**Universidade de Aveiro** Departamento de Engenharia Mecânica
**2017**

**Bruno Carvalho Vieira**

# Reconversão da Plataforma Robuter num AGV com Guiamento Visual

# Retrofitting of the Robuter's Platform to an AGV With Visual Guidance

**Universidade de Aveiro** Departamento de Engenharia Mecânica
**2017**

**Bruno Carvalho Vieira**

# Reconversão da Plataforma Robuter num AGV com Guiamento Visual

# Retrofitting of the Robuter's Platform to an AGV With Visual Guidance

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Automação Industrial, realizada sob a orientação científica de Vítor Manuel Ferreira dos Santos, Professor Associado do Departamento de Engenharia Mecânica da Universidade de Aveiro.

**o júri / the jury**

presidente / president

**Professor Doutor José Paulo Oliveira Santos**
Professor Auxiliar da Universidade de Aveiro

vogais / committee

**Professor Doutor José Luis Costa Pinto de Azevedo**
Professor Auxiliar da Universidade de Aveiro (arguente)

**Professor Doutor Vítor Manuel Ferreira dos Santos**
Professor Associado da Universidade de Aveiro (orientador)

**agradecimentos / acknowledgments**

"Quem caminha sozinho pode até chegar mais rápido, mas aquele que vai acompanhado, com certeza vai mais longe". Quero por isso, expressar aqui os meus profundos agradecimentos a algumas pessoas sem as quais não seria possível o sucesso desta dissertação.

Aos meus pais e irmão pela força e apoio incondicional que me deram durante a minha vida académica, mesmo quando isso representou a privação da minha presença durante longas jornadas.

À minha namorada, pelos excelentes conselhos de inglês que se revelaram úteis na realização deste documento. O seu constante acompanhamento e enorme compreensão nas alturas mais exigentes foi fundamental. Um obrigado por seres tão simples mas por marcares a minha vida académica de uma forma tão complexa, muita história foi escrita nestes anos.

Ao professor Vitor Santos pelo seu contagiante entusiamo, disponibilidade e partilha de conhecimento. Para além disso, a motivação que me transmitiu foi fundamental para conseguir superar as dificuldades encontradas.

Um particular agradecimento ao meu amigo Engenheiro Luís Emanuel Cristóvão, que foi a pessoa que, mesmo não estando envolvido neste projeto, mais o acompanhou e auxiliou. Sem a partilha do seu conhecimento prático, o fornecimento de material elétrico e a companhia nas maratonas de laboratório certamente não teria chegado tão longe.

Agradeço também a todos os restantes amigos e colegas, e em particular ao pessoal do LAR, com quem tive o privilégio de trocar ideias e partilhar conhecimento.

**palavras-chave** Robuter II, retrofitting, aquitetura modular, ROS, sistema de visão, AGV.

**resumo**

A automação industrial é uma parte fundamental para responder às crescentes necessidades do mercado. As empresas que desejam estar a par da competição, têm todas uma premissa em comum — melhorar a qualidade do produto, reduzindo custos e tempo de fabrico. Na última década, os veículos autonomamente guiados (AGV) têm ganho expressão devido ao facto de serem capazes de responder a esta última necessidade.

O contexto da presente dissertação insere-se no desenvolvimento desta temática através da recuperação de um equipamento com aplicações à escala industrial.

O objetivo principal passa por fazer o *retrofitting* da plataforma Robuter II, que se encontra obsoleta. Para isso, pretende-se substituir toda a parte de controlo por uma solução mais atual que permita a integração de diferentes módulos com base numa arquitetura distribuída baseada em ROS. Foi então necessário proceder a uma análise detalhada da plataforma existente para refazer toda a instalação elétrica da mesma, bem como proceder à instalação de diversos equipamentos.

No final, obteve-se um veículo capaz de navegar de forma manual (com controlo remoto) e foi ainda realizada uma prova de conceito de navegação autónoma através da realização de um algoritmo simples de visão artificial. Foi ainda verificada a versatilidade suficiente para integrar diferentes equipamentos e código desenvolvido de forma simples.

Neste documento é feita uma descrição geral do projeto onde são identificadas as várias etapas do mesmo e o trabalho desenvolvido até ao momento. Para além disso, é apresentada a forma como se irá proceder para a implementação das metas propostas. Por último, são apresentados resultados experimentais de navegação e integração de outros trabalhos na plataforma.

**keywords** Robuter II, retrofitting, modular architecture, perception system, ROS.

**abstract** The industrial automation has made its way in responding to the growing market's needs. Companies that wish to keep up with the competition have a common goal: improve the product's quality while reducing manufacturing costs and time.
In the last decade, the autonomous guided vehicles (AGV) have gained expression due to the fact that they are capable of responding this the last mentioned necessity.
The context of the current dissertation is inserted in the development of this thematic through the recovery of an equipment with industrial scale applications.
The main goal is centered in the Robuter II platform's retrofitting. For that, it is intended to replace the old control part by a modern solution that allows the integration of different modules based on a ROS distributed architecture. A detailed analysis of the existing platform was made in order to redo all the electrical installation, as well as to setup all of the used equipment.
In the end, a vehicle capable of navigating, both manual and autonomously, was achieved. The vehicle can be operated manually (via remote control) and as a proof of concept, autonomous navigation was carried out through a simple artificial vision algorithm. It was also verified the versatility of the developed solution by integrating different equipment and designed code in a simple way.
This document describes generally the project where all the different steps are identified, as well as all the work developed until the present moment. Furthermore, the guidelines to follow in order to achieve the proposed goals, is presented. Finally, navigation experimental results and the integration of different works on the platform are presented.

# Contents

# List of Figures

# List of Tables

# Acronyms

**AC** Alternating Current

**AGV** Automated Guided Vehicle

**API** Application Programming Interface

**BIOS** Basic Input/Output System

**CPU** Central Processing Unit

**DC** Direct Current

**DIN** Deutsches Institut für Normung

**DoD** Depth of discharge

**ETH** Ethernet

**FPS** Frames Per Second

**GUI** Graphical User Interface

**HSV** Hue Saturation Value

**$I^2C$** Inter-Integrated Circuit

**IDE** Integrated Development Environment

**IO** Input/Output

**IP** Internet Protocol

**ISR** Interrupt Service Routine

**LAR** Laboratory for Automation and Robotics

**LED** Light Emitting Diode

**LIDAR** Light Detection and Ranging

**NC** Normally Closed

**NO** Normally Open

**OS** Operating System

**PC** Personal Computer

**PCB** Printed Circuit Board

**PID** Proportional-Integrative-Derivative

**PWM** Pulse Width Modulation

**REST** Representational State Transfer

**ROS** Robotic Operating System

**SCL** Serial Clock

**SDA** Serial Data

**SSH** Secure Shell

**TCP** Transmission Control Protocol

**UDP** User Datagram Protocol

**USB** Universal Serial Bus

**YARP** Yet Another Robot Platform

# Chapter 1

# Introduction

## 1.1 Context

It is common knowledge that market demands are growing exponentially, either in a qualitative or quantitative level. The strong competition lead the companies into constant improvement in their quality standards, reduce their response time and make task planning. Industrial automation has made its way to become a crucial part in overcoming these needs, systematizing tasks that can be delivered by machines. One of those tasks, which is common to almost every industry and very time consuming, is the transportation of material inside the facilities.

In the past decade, the AGV's have gained expression due to the fact that they are capable of solving this issue. It is in this context that the importance of this type of vehicles arises — vehicles capable of performing tasks autonomously, allowing workers to be released from lower level tasks, maximizing the efficiency of the industrial environment.

This line of thought is being widely adopted and these emerging technologies not only help promoting industrial upgrades but can also be included on the recent concept of Industry 4.0 [1]. In an era where the cyber-physical systems allow the exchange of data between industrial processes, the ability of performing a wide variety of tasks can be enhanced with the inclusion of AGV's. So, in an intelligent factory with modular structures, besides navigating autonomously, they can be told where to navigate in the same way.

Nowadays, most of the AGV's used in an industrial scale are similar to forklifts or trailers, and they also have the advantage of being able to be integrated into their own assembly lines, by carrying material between stations without any human intervention. One of their other main characteristics is safety. In order to be used as a collaborative element, it follows a series of standards that prevent accidents, reducing significantly these incidents throughout the task performance that, according to OSHA (Occupational Safety and Health Administration), exceed the 90,000 occurrences per year in the United States of America [2].

## 1.2 Brief History

The development of AGVs began in 1954, when the engineer Arthur "MAC" Barret invented the first autonomous guided vehicle in his company, Barret Electronics. This vehicle was given the name of Guide-O-Matic (Figure 1.1) and was developed to be used in a warehouse. It consisted of an electrical vehicle, similar to a trailer, but with inferior dimensions, that could navigate by using wire trails on the floor. In order for them to move autonomously, underneath them, they contained magnetic field sensors that searched for a induced field,

which crossed the trail's wires. For it to stop in each station, there were magnetic codes that the vehicle was able to process [3].


Figure 1.1: First AGV, developed by Arthur Barret [4].

## 1.3 Motivation and Objectives

"Every major corporation in the world you can think of already uses AGVs" (Scott Kwilinski, director of Solutions Engineering at Egemin Automation Inc [5]) and the projections point out that this type of equipment will be integrated in a more easy and versatile way for the industrial sector. It has a wide range of applications and allows the workers to focus on high level tasks. So, the development of a platform from this nature meets the actual and future market demands.

Robuter is a robust platform with a solid metallic structure prepared to transport up to 120 kg that was stalled in the laboratory. This happened because some of its hardware was not operational and its programming method was archaic and not practical. Besides that, it was almost impossible to expand due to it being a closed and proprietary solution.

The main focus will be the development of a new control system to apply a perception unit (camera and possibly other sensor and devices). On the hardware level this has several major tasks, but generically, it is necessary to design and mount all the electrical installation and improve the necessary mechanical parts. On the software level, a versatile and open architecture will be implemented, such as ROS, that will allow the integration of different control and data acquisition units.

From another point of view, there is also an academic motivation. By accomplishing this project, it will be possible to convert an obsolete and unused equipment to a modern and didactic platform. It is still important to consider that it's a significant contribution to the Laboratory of Automation and Robotics (LAR), otherwise the institution would have to invest a considerable amount of money to acquire an equipment of this nature.

In the end, it is expected to endow the laboratory with a modular equipment that can be expanded and improved to future research and development. A particular application to this is another project already being developed in parallel and consists of the integration of the Fanuc LR Mate 200iD on the top of the platform. So, the big picture here is to have an AGV with a robotic manipulator.

## 1.4 Robot Software Frameworks

There are several open-source tools to develop robotic applications. Although this list does not contain all of them, the main concern was to introduce those which are up to date, i.e., the ones that still have support and recent released versions.

### 1.4.1 YARP

YARP stands for Yet Another Robot Platform. It is a free and open-source set of libraries, protocols and tools to keep modules and devices decoupled [6]. It supports building a robot control system as a collection of programs communicating in peer-to-peer way (TCP, UDP, multicast, among others) and can even be implemented without having to use YARP codebase. It was written to be OS neutral (Linux, Windows and macOS have been used [6] alongside with YARP) and for that reason it can easily stream data across different types of threads.

In YARP, the communications are ensured by a port network. A port is an object that can read and write values to peer objects spread throughout the network [7]. The objects that listen to this kind of data are called observers. As it is visible on Figure 1.2, the streamed information can be a camera image or motor commands to implement a visual tracking application. The camera captures and transmits the image to the **"/viewer1"** and **"/tracker/image"** ports. The tracker then receives the image and processes it, transmitting it to another viewer (**"/viewer2"**) and outputting position values that will further be sent to the **"/motor/position"** port. In the YARP topology, every port belongs to a process and each connection can be established using different protocols [8].

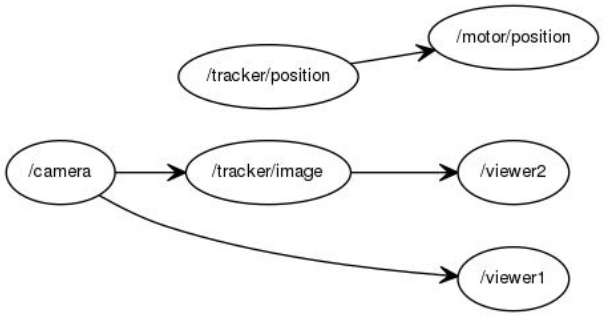



Figure 1.2: Example of a network of ports in YARP. Adapted from [8].

### 1.4.2 Orocos

Orocos (Open Robot Control Software) is an advanced framework, developed to deal with real-time constrains. The Orocos project supports four C++ libraries:

- Real-Time Toolkit ;
- Kinematics and Dynamics Library;
- Bayesian Filtering Library;
- Orocos Component Library.

These libraries are combined to develop the robot's application. In Orocos, the applications can be constructed using predefined components, by using the Orocos **"Control Component"** [9].

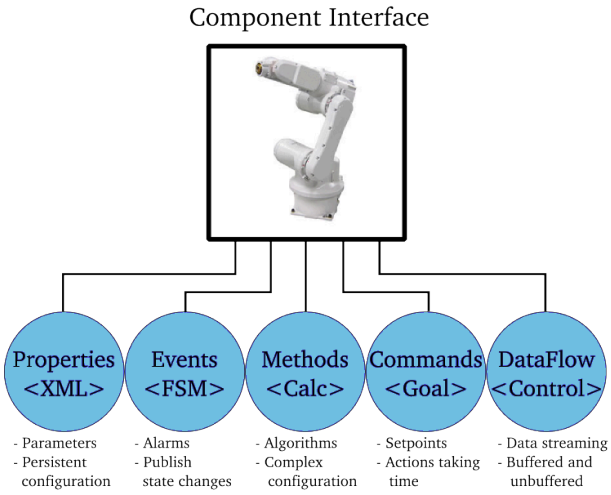



Figure 1.3: Example of the ways of interfacing an Orocos component. Adapted from [9].

In the given example (Figure 1.3), five ways of how a component can be interfaced are shown. Although a single component can control a whole machine, a network of components that communicate between them can also be implemented — using CORBA or MQueue protocols.

### 1.4.3 Microsoft RDS

Microsoft®Robotics Developer Studio, currently in version 4, is a Windows-based environment to create robotics applications for a wide range of hardware platforms [10] and it provides the following tools:

- CCR – Concurrency and Coordination Runtime;
- DSS – Decentralized Software Services;
- VPL – Visual Programming Language;
- VSE – Visual Simulation Environment.

Regarding CCR, it allows to write programs to handle with asynchronous sensor inputs and actuate outputs. The real-time feature is empowered by the DSS service, which enables monitoring the robot/system using a Web browser or Windows-based application. The reuse of code is also provided, since the programming can be made in modular services

The programs can run natively on a PC-based robot running Microsoft Windows OS or remotely scenarios can be implemented with serial port, Bluetooth, Wi-Fi or RF modem. To program the robots, two languages can be used — C#, in Microsoft Visual Studio environment, or the Visual Programming Language.

### 1.4.4 ROS

ROS [11], Robotic Operating System, is a software framework that provides a wide set of tools and libraries for robotic applications. It is an open-source, meta-operating system that promotes the sharing and reuse of code. This last characteristic is empowered by the capacity of executing programs independently (even in different programming languages) and integrating them in the system. This is possible due to the fact that it can run multiple processes at the same time (even in different machines), passing messages among them. Currently, ROS only runs on Unix-based platforms and it provides the services expected

from an operating system, including hardware abstraction.
In ROS, the processes are named "nodes". The nodes can communicate with each other with publish/subscribe semantics to a given topic (channels used for communication). Figure 1.4 shows an example of this topology.

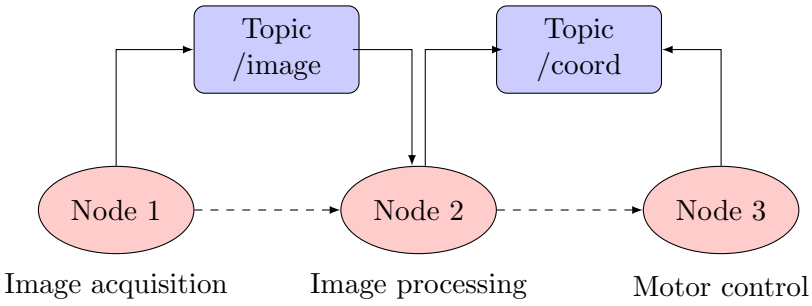



Figure 1.4: ROS topology example with three nodes exchanging messages on two topics.

In the diagram from Figure 1.4, the "**Node 1**", which is responsible for the image acquisition, publishes its data (in this case the acquired image) on the topic "**/image**". The "**Node 2**" will then subscribe to that same topic and collect that information so it can be processed and published (in this given example, the coordinates could be from a determined object so that the motor control system actuates accordingly and navigate to it).
Despite of not being represented in the diagram, each topic can be accessed by different nodes (this is valid both for publish and subscribe actions).

This framework is used at LAR and it is the most suitable one to use in order to carry on and integrate the developed work so far.

## 1.5 Navigation techniques

Navigation, and particularly autonomous navigation, has been widely focused by the mobile robotics area. As Leonard and Durrant-Whyte defined in the early 90's, it is a field of study that seeks to find solution to three fundamental questions: "Where am I?", "Where am I going?" and "How do I get there?" [12].

Regarding the first question, which intends to answer to the robot's position in its environment, it can be said that the existing solutions have difficulty to respond with a single and elegant solution. This means there are plenty of partial and/or combined solutions that, although they cannot be categorized, are usually divided in two groups: relative and absolute positioning. Due to the lack of an unique method, the AGV's commonly integrate technologies from both groups [13].

Nowadays the used techniques can go from solutions that imply restructuring the layout of the facility to non invasive methods.

**Absolute positioning**

- Active beacons – computation of the absolute position from measuring the direction of incidence of at least three actively transmitted beacons. The transmitters generally use light or radio frequencies and must be located at known sites in the environment so the position can be determined (triangulation) [14].
- Passive references – these references can be natural (inherent to the navigation environment) or artificial (placed in known sites and detectable). They are generally detected with perception techniques (for example artificial vision) and, similarly to the active

beacons, it is necessary to have at least three identifiers on the perception field so the position can be calculated.
- Model matching - this method compares the current information (acquired by the sensors) to an existing model or map of the environment. The model/map is created dynamically and then, using the existing one, the vehicle's absolute location can be estimated.

As major advantages, it is a non-invasive method that can present satisfactory results in terms of accuracy (when correctly referenced). The main disadvantages, particularly in the techniques that use perception, are the intensive data computation, difficulty of implementation and the robustness is heavily conditioned by the navigation environment.

**Relative positioning**

- Dead-Reckoning – use of encoders or inertial sensors (for example gyroscopes and accelerometers) to determine the direction and traveled distance from a starting reference.

One of the main disadvantages is the error accumulation due to its integrative method (the values of direction and distance are added to a given initial point). As advantages it is important to mention that these systems work regardless of the visibility circumstances and can even operate in adverse conditions (for example high altitudes).

**Guiding techniques**

- Wire – slot on the ground that contains an electric conductor that can be easily detected by a sensor. The wire is placed throughout the route.
- Magnetic or colored tape – detection of magnetic tape through sensors or colored (including black and white) with artificial vision or color sensors. As the wire technique, the tape is placed throughout the desired path.

The major advantages of wires and tapes are the navigation robustness, the easy implementation from a software point of view and great accuracy on the positioning (it can be implemented in a way where the robot position is always known and calibrated). As disadvantages it can be pointed out the need to restructure the facility's layout and, in the case of wire, the opening of gutters, which makes it a very limited and definitive approach.

## 1.6 Related Work

Regarding the recent related work on Robuter's retrofitting, no recent documentation has been found. However, in 2005, in the "Hands-On" Laboratory of the IPP-Hurray! Research Group, at the School of Engineering of the Polytechnic Institute of Porto the Robuter was subject to an hardware and software upgrade. After this intervention (on a similar Robuter, see Figure 1.5), the Robuter was equipped with the following items [15]:

- 2 x 300W DC motors, with gear boxes and brakes;
- 2 x incremental encoders;
- 2 x servo-amplifiers;
- 2 x axes interface between motors and boards;
- 4 x short range ultrasonic sonars (MIC130);
- 2 x medium range ultrasonic sonars (MIC340);
- 1 x RSMPC555 control board dedicated to multi-axis control (up to 4 axis);

- 1 x Pentium4 class embedded computer;
- 1 x Joystick 2-axis for manual operation with security button.

Figure 1.5: Previous version of Robuter used on "Hands on Laboratory", in Polytechnic Institute of Porto. Adapted from [16].

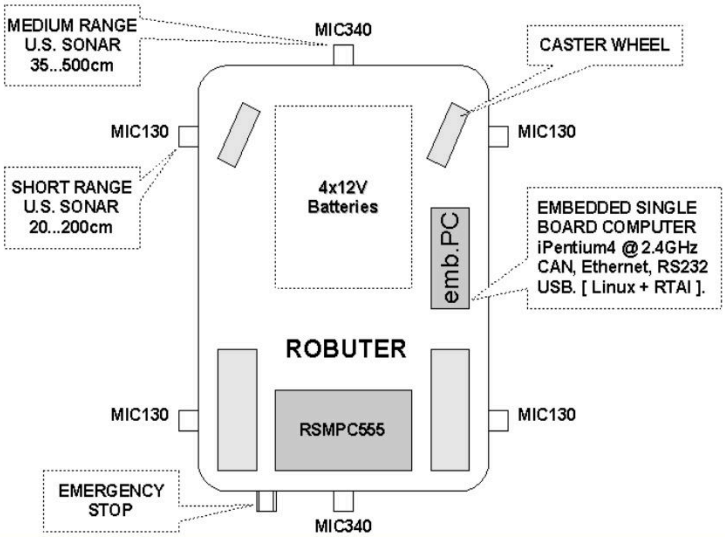



Figure 1.6: Location of retrofitted Robuter's main components in the "Hands on Laboratory" project [15].

The location of the components can be observed in Figure 1.6. Although it is not represented in the figure, some of the components are supplied with 24V and hence have dedicated DC/DC converters to adjust the power supply to appropriate levels.

About the software, the on-board CPU was running the Linux Operating System with Real-Time Application Interface (RTAI), which allows to write applications with strict timing constraints [17]. Even though it's a main unit, the CPU was used to load the executable control application into the RSMPC555 board and communicate with it (via RS232).

Another important work was carried out in 1995, at the University of Porto. The dissertation focus was to obtain a control system based on the platform's dynamics (forces and moments) by observing the controllers limitations based on kinematic models [18]. Despite the importance of this work to understand the platform's constraints, no physical intervention was performed.

Throughout the years this platform has had a meaningful role on the evolution of mobile robotics [19]. Some of the following listed themes are an example of that:

- Trajectory Tracking Strategies in Service Robotics Applications [20];
- An Integrated Learning, Planning and Reacting Algorithm Applied to a Real Mobile Robot [21];
- Indoor Robot Navigation with Single Camera Vision [22];
- Multi-agent Control Approach for Autonomous Mobile Manipulators: Simulation Results on RobuTER/ULM [23];
- Autonomous Navigation and Multi-Sensorial Real-Time Localization for a Mobile Robot [24].

## 1.7 Dissertation Structure

This dissertation is divided in seven chapters, including the present one.

Chapter 2 presents the platform in its original state and shows the necessary steps to discover the platform and all its elements. It exhibits pictures to illustrate the process and it is an important chapter to have an overview of the existing system.

Chapter 3 starts with a generic description of the proposed architecture and details the used hardware to achieve it.

Chapter 4 reports the implementation of the proposed solution where the reader can understand where, how and why each equipment was used. Besides that, it is explained the developed software to communicate between the existing subsystems and the reading/actuation of the main inputs/outputs of the system.

On chapter 5, after the retrofitting is complete, a simple vision application to detect and navigate through a line is made. Besides that, it is implemented a manual control with the Xbox controller.

Chapter 6 refers to the tests and results performed on the platform, such as the capacity of navigating through a straight line or a circle with the aid of vision tools to measure it. Also, the integration of a robotic manipulator and its software (work developed by Vitor Silva [25]) is analyzed.

Finally, chapter 7, presents the conclusions and points out the future work (where some enhancements are suggested).

A list of appendix is also presented in order to complement the documentation with relevant, but not crucial, information, allowing a more detailed knowledge on specific matters.

# Chapter 2

# The Robuter II

The purpose of this chapter is to present the Robuter II platform in its original state and report the major steps that took place in the platform recognition.

## 2.1 Original Features

The Robuter concept was created by a french company, Robosoft, and consists of a mobile robotic platform (Figure 2.1) with on-board CPU dedicated to research and development. In the 80's this was an innovating concept that allowed the researchers a great flexibility because, not only is it a programmable platform (in ALBATROS environment), but it also allowed the integration of additional hardware [19].


Figure 2.1: Robuter II original

The main characteristics are presented in Table 2.1. Note that this table could only be completed after the platform disassembly described in Section 2.2.

The ALBATROS is "A flexible multi-tasking and realtime network-operating-system-kernel" [27]. It was specifically designed with the purpose of control multi-axis and multi-sensorial systems.

## 2.2 Platform Disassembly

Due to the lack of technical documentation, the platform was fully disassembled. This step was crucial to understand its architecture and identify its components and physical connections at the hardware level.

The analysis started with a generic approach, removing all the protections to observe the elements that composed the platform. After their removal, the main case was detached

Table 2.1: Robuter II original characteristics.[26]

| | |
|---|---|
| Dimensions | [102.5 x 68 x 44] cm |
| Weight | 150 kg |
| Payload | 120 kg |
| Speed | up to 1 m/s |
| 4 Lead Batteries | 12V 60Ah each |
| 2 Motors | Permanent magnet DC 300W 48V |
| Control Type | Differential |
| Positioning | Optical encoders |
| Range sensing | Network of ultrasonic sensors |
| CPU | VME bus with 5 slots, Motorola 68020, 16 MHz |
| Operating System | ALBATROS |
| Programming | Software package and C application programs |

(Figure 2.2). This case had several assembled units in an Eurocard type mount. Different types of modules were identified:

- Two DC/DC converters (5V and 12V) — 1;
- Two DC drives — 2;
- One Security module — 3;
- One main CPU — 4.

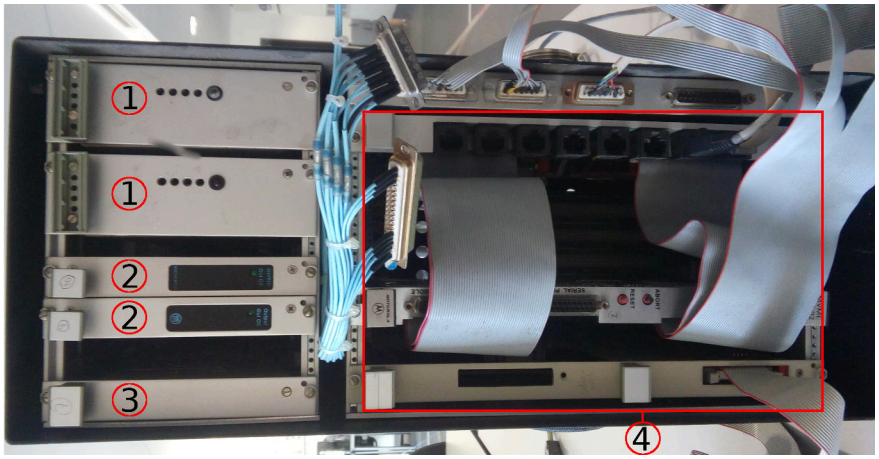



Figure 2.2: Main case containing various modules.

In parallel, both wheels were detached in order to observe their characteristics and test the motors. Each wheel setup is composed by an electromagnetic brake (coupled to the motor) and a two channel incremental encoder (Figure 2.3). The motor also has a gear box, but the ratio was not determined.

During this stage, particularly to test the motors, a main circular connector was detected (Figure 2.4 illustrates the connector type and Figure 4.1 its wiring). It establishes connection between the enclosed wheel setup (motors, encoders and electromagnetic brakes) and it was mapped with a multimeter to match the pins to the actual connections (the pinout is shown further in Figure 4.12). This step was crucial because without it, it would not have been possible to know that the motor had coupled brakes, hence they could not be actuated. At this stage, a DC supply was used to test the motors because the DC Drives were not

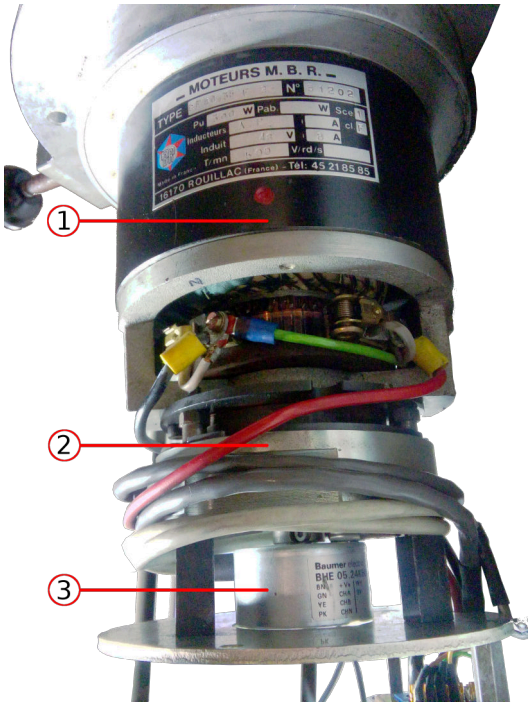



Figure 2.3: Wheel with coupled elements. 1 - Motor; 2 - Electromagnetic brake; 3 - Encoder.

operational yet. Additional information is presented in Appendix A in the Figures A.1 , A.2 and A.3.

Figure 2.4: Circular connector in perspective and front view.

After the previous listed steps, an examination was carried out to determine which elements could be used and which should be removed.

Since one of the main goals of the project is to implement an open architecture, the internal CPU components were not used and therefore removed.

Also, the network of ultrasonic sensors (24 in total) was detached due the lack of information and, in consequence, the time that was necessary to invest was not worthwhile given the complexity and functionality of the solution (there are better approaches these days, e.g. LIDAR). The four batteries were also tested, each one charged individually, and none of them was with an acceptable voltage level.

At last, the unnecessary wires were removed after the identification of the relevant ones. This step is registered in Figure 2.5, where it is possible to observe the amount of material taken from the original platform.

In the end of this process, the following was used from the original Robuter (resulting in the setup from Figure 2.6):

- Chassis
- Full wheels setup (Motors + Brakes + Encoders)
- DC drives
- DC/DC converters (5V and 12V)

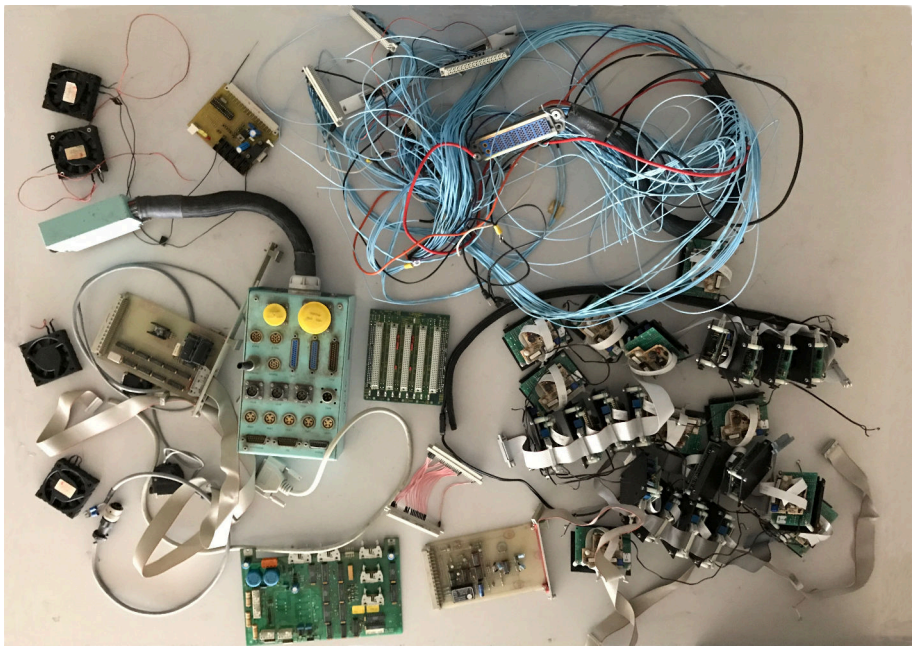


Figure 2.5: Robuter's parts after disassembly.

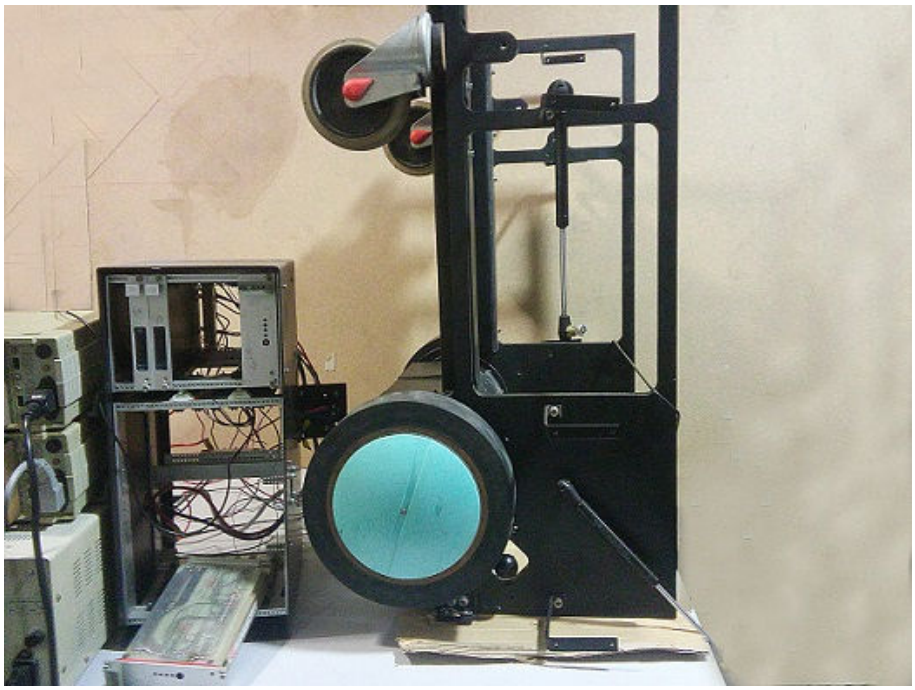


Figure 2.6: First setup used, main case aspect after disassembly.

## 2.3 Platform Kinematics

To perform the platform's kinematic analysis, first, it is important to understand the robot's locomotion. The movement is based on two separate driving wheels placed in one end of the robot. It can thus vary the direction by changing the rate of rotation of each wheel, eliminating the necessity of a steering system. The platform is balanced by two caster wheels, fixed on the opposite side.

From Figure 2.7, it is visible that the direction is determined by the relation between each motor's velocity, $v_l$ and $v_r$. However, in a robot's locomotion system, its position ($x$, $y$) and direction ($\theta$) are the important variables, so each wheel velocity must be related with them (expression (2.1)).

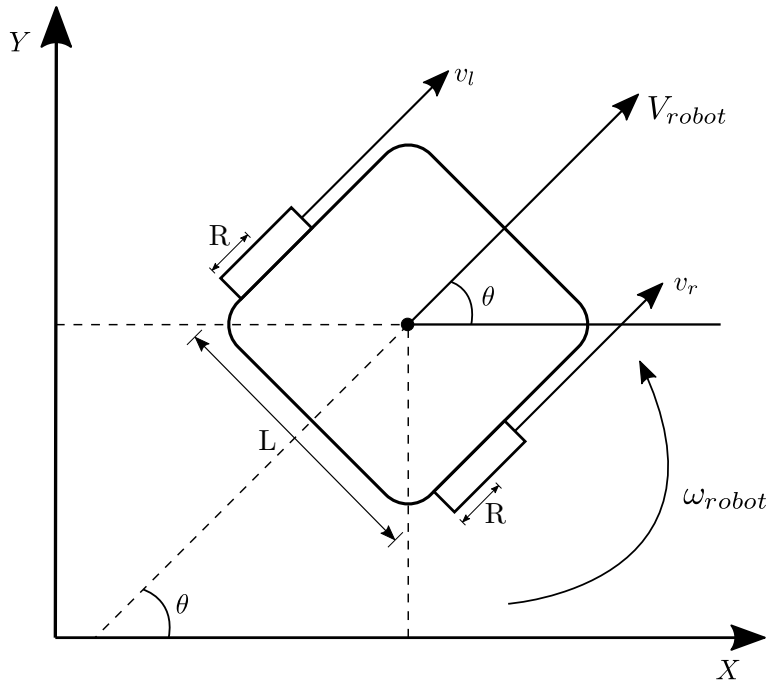



Figure 2.7: Differential drive kinematics. Image redesigned from [28].

$$\begin{cases} \dot{x} = \frac{R}{2}(v_r + v_l)cos\theta \\ \dot{y} = \frac{R}{2}(v_r + v_l)sin\theta \\ \dot{\theta} = \frac{R}{L}(v_r - v_l) \end{cases} \tag{2.1}$$

$$\begin{cases} \dot{x} = V_{robot}cos\theta \\ \dot{y} = V_{robot}sin\theta \\ \dot{\theta} = \omega_{robot} \end{cases} \tag{2.2}$$

Despite its valid relation, it is very cumbersome to think in terms of relative wheel velocities when designing a controller. For this reason, the unicycle model is used to have a more intuitive control type [29]. This is a common approach when designing controllers because it overcomes this issue by using the linear ($V_{robot}$) and angular($\omega_{robot}$) velocities (2.2).

Combining the equations (2.1) and (2.2), which is crucial because the actual inputs to the system are each wheel's velocity, the controller can be designed to accept linear and angular velocities — equations (2.3) and (2.4).

$$\begin{cases} V_{robot} = \frac{R}{2}(v_r + v_l) \\ \omega_{robot} = \frac{R}{L}(v_r - v_l) \end{cases} \tag{2.3}$$

$$\begin{cases} v_r = \frac{2V_{robot} + \omega_{robot} L}{2R} \\ v_l = \frac{2V_{robot} - \omega_{robot} L}{2R} \end{cases} \tag{2.4}$$

At this point, and given all this information, it was more clear to design a detailed solution. Chapter 3 gives an overview of the intervention plan.

# Chapter 3

# Proposed Solution and Intervention Plan

This chapter presents the proposed solution as well as an overview of how it is achieved. It starts with a generic solution for the global architecture, and afterwards the hardware will be specified individually.

## 3.1 Control Architecture Types

When considering what type of generic architecture should the system have, there are two major types that prevail: central and distributed systems (Figure 3.1).
Centralized systems use, in general, a CPU that is responsible for processing all tasks, meaning all the sensors/actuators communicate directly with the CPU through its peripherals. This requires a high processing capability in order to maintain good responses to the system. Since everything is centralized, these are usually lighter and more compact solutions than the others.
On the other hand, in a distributed architecture, the systems are divided in other subsystems, or modules. Each one of these modules is responsible for the control of a determined task and can even have its own CPU. In this topology, all the modules work individually to the overall operation of the system, either interconnected and functioning in an independent way or all connected to a central CPU (Master).

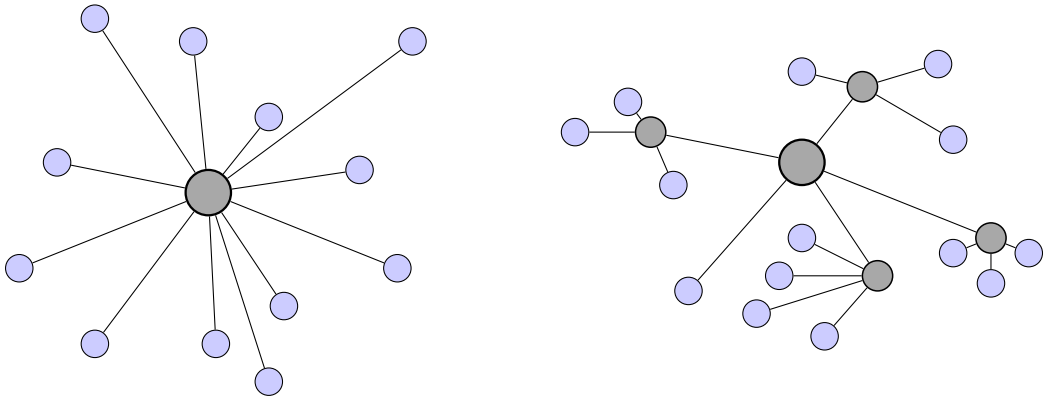


Figure 3.1: Central architecture type.

By comparing these architectures, it is easy to understand that a modular system is more

robust, allows development and expansion (particularly in the test stage) and it is more versatile than a centralized system. For these reasons, and because this is a project designed to be continuously improved, the option for a modular architecture becomes obvious.

## 3.2 System Diagram

The diagram in Figure 3.2 illustrates all the main blocks and explains how they interact among them, giving an overview of the proposed solution.

A video camera is responsible for acquiring the image that will then be received and processed by the main unit, `CPU1`. Depending on that information, `CPU2`, will generate electric signals to actuate the motors accordingly. These motors are actuated through a PWM signal on the DC drives and can feedback its velocity with encoders. This signal is processed by the `Counter` block and sent to `CPU2`, closing the loop.

To control the platform remotely, the operator will use a CPU, connected to the robot's network, that will grant access to the `CPU1` functionalities. This will allow to monitor the platform and perform actions/routines.

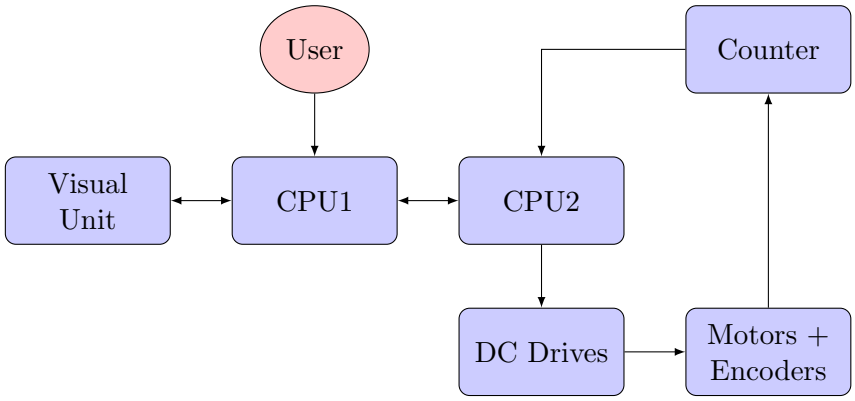



Figure 3.2: Generic system diagram for the platform's global architecture.

## 3.3 Communication Between Subsystems

To interact between subsystems, a main protocols is proposed: TCP/IP. Besides that, other interfaces (for example $I^2C$), which will be explained further in context on Section 4.5.2, are used.

The usage of TPC/IP protocol fulfills the concept of modularity and easy integration of subsystems. Another determinant aspect is the available technology on the market (both in social and industrial means), which is predisposed for this type of communication [30]. There are several units of microcontrollers and sensors which are capable of communicating through this protocol. Nevertheless, it also allows configurations where all the devices can communicate among themselves.

To transfer messages between the programmed processes (in `CPU1`), passing messages, it will be used ROS (see 1.4.4).

## 3.4 Hardware

This section contains a list and presentation of the used material on the platform, which will be further explained in context (Chapter 4). Some of the equipment was selected and

ordered, while the remaining was used from the previous robot or was already available in the laboratory. This is important to mention in order to justify some decisions of the used hardware.
Table 3.5 makes the correspondence between the CPU's blocks from Figure 3.2 with the presented hardware.

### 3.4.1 Motor Related Parts

The full wheel setup is illustrated in Figure 2.3. As explained in Section 2.3, the system has two driving wheels (which can be controlled individually) and two caster wheels.

**Full Wheel Setup**
**Motors:** Permanent magnet DC 300W 48V each (motor plate can be observed in Appendix A, Figure A.1).
**Encoders**: Baumer BHE 05.25K500 three channel encoders with 8024 pulses per wheel revolution (each pulse $\approx 0.045°$). Despite that, only two channels are being used (CHA and CHB) in the original setup. Power supplied with 5V.
**Brakes:** KEB electromagnetic 48V brakes. The brakes are disabled (loose wheels) with a 48V signal. Although each wheel has an individual brake, in the original connections the supply is common, so they are actuated together. Both brakes have approximately a power of 24W.
**Wheels**: 2 motored wheels with 75cm of perimeter and two caster wheels with 40cm of perimeter.

**DC Drives**
Model 306 from Copley Controls [31]. Input voltage from 16 to 80VDC. Maximum output current 10A. Controlled with an external PWM signal with minimum switching frequency of 22 kHz (avoids switching audible noise). These drives had to be repaired (as explained in Section 4.2) and were used in order to save costs and make use of its robustness.

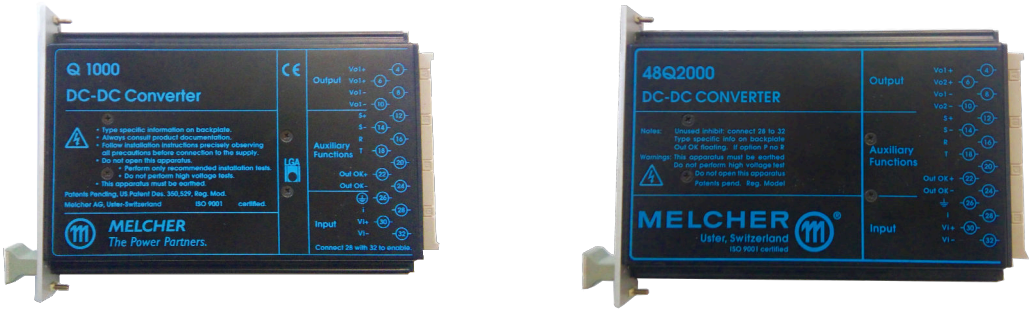

Figure 3.3: Copley 306 DC drives.

### 3.4.2 Power

**5V and 12V DC/DC Converters**
These two converters (Figure 3.4) were still operational and had useful voltage levels, therefore they were kept installed on the platform. Table 3.1 features the DC/DC converters' characteristics.

Table 3.1: Characteristics of the two original DC/DC converters [32].

| **Characteristic** | **Q1000** | **24Q2000** |
|---|---|---|
| Input Voltage (V) | 38–75 | 38–75 |
| Fixed Outputs (V) | 5.1; 12; 15 and 24 | 5.1; 12; 15 and 24 |
| Variable Output (V) | 3.3–48 | 3.3–48 |
| Power Rating (W) | 132 | 106 |
| Efficiency (%) | 90 | 86 |




(a) (b)

Figure 3.4: DC/DC converters Q1000 (a) and 24Q2000 (b).

**24V DC/DC Converter** [33]
Switching DC/DC Converter Mean Well SD-100C-24 with input voltage range from 36V to 72V (Figure 3.5). Single 24V output with adjust range from 22V to 30V and maximum power rating of 100W.

This converter was available in the laboratory and was necessary to supply `CPU1`. It was decided to use a single converter for that effect because of the computer specifications (see Sub-section 3.4.3).


Figure 3.5: DC/DC converter SD-100C-24. Adapted from [33]

### Batteries

There are numerous battery types on the market and choosing the right one for the right job is a demanding process. For this particular application, there were several aspects took into consideration such as size, weight, capacity, discharge rate and as in every project — price.

In a simplistic way, as it is possible to observe in Table 3.2, the lithium batteries have more energy density and charging cycles, therefore they are better and should be selected despite of their price (which does not include the more expensive and sophisticated way of its charging technology).

On the other hand, since this project is about retrofitting, the previous equipment should be analyzed. Figure 3.6 shows one of the two old Robuter's battery cases. These cases have an open space measuring [210x340x140]mm and a set of wheels underneath them, which makes manipulating the batteries easier.

Other aspect that is crucial to mention, despite not being the scope of this project, is the future application of the platform. As is mentioned in Section 1.3, this platform will have future applications, one of them is a robotic manipulator on the top of it, which brings new variables to the battery selection. It is important to lower the platform's center of mass and counterbalance the manipulator's movement (Fanuc LR Mate 200iD). Therefore, the weight is a relevant factor to grant safety and enable the maximization of its operation (for example in maximum payload scenarios, 6kg).

Table 3.2: Battery types characteristics. Adapted from [34] and last updated in 2016-04-11

| | **Lead** | **Li-ion** | **Ni-MH** |
|---|---|---|---|
| Energy Density (Wh/kg) | 30-50 | 150-250 | 60-120 |
| Cycle Life (80% DoD) | 200-300 | 100-500 | 300-500 |
| Charge Time (h) | 8-16 | 2-4 | 2-4 |
| Cell Voltage (V) | 2 | 3.6 | 1.2 |
| Cost (€/Wh) | Low | High | Moderate |

In Table 3.2 the Lithium polymer technology is not mentioned despite being widely implemented on the market. Accordingly to [35], the batteries sold in the market, commonly mentioned as polymer, are essentially the same as lithium-ion. Li-polymer offers slightly higher specific energy and can be made thinner than conventional Li-ion.

By the above reasons, and taking into consideration that in standalone applications a lead battery can have better payback [36], it was decided to order batteries with lead technology and with the same dimensions as the previous ones.

There were selected 4 batteries from Ultracell. The model is UCG75-12 and has a voltage of 12V and 75Ah of capacity [37]. Table 3.3 compares the old and new batteries where more detailed information is presented.

Table 3.3: Old vs. new batteries arrangement.

| | **Old** | **New** |
|---|---|---|
| Weight (kg) | 24 | 22.5 |
| Size (mm) | 210x168x208 | 210x168x208 |
| Type | Lead - AGM | Lead - GEL |
| Voltage (V) | 48 | 48 |
| Total Capacity (kWh) | 2.88 | 3.6 |

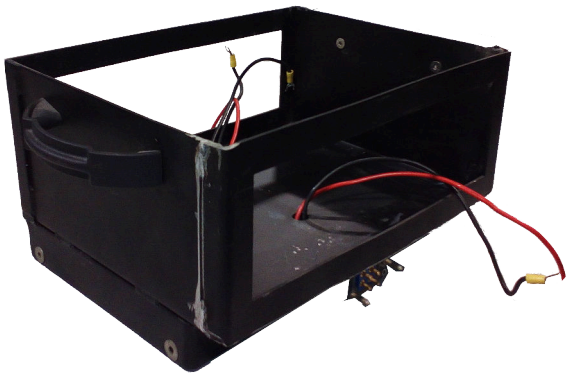


Figure 3.6: Robuter's battery case.

### 3.4.3 Processing Units

**CPU1 — Mini PC Cubi 2** [38]

Selecting a computer within the wide range of offers is challenging. To focus and limit the choices some specifications were pointed out, despite not being necessary for the current project. It is important that the CPU can cope with future applications, such as work with Point Cloud Library. For these reasons, the requirements for the search were:

- Good computational capacity — i5 processor and 8GB RAM;
- Vibration/impact tolerance — SSD memory;
- Compact size to fit in the main case — smaller than 150mm X 120mm X 50mm;
- Available IO's — 1 x Ethernet, 4 x USB, 1 x HDMI);
- Voltage supply level — 24V or lower;
- Affordable.

Once defined the specifications, various quotations were asked and the equipment that best fulfill them was the MSI Cubi 2 012BEU i5-7200U. This computer has built-in processor and both the SSD and RAM were ordered separately. The IO's are detailed in Figure 3.7 and the detailed specifications are presented in Table 3.4.

Table 3.4: `CPU1` specifications.

| **Spec** | **Detail** |
|---|---|
| Chipset | Intel® H110 |
| Processor | Intel® Core™i5-7200U Dual-Core, 2.5 GHz with Turbo up to 3.1 GHz, 3 MB Cache |
| Storage | SSD 2.5" Kingston V300 120 GB MLC SATA |
| Memory | RAM SO-DIMM Crucial Ballistic 8GB DDR4-2400MHz |
| Graphics | Intel® HD Graphics 620 |
| Network | Intel® Dual Band Wireless-AC 3168 (M.2 2230)<br>Realtek RTL8111H GigaLan 10/100/1000 |
| Power | 19V input, 65W |
| Dimensions | 115 X 111 X 35 [mm] |

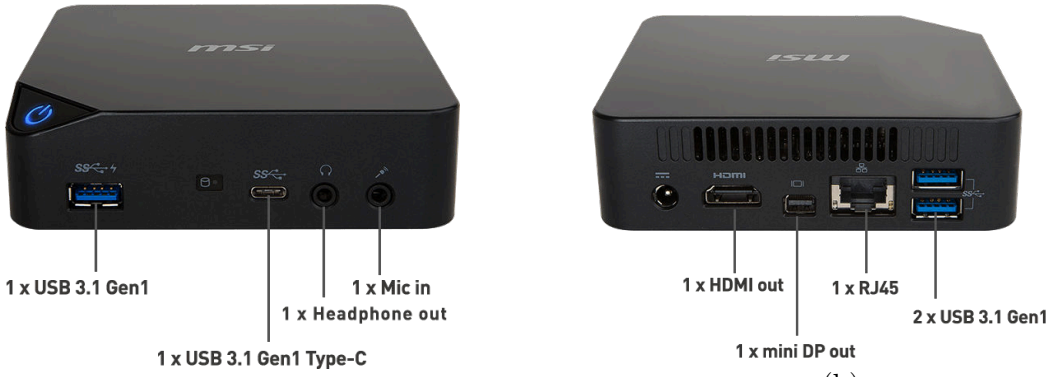



Figure 3.7: Mini PC MSI Cubi 2, front view (a) and rear view (b). Adapted from [38].

**CPU2 — Arduino Leonardo ETH** [39]
Microcontroller: Chip ATMega32U4 integrated in the development board Arduino Leonardo ETH with a 16MHz clock.
IO's: 20 IO pins of which 7 can be used as PWM channels and 12 as analog inputs. All the 20 pins can be used as digital input/output.
Power: Input voltage from 7-12V and power consumption until 900 mW.
Interfaces: RS232, SPI, $I^2C$, Ethernet.
Dimensions: 53.34 x 68.58 mm (WxD).
This board was available at the laboratory and has the necessary specifications to be used in this project: Ethernet communication, enough IO's (including PWM outputs) and clock speed.

Figure 3.8: Arduino Leonardo Ethernet. Adapted from [39].

**Counter — CPU3 — Arduino Micro** [40]
Microcontroller and IO pins are the same as described for Arduino Leonardo ETH.
Power: Input voltage from 7-12V and power consumption until 360 mW.
Interfaces: RS232, SPI, $I^2C$.
Dimensions: 48 x 18 mm (WxD).
This board had three major specifications: two or more interrupt pins, enough clock speed to read the encoders data and capacity of communication with the Arduino Leonardo ETH, `CPU2`, which can be achieved by `CPU3`, as will be explained in Sub-section 4.5.2.

Table 3.5: Identification of the blocks from Figure 3.2

| Name | Unit |
|---|---|
| CPU1 | Mini-PC MSI Cubi 2 i5-7200U |
| CPU2 | Arduino Leonardo ETH |
| CPU3 — Counter | Arduino Micro |

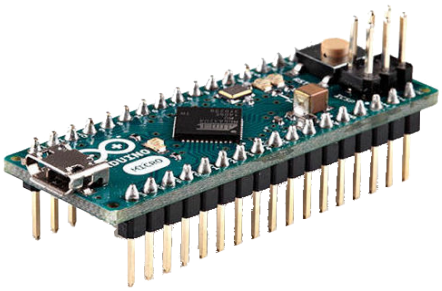


Figure 3.9: Arduino Micro. Adapted from [40].

### 3.4.4 Visual Units

**Camera** [41]

The chosen camera for this project was the Point Grey Flea3 GigE, model GE-28S4C-C (available from previous projects in the laboratory). It is a color camera with 2.8MP and it is equipped with a Sony sensor: ICX687 CCD, 1/1.8",9.69$\mu$m. Resolution 1928 x 1448 pixels at 15 FPS.
Power supplied with 12VDC.


Figure 3.10: Flea3 GigE, model GE-28S4C-C. Adapted from [41].

**Pan and Tilt Unit** [42]

Servo controlled pan and tilt unit, model PTU-D46-17 by FLIR (Figure 3.11). Its pan and tilt axis have 180°/111° positioning range respectively, and both support speeds up to 300°/s. The system allows control in position, speed and acceleration using RS232 communications. The resolution is 0.013°.
Power: Input voltage from 12 to 30VDC. Maximum power consumption 13W (full-power mode) and 6W on low-power mode.
This unit was available at the laboratory and will not have any particular function in this

project. Despite that, it will be installed to receive commands and move, being prepared to maximize the perception range for future applications.

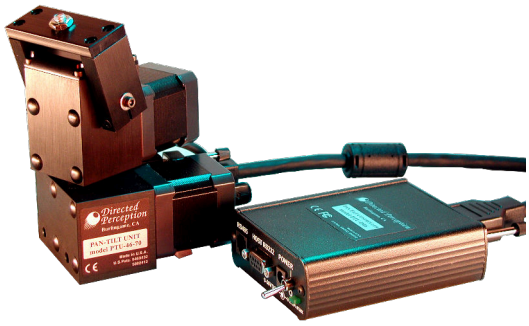



Figure 3.11: Pan-Tilt Unit FLIR PTU-D46-17. Adapted from [42].

### 3.4.5 Network

**Router** [43]
Router Asus AC750 RT-AC51U with 802.11ac standard (Figure 3.12). It is a wireless dual-band device (2.4 GHz and 5 GHz) with transfer rate up to 733 Mbps. It has wireless coverage improved with 5dBi antennas.
Ports: 1 x RJ45, 4 x RJ45 and 1 x USB 2.0.
Power: 12V supply and power until 12W.
Dimensions: 189 x 129 x 36 mm (WxDxH) and 270g of weight.

Figure 3.12: Router Asus AC750 RT-AC51U. Adapted from [43].

The router allows the creation of a wi-fi network to remotely access/monitor the platform. Its transfer rates are more than enough to transfer the necessary data but with the available budget the concern was to acquire a modern equipment.

Regarding the number of RJ45 ports, these are enough for the given application but it was decided to invest in an additional network switch. There were two main reasons for this network arrangement — the platform must be prepared for the future with additional ports and the price of an 8 port router was much higher than both (router and switch combined). The result was to spend less money and have 12 RJ45 ports, which is in line with a modular

system that grants the easy integration of new hardware.

**Switch** [44]
Network switch D-Link DGS-108 Gigabit (Figure 3.13). Contains 8 ports GigE with transfer rates up to 1000 Mbps. Input voltage 5V, maximum DC power 3W and 0.6W in stand-by mode.
Dimensions: 162 x 102 x 28 mm (WxDxH).


Figure 3.13: Switch D-Link DGS-108 Gigabit. Adapted from [44].

### 3.4.6 User

**Remote Controller** [45]
To control the robot manually the Xbox 360 Wireless Controller is used. This joystick uses the Microsoft Xbox 360 Wireless Receiver(Figure 3.14) to communicate in the 2.4 GHz band with ranges up to nine meters.
Power: Two AA batteries (the ones used are rechargeable)
The selection of this controller was natural because of its availability in the laboratory, the existing ROS libraries to interface with it and because it has several analog buttons (which allow a more accurate and versatile control experience to the user).

Figure 3.14: X-Box remote controller. Adapted from [45].

### 3.4.7 Switches

**Arduino 2 Relay Module** [46]
Module with two relays to use with Arduino (Figure 3.15) for brake control and enabling the 22V DC/DC converter. This module already has the relay's driver circuit built-in. Each relay is controlled with a 5V signal and can support up to 10A, 250VAC, 110VDC.
This solution is seen as an advantage since it is capable of actuating the brakes with a galvanically isolated circuit and it is a cheap circuit.

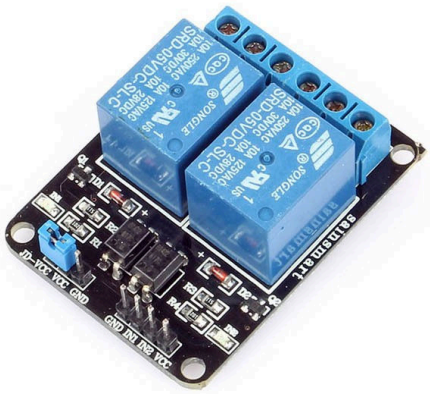


Figure 3.15: Arduino 2 Relay Module. Adapted from [46].

**Circuit Breaker** [47]
Since the platform already had one circuit breaker and the modifications did not require a new one, it was maintained. C32H-DC, Figure 3.16, is a circuit breaker that will protect the platform in case of short-circuit or overload current. It also allows cutoff the power manually. Power ratings: 250V DC, 1 to 40A at 40°C and break capacity up to 20kA. Type C tripping curve.


Figure 3.16: Circuit breaker C32H-DC.

# Chapter 4

# Solution Development

This chapter details the platform's retrofitting. In order to do this, the steps to achieve the proposed solution will be presented so that all the mechanical and electric interventions can be documented. Also, the developed code will be explained, mainly the code implemented for communicating between subsystems and motor control.

## 4.1 Electric and Mechanical Intervention

Although the core of this intervention is mostly electronic, several mechanical adjustments also took place. This section describes both of these procedures together because sometimes they are complementary.

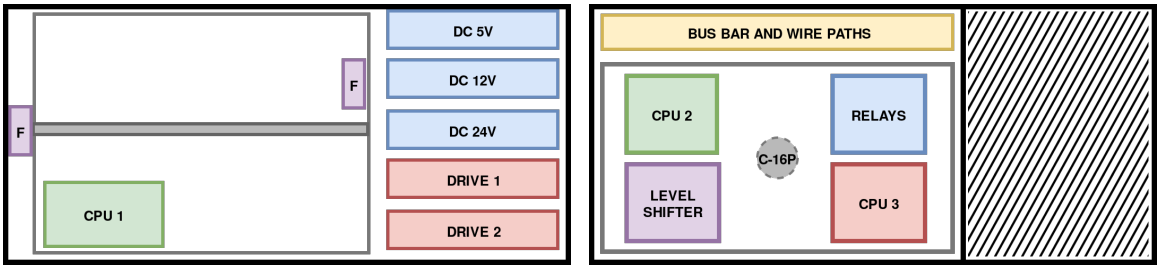
### 4.1.1 Components Layout

To decide which side is the front and the rear of the robot the possible future applications had to be studied. One of them is the addition of a robotic arm to manipulate objects up to 6kg. So, as explained in Sub-section 3.4.2, the counter-balance is an important factor and therefore the weight distribution must be analyzed. It is visible, from Figure 4.2, that the top part has more weight than the other half due to the batteries.

Based on the platform kinematics (Section 2.3) and the presented facts, it was decided to make the front side the nearest to the motors, leaving the manipulator aligned vertically with them. Also, to increase the manipulator's reachabily and to have a good field of vision it is convenient to have a wide area to operate.

Regarding the main case, the layout idea was kept, leaving the electronics and processing units on one side, and the power (drives and converters) on the other. Figure 4.1 displays the items location, where in (b) the dashed circle (C-16P) is located on the bottom of the case and the four blocks are fixed at a different height. The two metal structures with rails (Figure H.12) have revealed being very useful for fixing purposes.

Despite the main case having plenty of space left, it was decided to put the network equipment outside it, mainly for two reasons. One of them is to leave space for future processing units or electronics inside the case. The other is to make the Ethernet connections accessible from the outside, allowing to plug devices externally. The chosen place is represented in Figure 4.2 by number 2.




(a) (b)

Figure 4.1: Main case front view (a) and top view (b).

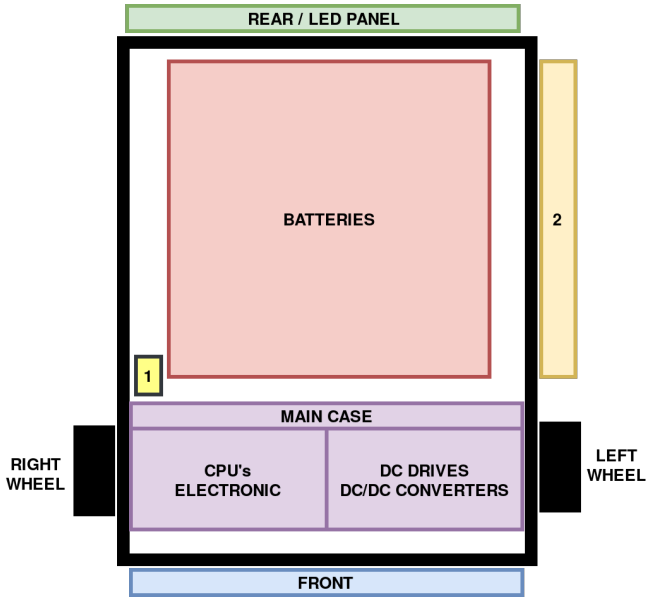



Figure 4.2: Platform's top view map.

### 4.1.2 Power

The platform is powered by 4 batteries in series, as stated in Sub-section 3.4. Since the old cases were used, some modifications had to be implemented. The cables were not in good condition and the connectors underneath the case (to connect the batteries in series) were damaged. To solve this, a new battery arrangement was made and new cables were placed, both for the series layout and charging purposes (Figure 4.4). As represented in the schematic from Figure 4.4 with a dark blue rectangles, two connectors were still used, the rear one to plug the charger and the front one to connect the robot's power. The final result is shown in Figure 4.3 where it is possible to observe some extra length in the cables so that the batteries can be operated easier and to allow its removal without disconnecting any cable. Figure H.1 shows an intermediate step of this stage and gives better detail on the battery cases.

The electric schematic is represented in Figure 4.5 where a circuit breaker, QF1, is responsible for disconnecting the main power. The circuit breaker was re-used from the old platform and it was mounted in a piece of DIN rail, fixed with rivets. It was fixed on the right side of the batteries, where it can be reached by the user from the outside. Figure 4.2 shows its location on the platform, represented by number 1, and Figure H.4 its assembling.

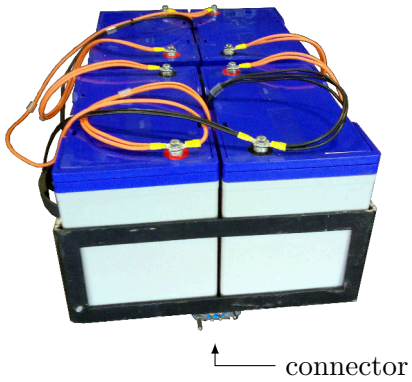



Figure 4.3: Mounted batteries.

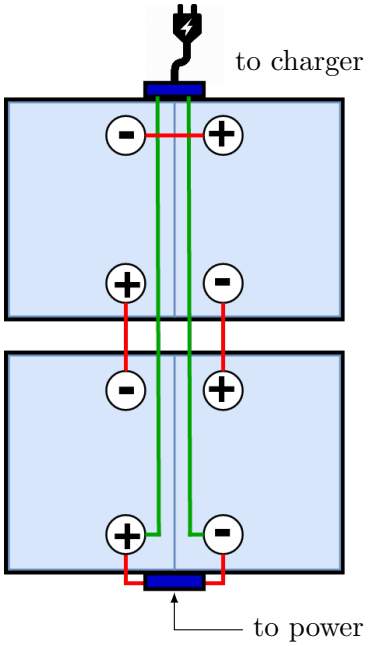



Figure 4.4: Batteries layout in the cases.

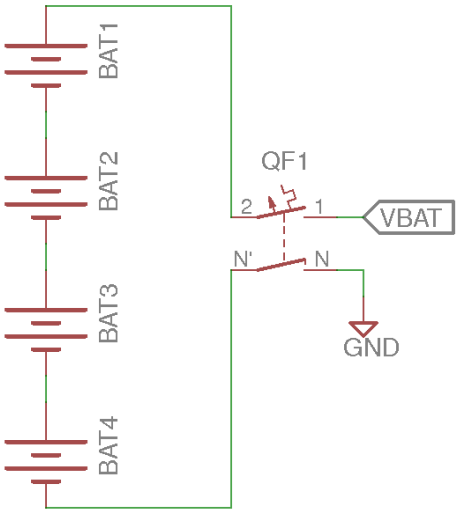



Figure 4.5: Main power supply, batteries schematics.

To charge the batteries the old charger was used. This device was not operational and required intervention to fix it. The connectors were damaged and therefore replaced and the connections on the charging plug had to be remade to match the new arrangement of the batteries in the cases (Figure H.2).

Concerning the installed DC/DC converters on the platform, 4 voltage levels are available: +5V, +/-12V and +22V, which are used for different applications (Table 4.4). The wiring of the DC/DC converters is represented in Figure 4.7. They are enabled with a ground signal (22A) and can be disabled with the ON/OFF button from the rear LED panel (Figure 4.15). This allows to have the robot powered with the electronics disconnected (useful for reset purposes). To have negative voltage (-12V), VO2+ and VO1- are connected to ground, creating a negative voltage on the VO2- pin.

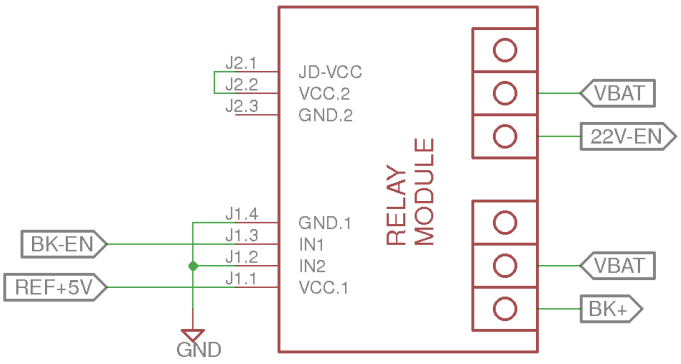



Figure 4.6: Relays Module schematic.

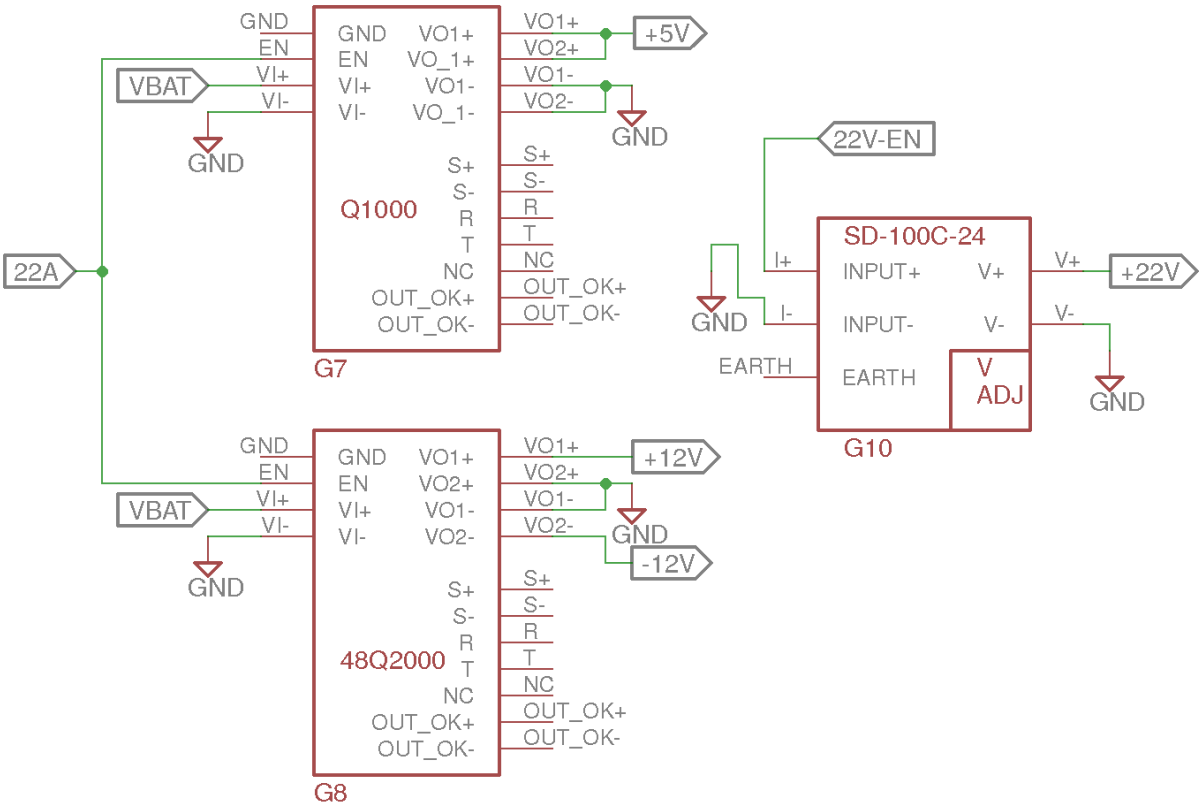



Figure 4.7: DC/DC converters schematic.

Since the 24V DC/DC converter is only used to power `CPU1` and does not have an enable input, the solution found to reset `CPU1` was to use a relay to cutoff its power. The relay is supplied with +5V, which is enabled by the ON/OFF switch (see further Figures 4.15 and 4.16 to identify the switch — BF2) on the rear. By connecting the converter's positive input to the relay's NO (normally open) contact, the `CPU1` can be connected/disconnected together with the remaining electronics (Figures 4.7 and 4.6).

Even though the converter is 24V, the potentiometer (V ADJ), that allows to adjust the voltage output level (Section 3.4), was used to drop the voltage to its minimum: 22V. Despite of the `CPU1`'s voltage supply is 19V, it was ensured by its vendor that it could be supplied with 22V due to its internal circuitry. Validation tests were carried out by connecting `CPU1` for an entire day and it presented normal behavior.

### 4.1.3 General

This sub-section presents the electric schematics for the fans and battery level.

**Fans**

Inside the main case, there were four fans used for air circulation. When tested they made too much noise (probably they were seized up) and had to be replaced for new ones. Instead of ordering four similar fans, two 12V fans were taken from old computers and mounted on a similar structure to the one represented on Figure H.12. Since they had bigger dimensions than the previous ones, one of the fans had to be installed inside the structure to fit everything inside the main case (in Figure 4.2 the fans are represented with letter "F"). The wiring used to connect the fans is shown in Appendix B, where the full platform's schematic can be observed.

**Battery Level**

The battery level is measured with a designed circuit (Figure 4.8) that has a resistor in series with a potentiometer (X2). The main reason for this was because the optimal battery levels to the platform's functionality are not known yet, and with X2 it is possible to tune them in the future. This way, the voltage levels can be optimized so the analog read can have the maximum scale (5V). The components were dimensioned with use of equation (4.1).

For the potentiometer installation, and to grant its accessibility, a hole on top of the main case was drilled (Figure H.3).

$$5V = \frac{X2_{bottom}}{R9 + X2} V_{BAT} \tag{4.1}$$

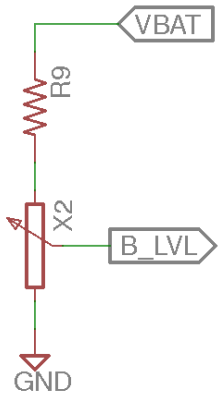



Figure 4.8: Battery level acquisition circuit.

### 4.1.4 CPU2 Mount

`CPU2` is in charge of establishing communication between the hardware and `CPU1` through Ethernet and for controlling the motors. The connections are represented in the schematic from Figures 4.9 and 4.23 and the labels detailed in Table 4.1.

Regarding the relay module, (schematic on Figure 4.6) it contains optocouplers and the input circuit can be isolated from the relay circuit by removing the JD-VCC jumper and providing a separate power supply for the relay, which was not necessary in for its application [46]. For safety reasons, the used contact for the brakes was NO, preventing an unbraked system in case of power failure.

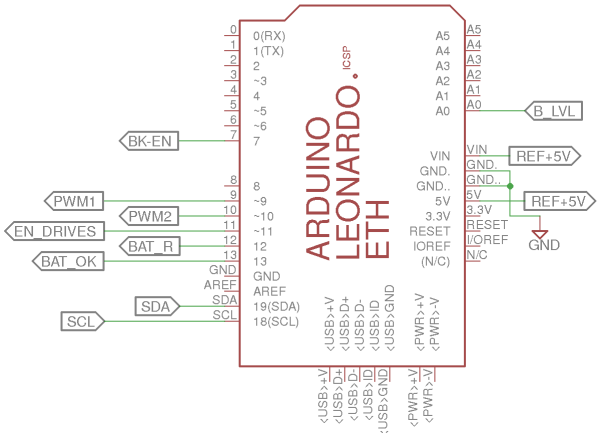



Figure 4.9: CPU2 IO circuit.

Table 4.1: CPU2 IO's list.

| Pin | Description | Label |
|---|---|---|
| 11 | Digital input. Emergency/ Bumper state | EN-DRIVES |
| 7 | Digital output. Brake actuation | BK-EN |
| 12 | Digital output. Batteries Recharge LED | BAT_R |
| 13 | Digital output. Batteries OK LED | BAT_OK |
| A0 | Analog input. Batteries level | B_LVL |
| 9 | Analog output. Connects to REF+ from Drive 1 | PWM1 |
| 5 | Analog output. Connects to REF+ from Drive 2 | PWM2 |
| SDA | Data line. $I^2C$ communication | SDA |
| SCL | Clock line. $I^2C$ communication | SCL |

### 4.1.5 CPU3 Mount and Circuit

CPU3, Arduino Micro, is responsible for acquiring the data from the encoders and transferring it via $I^2C$ to CPU3. For those purposes, the unit was installed in a prototype board with additional components (Figure 4.10). To place CPU3, female PCB sockets with 2.54mm of pitch (same as the prototype board) were used. The remaining wires were connected with PCB screw terminal blocks, again with the same pitch. This kind of connections allows the replacement of components easier, which is useful in case of malfunctions.

Regarding the schematic (Figure 4.11), the four capacitors are used to filter the noise coming from the encoder signal. Even without resistance (to make a RC filter) the capacitors work, eliminating high frequency noise. Since the encoders can reach up to 15 kHz frequencies, a 10nF ceramic capacitors were used.

The two wires for $I^2C$ communication were directly connected between CPU2 and CPU3. On the schematics (Figures 4.9 and 4.11) there are no pull-up resistors on the SDA and SCL lines because CPU2 already has them internally [39]. The labels description for the IO's can be found in Table 4.2

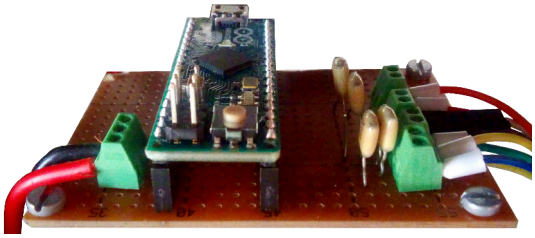


Figure 4.10: `CPU3` mounted on prototype PCB board.

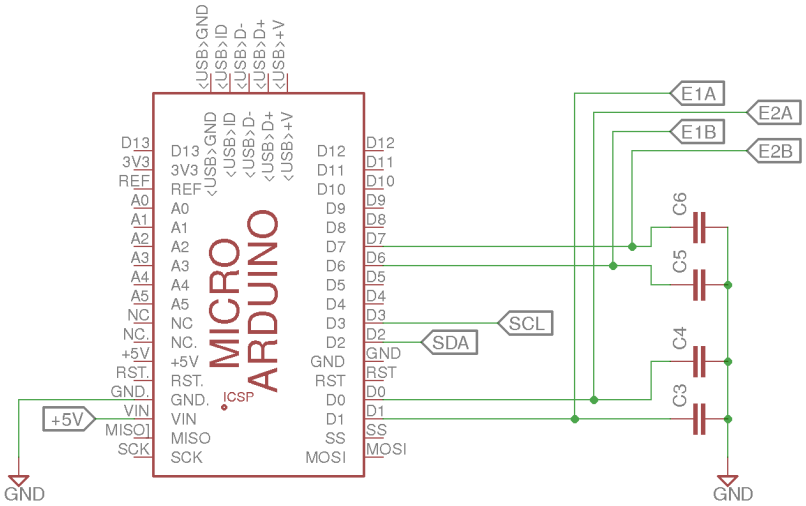



Figure 4.11: `CPU3` IO's circuit.

Table 4.2: `CPU3` IO's list.

| Pin | Description | Label |
|---|---|---|
| 2 | Serial data | SDA |
| 3 | Serial clock | SCL |
| 0 | Encoder 1 signal, channel A | E1-CHA |
| 1 | Encoder 1 signal, channel B | E1-CHB |
| 6 | Encoder 2 signal, channel B | E2-CHB |
| 7 | Encoder 2 signal, channel A | E2-CHA |

### 4.1.6 Wheel Setup and Circular Connector (C-16P)

As previously stated in Section 2.2, a main circular connector to access the enclosed wheel setup is available. This connector wiring is represented in Figure 4.12 alongside with the encoders, brakes and motors. Despite of the connector being circular, in order to make the interpretation easier, it is represented by a rectangle where its numbering matches with the actual pin numbers in the connector. Both male (C-16P-M) and female (C-16P-F) parts are represented, establishing correspondence with the schematics from Figures 4.9 and 4.11.

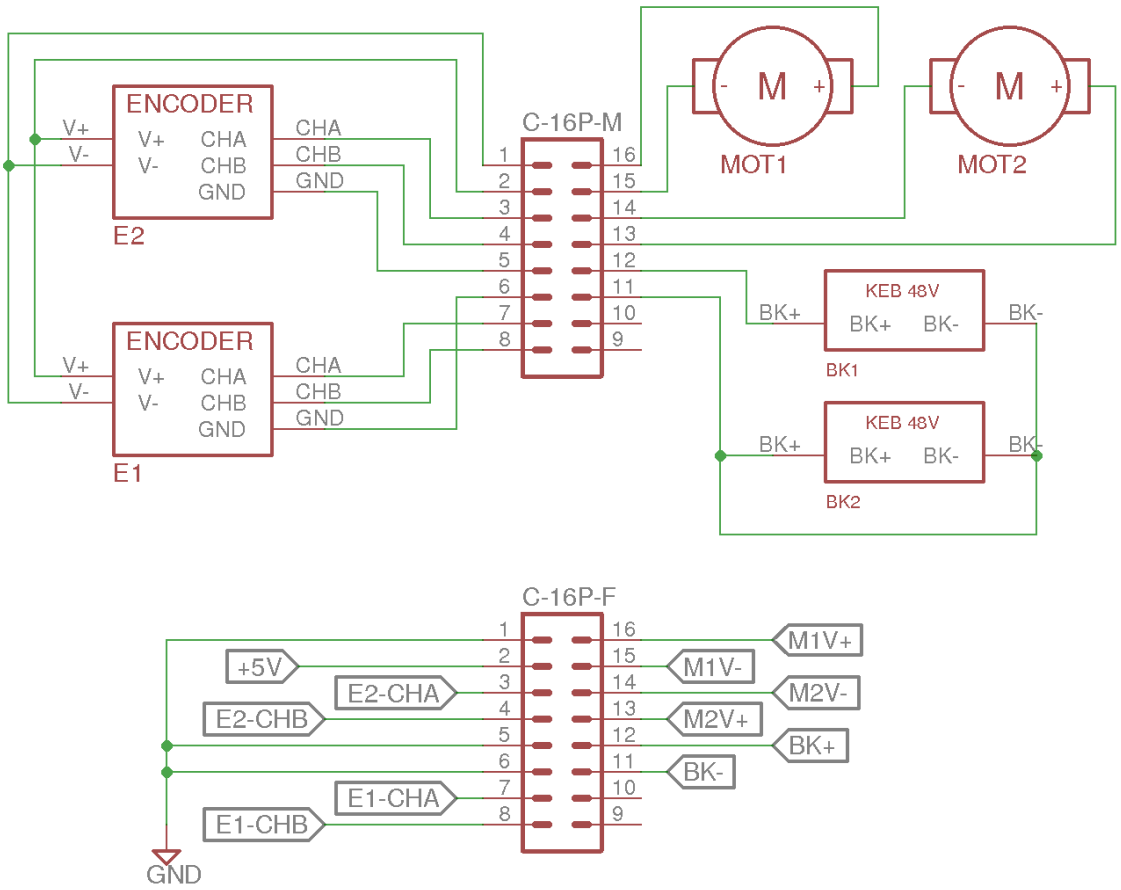



Figure 4.12: Pinouts from 16 pin circular connector (C-16P-M and C-16P-F).

### 4.1.7 Level Shifter

A circuit was projected to convert a PWM signal to an analog signal from [-5;5V] (Figure 4.14). The design of the circuit will be explained in context on Section 4.2.

Similarly to the `CPU2`, a PCB prototype board, illustrated in Figure H.5, which was particularly chosen to avoid making wire or solder paths, was used to implement this circuit. To cut the paths when needed, a Dremel tool was used to erode them. The implementation of the circuit, as connected in the platform, is presented in Figure 4.13.

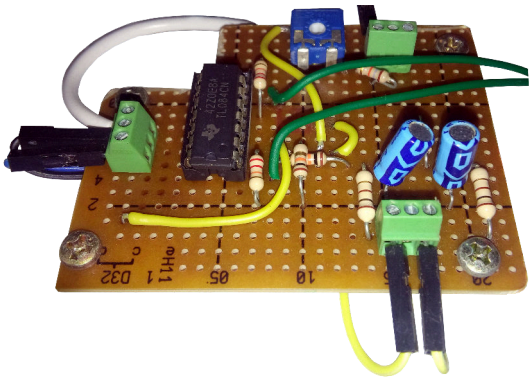


Figure 4.13: Level shifter circuit mounted.

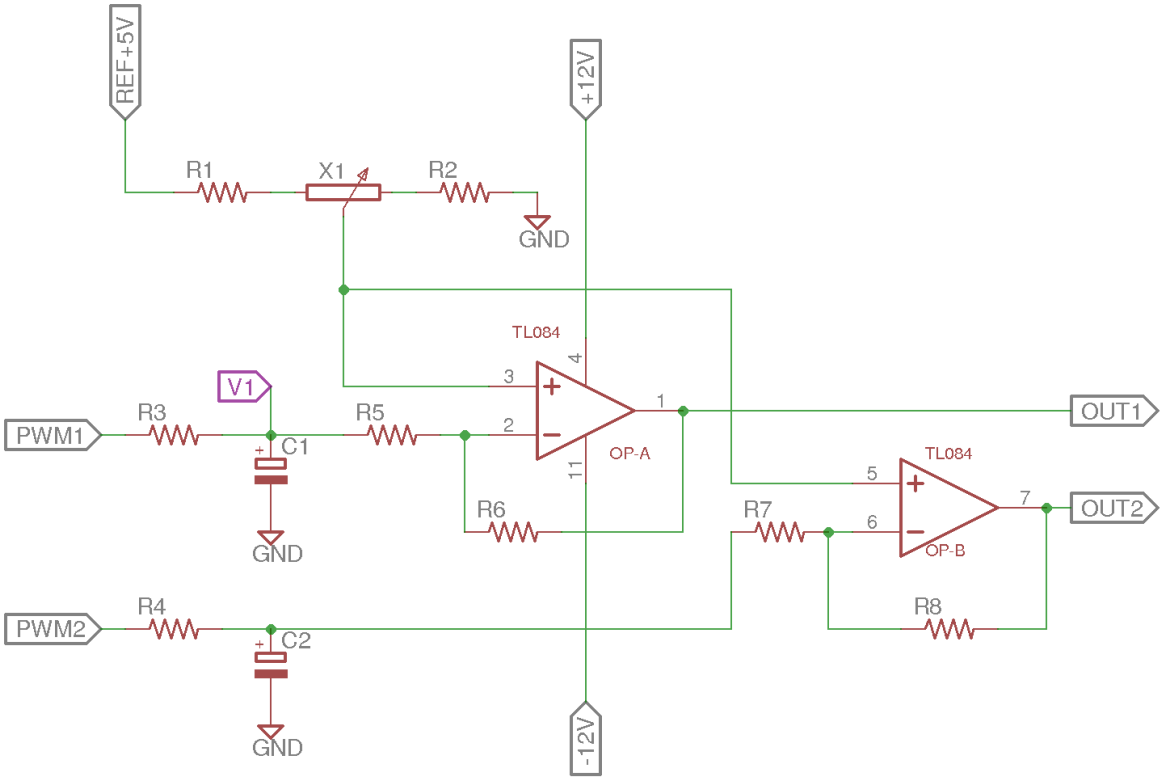



Figure 4.14: Circuit to convert 5V PWM signal into [-5;5]V analog signal. Components values can be found in Table G.

### 4.1.8 Status Panel

To be aware of some machine information and enable/disable modules, the old Robuter already had a LED status panel on the rear. On the original panel it was possible to:

- Monitor the battery level (3 states - OK, RECHARGE and N.OK);
- Monitor the voltage levels (+5V, +/-12V and +48V);
- Actuate the emergency and monitor its state (red LED);
- Reset the old on-board computer;
- Enable/disable the DC/DC converters.

Despite the usefulness of all these features, the panel's PCB layout did not allow it. In Figure H.6 is shown the existing PCB that had to be adapted so it could be used with the new electronics. After deep analysis and identification of every pin from the DB25 connector to match with the PCB pads (Table 4.3), some modifications were made:

- Emergency contact changed from NC to NO;
- Modification of the LED paths to connect them individually;
- Addition of resistors for the LEDs — $R_{LED} = \frac{V_{supply} - V_{LED}}{I_{LED}}$;
- Modification of the ON/OFF button connection to NC contact (DC/DC converters need 0V to be enabled).

During this intervention, the red LED to inform the emergency state had to be disconnected. The same happened with the "BATTERIES N.OK" red LED. As mentioned before, some resistors were added to limit the LEDs current.


Figure 4.15: LED status panel on Robuter's back. Check description on Table 4.3.

Since the space on the PCB was limited, it was decided to solder the resistors directly on the cables (see Figure H.7). While doing this, each cable was extended because they did not had the necessary length.

Due to the previous connections, the battery status LEDs were connected to 5V (28a pad in the PCB, see Table 4.3). This way, to connect them from `CPU2`, the digital output needs to be set "LOW" ("HIGH" to disconnect). On the schematics from Figure 4.16 are represented all the necessary connections to understand the modifications made.

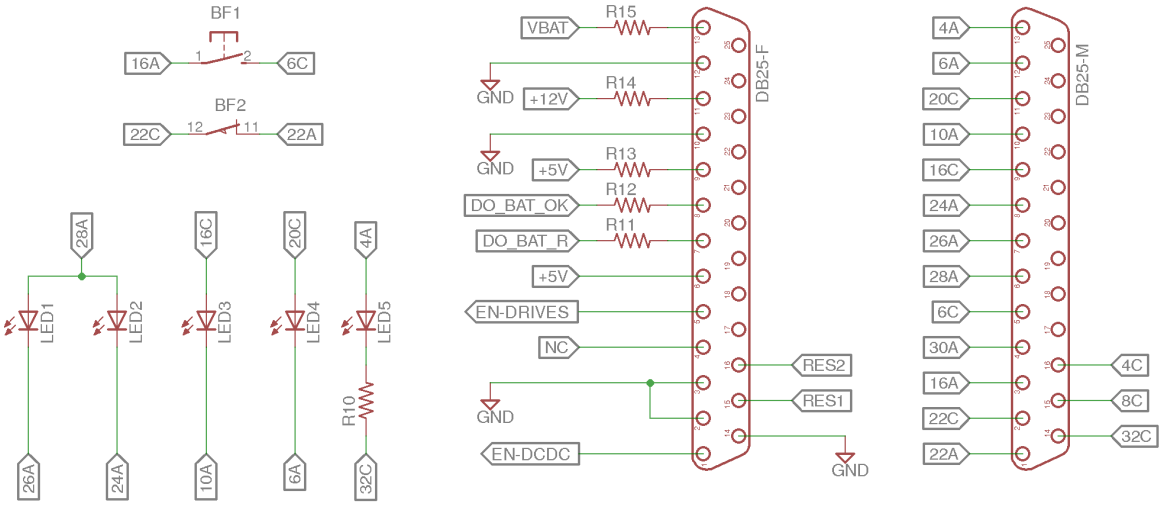



Figure 4.16: LED status circuit with DB25 pinout.

After removing the ultrasonic sensors (Section 2.2) a lack of mechanical resistance was detected on the rear part (Figure 4.15). In order to solve this, pieces of roofmate (Figure H.8) were placed, increasing the safety in case of unexpected collisions.

### 4.1.9 Fixing Plates

To install and fix some of the components it was necessary to design, in SolidWorks, parts for that purpose or, in other cases, use a different method, as it will be explained in the present sub-section.

#### Circuitry Fastening

For the circuitry, a polycarbonate sheet (technical drawing on Appendix C) with many openings was used to allow the air circulation inside the main case and to arrange/fix the

Table 4.3: Signal matching between LED panel PCB and DB25 connector from Figure 4.16.

| Num. DB25 | ID PCB | Function / Notes | Label in schemes |
|---|---|---|---|
| 1 | 22a | NC contact pin 14. OFF – close, ON – open | EN-DCDC |
| 2 | 22c | NO contact pin 13. OFF – open, ON – close | GND |
| 3 | 16a | Emergency, BF1, ground contact. | GND |
| 4 | 30a | GND Red LED (EMR) - not connected | NC |
| 5 | 6c | Emergency, BF1, NO contact DC drives enable | EN-DRIVES |
| 6 | 28a | Common 5V for LED1 and LED2 | +5V |
| 7 | 26a | GND Yellow LED1 (Recharge) | BAT_R |
| 8 | 24a | GND Green LED2 (Batteries OK) | BAT_OK |
| 9 | 16c | V+ Green LED3 (+5V with 220Ω resistor) | +5V |
| 10 | 10a | GND Green LED3 +5V | GND |
| 11 | 20c | V+ Green LED4 (+12V with 680Ω resistor) | +12V |
| 12 | 6a | GND Green LED4 +/-12V | GND |
| 13 | 4a | V+ Green LED5 (VBAT with 2.7kΩ + 1kΩ resistor) | VBAT |
| 14 | 32c | GND Green LED5 VBAT | GND |
| 15 | 8c | Not used, Reserve1 | RES1 |
| 16 | 4c | Not used, Reserve2 | RES2 |

wires properly. To lock the PCB boards, holes were drilled on the extremities (four holes per board) and attached with a set of M3 screw, nut, washer and spacer screw (Figure H.10 shows this in detail). The final result, with all the installed boards, is presented on Figure 4.17

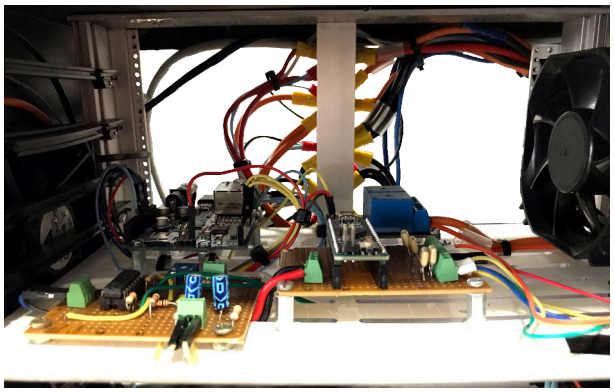


Figure 4.17: Polycarbonate sheet inside the main case with all PCBs mounted.

### Placement of Network Devices

For the network setup (router and switch), a metal plate was designed to fix them in a wall mount type (while selecting the equipment this was a relevant concern). On the technical drawing (Appendix D) three half circles, that were made to pass the cables more smoothly, are represented. Concerning the fixation of the plate to the vehicle's structure, it was decided to make use of the existing M6 holes (for the bottom part) and additionally were placed holes to fasten with nylon cable ties (more specifics on Figure H.11). Diagram 4.2 indicates where the network hardware is located with the number 2 and Figure 4.18 exhibits the final mount.

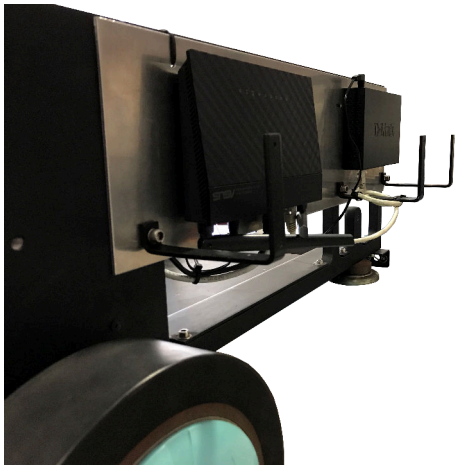


Figure 4.18: Metal plate with router and switch installed.

### Placement of Visual Units

In order to use the camera, two different aluminum supports were designed, Appendixes E and F. The decision of the camera's location was based on some constraints, such as:

- Safety of the equipment — it should not exceed the platform's dimensions;
- Manipulator's operability — the robotic arm's movements can not be conditioned by the camera location;
- Pan and tilt (PTU) full range — the PTU, where the camera is fixed, must have space to reach its rotation range;
- The acquired image must be from the front side of the robot.

The ideal camera location is in the center of the platform in order to simplify the navigation algorithm, but at the time of the first design and due to the presented constraints that was not possible. So, on the first designed support, the camera was placed in the left side of the platform, as shown in Figure 4.19.

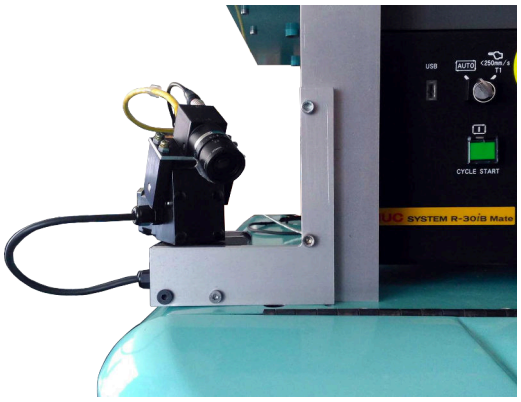


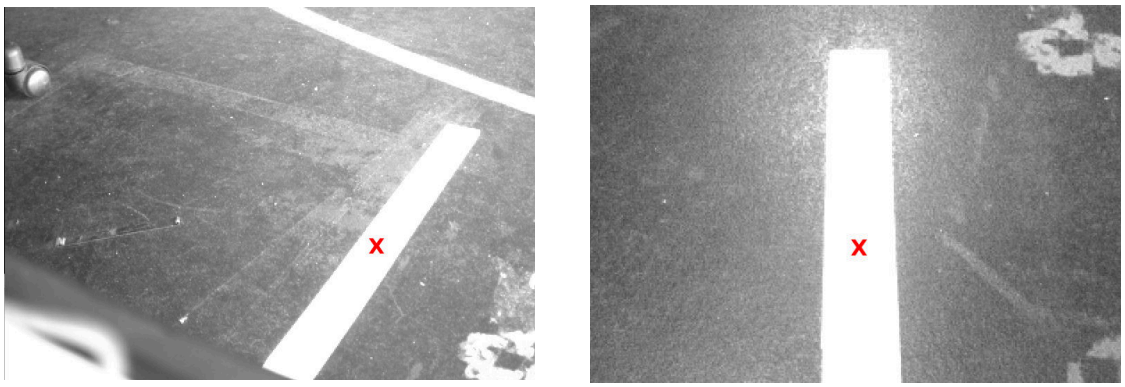

Figure 4.19: First camera setup used for navigation. Detailed view of the camera mounted on the PTU.

Despite this fact, after performing the tests with a simple navigation algorithm, that detected a line and its orientation, it was noticed that a more complex one had to be designed (the camera lost track of the line). By that time, it was already possible to rearrange the

components on the top of the platform so the camera and PTU could be centered with it, minimizing/eliminating the effect that the perspective introduced to the image. For that purpose, a new support was designed, Appendix F, and the complete setup is documented on Figure 4.20.


Figure 4.20: Final camera setup used to implement the navigation algorithm. Detailed view of the camera mounted on the PTU.

The difference between the acquired raw images (mono8 format) can be observed in Figure 4.21 where the white line (marked with a red 'X') on both images is similar, only the camera's location and orientation changes.




(a) (b)

Figure 4.21: Raw images acquired from the first setup (a) and second setup (b). The navigation line is marked with a red 'X'.

### 4.1.10 Wire Labeling

Another important procedure during the electrical installation was the definition of metrics/rules for the wiring. A color code was specified, presented on Table 4.4, making the identification of the cables easier for future enhancements/repairs. Additionally, labels to place in marker sleeves TM-I 18 from Weidmüller (reference 1798620000) were printed, which identify the majority of the cables (the printed label list can be seen in Appendix I).

In order to connect all the wires on the platform in an arranged and organized way, two power distribution bus bars (see Figure H.13) were used. These bars were drilled to be aligned vertically, and since they have three available screw terminals in each, for the most common wires (VBAT and GND), two were used. Figure 4.22 shows the rear view of the main case, where the bus bar and all the connections can be seen.

Table 4.4: Cable identification from Figure 4.22.

| Code | Description | Signal | Color |
|---|---|---|---|
| VBAT | Battery level supply for motor drives, DC/DC converters | +48V | Orange |
| +22V | Supply for `CPU1` | +22V | Striped blue |
| +12V | Supply for `CPU2` and Router | +12V | Blue |
| -12V | Voltage reference for level shifter circuit | -12V | White |
| +5V | Supply for encoders, `CPU3` and Switch | +5V | Red |
| GND | Ground | 0V | Black |
| M1-V+ | V+ Motor 1 supply, signal from drive 1 | 0-48V | Brown |
| M1-V- | V- Motor 1 supply, signal from drive 1 | 0-48V | White |
| M2-V+ | V+ Motor 2 supply, signal from drive 2 | 0-48V | Brown |
| M2-V- | V- Motor 2 supply, signal from drive 2 | 0-48V | White |
| PWM1 | PWM signal from the controller to the drive 1, REF+ | [-5;5] V | White |
| PWM2 | PWM signal from the controller to the drive 2, REF+ | [-5;5] V | White |
| BK+ | V+ Common supply for electromagnetic brakes | 48V | Orange |
| BK- | V- Common supply for electromagnetic brakes | 0V | Black |

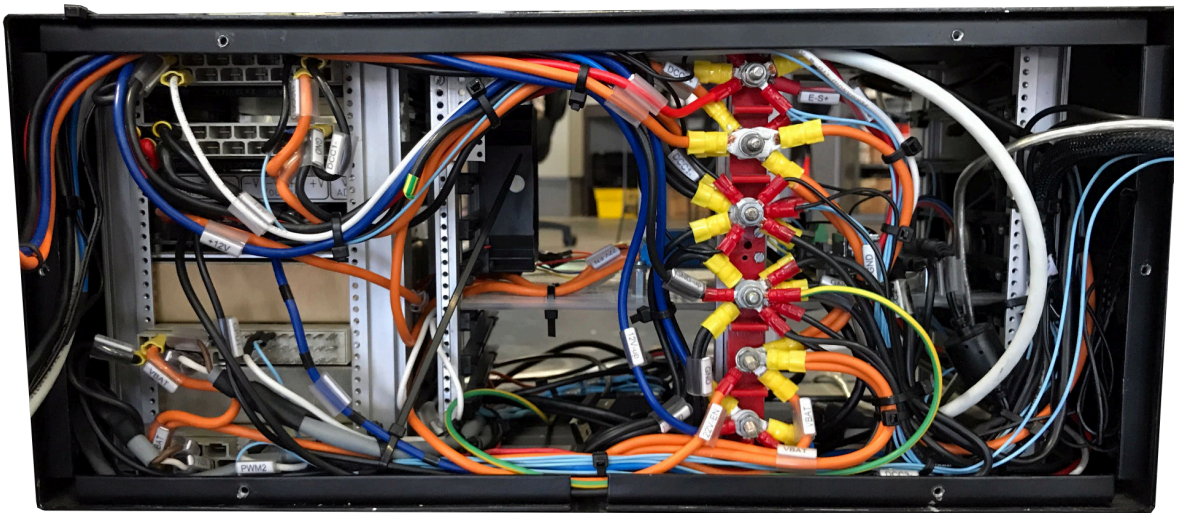


Figure 4.22: Wire arrangement in the power distribution bars.

## 4.2 Motor Actuation

To actuate the motors, two DC drives are used (Copley 306), but only a deep analysis and one repair was needed. One of the drives had two soldered pins from the PCB that fitted on top of the drive (Figure 3.3 for more details) that could not be soldered together. To solve this, it was necessary to cut some pins and remove the PCB using a soldering station with infrared technology (to maximize and uniform the heat on the row of pins). After cleaning the solder from unwanted places, the pins were leveled to fit a row of female PCB sockets (same pitch as the pins: 2.54mm).

With this solution (Figure H.9) it is still possible to remove the top PCB to access the rest of the DC drive. This PCB had the purpose of making the necessary connections to the driver and convert the pinout into an Eurocard mount. Besides that, the type of signal to control the motors was modified. The original drives had two modes of operation:

- PWM signal [0-5]V on REF+ pin and 0/5V on DIR pin;
- Single [0-5]V PWM signal where 50% of duty-cycle produces zero current at the output of the amplifier (motor stopped).

After the modification, each drive works only with a single analog signal from [-5;5]V, where the [-5;0] range makes the wheels rotate forward and the [0;5]V range backwards.

After understanding this feature, a circuit had to be projected to allow the control from `CPU2` (Figure 4.14 ). This is necessary since the Arduino board only has a variable output type: PWM from [0;5]V with adjustable frequency. The signal modulation was performed with a difference amplifier (expression (4.3) is for OUT1 but the same relation can be made for OUT2). Note the variables, are identified in the schematic from Figure 4.14.

The simple RC low-pass filter (R3-C1 and R4-C2) shown in the circuit converts the PWM signal to a voltage proportional to the duty cycle. Expression (4.2) shows how these values can be calculated, but note that two major aspects had to be considered: ripple and settling time. If the cutoff frequency is too low, the RC constant will be too high, meaning the capacitor will take more time to output the desired voltage. On the other hand, if the cutoff frequency is too high, the ripple will be larger. To solve this problem, a lower resistor was used, and the PWM frequency was adjusted to 31.25 kHz.

$$f_c = \frac{1}{2\pi RC} \tag{4.2}$$

$$OUT1 = 5V \times \frac{R_2 + X_{1VAL}}{R_1 + R_2 + X_{1TOT}} \left(1 + \frac{R_6}{R_5}\right) - V_1 \frac{R_6}{R_5} \tag{4.3}$$

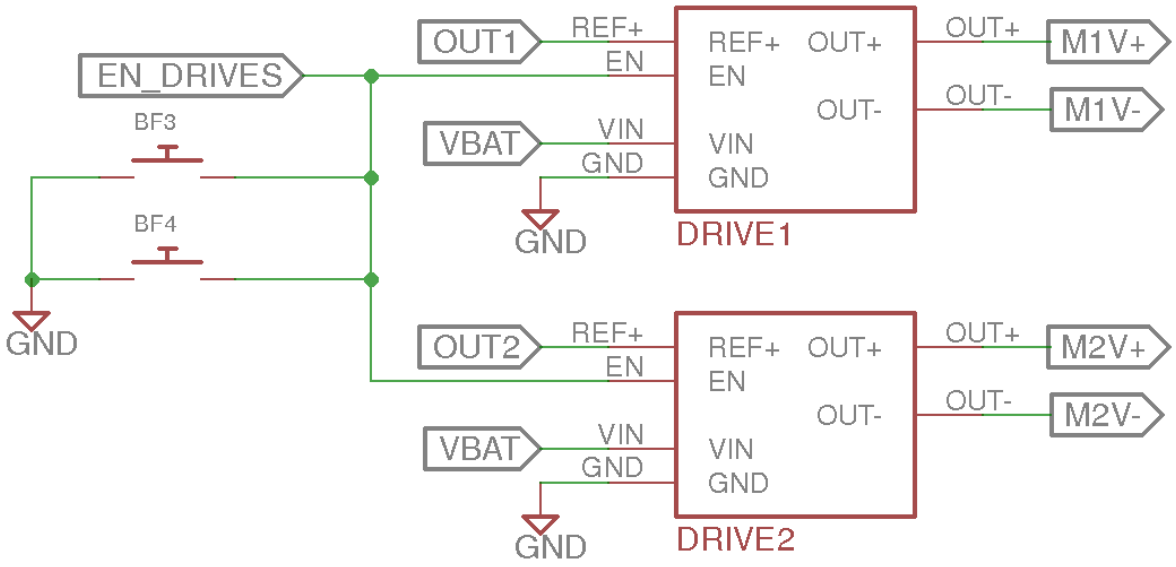



Figure 4.23: DC motor drives schematic.

There is also a potentiometer, X1, that allows a fine tuning on the voltage reference. This is important since the zero voltage applied to the motors will occur when the Duty Cycle is at 50%, and this tuning will grant that. Note that the values for the components presented in Figure 4.14 are presented in Table G.1.

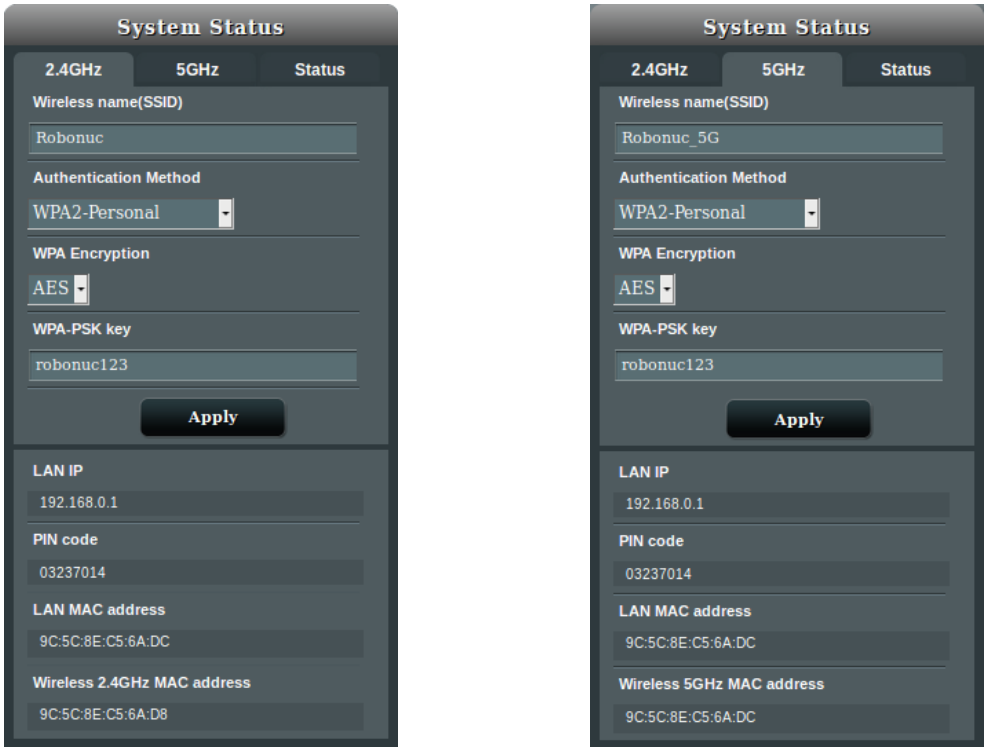
Still concerning the DC drives (schematics on Figure 4.23), they have a disabling input (with internal pull up resistor) that when it has a 0V signal, it activates the emergency mode, stopping the motors. This emergency can be activated in two ways:

- Emergency push button pressed (on the rear LED panel);
- One of the two bumpers (BF3 and BF4) detects collision. At this stage only two wires were passed for each bumper to install them in the future, but connecting them together simulates a collision event.

## 4.3 Configurations

### 4.3.1 Network Creation

To share data between subsystems, a network was created on the platform with a router and switch. On the router, the DHCP was disabled in order to use static IP addresses, allowing the manual configuration of the network equipment. Figure 4.24 shows the details of the configured network for the platform, while Figure 4.25 illustrates the network diagram used on the robot.




(a) (b)

Figure 4.24: Print screen from the Asus router webpage with both configured networks: 2.4 GHz band (a) and 5GHz (b).

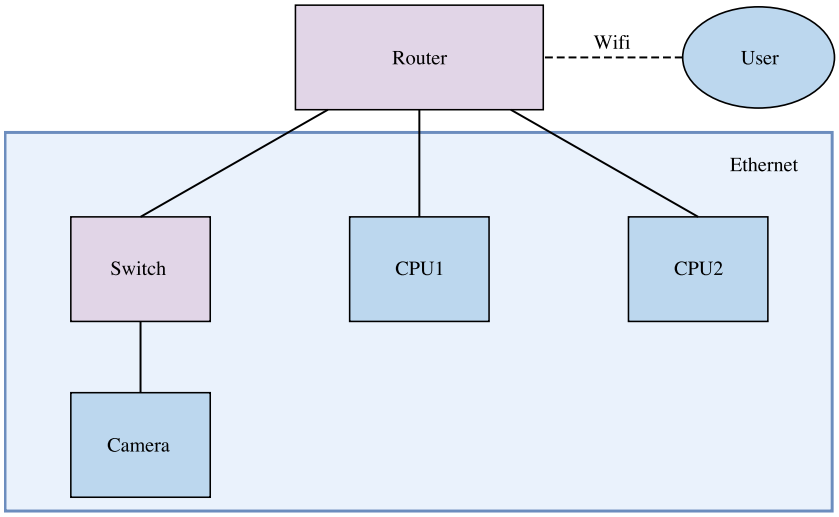



Figure 4.25: Diagram with implemented network topology.

### 4.3.2 CPU1

The `CPU1` was the unit that required more configurations. It is the main processing unit of the system and where ROS will be running. Firstly, an operating system was installed, Ubuntu 16.04 LTS. Ubuntu is a complete Linux operating system, freely available with both community and professional support [48].

After setting up the operating system, several tools had to be installed. In Appendix J is a simple guide, with all the installation commands and some notes about each one of them. This way, the PC can be configured exactly the same way for a fresh reinstall, if necessary.

Another important aspect to mention is the BIOS configurations. Usually, every PC starts when the power button is pressed, but in this case that is not a viable solution due to the `CPU1` being inside the main case. To solve that, it was enabled the AC Power Recovery setting. For better comprehension and mainly for re-installation purposes, a print screen from the BIOS setup is shown in Appendix H, Figure H.14.

### 4.3.3 Camera

As previously stated in Sub-section 3.4.4, the camera was already used in another project, so it had a different IP range from the platform. The predefined IP was in the range **"169.254.0.XXX"** while the platform's network in the range **"192.168.0.XXX"**.

In order to change the camera's IP, it was necessary to install its API, Flycapture 2, and create a network inside the camera's IP range to be able to detect it. After detecting the camera, it was possible to access its configuration menu, and its registers. Figure 4.26 illustrates this process, where it was necessary to write in the hexadecimal registers to configure permanently the desired IP address range. This API also provides access to a wide range of camera settings that can be configured manually on it, or by using FlyCapture2 library, written in C++.

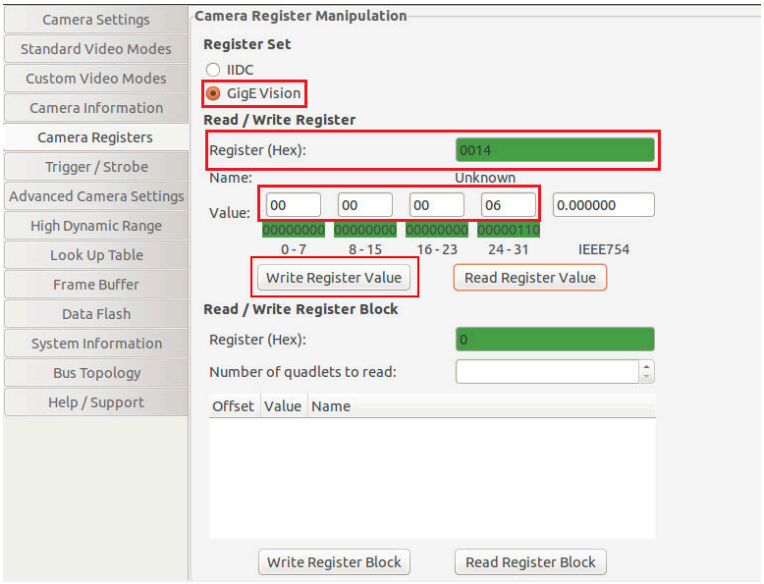



Figure 4.26: Setting the camera's IP range to the predefined in the Robonuc's platform. Illustration adapted from the FLIR documentation [49].

## 4.4 Generic ROS Nodes

### 4.4.1 Communication between CPU1 and CPU2

The most important part of the `CPU1` programming is the communication between the ROS and the `CPU2`. This communication is established via Ethernet and it is crucial, because it allows to actuate the system and receive data from it while taking advantage of the processing capacity of `CPU1`. So, the data transmission between units, is carried out by a client node, named "**client_node**". After initiating the ROS environment (with `roscore`), the node is initialized with the following command, where "-e" represents the echo mode:

```
$ rosrun comm_tcp client_node 192.168.0.10 50000 -e
```

When the connection is established, this node subscribes to the "**\client_messages**" topic, that contains the messages to be transferred, at a rate of 20 Hz. The type of messages are presented on Sub-section 4.5.1.

### 4.4.2 Velocity Conversion

Since the system was designed to output linear and angular velocities, these values had to be converted in motor commands (see Section 2.3). Node "**vel_decompose**" is the publisher and subscriber node responsible for that particular task. Velocities are subscribed from the "**\navi_commands**" topic and equations (2.3) and (2.4) are used to perform the conversion.

After the conversion, each wheel velocity is determined to correspond to the desired linear and angular velocity required by a navigation oriented node. This relative velocity is obtained in radians/second; however, the input motor commands are given in pulses, as will be presented in Sub-section 4.5.1. Equation (4.4) relates the velocity with the pulses (encoder increments), where "$F$" represents the sampling rate (20 Hz) and "$P_W$" represents the number of pulses to complete a full wheel rotation (8024, note that this value is multiplied

Table 4.5: Communication commands between CPU1 and CPU2.

| Type | Message |
|---|---|
| Motors | MM B/F [0-200] – B/F [0-200] > |
| Request encoder | EN > |
| Atuate brakes | BK 1/0 > |
| Battery level | BT > |

by the gear box ratio). The result, $P_r$, represents the necessary number of pulses to be counted in $\frac{1}{F}$ of a second. Although the equation refers to the right wheel, the same relation is valid for the left one.

$$P_r = \frac{v_r P_W}{2\pi F} \tag{4.4}$$

In the end, the message is packed in a *string* datatype to match the required format. When this process is concluded, the motor's message is then published in the topic "**\client_messages**".

## 4.5 Microcontrollers Programming

### 4.5.1 CPU2 Program

CPU2 was configured as an Ethernet server (**IP address:** 192.168.0.10, **Port:** 50000) so the ROS application can access its data through a client request, as explained previously.

Figure 4.27 shows the main loop running in CPU2. The program starts on "Setup", where all the necessary initializations are made, such as:

- Adjustment of the PWM frequencies to 31.25 kHz;
- Definition of input/output pins;
- Brake system and stop motors;
- Start $I^2C$ module as master;
- Start Ethernet connection and server;
- Definition of timer interrupt frequency.

After the client connection is established (in CPU1), the received messages are validated and then processed. As it can be observed in Figure 4.28 and Table 4.5, there are four main types of messages that can be distinguished by its first two characters: MM, BK, EN and BT. These last three messages (BK, EN and BT) are more simple to comprehend, while the other requires more attention.

The type MM defines the setpoint of each motor, in pulses (which is directly proportional to rpm's), and its direction of rotation. For example, the message "MMB50-F30>", defines the setpoint for motor 1 at 50 pulses in the backward direction and the motor 2 with 30 pulses in the forward direction. Highlighted in red is the type of message, in blue the direction of rotation, in black the corresponding references and in green the end character of the message, which is common to all types.

As stated in Section 4.2, each motor is controlled with a single PWM signal. The PWM resolution in the Arduino has 8 bytes, meaning the value can vary from [0; 255], where:

- From [0;127] — backward direction;
- From ]128;255] — forward direction;
- The value 128 — outputs 0V to the motor.

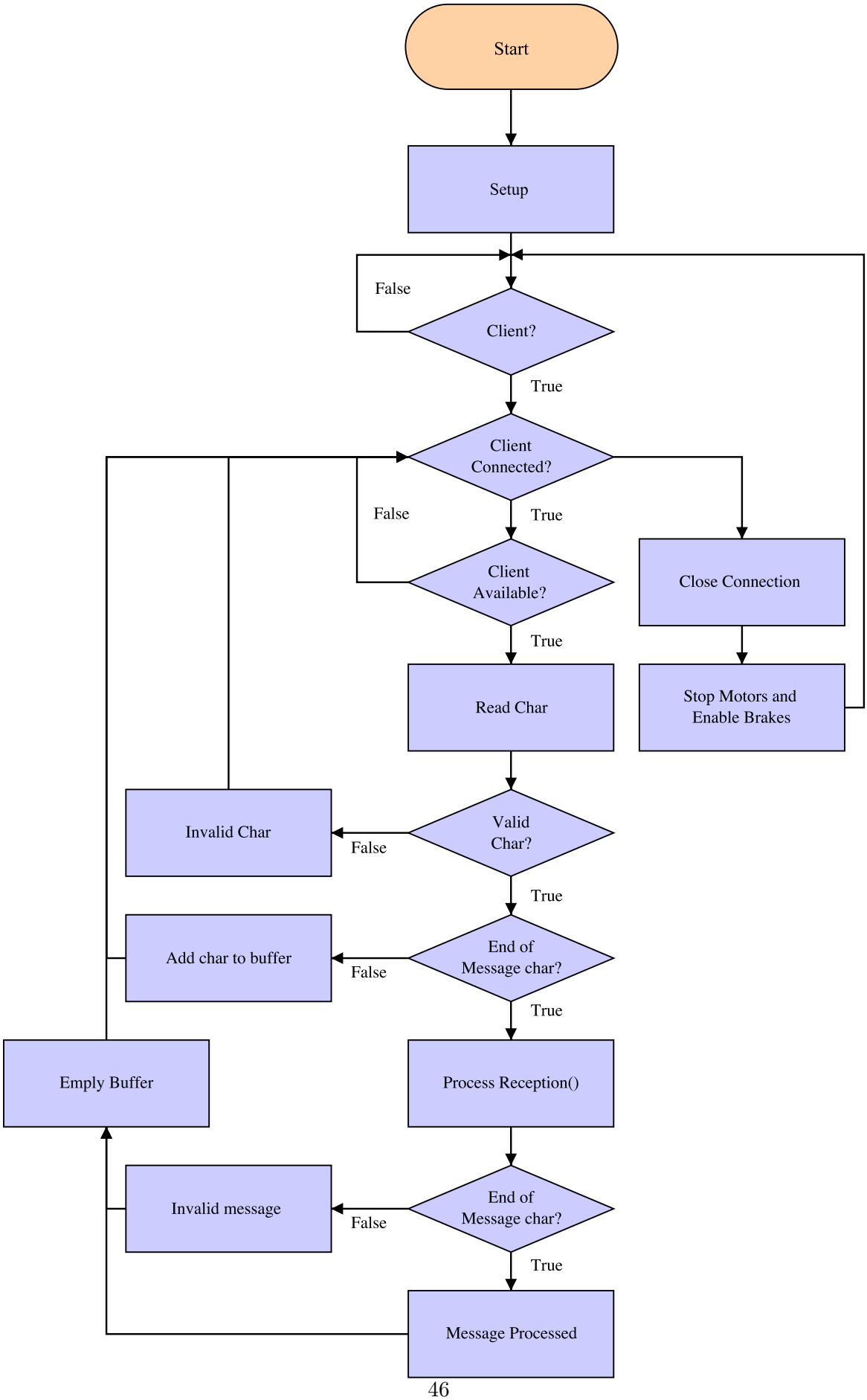





Figure 4.27: Flowchart from the main loop in `CPU2`.

This type of control has the advantage of saving IO's on `CPU2` (a single wire is enough to control each motor) but has the drawback of its control method, i.e., it is difficult to design a control system that within its range has increasing and decreasing values depending on the direction. To avoid a complex solution, a subroutine that accepts a direction input (B or F) and values from [0;128] that are later converted to the full range output ([0;255]) was created .

Running aside, as a timer overflow interrupt, a routine similar to a watch dog timer that monitors the message income from `CPU1` was created (Figure 4.29). If the defined number of messages are not received on the timer's interval, the system detects it and enters in an emergency state, braking the system and stopping the motors (PWM = 128).

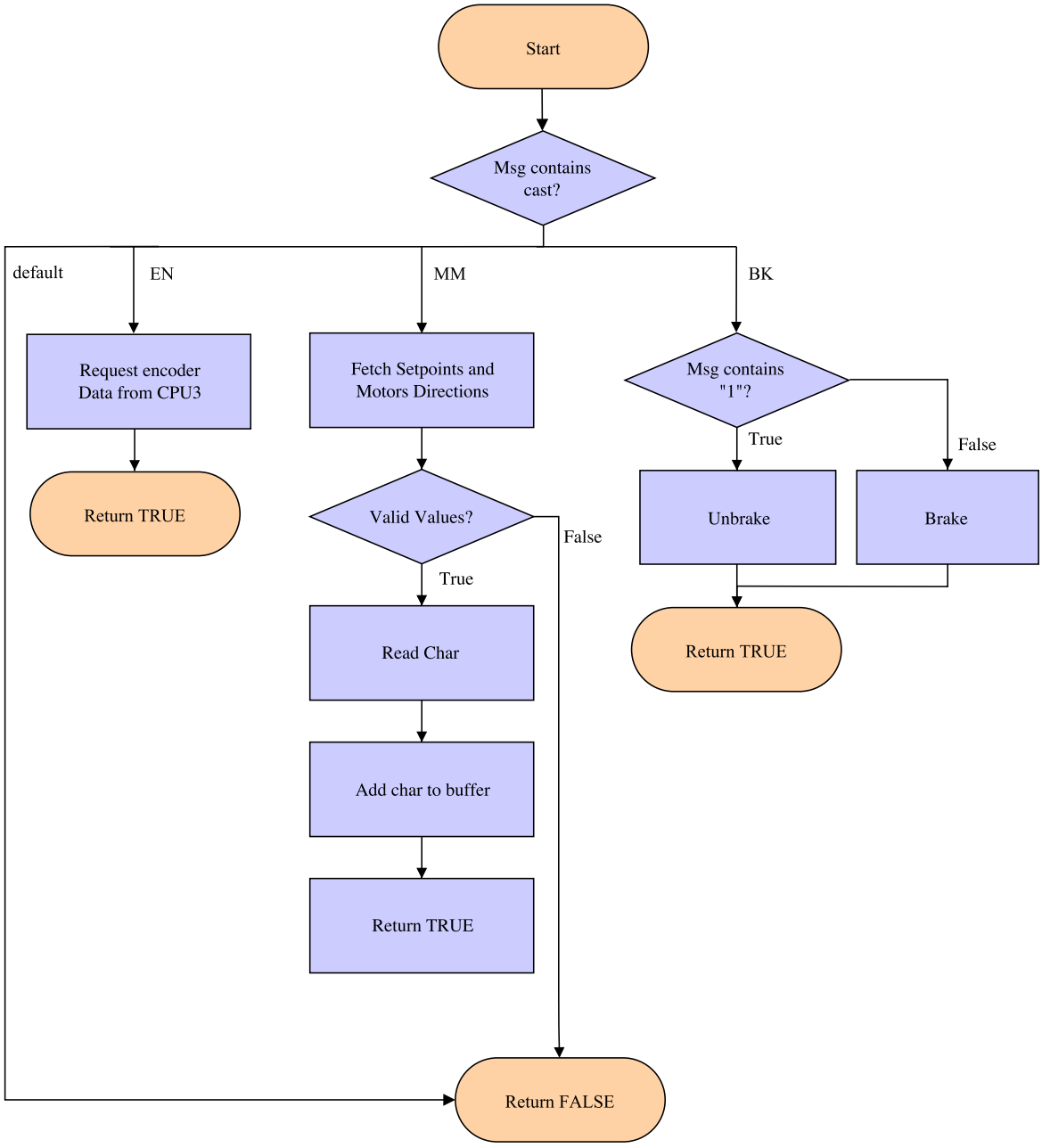



Figure 4.28: Flowchart from routine "Process Reception".

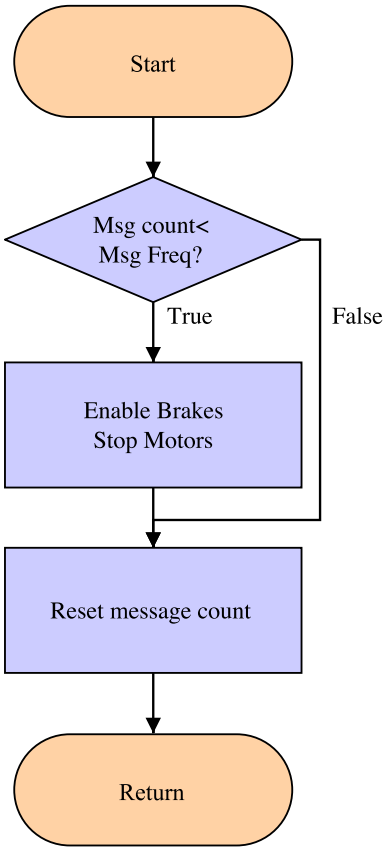



Figure 4.29: Flowchart from Timer 2 interrupt routine.

### 4.5.2 Read and Transfer Encoders Data

The encoder data is acquired with the Arduino Micro (Sub-section 3.4.3) because the Arduino Leonardo ETH can not respond fast enough (equation 4.5) while handling the Ethernet communication. To do it, some considerations took place on the selection of this unit:

- The platform's maximum velocity is set in 1m/s;
- The encoders have 8024 pulses per wheel revolution;
- The wheel perimeter is 75cm.

Having this in mind, the proposed solution must be capable of responding to a frequency of 16 kHz per motor (equation 4.5). In the equation, a slightly increased value for the velocity is used to give a safety factor (in case of higher velocities).

$$1.5m/s = 2rps \rightarrow f_{\text{acquisition}} \geq 16kHz. \tag{4.5}$$

To validate this solution, some tests took place. It was decided to read the pulses with interrupt service routines (ISR) instead of the polling method to have a better code structure running in the microcontroller [50]. To do it, a signal generator was used (configured with a square wave [0;5]V with 50% of Duty-Cycle) in order to determine the maximum interrupts frequency of the Arduino Micro while transferring its data via $I^2C$ or SPI.

The performed tests have shown capacity of response up to 145 kHz in a single interrupt pin and 72 kHz in two interrupt pins (with synchronous waves).

After the proof of concept, this method was implemented on `CPU3`. Since each encoder has two channels, A and B, to determine the sense of rotation, one of the channels is detected by an interrupt routine, channel A, while the other, B, is polled (with `digitalRead()` function). From Figure 4.30, it is possible to observe that both waves are 90° out of phase. While rotating backwards, when channel's A ISR is triggered, the value of channel B is `LOW`. The opposite happens when rotating forward, where channel's B value is `HIGH` when the interrupt is called. Depending on the sense of rotation, the variable containing the number of interrupts (one per encoder) is incremented/decremented.

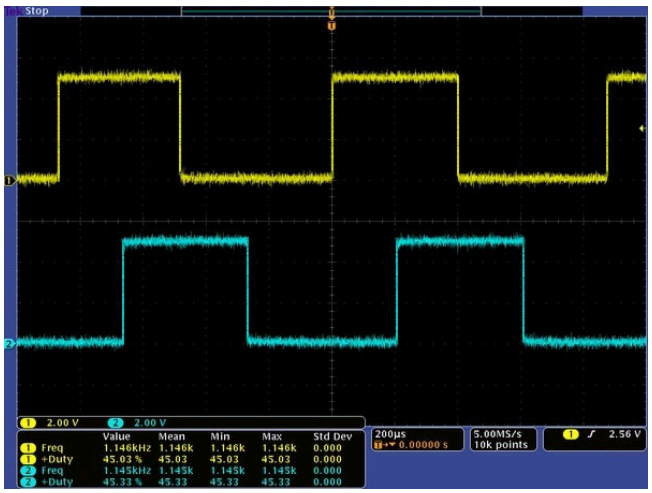



Figure 4.30: Encoder signals with wheel rotating forward. Channel A in yellow, channel B in blue.

To transfer data between `CPU2` and `CPU3`, the $I^2C$ protocol was used. Despite that, the first approach was the SPI protocol because the communication only needed to be set between two units and it is faster than $I^2C$ [51]. The code for the communication was developed and successfully tested, nevertheless, when combining it with the remaining one it was detected a conflict between the Ethernet and SPI libraries. Even though it was not the first choice, the $I^2C$ protocol has proved to be enough for this application and has one advantage that can be useful for future applications — it allows to communicate with multiple devices if needed.

Similarly to the encoder's readings, the data transfer was also configured with an interrupt routine. `CPU3`, when prompted by its master, `CPU2`, transfers its data via $I^2C$. The data is requested at a 20 Hz frequency and, in order to save time in its transfer, two unsigned *integer* variables (2 bytes each) were used.

Since the $I^2C$ protocol only can transfer one byte at the time, each variable is split before being transferred (`CPU3`) and concatenated on the reception (`CPU2`). The flowchart that represents the microcontroller's program is shown on Figure 4.31. Note that only one ISR is represented for the encoders reading because the process is exactly the same for both (only the input pin changes).

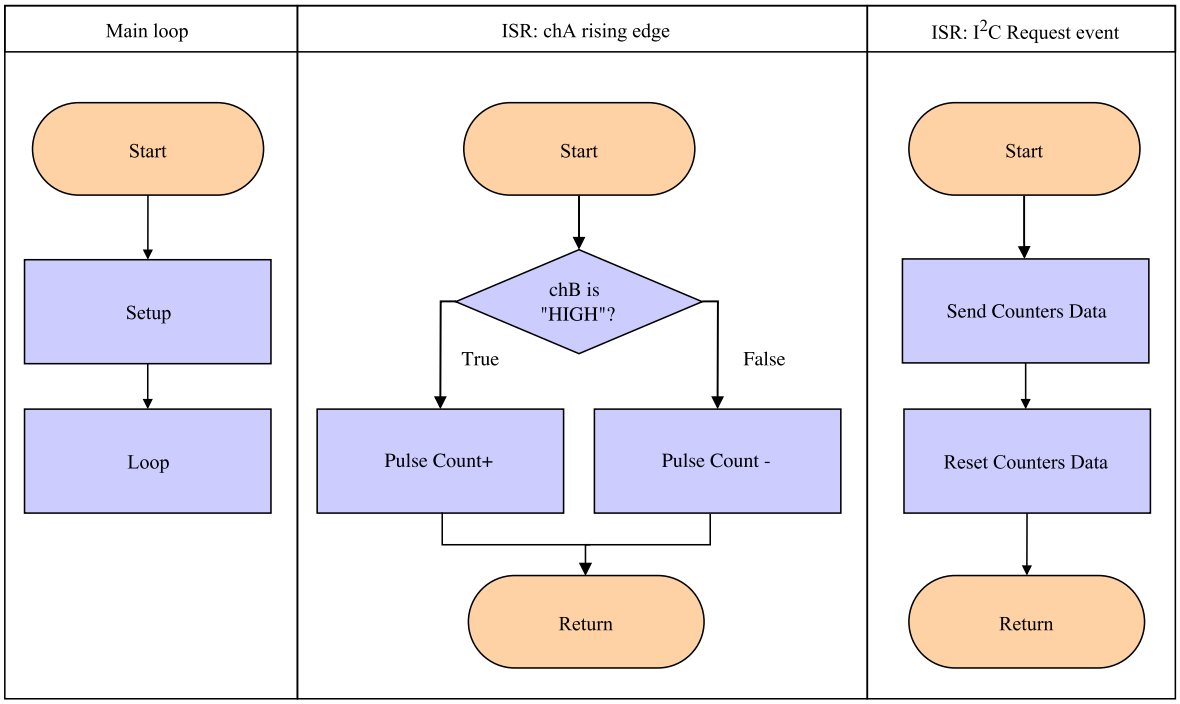



Figure 4.31: Flowchart from `CPU3` with interrupt service routines.

## 4.6 Control System

### 4.6.1 Generic Description

To close the control loop it was decided to use `CPU2` instead of `CPU1`. The main reason for this was to minimize the number of communications to perform the control. Figure 4.32 illustrates a classical PID control system (despite only one diagram being represented, note that two loops are implemented, one for each motor).

The reference is set in pulses/0.05s ($\frac{1}{F}$) by `CPU1`. After that, the actual rotation value, also with the same units as before, is required to the `CPU3` via $I^2C$ so that the error can be calculated. This error is then multiplied by a proportional, integral and derivative gains ($K_p, K_i$ and $K_d$, respectively) to calculate the necessary motor's output to reach the desired setpoint.

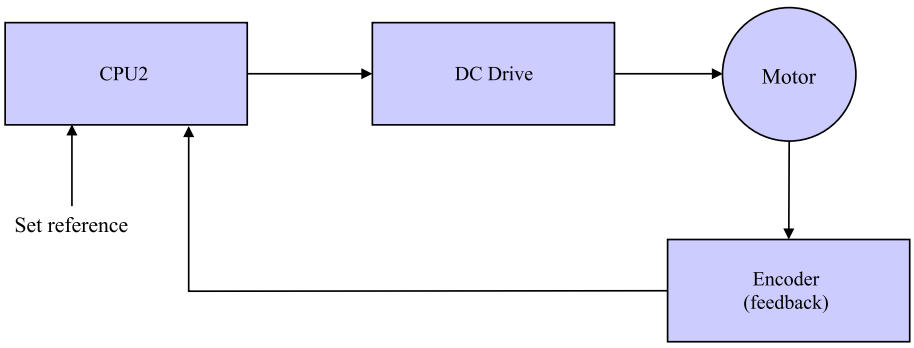



Figure 4.32: Implemented PID block diagram.

### 4.6.2 Tuning

Finding the optimal PID parameters is a meticulous task, nearly impossible without an accurate system's model. To determine the vehicle's transfer function it is necessary to model it considering all its numerous particularities, which is not feasible with the current time constrains and system's complexity.

Nevertheless, some approaches use heuristic methods, which intend to bypass these difficulties. Like the trial and error method, Ziegler and Nichols also defined commonly used approaches. One of them being the closed loop Ziegler-Nichols method, which consists of applying sustained oscillations to the system and relate them with a pre-determined parameter table [52]. However, this method forces the system into a condition of critical stability that may lead to an hazardous situation (due to the motors being constantly oscillating).

Despite of the Ziegler-Nichols method not being used, some of its guidelines were taken into consideration (like first tuning the proportional parameter) when tuning the system with an error and trial approach. To do it, a setup was created so all the important variables (encoders data and desired setpoints) could be outputted. Also, for safety reasons, the first tests were carried out with the lifted platform, minimizing possible damages on the initial tuning.

To monitor the variables, "**rqt_plot**" from ROS was used. This tool provides a GUI plugin to visualize numeric values in a 2D plot [53]. Figure 4.33 shows the created graphs alongside with the terminals to change each wheel's setpoint and each PID parameter.

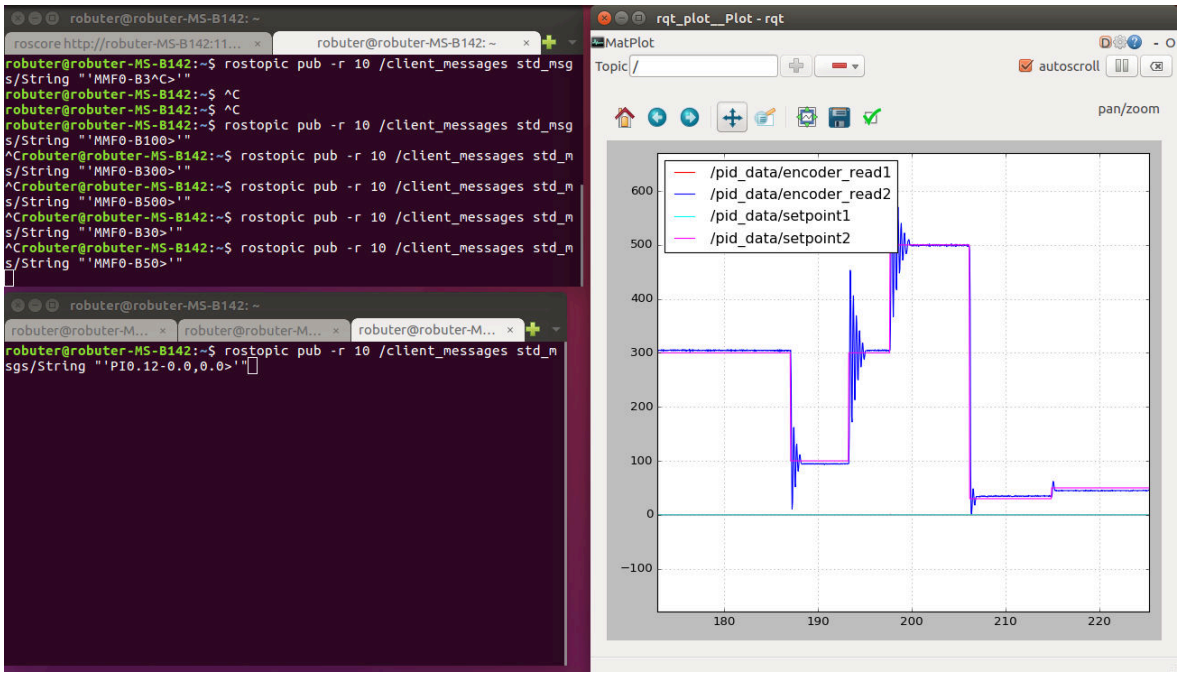



Figure 4.33: Used setup to tune the PID parameters. Rqt_plot tool on the right and terminals publishing PID values and setpoints to `CPU2` on the left.

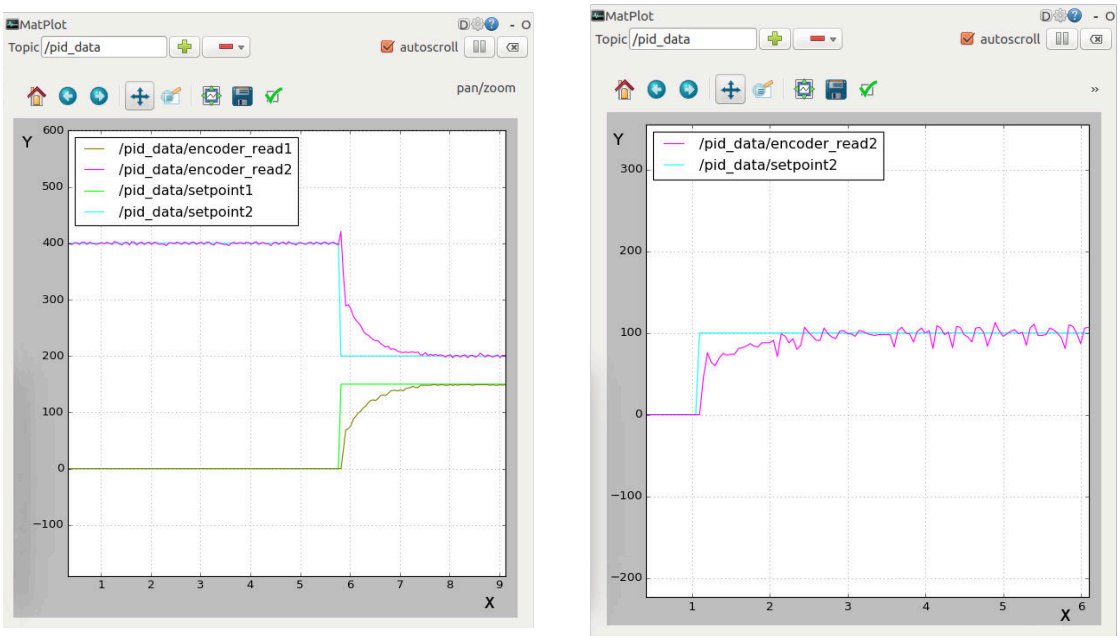
Also, it was decided to change the PID parameters while the program was running to avoid the code compilation each time a test had to be performed. Since the control loop is implemented in the `CPU2`, the PID parameters, from `CPU1` to `CPU2`, had to be transferred. This was made by creating a new type of message to `CPU2` process: "`PI`".

An example of this message can be "PI0.05-0.3,0.001>", where in red is the type of messages, in blue the separators of each parameter, in green the terminating character and in black the actual PID values (PI<Proportional>-<Integrative>,<Derivative>>). To publish the setpoints and PID messages to "**client_node**" a standard publication was used :

```
$ rostopic pub -r 10 /client_messages std_msgs/String "'MMF50-B70>'"

$ rostopic pub -r 10 /client_messages std_msgs/String "'PI0.12-0.0,0.0>'"
```

It is visible from Figure 4.33, where only proportional control is implemented, a high overshoot. This kind of response is not desirable for a system of this nature. So, at this stage, it was necessary to add new parameters, like the integrative and derivative components. The integrative intends to minimize the steady-state error, while the derivative tends to compensate the variations. With this in mind, different values were tested for these three variables. The main concern was to avoid the overshoot, even if the system is slower to respond (i.e. takes more time to reach the desired setpoint). Additionally, the available space to perform the tests was not the ideal, due to the LAR space constraints. In Figure 4.34 two examples of the system's response with the implemented PID values are represented.




(a) (b)

Figure 4.34: Final system's response with tuned PID parameters, where (a) and (b) represent different experiences. X axis — seconds; Y — pulses/0.05s.

An important note regarding the images is the rate of publication, "**-r**", which is set to **10** in the images. By the time these pictures were acquired, the rate was set at 10 Hz. This also had influence in the output velocity, which was half by comparing with the system running at 20 Hz (see equation (4.4)).

## 4.7 Remote Access

To access the platform from an external computer the OpenSSH is used, which is a free available version of the Secure Shell (SSH) protocol family of tools for remotely controlling, or transferring files between, computers [54].

When accessing the platform, the user must be connected to its network, "**Robonuc**" or "**Robonuc_5G**", and then run the following command:

```
$ ssh -X robuter@192.168.0.123
```

The user will then be prompted with a password, which is "**robuter**", to establish the connection. The argument "**-X**" is optional since it is only used to bring the graphic environment to the user's PC.

With this tool it is possible to fully access `CPU1`'s functionalities through command line, therefore operations like running nodes, power off the CPU and having visual feedback can be made remotely, without having anything connected through cables to the platform.

# Chapter 5

# Application Level

This chapter intends to demonstrate the versatility of the platform by implementing two navigation methods: autonomous and manual. An application with a remote controller is presented as well as the integration of a simple navigation algorithm based on line recognition.

## 5.1 Manual Navigation

### 5.1.1 Remote Controller

The platform can be operated manually with an Xbox remote controller (Figure 5.1). To perform this task, first it was necessary to install the controller's drivers [55] and then to write a teleoperation node. To get the joystick data published over ROS, "**joy_node**" must be launched. This node will publish the buttons state in "**/joy**" topic whenever any alteration occurs. Nevertheless, as referred in subsection 4.4.2, the velocity data must be received at a rate of 20 Hz, therefore this default had to be modified. In order to do it, the configuration parameter "**autorepeat_rate**" was set to **20** (value in Hz) when launching the "**joy_node**", which will guarantee the data publishing at the defined rate, even if the buttons' status does not change.

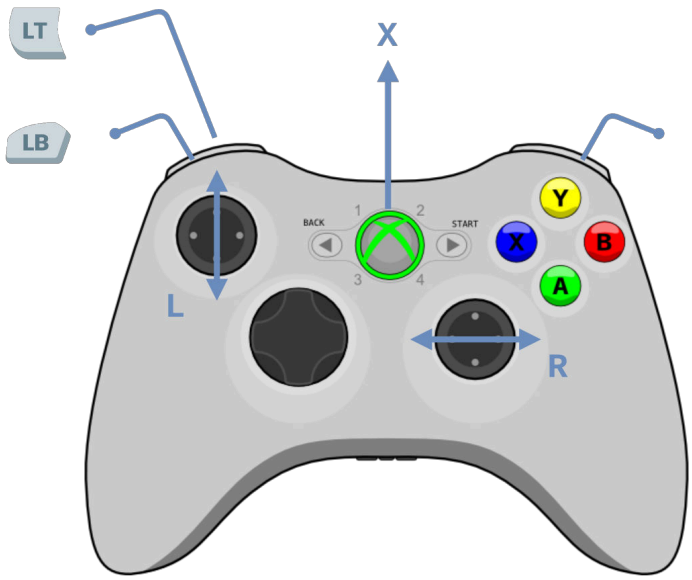



Figure 5.1: Remote controller's used buttons. Table 5.1 describes the buttons' funtions.

Table 5.1: Xbox remote buttons description.

| Button | Description |
|---|---|
| X | Power on remote controller by holding for 1s |
| L | Linear velocity, forward and backward on vertical axis |
| R | Angular velocity, left and right on horizontal axis |
| LB | Decrease velocity rate constant in 0.05 |
| RB | Increase velocity rate constant in 0.05 |
| LT | Dead-man switch |

Also, by operating the joystick and observing its response, it was detected that the axis values near the zero position had too much sensibility. So, to avoid moving the platform unintentionally, the "**deadzone**" parameter was set to **0.1**, which is the amount by which the joystick has to move before it is considered to be off-center.

The core node of the manual navigation is the "**r_teleop** node which is both a subscriber and publisher. It subscribes to the "**/joy**" topic and whenever a new message arrives it processes the income data from the controller. Depending on each buttons' status, a message containing the linear and angular velocity will be published in the "**/navi_commands**" topic.

Figure 5.1 identifies the used buttons to operate the platform, while Table 5.1 contains its description. An important thing to mention is the addition of a dead-man switch, which follows the industry's practices. Therefore, to validate and transfer any velocity to the platform, **LT** button must be pressed, otherwise the velocity will be set to zero.

### 5.1.2 Overview

In order to launch all the created nodes, a launch file was written (from **roslaunch** tool) — **robonuc_teleop.launch**. The **roslaunch** is a tool for easily launching multiple ROS nodes locally and remotely via SSH, as well as setting parameters on the Parameter Server [56]. Also, a file with parameters was created, in order to load them to the multiple files without having to compile the code.

Figure 5.2 was obtained by launching "**rqt_graph**", which provides a GUI for visualizing the currently active ROS nodes and topics. The image presents the information flow throughout the nodes and complements the previous description of each one, giving an overview of the implemented system.

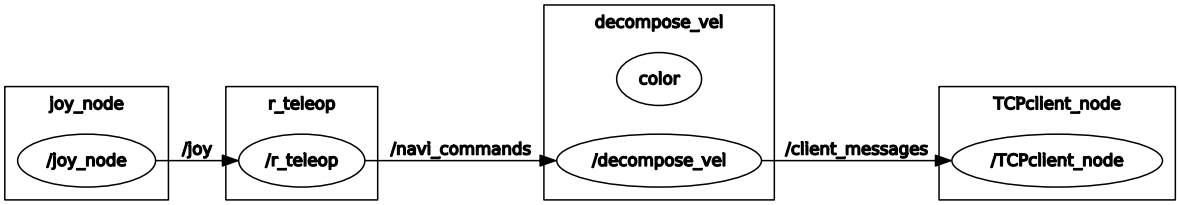



Figure 5.2: Nodes involved in the platform's teleoperation. This diagram was obtained with **rqt_graph** tool, where the inter-node communication, in topics, is represented.

## 5.2 Autonomous Navigation

A simple vision application to perform autonomous navigation was integrated in the platform. Again, this integration proves its capacity of enhancements and inclusion of add-ons, which were crucial guidelines taken into consideration while designing it.

### 5.2.1 Camera Calibration

To ensure an important condition on the system — constant camera orientation, a servo controlled pan and tilt unit (model PTU-D46-17 by FLIR [42]) was used. The PTU controller has a serial port interface, RS232, to receive commands and send status information. Communication was established via serial terminal in Ubuntu, with *gtkterm*, where three main commands were used:

- **R** — Performs reset calibration, the pan and tilt axis are set to its '0' position;
- **PP** or **TP** — to query the absolute pan and tilt axis' position, respectively;
- **PP<position>** or **TP<position>** — to set a desired pan and tilt axis' position, where <**position**> is the desired value.

The camera was connected and different configurations were tested to analyze the obtained views. In the end, the predefined position was set to: **PP0** and **TP-650**. Even though different methods (i.e. simpler and inexpensive) could be used for this purpose, the installation of this PTU is useful for future applications (e.g. searching algorithms), since it can be easily integrated in the ROS environment by using the manufacturer's available node and drives.

### 5.2.2 Image Acquisition

To acquire image, the "**r_pointgrey_FL3_28S4**" node was used. This code, originally developed by Marcelo Pereira [57] and David Silva [58], was integrated by including its source code and dependencies in the ROS environment, i.e., *packages* and *CMakeLists* (see Appendix K, where a platform's operation guide is presented, for more details).

This node is responsible for establishing communication with the camera while configuring some of its properties (i.e. image size) and acquiring its image. After the acquisition, the image is then published in the topic "**\RawImage**" where can be subscribed by different nodes.

### 5.2.3 Image Processing

Since the purpose of the current document is mainly to detail the integration of the navigation algorithm instead of its development and experimental results, it will only the used approach is presented.

The image processing node, "**r_image_process**" is both a subscriber and a publisher. It subscribes to the topic "**\RawImage**" to receive the image for further processing (in mono8 format) and afterwards, it publishes the necessary velocity (linear and angular) in the topic "**\navi_commands**" to ensure the line following.

The process is triggered when an image is loaded. Subsequently, the first step is to crop it, which will reduce the amount of information to be processed while eliminating sources of potential error in areas where no useful information (for this algorithm purpose) is contained.

Image is loaded with 964x724 pixels and after the resize is 700x300 pixels.

The diagram from Figure 5.3 represents the major steps of the implemented computer vision algorithm to obtain $\Delta\theta$ and $\Delta X$, which are the angular difference from the reference and the offset of the detected line, respectively. The first used vision algorithm from the OpenCV library was an adaptive threshold operation, which calculates the threshold for small regions of the image, allowing to surpass the illumination variation issue. This operation alongside with a morphological operation, erosion, are performed in the block "Image Filtering", from Figure 5.3.

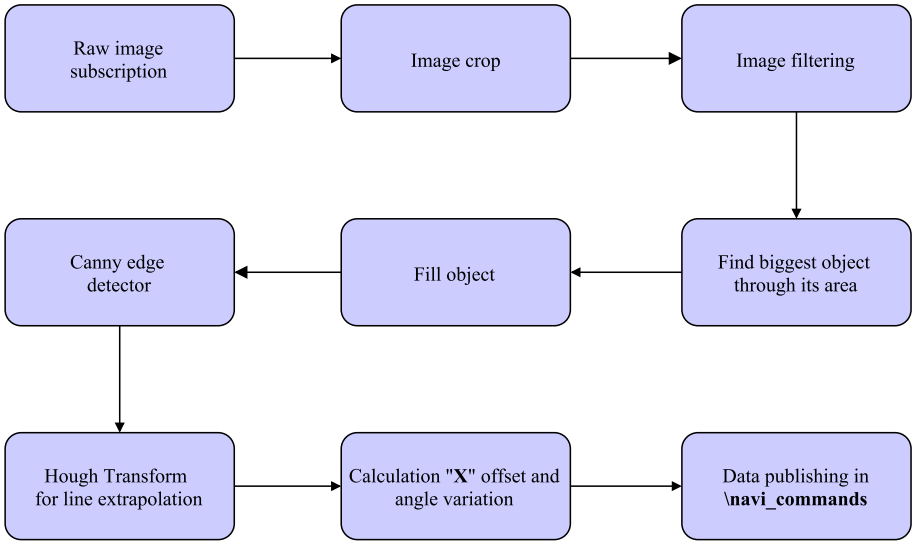



Figure 5.3: Computer vision algorithm applied to detect line and determine $\Delta\theta$ and $\Delta X$.

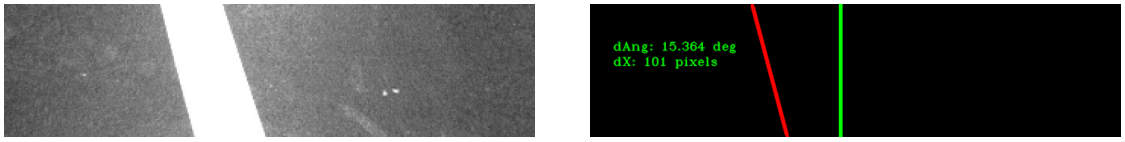
In the end, linear and angular velocities are published in the "**\navi_commands**" topic. The linear velocity is set to constant ($L_{const} = 0.06$) while the angular velocity is calculated with equation (5.1). In the expression, the constants values were determined empirically, by observing the platform's behavior. Besides that, each one of them can be modified in the "*default.yaml*" file.

$$\omega_{robot} = 0.08\Delta\theta - 0.001\Delta X \tag{5.1}$$

The system's response can be reduced to three simple actions:

- $\omega_{robot} < 0$ — system turns right;
- $\omega_{robot} > 0$ — system turns left;
- $\omega_{robot} = 0$ — system moves forward, zero angular velocity.

For better comprehension, a resized raw image is presented alongside with the final processed image, Figure 5.4.




(a) (b)

Figure 5.4: Raw image (a) and final output image (b) from a "turning left" situation. The $\Delta\theta$ and $\Delta X$ values can be observed. The green line represents the reference for these values and the red line one side of the line.

An important note regarding the image process is its processing rate. The 20 Hz, predefined for the control loop, are not achieved while processing and acquiring image (Flea3 camera only acquired images at a maximum rate of 15 Hz). So, for that reason, the velocity is published at a slower rate which did not represent a problem on the system's behavior.

### 5.2.4 Overview

In order to launch all the created nodes for the autonomous navigation, a launch file was created — **robonuc_aut_navi.launch**. Figure 5.5 was obtained by launching "**rqt_graph**" and presents the information flow throughout the nodes to give an overview of the implemented autonomous navigation system.

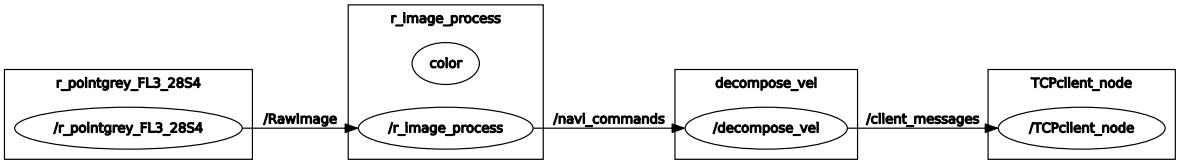



Figure 5.5: Vision system nodes diagram obtained by **rqt_graph** where the inter-node communication, in topics, is represented.

# Chapter 6

# Tests and Results

This chapter presents some platform tests and results, along with the created infrastructures to perform them, and finally shows the integration with a robotic manipulator.

## 6.1 Assessment of Trajectory Execution

The purpose of the following tests is to assess the quality of open loop path execution.

### 6.1.1 Straight Line

To carry out the straight line tests, a ROS node was developed, "**r_exp_results**", alongside with a launch file for starting the experiments, "**experience.launch**". For a similar reason presented on Sub-section 5.2.3, the designed code will only be reported with a flow diagram (Figure 6.1) and a brief explanation to obtain the final image (Figure 6.3). Since limited vision algorithms were applied, a crucial condition to perform the tests is a light controlled environment. A 6 meter line was placed on the floor with the guidance of a stretched rope to serve as straight path reference (Figure 6.2). Afterwards, the joystick was used to actuate the platform without the angular velocity's component, applying a same speed message for both wheels.

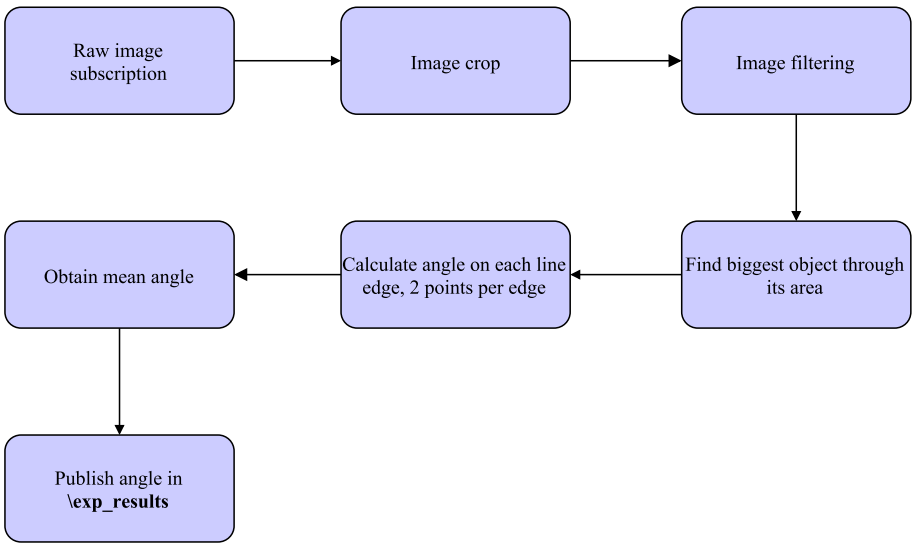



Figure 6.1: Image processing algorithm to extract platform's angle in relation to a straight line on the floor.

Figure 6.2: Straight line used as reference at LAR.

Despite the fact that the created infrastructure is theoretically adequate for the problem, some sources of error were verified, such as in the initial platform alignment with the line and the floor irregularities, which conditioned the tests reproducibility. Therefore, many tests had to be performed to obtain some repeatability.

To start the experiences some common initial conditions had to be ensured:

- Caster wheels turned for the navigation direction;
- Platform centered with the floor's guideline;
- Platform with the same orientation as the floor's guideline;
- All nodes from "**experience.launch**" file running.

The first test was conducted at a constant speed (5 cm/s), in the forward direction (Figure 6.4). The graph was obtained with **rqt_plot** tool, where the vertical axis represents $\Delta\theta$, in degrees, and the horizontal axis the sample number, which is published by the "**r_exp_node**", at the rate of its processing, in the "**experimental_results**" topic.

Since the tests were performed at low speed, even if the values are given with slight variable rates it is not significant for the results evaluation, since the objective is to evaluate the whole path. Another important note is the presented graphs starting point, which is not relevant, since the **rqt_plot** sometimes had to be used during a long period of time. This explanation can be extrapolated to all the graphs in the current sub-section.

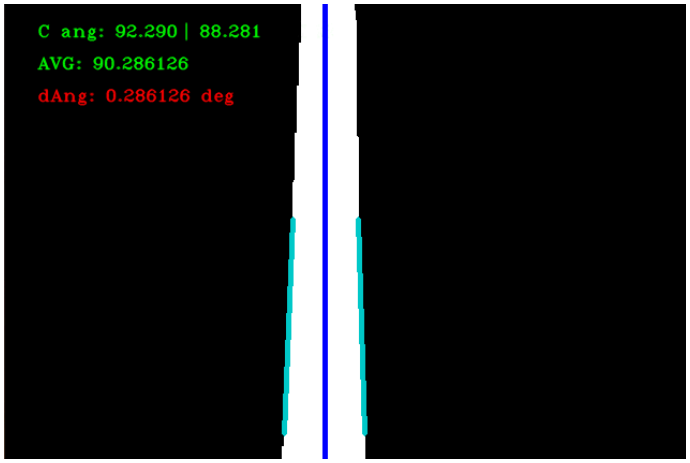



Figure 6.3: Image processing result to measure angle.

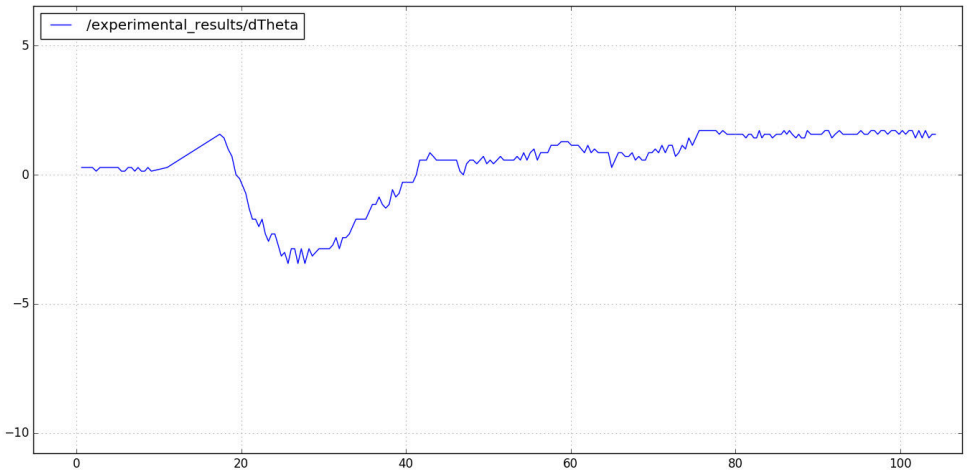



Figure 6.4: Straight line path in forward direction with constant speed. Vertical axis in degrees, horizontal in seconds.

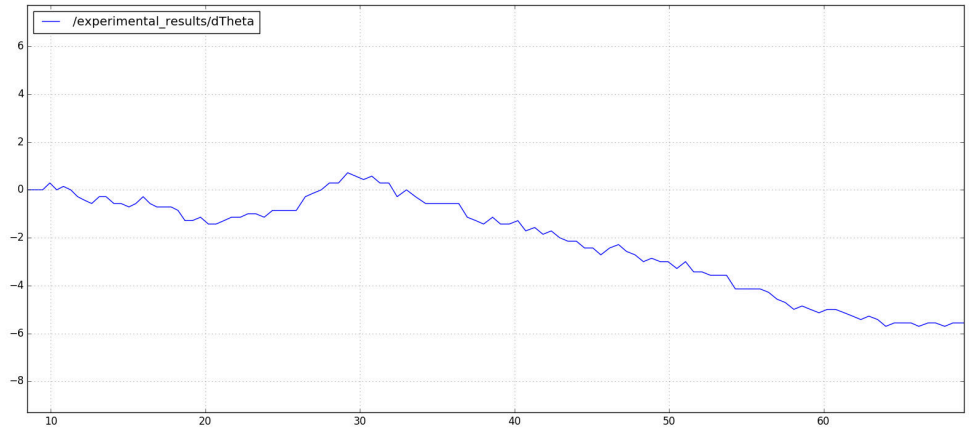



Figure 6.7: Straight line path in backward direction with variable speed.Vertical axis in degrees, horizontal in seconds.

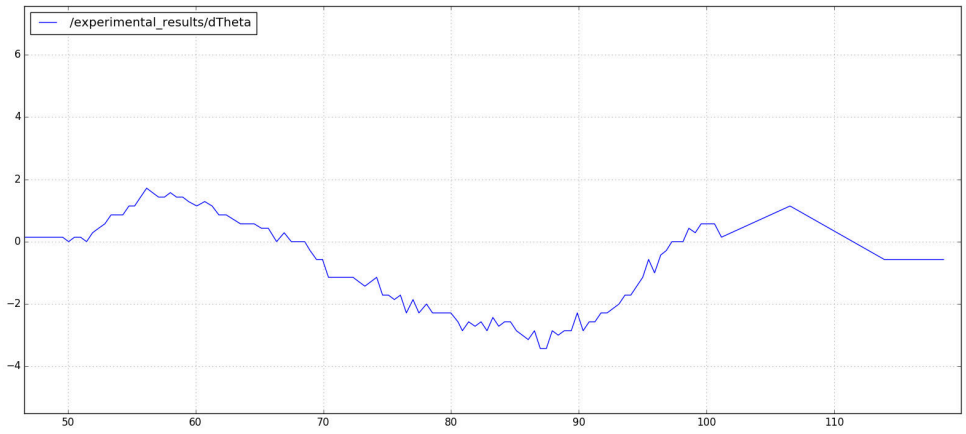



Figure 6.5: Straight line path in forward direction with variable speed.Vertical axis in degrees, horizontal in seconds.

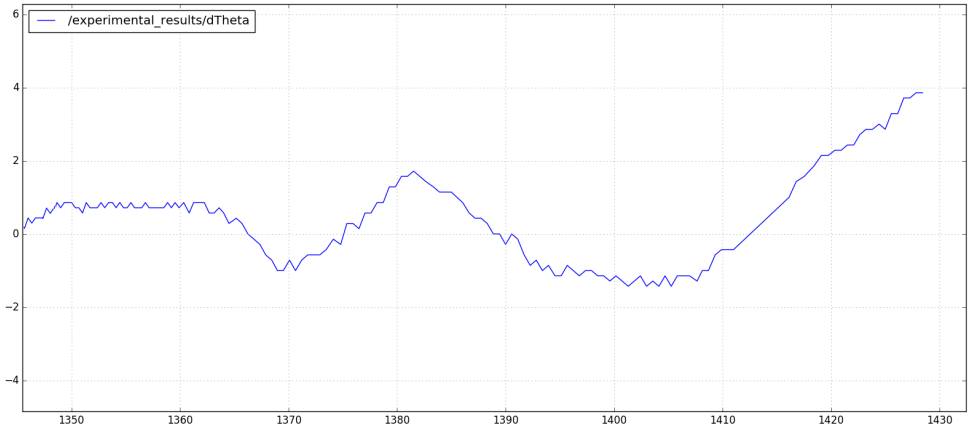
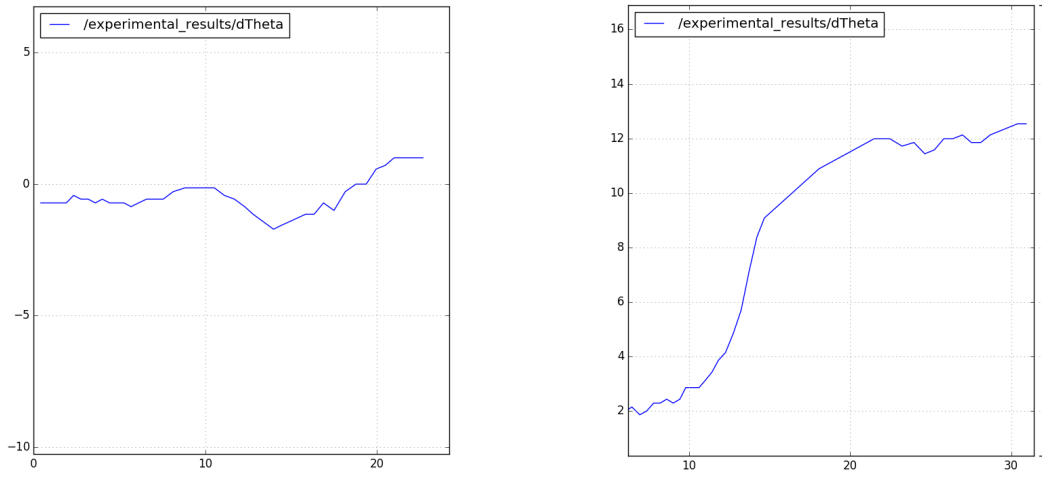



Figure 6.6: Straight line path in backward direction with constant speed.Vertical axis in degrees, horizontal in seconds.

By the above graphs (Figures 6.4, 6.5, 6.6 and 6.7), it is possible to conclude that the system presents a better response on the forward direction (maximum error around 4 degrees) than in the backward one (maximum error almost reaches the 6 degrees). This is due to the system's dynamics: while rotating forward, it is easier to define a movement, since the motored wheels are nearer the robot's front. This also validates the decision of the front and rear of the platform, mentioned in Sub-section 4.1.1. Also, no significant differences were observed by varying the speed during the path, showing the system's capacity of responding to them.

Also, a different kind of test was performed, with the caster wheels not aligned with the navigation direction (Figure 6.8). For this purpose, only the initial behavior was plotted, and as expected, unpredictable directions were visible (even small angle variations were observed). This depends on different factors, such as the initial caster wheel's position alongside with the platform's navigation direction and the floor irregularities




(a) (b)

Figure 6.8: Caster wheels not aligned in the forward (a) and backward (b) direction. Vertical axis in degrees, horizontal in seconds.

### 6.1.2 Circle

To carry a circular trajectory analysis a script was designed (diagram showing major steps is shown in Figure 6.10). These tests, alongside with the straight line tests, allows to evaluate the platform's response for different kinds of movement.

To obtain the images, a standard video camera was used. The camera was placed near the ceiling, parallel to the floor and a video was recorded while the platform was rotating with symmetric wheel speed values. To actuate the motors, the remote controller was used: only with angular velocity by using the joystick's "L" button, see Figure 5.1 for more details). Figure 6.9 shows an example of two acquired images during this rotation, where the objective is to detect and plot the centroid of the red emergency button (Matlab was used for that purpose). The objective of this test was to evaluate the robot's capacity of performing a circular trajectory, rotating upon its axis.

In order to separate and numerate the frames for further Matlab processing, "FFmpeg" tool was used. This tool is a multimedia framework, able to decode, encode, transcode, mux, demux, stream, filter and play any video or image format [59]. It was used by installing it in Ubuntu and running a command with the following syntax:

```
$ ffmpeg -ss 20 -i input.mp4 -t 35 -vf fps=24 out%d.png
```

In the previous command, the option "**-ss**" defines the starting point (in seconds), the "**-t**" the duration (in seconds) and "**fps**" the frame rate (in Hz). So, for each experience, 840 frames were extracted from the recorded video and then processed.

(a) (b)

Figure 6.9: Aquired images from the top view in two different positions (a) and (b) while the robot was rotating around its wheel center point.

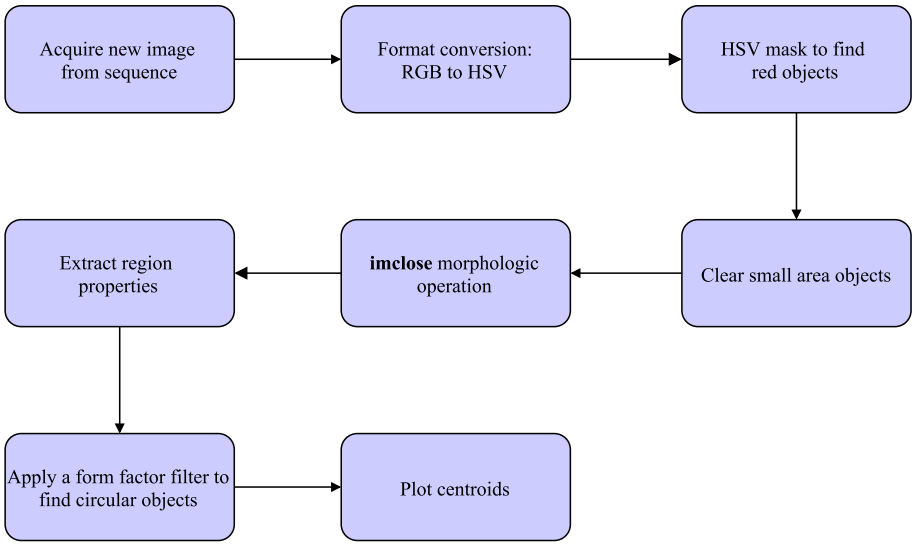



Figure 6.10: Image processing algorithm to extract and plot centroid points of the emergency button used as marker for easy detection in the images.

Figures 6.11 and 6.12 illustrate the results of a circular trajectory, where an accurate circumference was obtained in both scenarios (rotating left and right). Some points outside the circumference are observed in both experiences, which can be considered as outliers since the trajectory was kept.

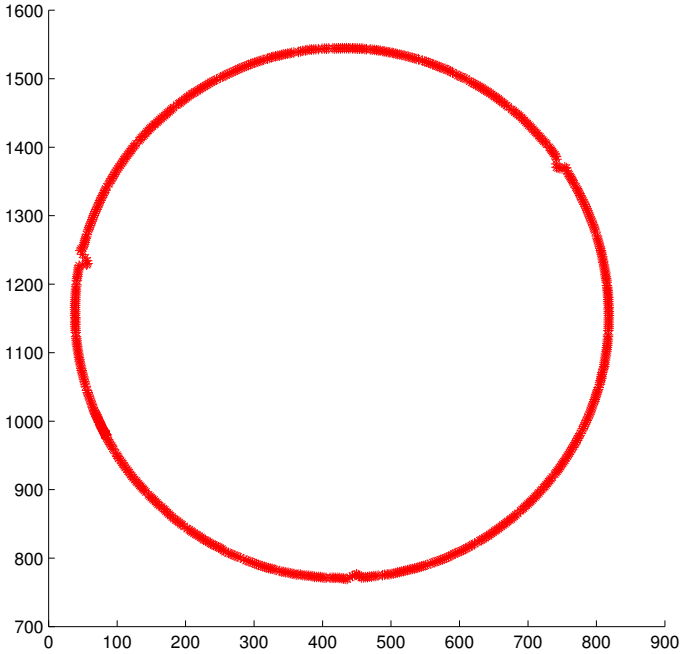



Figure 6.11: Plot of a robot point (the detected object's centroid) during rotation.. This experience was performed with a rotation in the counter-clockwise direction. Graph in pixels.

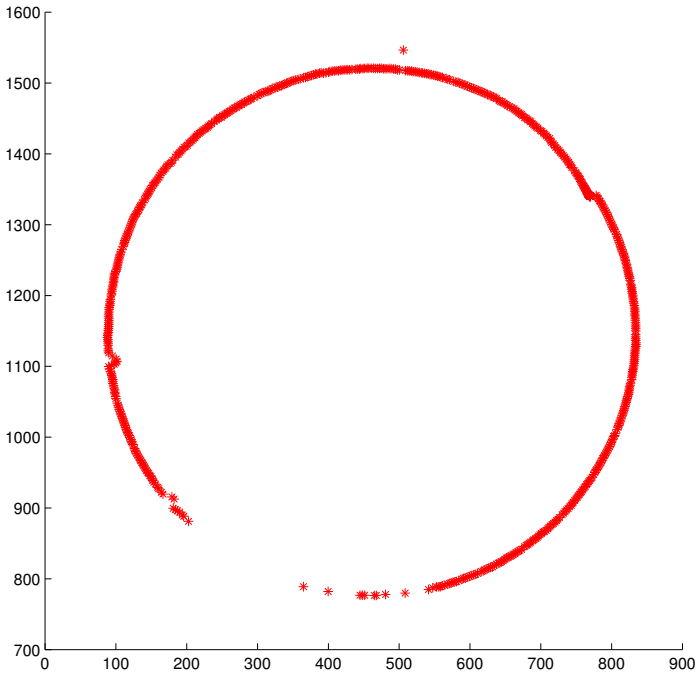



Figure 6.12: Plot of a robot point (the detected object's centroid) during rotation. This experience was performed with a rotation in the clockwise direction. Graph in pixels.

It is visible on the clockwise rotation (Figure 6.12) sense an absence of points during a part of the experiment. This was due to light conditions (i.e. shadows were observed) that

altered the HSV values. In spite of that absence, a circular trajectory can be observed in both scenarios, validating the experimental results.

## 6.2 Velocity

For safety reasons the maximum velocity was limited to 0.4 m/s. This was implemented on low level (CPU2) by truncating the PWM values and on the software level by sending a maximum pulse value for each wheel of 200 (which corresponds to 0.4 m/s). Although higher speeds can be achieved, it is necessary to perform tests in wider areas to accurately determine the maximum velocity maximum.

Another detected problem was the encoder's response on high velocities. Measuring the signal in an oscilloscope was verified that, at higher velocities (such as 0.6 m/s or higher) the signal stopped being a square wave, presenting irregular behavior. Despite of several attempts, no plausible reason or solution was found, once the signal at lower velocities was the expected from an incremental encoder.

## 6.3 Integration With Robotic Manipulator

One of the most relevant results was the capacity of integrating a robotic manipulator on the platform's architecture, both on the physical and hardware/software architectures. This work was carried out by Vitor Augusto Silva [25] in the final stage of this dissertation.

The used manipulator was the Fanuc LR Mate 200iD and the assembly can be observed in Figure 6.13. This reinforces the platform's versatility, by including an industrial equipment on it. At this stage, tests and code design are being carried out to integrate both works in order to perform simple mobile manipulation demonstrations. Despite of that, in this document that will not be presented.

## 6.4 Retrofitted Robonuc

The final result can be observed on Figures 6.14, 6.15 and 6.16, where different views are presented.

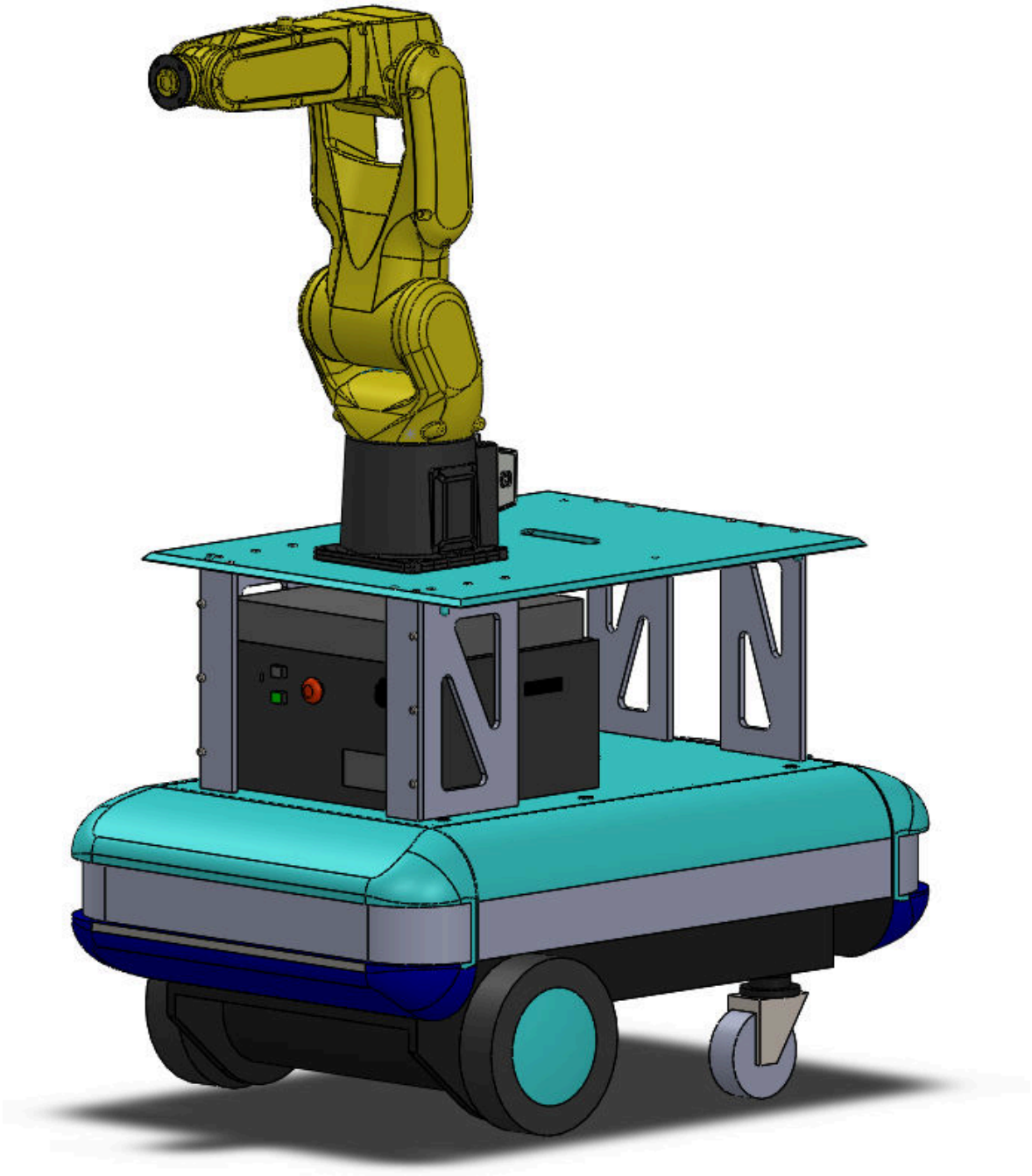


Figure 6.13: Complete platform draw. Adapted from [25]


Figure 6.14: Complete platform after the retrofitting. Perspective view.

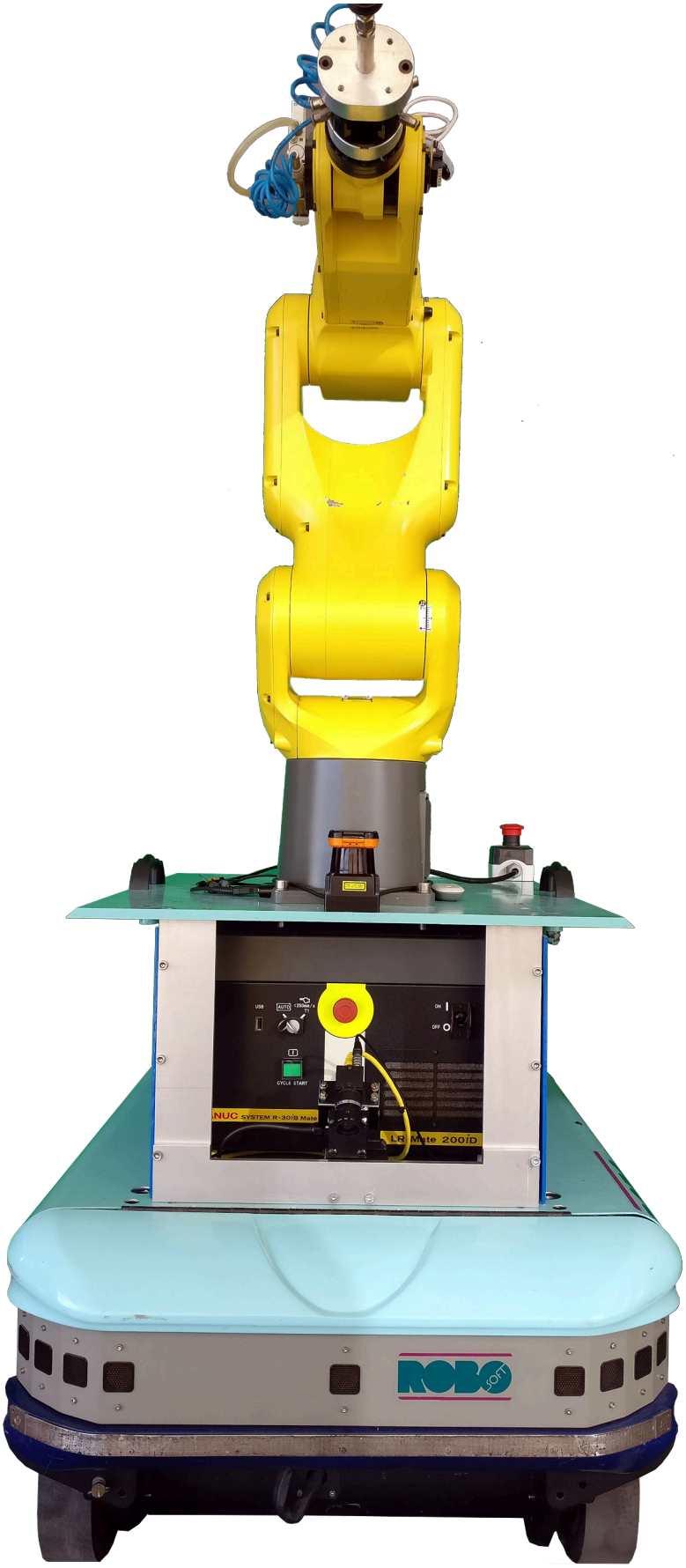



Figure 6.15: Complete platform after the retrofitting. Front view.

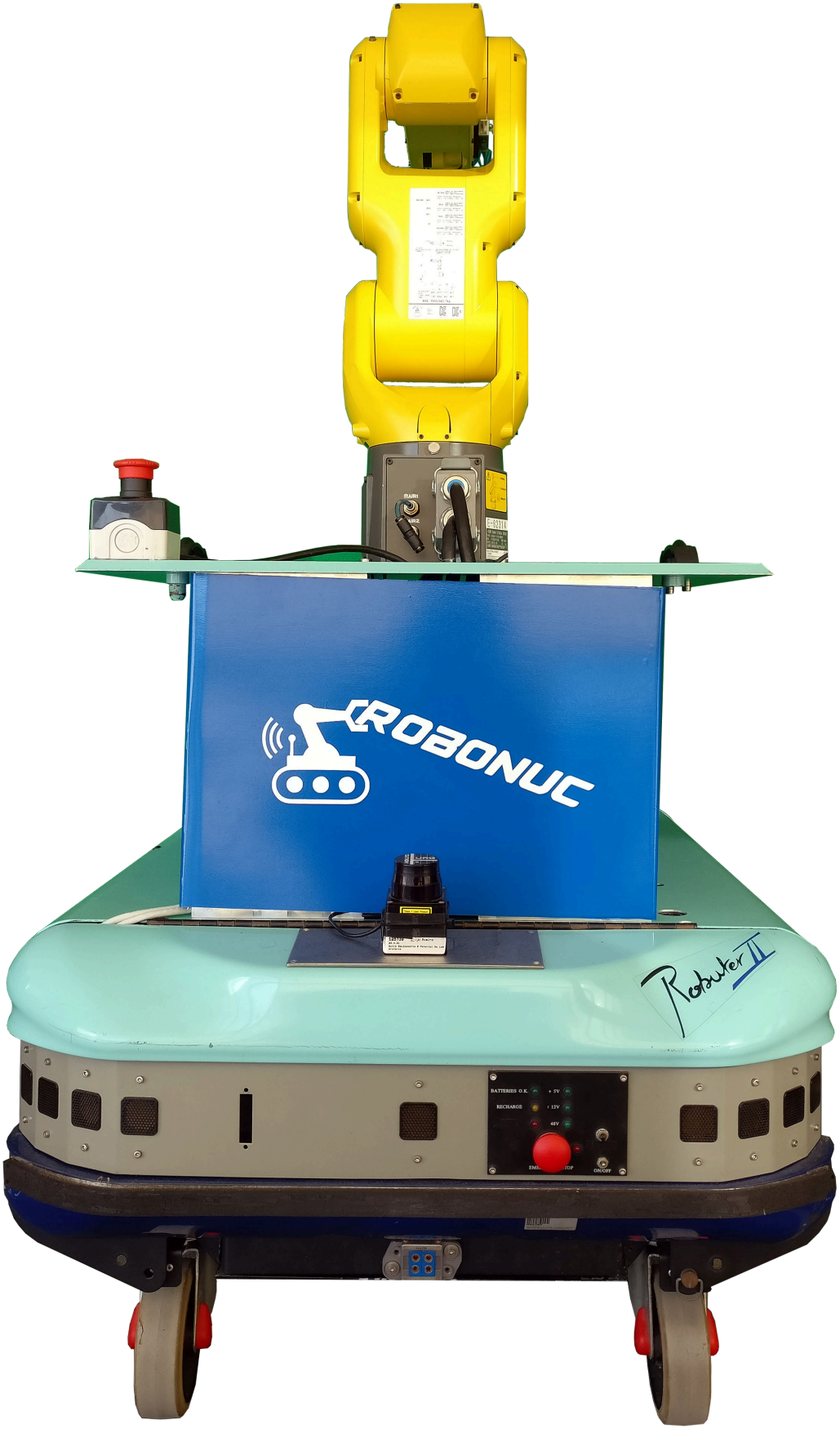



Figure 6.16: Complete platform after the retrofitting. Rear view.

# Chapter 7

# Conclusions and Future Work

## 7.1 Conclusions

The main goal of this project was the Robuter's retrofitting, hence it can be said that it was successfully achieved. Several features were enhanced (presented in Table 7.1). The computational power was significantly increased and it is now distributed, which is commonly used in modern robot architectures.

Also, the setup of a network within the robot (wired and wireless) offers an easy way of integrating different kinds of equipment while allowing remote access to it.

Table 7.1: Comparison of the old Robuter with the retrofitted. Only the modified features are represented.

| **Feature** | **Old Robuter** | **Retrofitted Robuter** |
|---|---|---|
| Batteries capacity | 2.88 kWh | 3.6 kWh |
| Sensing | Network of ultrasonic sensors | Camera Unit |
| CPU | VME bus with 5 slots, Motorola 68020, 16 MHz | Mini-PC MSI Cubi 2 i5-7200U<br>Arduino Leonardo ETH<br>Arduino Micro |
| Operating System | ALBATROS | Linux OS with ROS running |
| Programming | Software package and C application programs | C++ and Python |

During the development of this project, a guideline was followed: since the project is about retrofitting, the maximum number of components should be recovered and used, but like every philosophy, this method had advantages and disadvantages. For instance, the DC Drives that were repaired and used — a considerable amount of money was saved by integrating them in the platform (see section 4.2) but the time consumed to do it can only be justified and accepted in an academic project (in the industrial sector a new equipment should be quickly acquired). During this stage it was difficult to manage the compromise between time and money, which is an important skill to improve.

Still, it is important to mention, that the budget spent to retrofit the original platform (that 20 years ago costed around 35.000€) was around 1300€. Naturally, this budget does not include the used equipment from the previous platform (structure, motors, etc) nor the available at the laboratory (camera and PTU). Despite of that fact, it is important to mentioned it, because a big intervention took place and a low-cost solution (proportionally to the platform's original value) was achieved.

A distributed and ROS-based architecture made the development of the solution easier by facing each challenge individually. This is a fundamental premise since this project can be considered the base for further developments and therefore must be prepared to integrate them. The potential and growing of the ROS community was verified several times during the realization of this project, where several tools and nodes could be used from it, revealing to be a great environment for high level developers and researchers.

To accomplish this project, it was necessary to combine and develop skills from a wide range of areas. It was carried out an engineering work where the approach was from the design of the solution to the contact with companies in order to get the necessary material to be assembled. This contributed significantly to the improvement of multiple skills.

## 7.2 Future Work

Future work may include the design and installation of an electric switchboard to place on the top of the platform. It is a safe and proper solution to connect additional equipment without having to disassemble anything to do it.

To prevent noise interference or a possible faulty/loose contacts in the developed circuitry, it is suggested the design of PCB boards to replace the circuits mounted on prototype boards. Those PCB's can also have room for connectors, in order to increase the robustness of the electrical installation

From the software architecture point of view, in ROS, it is strongly recommended to evaluate the integration of the "**actionlib**" package on the platform. The main advantage of its implementation is the possibility of canceling a request during its execution or get periodic feedback about how the request is progressing [60].

To make use of the on-board computational power, the development of more sophisticated navigation algorithms or the integration of past projects developed at the laboratory is proposed. After the development is complete, the integration of these algorithms should not be a problem due to the ROS architecture.

Another important work to be developed is the inclusion of an odometry node. This feature alongside with a laboratory's navigation map would allow referencing the robot to determine its position, which could expand its use and applications.

With the created infrastructures (i.e. nodes and scripts) for obtaining experimental results (for instance, PID tuning, straight line and circular trajectory), it is recommended to carry out additional tests. They should be performed in location with large enough area to avoid any kind of constrains, thus maximizing the results.

The collected data should then be analyzed so an intervention to improve the platform can take place. In this way, it will be possible to tune the platform's parameters more precisely. Also, to evaluate the trajectories' precision, it is suggested to find a method of granting more precision in the platform's positioning/alignment with the reference line or to use a marker (e.g. chalk) that can print on the floor/sheet the traveled path for further analysis.

At last, the integration of a GUI is seen as a good enhancement to the platform's functionality and monitoring. This addition makes the robot more attractive and user friendly, while providing important information and enabling a more intuitive operation.

# References


[1] Xiaomin Li et al. "A review of industrial wireless networks in the context of industry 4.0". In: *Wireless networks* 23.1 (2017), pp. 23–41.

[2] *Forklift Safety: Facts, Stats and Tips for Safe Operation [Infographic]*. July 2015. URL: https://www.oshasafetymanagement.com/blog/forklift-safety-infographic/ (visited on 12/14/2016).

[3] "Eletrical Engineering". In: *Indoor Electronic Train Pulls 10-Ton Load* 77.7 (July 1958), p. 664.

[4] *History of AGVS*. Aug. 2012. URL: http://www.agvsystems.com/history-agvs/ (visited on 12/01/2016).

[5] Christine Lagorio-Chafkin. *Automated Guided Vehicles: Behind the Swift Business of a Heavy Industry*. Mar. 2014. URL: https://www.inc.com/christine-lagorio/best-industries/automated-guided-vehicles.html (visited on 11/12/2016).

[6] *YARP: What exactly is YARP?* URL: http://www.yarp.it/what_is_yarp.html (visited on 04/11/2017).

[7] *YARP: The YARP OS library*. URL: http://www.yarp.it/yarp_os.html (visited on 06/11/2017).

[8] *YARP: Getting Started with YARP Ports*. URL: http://www.yarp.it/note_ports.html (visited on 06/11/2017).

[9] *The Orocos Component Builder's Manual*. URL: https://people.mech.kuleuven.be/~orocos/pub/documentation/rtt/current/doc-xml/orocos-components-manual.html (visited on 06/11/2017).

[10] *Robotics Developer Studio Overview*. URL: https://msdn.microsoft.com/en-us/library/bb483024.aspx (visited on 06/11/2017).

[11] Aaron Romero. *ROS/Concepts - ROS Wiki*. June 2014. URL: http://wiki.ros.org/ROS/Concepts (visited on 01/12/2017).

[12] Johann Borenstein, H. R. Everett, and Liqiang Feng. *Navigating Mobile Robots: Systems and Techniques*. Natick, MA, USA: A. K. Peters, Ltd., 1996.

[13] Moshe Kam, Xiaoxun Zhu, and Paul Kalata. "Sensor fusion for mobile robot navigation". In: *Proceedings of the IEEE* 85.1 (1997), pp. 108–119.

[14] João Sena Esteves, Adriano Carvalho, and Carlos Couto. "Generalized geometric triangulation algorithm for mobile robot absolute self-localization". In: *Industrial Electronics, 2003. ISIE'03. 2003 IEEE International Symposium on*. Vol. 1. IEEE. 2003, pp. 346–351.

[15] Emmanuel Lomba and Mário Alves. *On the hardware and software architecture of the robuter mobile platform: a hands-on approach*. Tech. rep. CISTER-Research Centre in Realtime and Embedded Computing Systems, 2005. URL: http://www.cister.isep.ipp.pt/docs/on_the_hardware_and_software_architecture_of_the_robuter_mobile_platform__a_hands_on_approach/230/ (visited on 11/30/2016).

[16] Dupourque. *Robuter*. 1988. URL: https://commons.wikimedia.org/wiki/File:Robuter88_1.jpg (visited on 12/06/2016).

[17] *RTAI - the RealTime Application Interface for Linux*. URL: https://www.rtai.org/?Homepage&id=38 (visited on 11/30/2016).

[18] Alfredo Manuel de Oliveira Martins. "Modelização e controlo de um veículo autónomo terrestre". In: (1995).

[19] *B2B service robots Robosoft*. Sept. 2007. URL: http://www.lokarria.com/corporate/museum.html (visited on 10/30/2016).

[20] P. Pomiers et al. "Trajectory tracking strategies in service robotics applications". In: *Advanced Motion Control, 1998. AMC '98-Coimbra., 1998 5th International Workshop on*. 1998, pp. 328–333. DOI: 10.1109/AMC.1998.743558.

[21] Alex Weiser and Pedro Lima. "An Integrated Learning, Planning and Reacting Algorithm Applied to a Real Mobile Robot". In: 1996. URL: https://www.researchgate.net/profile/Pedro_Lima4/publication/2718966_An_Integrated_Learning_Planning_and_Reacting_Algorithm_Applied_to_a_Real_Mobile_Robot/links/559d066f08aeefefa1b8329e.pdf (visited on 01/13/2017).

[22] Giuseppina C Gini and Alberto Marchi. "Indoor robot navigation with single camera vision." In: *PRIS* 2 (2002), pp. 67–76.

[23] Abdelfetah Hentout, Mohamed Ayoub MESSOUS, and Brahim BOUZOUIA. "Multi-agent control approach for autonomous mobile manipulators: simulation results on RobuTER/ULM". In: *IFAC Proceedings Volumes* 47.3 (2014), pp. 8503–8508.

[24] Caetano Filipe Costa de Noronha Ferreira. "Autonomous navigation and multi-sensorial real-time localization for a mobile robot". PhD thesis. Universidade de Aveiro, 2008. URL: https://ria.ua.pt/handle/10773/2468 (visited on 02/23/2017).

[25] Vitor Augusto Silva. "Integração de Manipulador Fanuc na Plataforma Robuter para Manipulação Móvel". MA thesis. Universidade de Aveiro, 2017.

[26] J. Borges Sousa et al. "On the design and implementation of a mobile robotic system". In: *Intelligent Control, 1995., Proceedings of the 1995 IEEE International Symposium on*. IEEE, 1995, pp. 617–622. URL: http://ieeexplore.ieee.org/abstract/document/525123/ (visited on 05/04/2017).

[27] Ewald von Puttkamer and Uwe R. Zimmer. "An Operating-System under hard Realtime-Constraints". In: *Real-Time Magazine* 5 (). URL: ftp://ftp.fhg.de/archive/gmd/ai-research/Publications/1991/Zimmer.91.albatross.pdf (visited on 05/04/2017).

[28] S Samuel Abhishek and K. Veladri. "Trajectory Planning of a Mobile Robot". In: July 2016, p. 8.

[29] Alessandro De Luca, Giuseppe Oriolo, and Marilena Vendittelli. "Control of wheeled mobile robots: An experimental overview". In: *Ramsete*. Springer, 2001, pp. 181–226. URL: http://link.springer.com/chapter/10.1007/3-540-45000-9_8 (visited on 05/29/2017).

[30] N. Muskinja, B. Tovornik, and M. Terbuc. *Use of TCP/IP protocol in industrial environment*. Vol. 2. 2003, 896–900 Vol.2.

[31] Copley Controls. *300 Series Amplifier*. URL: http://www.copleycontrols.com/Motion/pdf/300Manual.pdf (visited on 12/08/2016).

[32] Power-One. *100 Watt DC-DC Converters Series Q*. (Visited on 12/06/2016).

[33] Mean Well. *100W Single Output DC-DC Converter*. URL: http://www2.mouser.com/ProductDetail/Mean-Well/SD-100C-24/?qs=7vGxPDjevFMtpqB6ytqsVQ%3D%3D (visited on 05/14/2016).

[34] Battery University. *BU-107: Comparison Table of Secondary Batteries*. URL: http://batteryuniversity.com/learn/article/secondary_batteries (visited on 01/22/2017).

[35] Battery University. *BU-206: Lithium-polymer: Substance or Hype?* URL: http://batteryuniversity.com/learn/article/the_li_polymer_battery_substance_or_hype (visited on 01/23/2017).

[36] Suratsawadee Anuphappharadorn et al. "Comparison the Economic Analysis of the Battery between Lithium-ion and Lead-acid in PV Stand-alone Application". en. In: *Energy Procedia* 56 (2014), pp. 352–358. ISSN: 18766102. DOI: 10.1016/j.egypro.2014.07.167. URL: http://linkinghub.elsevier.com/retrieve/pii/S1876610214010297 (visited on 05/22/2017).

[37] Ultracell. *UCG75-12*. URL: http://ultracell.co.uk/products/ucg-batteries/12v (visited on 02/23/2017).

[38] MSI. *Cubi 2 Desktop*. URL: https://www.msi.com/Desktop/Cubi-2.html (visited on 05/25/2017).

[39] Arduino. *ArduinoBoardLeonardoEth*. URL: https://www.arduino.cc/en/Main/ArduinoBoardLeonardoEth (visited on 12/10/2016).

[40] Arduino. *ArduinoBoardMicro*. URL: https://www.arduino.cc/en/Main/arduinoBoardMicro (visited on 03/01/2017).

[41] *Flea3 2.8 MP Color GigE Vision (Sony ICX687)*. URL: https://www.ptgrey.com/flea3-28-mp-color-gige-vision-sony-icx687-camera (visited on 02/22/2017).

[42] DPerception. *PAN-TILT UNIT Model PTU-D46*. URL: http://www.flir.com/uploadedFiles/Directed_Perception/Documents/Products/PTU-D46/PTU-manual-d46.pdf (visited on 04/27/2017).

[43] ASUS. *RT-AC51U | Redes*. URL: https://www.asus.com/pt/Networking/RTAC51U/ (visited on 05/16/2017).

[44] D-LINK. *5/8 Port Gigabit Ethernet Switches*. URL: http://www.dlink.com/pt/pt/products/dgs-108-8-port-gigabit-ethernet-switch (visited on 04/05/2017).

[45] *Xbox 360 Wireless Controller*. URL: https://support.xbox.com/en-US/xbox-360/console/manual-specs (visited on 05/11/2017).

[46] *Songle Relay*. URL: https://www.ghielectronics.com/downloads/man/20084141716341001RelayX1.pdf (visited on 02/17/2017).

[47] Schneider Eletric. *C32H-DC circuit-breakers*. URL: http://www.schneider-electric.com/en/product-category/4200-circuit-breakers-and-switches/?filter=business-4-low-voltage-products-and-systems (visited on 11/18/2016).

[48] *1.1. What is Ubuntu?* URL: https://help.ubuntu.com/lts/installation-guide/s390x/ch01s01.html (visited on 06/07/2017).

[49] FLIR. *FLIR Knowledge Base*. URL: https://www.ptgrey.com/KB/10933 (visited on 06/15/2017).

[50] Gunther Gridling and Bettina Weiss. *Introduction to Microcontrollers*. Vienna University of Technology, 2007. URL: https://ti.tuwien.ac.at/ecs/teaching/courses/mclu/theory-material/Microcontroller.pdf (visited on 04/25/2017).

[51] Frédéric Leens. "An introduction to I 2 C and SPI protocols". In: *IEEE Instrumentation & Measurement Magazine* 12.1 (2009), pp. 8–13. URL: http://ieeexplore.ieee.org/abstract/document/4762946/ (visited on 06/06/2017).

[52] Mohammad Shahrokhi and Alireza Zomorrodi. "Comparison of PID controller tuning methods". In: *Department of Chemical & Petroleum Engineering Sharif University of Technology* (2013). URL: http://www.ie.tec.ac.cr/einteriano/control/clase/Zomorrodi_Shahrokhi_PID_Tunning_Comparison.pdf (visited on 06/29/2017).

[53] Isaac Saito. *rqt_plot - ROS Wiki*. URL: http://wiki.ros.org/rqt_plot (visited on 05/01/2017).

[54] *OpenSSH Server*. URL: https://help.ubuntu.com/lts/serverguide/openssh-server.html (visited on 06/01/2017).

[55] Matthew Wilson. *Configuring and Using a Linux-Supported Joystick with ROS*. Nov. 2016. URL: http://wiki.ros.org/joy/Tutorials/ConfiguringALinuxJoystick (visited on 06/28/2017).

[56] Isaac Saito. *roslaunch - ROS Wiki*. May 2017. URL: http://wiki.ros.org/roslaunch (visited on 06/12/2017).

[57] Marcelo Silva Pereira. "Automated calibration of multiple LIDARs and cameras using a moving sphere". MA thesis. Universidade de Aveiro, 2015.

[58] David Silva. "Multisensor Calibration and Data Fusion Using LIDAR and Vision". MA thesis. Universidade de Aveiro, 2016.

[59] *About FFmpeg*. URL: https://ffmpeg.org/about.html (visited on 07/04/2017).

[60] Eitan Marder-Eppstein and Vijay Pradeep. *actionlib - ROS Wiki*. URL: http://wiki.ros.org/actionlib (visited on 06/08/2017).

# Appendix A

# Motor and Encoder References


Figure A.1: Motor characteristics plate.

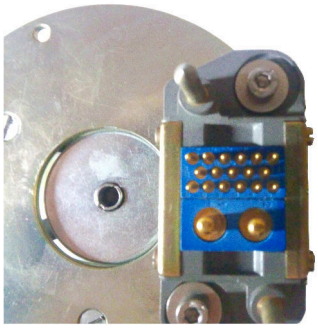
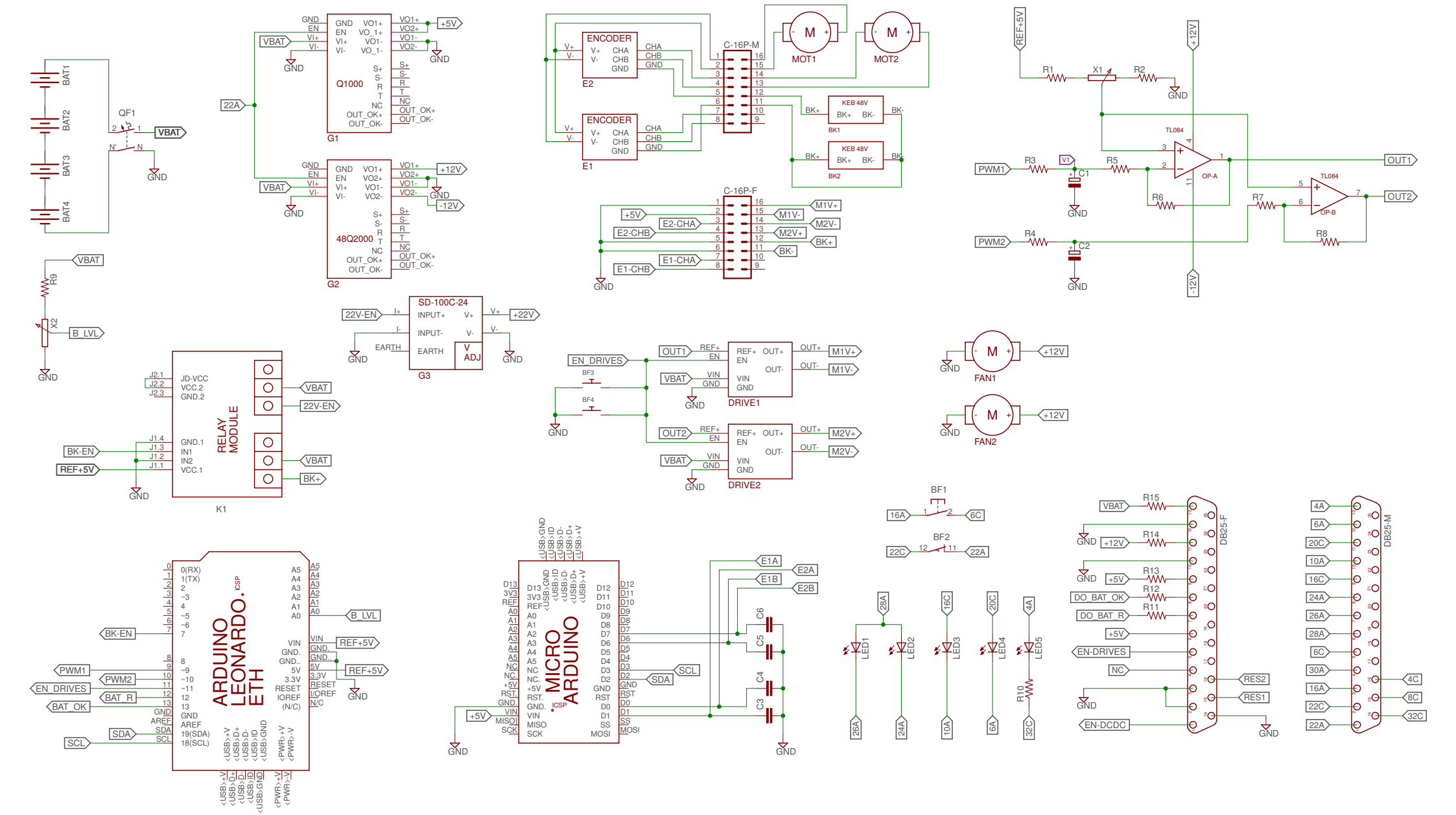


Figure A.2: Connector from each motor to the main circular connector.


Figure A.3: Encoder reference.

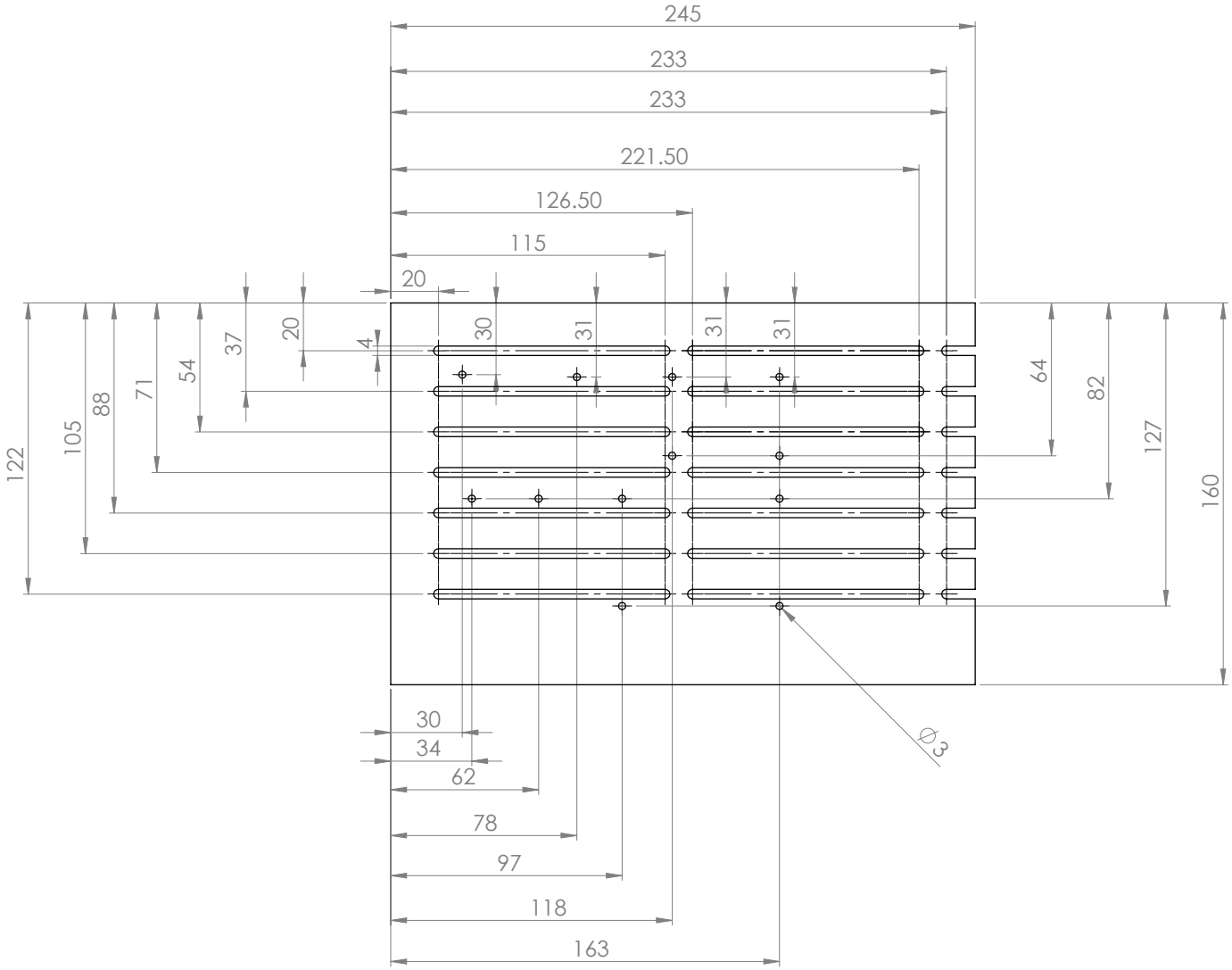
# Appendix B

# Electrical Schematics


BAT1
BAT2
BAT3
BAT4
QF1
VBAT
GND
Q1000
G1
48Q2000
G2
+5V
+12V
-12V
22A
ENCODER
E1
E2
C-16P-M
C-16P-F
MOT1
MOT2
KEB 48V
BK1
BK2
M1V+
M1V-
M2V+
M2V-
BK+
BK-
E1-CHA
E1-CHB
E2-CHA
E2-CHB
REF+5V
R1
R2
X1
TL084
OP-A
OP-B
PWM1
PWM2
R3
R4
R5
R6
R7
R8
C1
C2
OUT1
OUT2
R9
X2
B_LVL
SD-100C-24
22V-EN
+22V
G3
RELAY MODULE
K1
BK-EN
EN_DRIVES
DRIVE1
DRIVE2
FAN1
FAN2
BF1
BF2
16A
6C
22C
22A
ARDUINO LEONARDO ETH
MICRO ARDUINO
SDA
SCL
BAT_R
BAT_OK
LED1
LED2
LED3
LED4
LED5
R10
28A
26A
24A
16C
10A
20C
6A
4A
32C
R11
R12
R13
R14
R15
DO_BAT_R
DO_BAT_OK
EN-DRIVES
NC
EN-DCDC
RES1
RES2
DB25-F
DB25-M
C3
C4
C5
C6

# Appendix C

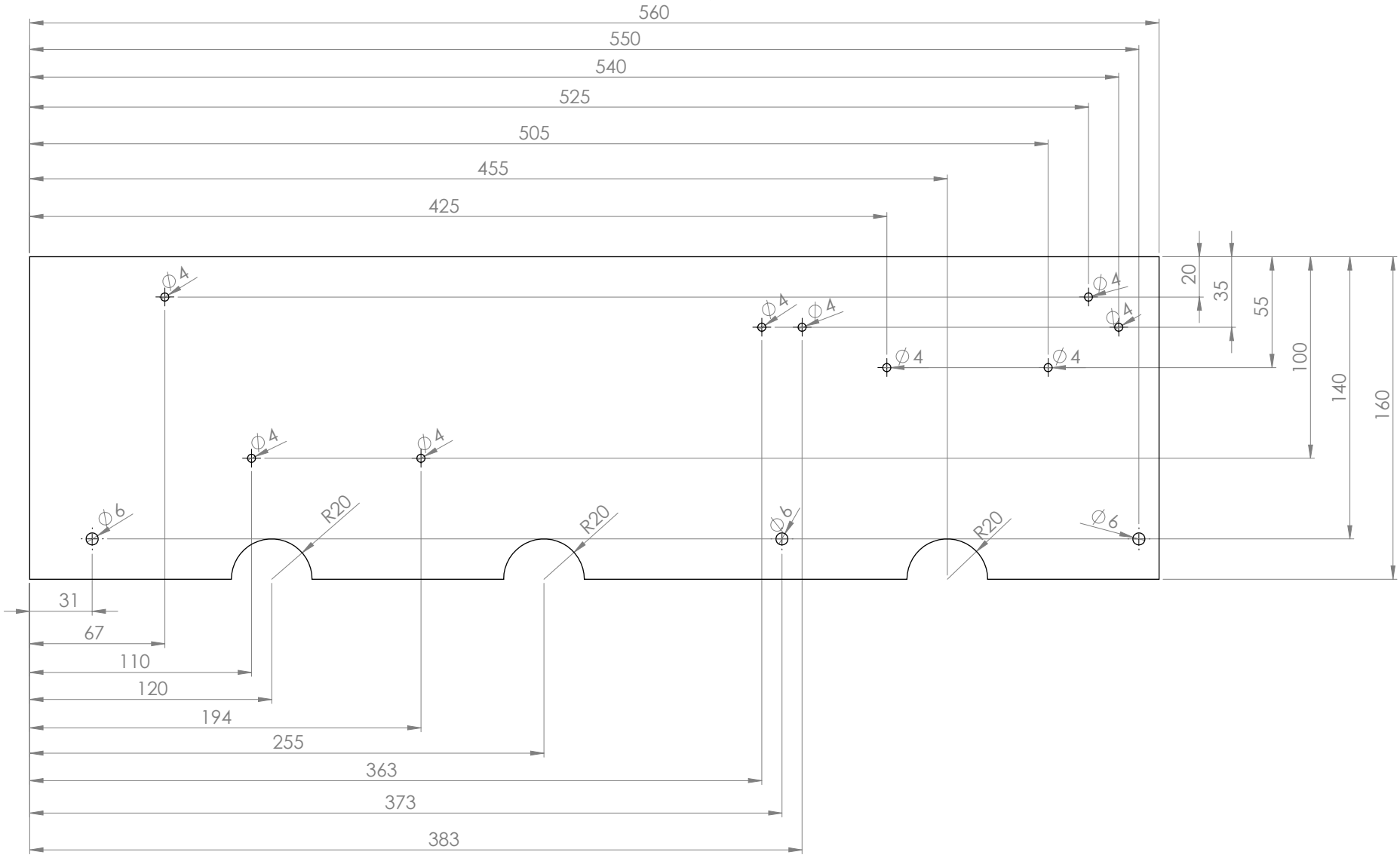
# Polycarbonate Board to Lock PCBs in Place




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>ANGULAR: | | FINISH: | | DEBURR AND BREAK SHARP EDGES | DO NOT SCALE DRAWING | | REVISION |
| | NAME | SIGNATURE | DATE | | TITLE: Author: Bruno Vieira<br>Tutor: Dr. Vitor Santos | | |
| DRAWN | | | | | | | |
| CHK'D | | | | | | | |
| APPV'D | | | | | | | |
| MFG | | | | | | | |
| Q.A | | | | MATERIAL: | DWG NO. Acrylic Arduinos | | A3 |
| | | | | WEIGHT: | SCALE:1:2 | SHEET 1 OF 1 | |

# Appendix D

# Plate to Hold Network Equipment

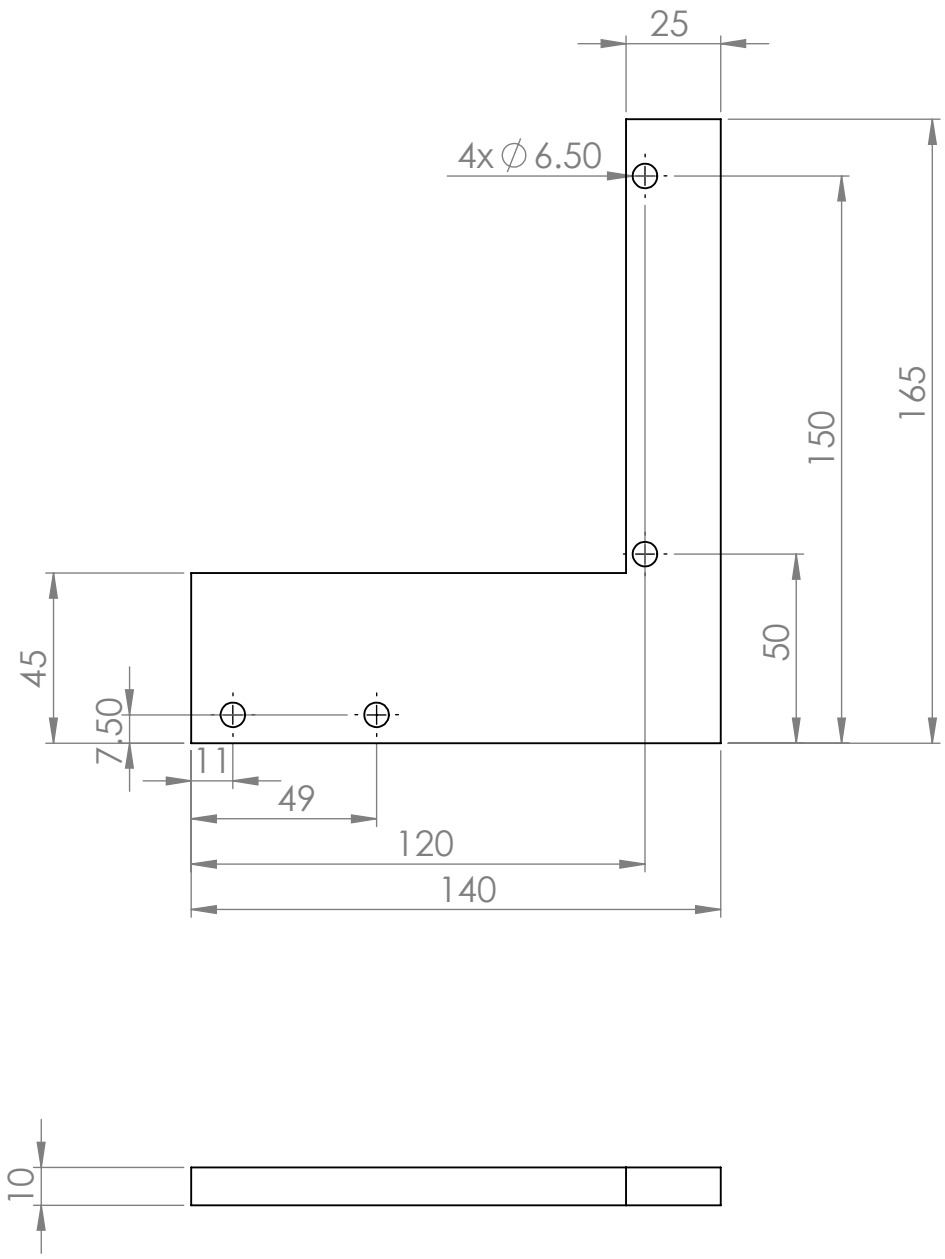



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>ANGULAR: | | FINISH: | | DEBURR AND BREAK SHARP EDGES | | DO NOT SCALE DRAWING | REVISION |
| | NAME | SIGNATURE | DATE | | | TITLE: | |
| DRAWN | | | | | | Author: Bruno Vieira<br>Tutor: Dr. Vitor Santos | |
| CHK'D | | | | | | | |
| APPV'D | | | | | | | |
| MFG | | | | | | | |
| Q.A | | | | MATERIAL: | | DWG NO. Plate Network | A3 |
| | | | | WEIGHT: | | SCALE:1:5 | SHEET 1 OF 1 |

**SOLIDWORKS Student Edition.**
**For Academic Use Only.**

# Appendix E

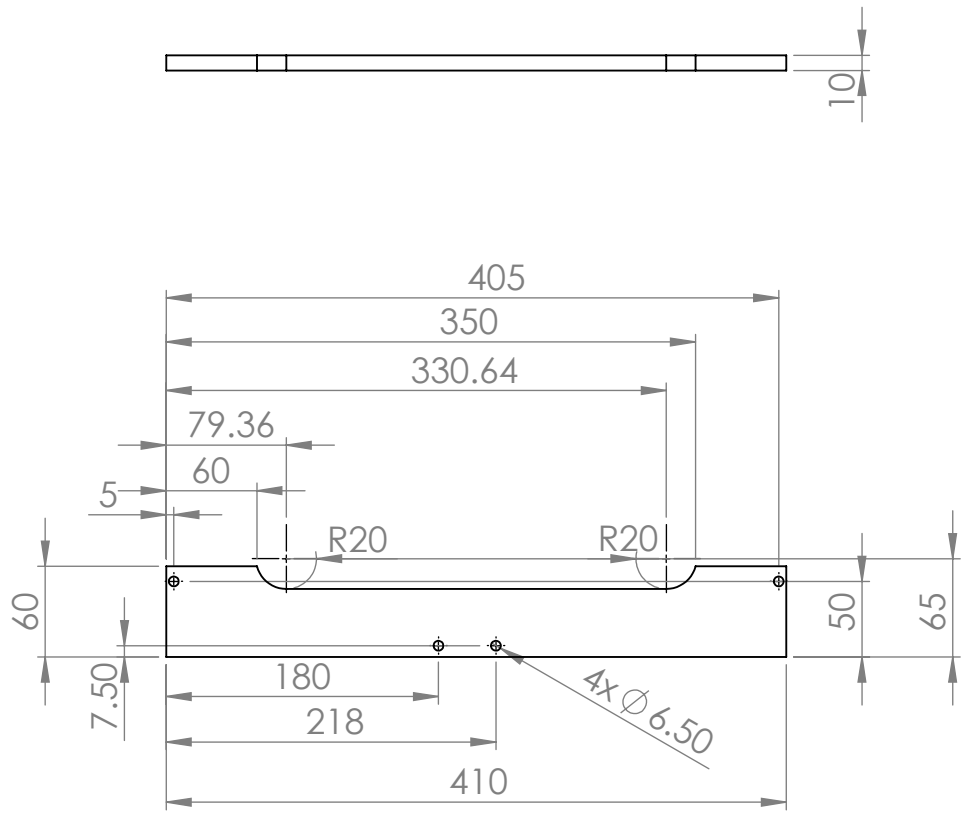
# First Visual Units Plate




| | | | | | | |
|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>ANGULAR: | FINISH: | | DEBURR AND BREAK SHARP EDGES | DO NOT SCALE DRAWING | REVISION | |
| | NAME | SIGNATURE | DATE | TITLE: | | |
| DRAWN | | | | Author: Bruno Vieira<br>Tutor: Dr. Vitor Santos | | |
| CHK'D | | | | | | |
| APPV'D | | | | | | |
| MFG | | | | | | |
| Q.A | | | | DWG NO. | | A4 |
| | | | | PTU support | | |
| | | | WEIGHT: | SCALE:1:2 | SHEET 1 OF 1 | |

# Appendix F

# Final Visual Units Plate




| | | | |
|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>ANGULAR: | FINISH: | DEBURR AND BREAK SHARP EDGES | DO NOT SCALE DRAWING / REVISION |

| | NAME | SIGNATURE | DATE |
|---|---|---|---|
| DRAWN | | | |
| CHK'D | | | |
| APPV'D | | | |
| MFG | | | |
| Q.A | | | |

TITLE:

Author: Bruno Vieira
Tutor: Dr. Vitor Santos

**SOLIDWORKS Student Edition.**
**For Academic Use Only.**

MATERIAL

DWG NO.

# PTU Fix 2

A4

WEIGHT:

SCALE:1:5

SHEET 1 OF 1

# Appendix G

# Full List of Components Used for the Platform

Table G.1: Components list and identification.

| **Name** | **Type** | **Value / Ref** |
|---|---|---|
| R1 | Resistor | 18 kΩ |
| R2 | Resistor | 10 kΩ |
| R3 | Resistor | 100 Ω |
| R4 | Resistor | 100 Ω |
| R5 | Resistor | 10 kΩ |
| R6 | Resistor | 22 kΩ |
| R7 | Resistor | 10 kΩ |
| R8 | Resistor | 22 kΩ |
| X1 | Potentiometer | 1 kΩ |
| C1 | Capacitor | 1 $\mu$F |
| C2 | Capacitor | 1 $\mu$F |
| C3 | Capacitor | 10 nF |
| C4 | Capacitor | 10 nF |
| C5 | Capacitor | 10 nF |
| C6 | Capacitor | 10 nF |
| U1A | Operational Amplifier | TL084CN |
| U1B | Operational Amplifier | TL084CN |
| R9 | Resistor | 120 kΩ |
| X2 | Potentiometer | 12 kΩ |
| R10 | Resistor | 2.7 kΩ |
| R11 | Resistor | 220 Ω |
| R12 | Resistor | 220 Ω |
| R13 | Resistor | 220 Ω |
| R14 | Resistor | 680 Ω |
| R15 | Resistor | 1 kΩ |
| LED1 | Yellow LED | 2.2V |
| LED2 | Green LED | 2.2V |
| LED3 | Green LED | 2.2V |
| LED4 | Green LED | 2.2V |
| LED5 | Green LED | 2.2V |

Table G.1: Components list and identification.

| **Name** | **Type** | **Value / Ref** |
|---|---|---|
| BF1 | Emergency Push Button | NO |
| BF2 | ON/OFF Switch | NC |
| BF3 | Switch (Front Bumper) | NO |
| BF4 | Switch (Rear Bumper) | NO |
| QF1 | DC Circuit Breaker | C32H-DC |
| BAT1 | Lead battery | 12V |
| BAT2 | Lead battery | 12V |
| BAT3 | Lead battery | 12V |
| BAT4 | Lead battery | 12V |
| MOT1 | Left motor | 48V/300W |
| MOT2 | Right motor | 48V/300W |
| FAN1 | Cooling Fan | 12V |
| FAN2 | Cooling Fan | 12V |
| G1 | DC/DC Converter | 5V |
| G2 | DC/DC Converter | 12V |
| G3 | DC/DC Converter | 22V |
| DRIVE1 | DC Drive for Left Motor | 75V/10A |
| DRIVE2 | DC Drive for Right Motor | 75V/10A |
| E1 | Encoder for left motor | 0.045° |
| E2 | Encoder for right motor | 0.045° |
| K1 | 2 Relays Module | 10A |

# Appendix H

# Platform's Installation Steps

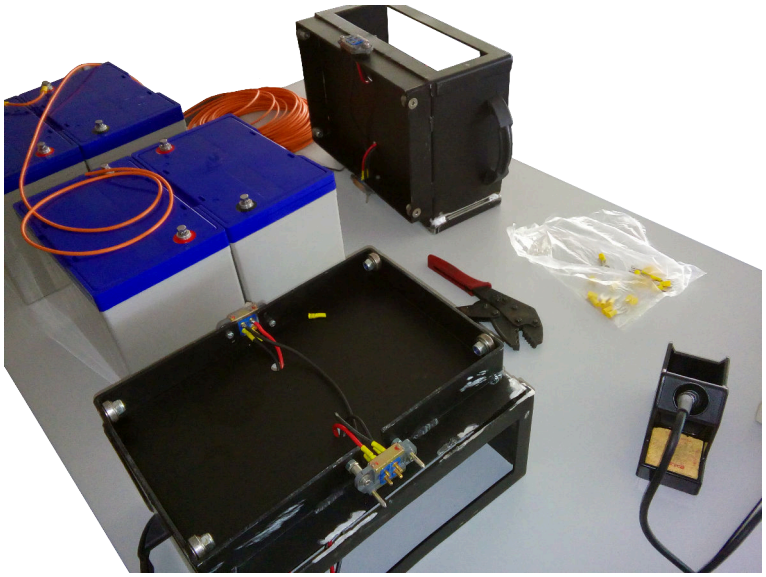


Figure H.1: Battery installation setup.


Figure H.2: Battery charger.

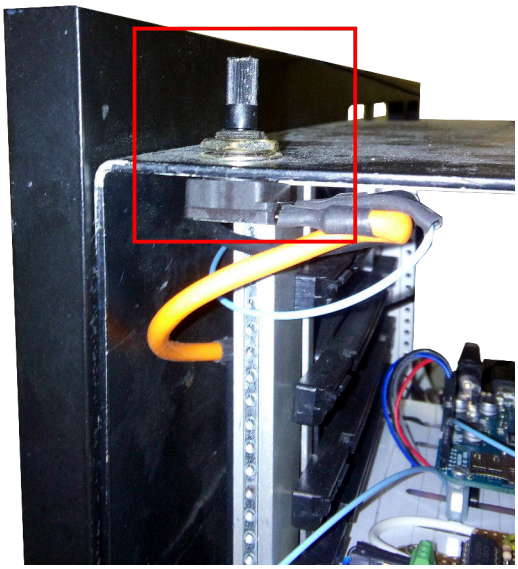


Figure H.3: Installation of the potentiometer X2.

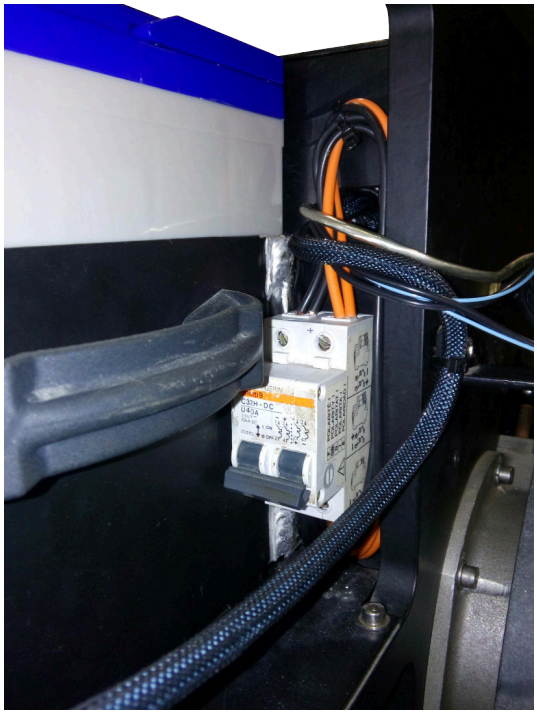


Figure H.4: Mounted circuit breaker.

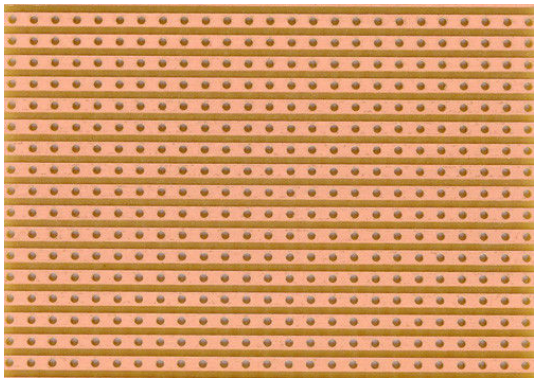


Figure H.5: Type of prototype board used to circuit implementation (CPU2 and level shifter).

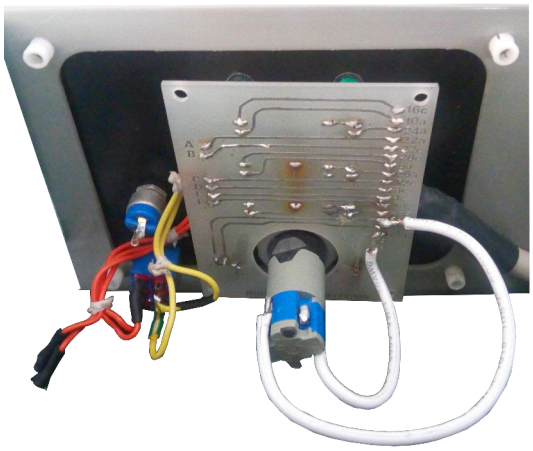


Figure H.6: PCB board for the status panel.

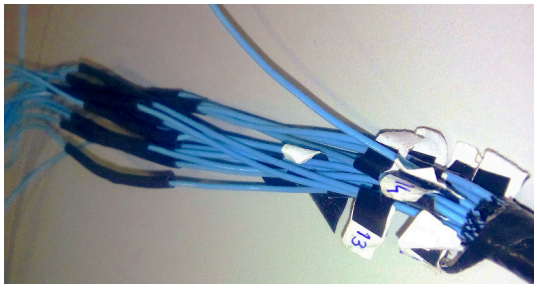


Figure H.7: Wires from DB25 connector identified and extended with soldered resistors.

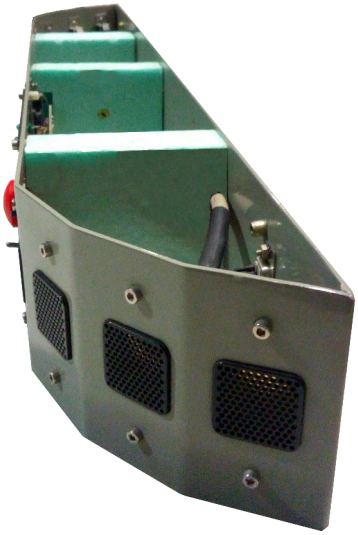


Figure H.8: Intervention performed to add resistance on the rear zone.


Figure H.9: Intervention performed to repair one DC drive.

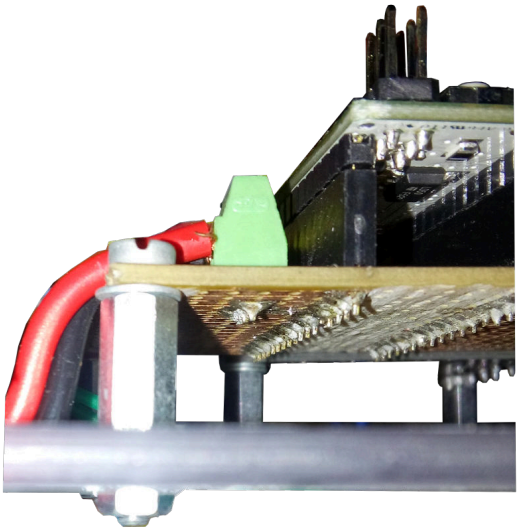


Figure H.10: Type of fastening used to the developed circuitry.

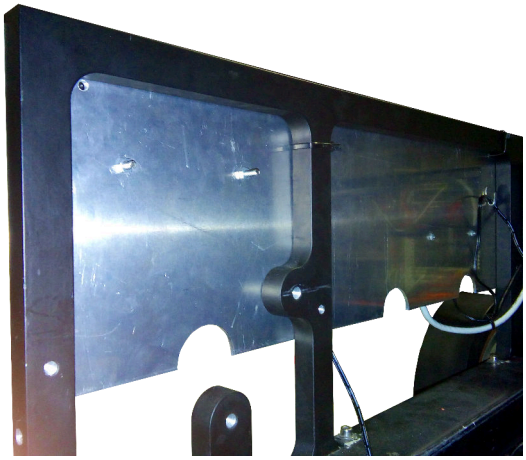


Figure H.11: Fastening of the router and switch.

Figure H.12: Type of structure used inside main case.

Figure H.13: Type of bus-bars used to make the power distribution.

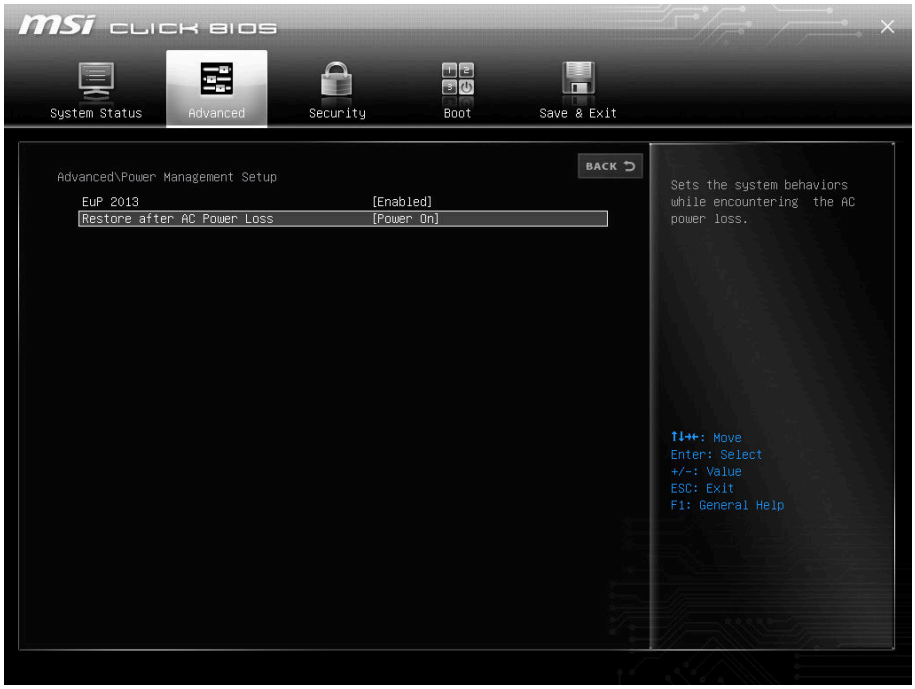



Figure H.14: BIOS setup used on CPU1.

# Appendix I

# Printed Labels

Table I.1: Printed labels to place inside the label sleeves.

| Label | Description | Quantity |
|---|---|---|
| E-C1 | Ethernet CPU1 | 1 |
| E-C2 | Ethernet CPU2 | 1 |
| E-Cam | Ethernet Camera | 1 |
| E-RS | Ethernet Router-Switch | 1 |
| USB-up | USB extension to the top of the platform | 1 |
| B-CH+ | Charge batteries, positive wire from the series of batteries | 1 |
| B-CH- | Charge batteries, positive wire from the series of batteries | 1 |
| B-S+ | Batteries series, +48V | 1 |
| B-S- | Batteries series, GND | 1 |
| BP-R | Rear bumper reserved cables | 1 |
| BP-F | Front bumper reserved cables | 1 |
| 5V-up | 5V to the top of the platform | 2 |
| 12V-up | 12V to the top of the platform | 2 |
| VBAT-up | VBAT to the top of the platform | 2 |
| GND-up | GND to the top of the platform | 2 |
| SW-S | Switch supply (+5V and GND) | 1 |
| RT+ | Router supply (+12V) | 1 |
| RT- | Router supply (GND) | 1 |
| CPU1+ | CPU1 supply (+22V) | 1 |
| CPU1- | CPU1 supply (GND) | 1 |
| CPU2+ | CPU2 supply (+12V) | 1 |
| CPU2- | CPU2 supply (GND) | 1 |
| CPU3+ | CPU3 supply (+5V) | 1 |
| CPU3- | CPU3 supply (GND) | 1 |
| DCC1+ | DC/DC converter to +/-12V, positive supply (VBAT) | 2 |
| DCC1- | DC/DC converter to +/-12V, negative supply (GND) | 2 |
| DCC2+ | DC/DC converter to +5V, positive supply (VBAT) | 2 |
| DCC2- | DC/DC converter to +5V, negative supply (GND) | 2 |
| DCC3+ | DC/DC converter to +22V, positive supply (VBAT) | 2 |
| DCC3- | DC/DC converter to +22V, negative supply (GND) | 2 |
| DRV1+ | DC Drive1 (Right motor), positive supply (VBAT) | 2 |
| DRV1- | DC Drive1 (Right motor), negative supply (GND) | 2 |

Table I.1: Printed labels to place inside the label sleeves.

| **Label** | **Description** | **Quantity** |
|---|---|---|
| DRV2+ | DC Drive1 (Left motor), positive supply (VBAT) | 2 |
| DRV2- | DC Drive1 (Left motor), negative supply (GND) | 2 |
| 22V-EN | DC/DC converter supply (VBAT), relay contact (to enable/disable) | 2 |
| BK+ | Brakes supply (VBAT), relay contact (to break/unbreak) | 2 |
| BK- | Brakes supply (GND) | 1 |
| F1+ | Fan 1 supply (+12V) | 1 |
| F1- | Fan 1 supply (GND) | 1 |
| F2+ | Fan 2 supply (+12V) | 1 |
| F2- | Fan 2 supply (GND) | 1 |
| E1-R | Encoder 1, Right encoder (driver's perspective) | 1 |
| E2-L | Encoder 2, Lef Encoder (driver's perspective) | 1 |
| E-S+ | Encoders supply (+5V) | 1 |
| E-S- | Encoders supply (GND) | 1 |
| M1V+ | Motor 1 (Right) supply (0-48V) | 1 |
| M1V- | Motor 1 (Right) supply (GND) | 1 |
| M2V+ | Motor 2 (Left) supply (0-48V) | 1 |
| M2V- | Motor 1 (Left) supply (GND) | 1 |
| PWM1 | PWM signal from the controller to the drive 1, REF+ | 1 |
| PWM2 | PWM signal from the controller to the drive 2, REF+ | 1 |
| VBAT | Battery level. Supply for motor drives, DC/DC converters | 6 |
| +12V | Supply for Arduino and switch | 6 |
| -12V | Voltage reference for level shifter circuit | 2 |
| +5V | Encoders supply | 6 |
| GND | Ground | 10 |

# Appendix J

# CPU1 Install Commands

The following guide was elaborate while configuring `CPU1`, so if is necessary to format it, this guide can be used as starting point to reinstall everything.

**WIFI and Bluetooth install**

```
$ sudo apt update
$ sudo apt install linux-generic-hwe-16.04
```

**ROS**

```
$ sudo sh -c 'echo "deb␣http://packages.ros.org/ros/ubuntu␣$(
    lsb_release␣-sc)␣main" > /etc/apt/sources.list.d/ros-latest.list'

$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --
    recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116

$ sudo apt-get update

$ sudo apt-get install ros-kinetic-desktop-full

$ sudo rosdep init
$ rosdep update

$ echo "source␣/opt/ros/kinetic/setup.bash" >> ~/.bashrc
$ source ~/.bashrc

$ sudo apt-get install python-rosinstall
```

**Arduino Related**

To instal the Arduino IDE, first download the version from `https://www.arduino.cc/en/Main/Software` and run the install.sh file. After that, an error comes up: "xdg-icon-resource: icon name 'cc.arduino.arduinoide' does not have a proper vendor prefix ".

Solution (from `https://github.com/arduino/Arduino/issues/6116`) says that the re-

source name in the "**install.sh**" file is wrong. in that file, edit the field "RESOURCE_NAME = cc.arduino.arduinoide" to "RESOURCE_NAME = arduino-arduinoide". After that, it is only necessary to run the "**install.sh**" normally.

```
$ sudo usermod -a -G dialout robuter
```

Arduino Leonardo requires a specific command to be detected. Solution found at: http://starter-kit.nettigo.eu/2015/serial-port-busy-for-avrdude-on-ubuntu-with-arduino-leonardo-eth/

```
$ echo 'ATTRS{idVendor}=="2a03", ENV{ID_MM_DEVICE_IGNORE}="1"' | sudo
    tee /etc/udev/rules.d/77-arduino.rules
```

**Used libraries for the Arduino (install from the Arduino IDE)**
Ethernet2, TimerOne, MsTimer2

**Joystick** (`http://wiki.ros.org/joy/Tutorials/ConfiguringALinuxJoystick`)

```
$ sudo apt-get install ros-kinetic-joy
```

**Others**

```
$ sudo apt-get install doxygen
$ sudo apt-get install opencv-doc
```

**PTU unit**
`https://github.com/ros-drivers/flir_ptu` `https://github.com/wjwwood/serial`

**FLEA Camera**
`https://github.com/ros-drivers/pointgrey_camera_driver`

# Appendix K

# Robonuc's Operation Guide

This guide intends to present an overview of the **r_platform** package as well as the necessary commands/requisites to operate the platform.

## Packages Trees

```
/
├── CMakeLists.txt
├── package.xml
├── launch
│   ├── experience_launch
│   ├── robonuc_teleop.launch
│   └── robonuc_aut_navi.launch
├── msg
│   └── navi.msg
├── params
│   ├── default.yaml
│   └── joy_params.yaml
└── src
    ├── decompose_vel.cpp
    ├── r_teleop.cpp
    ├── r_pointgrey_fl3-ge-28s4-c_driver.cpp
    └── r_exp_results.cpp
.
```

Most of the the designed code is contained on the **r_platform** package, where the cpp files have the same name as the nodes. The remaining code, TCP/IP client, is contained in the **comm_tcp-client** package. Its tree is presented below:

```
/
├── CMakeLists.txt
├── package.xml
├── msg
│   └── pid.msg
└── src
    ├── client_node.cpp
    └── server_node.cpp
.
```

## Launch the system

- Power on the system — QF1 ON;
- Ensure that the ON/OFF switch on the robot's rear status panel is in the ON position (up);
- Verify that the 4 green LEDs are ON (BATTERIES O.K., +5V, +/- 12V AND +48V);
- Connect to the Robonuc's network (with an external PC) — **Robonuc** or **Robonuc_5G**;
- Access CPU1 via SSH bridge;

When this point is achieved, the user must have an available remote terminal from CPU1 on its machine. The user can now proceed in three different ways:

- **Manual operation with joystick** — robonuc_teleop.launch;
- **Autonomous Navigation** — robonuc_aut_navi.launch;
- **Straight line experimental results** — experiment.launch.

Each one of these launch files can be executed from its directory simply by running: "roslaunch launchfilename.launch".

Recommendation for the operator: before executing any node, press the emergency button and verify that every node is running properly by publishing the expected data. After that, proceed with the selected application.

## IP table

| **Unit** | **IP** |
|---|---|
| CPU1 | 192.168.0.123 |
| CPU2 | 192.168.0.10 |
| Flea3 Camera | 192.168.0.43 |
| Fanuc Manipulator | 192.168.0.231 |
| Arduino ETH 2 | 192.168.0.50 |

## Battery Charging

In order to charge the platform, the appropriate charger must be used. Before plug the connector on the platform's rear, the circuit breaker, **QF1**, must be **OFF**. Also, the connector as a proper position, the user should look for the "UP" and "DOWN" inscriptions on it, which should be respected when plugging the cable.

## CPU2 and CPU3 Programming

To upload new code for CPU2 and CPU3, two USB extensions are available inside Robonuc's main case. For CPU2, the cable is labeled with "USB1" and for CPU3 with "USB2". Note that while uploading code for the Arduinos, the **EMERGENCY BUTTON MUST BE PRESSED**, otherwise, fluctuations in the PWM output pins are observed,

which will actuate the motors while they are braked: harming the system and moving the platform during the upload process.

## Troubleshooting

If the message "There is an image consistency issue with this image" is prompted while trying to acquire image, proceed in the following way:

- Increase the package delay on the Flycap API sub-menu "Custom Video Modes" or by accessing the camera registers through its library (FlyCap2.h);
- Decrease the package size using the same method;
- Test the connection with a higher category Ethernet cable.

The motors are rotating but the platform is not moving: insert the wheel's clutch.