both the standard Arduino IDE for the code concerning the IMU and, in parallel, the utilization of the Arduino Pro IDE (also found on the desktop) for the motors bring successful results. Note the latter IDE version does allow opening multiple tabs, so it could perfectly be used alone (without the standard IDE), but the previous solution (using both) is preferred for stability (Pro IDE is in early beta version and may present bugs).

1.2.2. Code: Github repository

The code needed to run the implemented robots can be found in the Github repository "T2I-Mechanum".
Inside it, two main folders can be seen. The first one, denoted as "latest_mecanum" contains the final and useful results, to be run. Alternatively, older project status work can be found in the folder "previous_work". Note it corresponds to older approaches, some which presented a different block diagram than the one introduced (the main structural change being the use of sonars instead of Lidar); but also some foundational steps in an MQTT protocol of communication design, for future maneuvers that require bots communication. Therefore, we recommend checking the content of the folder before starting something from scratch, as it may have been tried in the past already.
Assuming the first folder is accessed, two folders follow: "Final_Integration" and "IndividualComponents_code". The former contains all the code needed to get the autonomous movement up and running; whereas the latter just holds each individual implementation for completeness.

1.2.3. Code: Running the robot

To run the robot, the following steps have to be completed:

- Place the RTLS beacons in the room, making sure they create an area of around 5x5m, where the corners are given by the beacons :

```
UL  —------  UR
 |            |
LL  —-------  LR
```

- Locate the robot in the room in its starting position. Note! Its orientation (heading angle) will be the one that the IMU will consider as reference (0 degrees) from now on (until python program shutdown) so make sure you agree with it. We recommend placing it parallel to the x axis of the 5x5 square sides.

- Open and run the Motors_control_arduino.ino with Arduino Pro IDE.

- Open and run Orientation_IMU_arduino.ino with Arduino standard IDE.
- Run the python file “Autonomous_Robot.py” in the Visual Studio terminal (being placed at the file location) to see the robot move!
- Stop its execution anytime by pressing Ctrl+c in the terminal window.

2. Components

In this chapter, every vital component's functionality is described.

2.1. List of components

The following components are used for the robots:

- Microcontroller - Lattepanda Alpha 864s.
- Touch-Screen - Viewsonic TD2230
- Battery - 12 V 12 Ah LiFePO4
- Motorboard - Nexus kit Arduino 328
- 4 DC motors with omnidirectional wheels
- RTLS - Decawave MDEK1001
- IMU - Grove - IMU 9DOF v2.0
- LiDAR - RPLidar A1

2.2. Main building-blocks

2.2.1. Microcontroller: Lattepanda Alpha 864s

The Main controller for the robot is the Lattepanda. This is both an Arduino and Windows microcontroller. The controller is connected to Lidar, screen, RTLS and motor controller board via USB. The screen is also connected via HDMI. The screen needs the USB input for touch capability. The IMU is connected to the Arduino pins of the board according to the figure and table below.

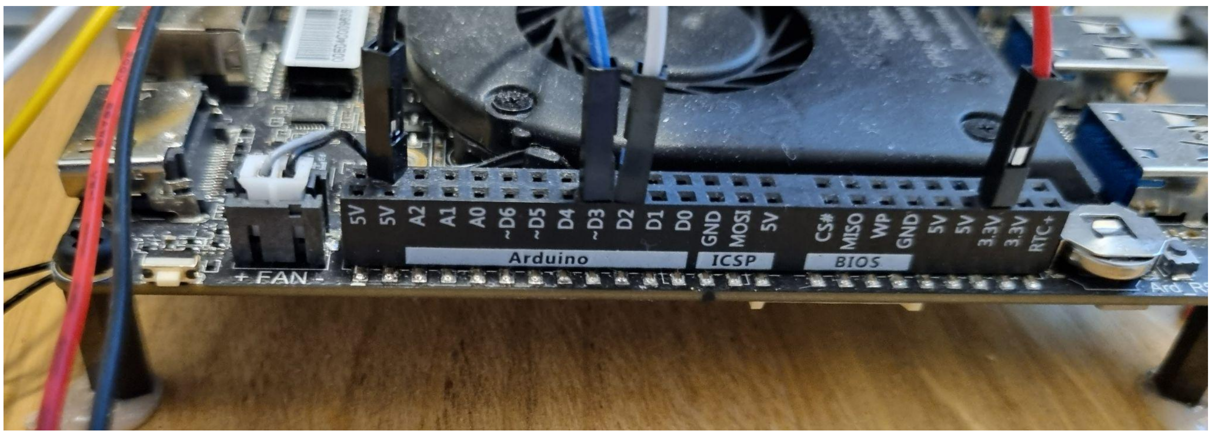


Figure 4: Lattepanda connection to IMU

Table 2: Lattepanda connection to IMU

Red Cable	VCC
Black cable	gnd

White cable(D2)	SDA
Blue cable(D3)	SCL

The Lattepanda can sometimes be rather slow when operating. Thus it is recommended to use github to be able to work on a better computer and only use Lattepanda when needed. However just running the robot as it is intended works well, it only becomes slow when performing computationally heavy tasks.

2.2.2. Battery

The batteries used for the different robots are rated at 12 Volt and a capacity of 12 Ampere hours (Ah). The batteries power the Lattepanda, motor board and screen. These are connected in parallel to each other. Further specifications of the battery can be found in the data sheet.

The batteries in general have good battery life, but there is currently no possibility to check the batteries state of charge when the bot is in use. To charge, the included charger should be plugged into the 3 pin battery charger. If the battery completely runs out of charge, the charger might not be able to charge the battery if the battery is connected to the rest of the circuit. Hence the battery needs to be charged whilst no other components are connected. Thus, the rest of the robot can not be connected to the battery at the same time. If the battery still has charge in it, then it can be charged whilst the bot is turned on. When the robot is turned off, removing the cable from the anode of the battery will save battery life, since some of the components will still draw power even though the bot is off. For example the Lidar will still spin even though the computer is off.

2.2.3. Motors driver

The motors used for the movement are controlled via an Arduino powered controller. The wheels are so called omnidirectional, which means that the bot can move in any direction, not just backwards and forwards as conventional wheels. This allows the wheels to have higher mobility in the limited space available. The motors are controlled using Arduino IDE, the code receives a command from the windows side (Lattepanda). The speed at which the bot is supposed to move is controlled by constants in the arduino code. The speed is set to values that give satisfactory movement speed and friction to the floor. The motors are not perfectly mounted, this means that there will be differences in the rotational speeds and the robot will not travel exactly as expected. Checking the physical connection of the motors one can see that some motors are slackly mounted, hence the resistance for the motor will be larger and they will have small deviations in the speed. This could be solved by disassembling the motors from the bot and also dismantling the wheels from the motor shaft and straightening out the motor wheel. Due to time constraints, there was no time to attend to this issue.

The arduino code to drive the motor, applies a signal to the wheels that corresponds to the certain action that is activated. Each motor is connected to different pins that control the speed and direction of the motor.

2.2.4. Real Time Locating System (RTLS)

RTLS stands for Real Time Location System and is used to locate the robots position in the room. The system used is from Decawave and the model is MDEK1001 Development Kit. The system is based on using anchors and tags. The anchors in this case are used to define the edges of the room/space in use. This is setup for a 5x5 space but it can be changed. The tags are the devices that are connected to the bots using usb cable. The tags then output its own position in the room to the Lattepanda.
There is an apk available for this system to be able to change settings of the different devices. The android app can be downloaded from the MDEK1001 product page.

When using the RTLS to determine the location of the bot, it is essential to place the 4 different anchors at the correct spot (seen before in the running bot section) and at the correct distance from each other. The anchors are labeled for LL- lower left, UR-Upper left e.t.c corresponding to the anchors placement. By placing the sensors at correct distances from each other there will not be as many sample errors. Small distance deviation for the anchors can cause some samples to be incorrect. The position data sent through usb is then processed using python.

The anchors are powered using batteries, whilst the tags connected to the pc are powered through USB. The battery life is great and recharge time is short using the included cell charger.

2.2.5. Inertial Measurement Unit (IMU)

The IMU or Inertial Measurement Unit is used to determine the orientation of the robot. The sensor used is an Grove - IMU 9DOF v2.0. This board is compatible with Arduino and hence is connected to the LattePanda. The board has an Accelerometer, Magnetometer and Gyroscope. All of these have 3 dimensional measurement capabilities. But to acquire the orientation of the bot the sensor is mounted to the wooden floor in the direction of the bot. Then the orientation is obtained by integrating the output of the Gyroscope Z-value (yaw-axis). Then by integrating the result and adding it to the already existing orientation, the orientation is determined. But this causes some errors over time for the sensor. This is eliminated by removing the DC-drift of the signal. However there is still an aquleminative error present when turning. For now this error is too small to be significant, and there are larger errors when using the motors. So if new motors are used, then the gyro-drift could cause errors.

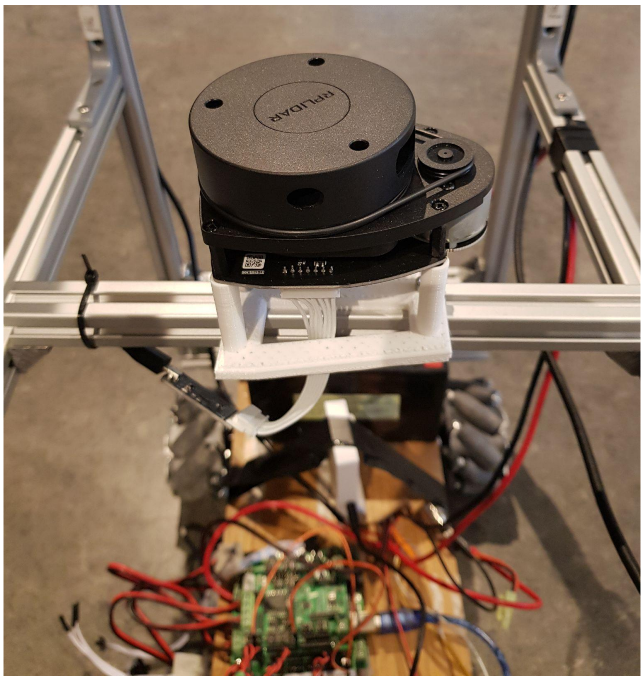
The code is set up in such a way that the front of the robot needs to be aligned with the X-axis to get a correct initial setup. Then the bot is able to receive the orientation over serial from the arduino. This is done by running a Python script that collects the data sent through arduino. A small bug that was discovered is when uploading the python script without first recompiling the arduino code, the actual orientation gets a +10 offset to the actual orientation. This is not a problem when the code does not need to recompile too many times.

It is common to filter the gyrodata using the Magnetometer when using the yaw axis data, however the magnetometer on this device seems to be having some problems. We found it really hard to make the sensor be able to give good data.

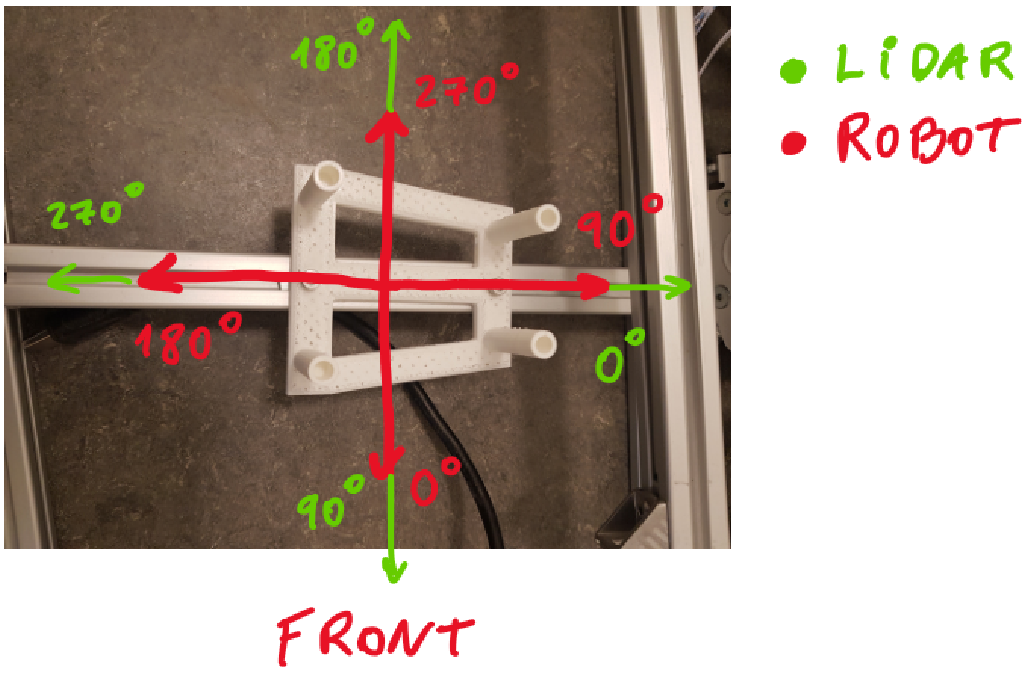
2.2.6. LiDAR

The LiDAR, corresponding to the model RPLidar A1, is used for obstacle detection and programmed using Python, through the library RPLidar from Adafruit.

Its placement on the robot is located in the middle of the robot, on top of bars and resting on a 3D printed mounting platform (3D model to print again can be found on github repository), as seen in the figure below. Note the robot's front corresponds to the place from which the picture is taken.



OBS! A very important mounting detail has to do with the angle correspondence between Lidar and robot frames. With the mounting in the images, the robot's front, 0 degrees in red, would correspond to measurements from the Lidar at 90 degrees and so on. This is compensated for in the code (read coding comments in the lidar implementation file), but it is vital to know.



The LiDAR algorithm implemented has two distinguished parts: data acquisition and data processing. The first part, outcome of an infinite loop (all the time gathering data), obtains pairs of angles and distances of objects placed around, from 0 to 159 degrees.

Once collected, some decisions have to be done based on data. Two different implementations were explored: clustering (polar/cartesian coordinates) or command-based, but finally the latter was implemented.

The first technique suggests clustering the data, i.e. grouping the detections in clusters with the aim of getting points with similar information, which will likely correspond to the same object in space. This way, a meaningful analysis of obstacles around can be done, to keep track of them over time. This approach was implemented mainly in cartesian coordinates with the aim of, afterwards in the integration (autonomous algorithm), dodging the obstacles based on their position in x and y.

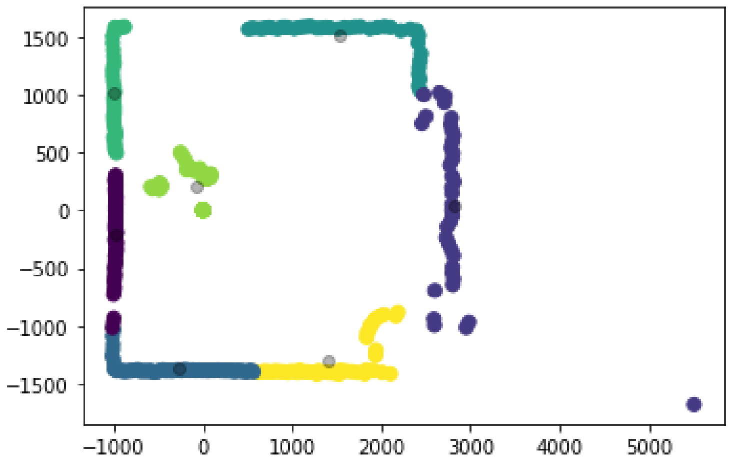


Figure 4: Clustering approach (in Cartesian)

Given the fact the final autonomous decision-making algorithm was done based on angles, instead on positions of x and y, an alternative lidar coding scheme was suggested, working instead in polar coordinates, as it comes from the lidar (angle, radius) and checking obstacles based on the command the robot is executing, which proved to be simple and efficient enough to get great results. However, the previous path is explored and implemented enough to continue from there if desired.

Just, as small observations, note the bars corresponding to the robot's structure are filtered out from the detections; and recall when mounting the robots, the LiDARs must be placed at slightly different heights to avoid laser reflections from each other.

3. Conclusions and future work

A successful autonomous robotics solution is provided for the T2I project.

The handed robots are capable of, given an initial position in a room and a desired setpoint, move to the target autonomously, while dealing with obstacles on the way (waiting till the obstacle is no longer present).

This is achieved by putting together valuable information coming from position, orientation and obstacle detection sensors.

The implementation suits the hardware installed, where two well-defined environments can be found (Arduino and Python), but which are integrated using the second (mainly PySerial library).

3.1. Possible improvements

The natural continuation of work from the point in which the project is found would be:

- **Developing a better heading (yaw angle) estimation algorithm**. The implemented solution integrates the gyro data and therefore suffers from drifting, which damages the estimation (progressively) in the long term.

  The implementation of an Extended Kalman Filter (EKF) or similar is the recommended approach. This would consider making use of the Magnetometer, fusing it with the accelerometer and gyroscope. Note the Magnetometer requires careful calibration and it is heavily affected by surrounding magnetic disturbances, so an outlier rejection approach will probably be needed.

- **Improving the motion algorithm:** Linear motion (x and y- completion) has been implemented, but it is obviously still far from optimal, so a more complex algorithm (path planning) could be explored.

  This would also fix an existing problem which takes place when the wheels slide over time, consisting in reaching the setpoint in x, changing to moving in y, but when achieving the setpoint in y, x is no longer as close as desired to it. Note of course further specific code could be implemented to fix this, ensuring the final setpoint is reached once both booleans of confirmation of x and y movements are received.

- **Improving the obstacle detection:** In the implemented code, obstacles on the way the robot moves are the only ones of interest. Further work could make decisions also based on other directions; and obstacles could be tracked over time (comparing with previous time instants) to see if they are moving or not.

  Moreover, in line with the latter, it could be programmed that the robot does not wait for an obstacle to go away if it is static for some time, replanning its trajectory based on it.

3.2. Issues and suggestions

The possible improvements described above correspond to the direct continuation of work, keeping the main structure as it is right now.

The work provided deals with many main choices of hardware imposed from the past. Even though it is perfectly possible to outperform the handed behavior by applying the suggestions listed, at this point in which the project is found, we consider it is a good moment to stop and carefully plan critical future steps.

<u>Small possible changes:</u>

Far from ideal, the robots present hardware **components that need urgent replacement**.

Some attention needs to be put to the wheels and overall structure, to ensure stability and proper movement, as it heavily conditions the final motion performance.

The motor driver does not work as supposed by the manufacturer (neither software nor hardware wise). A newer compatible substitute should be considered, moving towards better speed control capabilities.

These two points are heavily error prone, which may generate lower productivity in the development phase, as it was experienced by the consultants.

<u>Large possible changes:</u>

The development of robotic applications is rarely done in Windows, but instead using **Linux** as the operating system.

The choice of the LattePanda as microcontroller is actually a non-common pick. It is undeniable owning an Arduino in-built and Windows 10 are strong features. The latter might especially be useful for User Interfaces design for the touch-screen. However, using Windows instead of Linux is a great inconvenience from a software engineering perspective.

It highly seems future specifications will require the robot to communicate with other robots. **ROS/ROS2**, which works with Linux, is the standard platform choice for robotic development applications, as it sets the perfect environment for the integration and communication of different sensors and peripherals with each other to take place.

Its equivalent solution using Windows, with high chances, implies the development of a custom protocol of communication. Previous consultants explored the option of using MQTT, but turned out to be error-prone and slow.Such a path could be continued, but ROS would make life much easier in many aspects.

An Nvidia Jetson Nano is the suggested way to go if this path was considered. The fact it owns a gpu is perfect for AI applications, which could perfectly be added in the future if other sensors (cameras, etc) were considered. An additional Arduino board would be necessary to be bought, but not many changes in the already implemented code would be required.