Universidade de Aveiro

Mestrado em Engenharia de Automação Industrial

Projeto em Engenharia de Automação

Development and integration of a visual guidance system into the Robonuc platform

Author:
Bruno Vieira

Supervisor:
Dr. Vitor Santos

June 25, 2017

Abstract

The current project is being carried out as final discipline's work and is a contribution for the Robonuc's platform (Laboratory of Automation and Robotics).
The main goal is centered in implementing a vision-based navigation algorithm on a ROS distributed architecture. The navigation will be based on line recognition, where the direction will be extracted and converted in motor commands.
As final result it was possible to install a camera setup in the platform and navigate autonomously, guided by a line.
This document presents a general description of the problem where all the different steps are identified, as well as all the work developed to achieve the proposed objectives.

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Objectives . . . 1

2 Experimental Infrastructure **3**
- 2.1 Robonuc . . . 3
  - 2.1.1 Platform Commands . . . 4
  - 2.1.2 Platform Kinematics . . . 5
- 2.2 Visual Units . . . 6
- 2.3 ROS . . . 7
- 2.4 OpenCV . . . 8

3 Solution Development **9**
- 3.1 System Diagram . . . 9
- 3.2 Camera Setup . . . 10
  - 3.2.1 Camera Fix . . . 10
  - 3.2.2 Camera Configuration . . . 12
  - 3.2.3 Camera Orientation . . . 12
- 3.3 ROS nodes . . . 13
  - 3.3.1 Image acquisition . . . 13
  - 3.3.2 Image processing . . . 13
  - 3.3.3 Velocity conversion . . . 15
  - 3.3.4 Motor's communication . . . 15
  - 3.3.5 Vision system overview . . . 16

4 Experimental Results **17**
- 4.1 Test conditions . . . 17
- 4.2 Obtained images . . . 17
  - 4.2.1 Pre-Processors . . . 17
  - 4.2.2 Line segment determination . . . 18

5 Conclusions **25**
- 5.1 Future Work . . . 25

Bibliography **27**

A Technical Drawing for Camera Support **29**

List of Figures and Tables

2.1 Under development Robonuc platform. 4
2.2 Differential drive kinematics. Image from [2] 5
2.3 Flea3 GigE, model GE-28S4C-C. Adapted from [4]. 6
2.4 Pan-Tilt Unit FLIR PTU-D46-17. Adapted from [8]. 7
2.5 ROS topology example with three nodes exchanging messages on two topics. 8

3.1 Generic system diagram for the platform's global architecture. 9
3.2 First camera setup used for navigation. Detailed view of the camera mounted on the PTU. 10
3.3 Final camera setup used to implement the navigation algorithm. Detailed view of the camera mounted on the PTU. 11
3.4 Raw images acquired from the first setup (a) and second setup (b). The navigation line is marked with a red 'X'. 11
3.5 Setting the camera's IP range to the predefined in the Robonuc's platform. Illustration adapted from the FLIR documentation [7]. 12
3.6 Printed lines with the angle references from the first setup (in red) and the second (green). 14
3.7 Computer vision algorithm applied to detect line and determine $\Delta\theta$. 14
3.8 System nodes diagram obtained by **rqt_graph** where the inter-node communication, in topics, is represented. 16

4.1 Used path to test the navigation algorithm. 17
4.2 Pre-processing algorithms with top incident light. 18
4.3 Pre-processing algorithms with side light (from the left side). 19
4.4 Pre-processing algorithms with low light conditions. 20
4.5 Canny edge detection after pre-processing with top incident light. 21
4.6 Canny edge detection after pre-processing with side light (from the left side). 22
4.7 Canny edge detection after pre-processing with low light conditions. 23
4.8 Results with standard Hough transform (a) Probabilistic Hough Transform (b). 24
4.9 Hough transform with right slope (a) and left slope (b). 24

2.1 Robonuc characteristics. 3

3.1 Identification of the blocks from Figure 3.1 10

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. The 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 ease from lower level tasks, maximizing the efficiency of the industrial environment.

This project in particular, intends to integrate this technology in Robonuc, a retrofitted robotic platform (from Robuter II), which is available at Laboratory of Automation and Robotics, LAR. This platform is still under development, thus further improvements will be made.

1.2 Objectives

The final objective is to implement an algorithm to navigate autonomously, guided by a line. In order to do it, there are several steps, or sub-goals, that can be pointed out:

- Study the platform hardware and kinematics;
- Design a support to install the camera;
- Find a reference for the system;
- Line detection and extraction of linear and angular velocities;
- Implement a navigation algorithm that outputs motor commands.

Chapter 2

Experimental Infrastructure

2.1 Robonuc

The Robonuc (Figure 2.1) is a retrofitted platform from the Robuter II. It has an open software architecture, ROS, and distributed/modular hardware. Table 2.1 features some of the robot's main characteristics.

Table 2.1: Robonuc characteristics.

Dimensions	[102.5 x 68 x 44] cm
Weight	approximately 200 kg
Payload	120 kg
Speed	up to 1 m/s
Batteries	4 x 12V 75Ah. Total capacity 3.6kWh
2 Motors	Permanent magnet DC 300W 48V
Control Type	Differential
Positioning	Optical encoders
CPU	Mini-PC MSI Cubi 2 i5-7200U Arduino Leonardo ETH Arduino Micro
Operating System	Linux OS with ROS running
Programming	C++ and Python

Figure 2.1: Under development Robonuc platform.

2.1.1 Platform Commands

There are three main types of messages, that can be distinguished by the first two characters: "**MM**", "**BK**" and "**EN**". Both of the last two messages (BK and EN) are simply for actuating brakes and request for encoder data, respectively, while the other requires more attention. This last type of message defines the setpoint of each motor, in pulses (which is directly proportional to rpm's), and its direction of rotation. For example, the message "$MMB500 - F300>$", defines the setpoint for motor 1 at 500 pulses in the backward direction and the motor 2 with 300 pulses in the forward direction. Highlighted in red is the type of the message, in blue the direction of rotation, in black the corresponding references and in green the end character of the message.

After processing the message, each motor is controlled individually with a single PWM signal. The resolution of PWM in the Arduino has 8 bits, 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.

2.1.2 Platform Kinematics

To understand and perform kinematic analysis of the platform it is important to understand how is the robot's locomotion. The movement is based on two separate driving wheels placed on one end of the robot. It can thus vary the direction of its movement only by changing the rate of rotation of each wheel, eliminating the necessity of a steering system. To balance the platform and follow the movement, there are two caster wheels (on the opposite side of the driving wheels).

From Figure 2.2 and the previous explanations, in order to control the platform's speed and direction, it is visible that the inputs are each motor's velocity (V_l and V_r). However, typically the important things about a robot's locomotion are its position (x,y) and direction (ϕ) and therefore each wheel velocity must be related with these variables. Expressions (2.1) show this relation, where $\dot{x}$, $\dot{y}$ and $\dot{\phi}$ represent how these variables, linear and angular velocities, change [1].

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

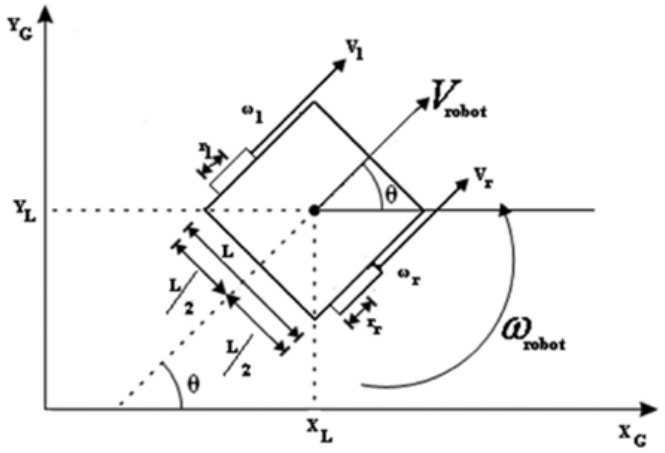


Figure 2.2: Differential drive kinematics. Image from [2]

Despite its valid relation, it is very cumbersome to think in terms of relative wheel velocities, so the unicycle model is used in order to control it more intuitively [3]. This approach is more common when designing controllers because it overcomes this issue by using the linear (V) and angular(ω) velocities (expressions (2.2)).

Relating the equations from (2.1) and (2.2), which is crucial because the actual inputs to the system are the wheels velocities, the controller can be designed to accept the linear and angular velocities — equations (2.3) and (2.4).

$$\begin{cases} \dot{x} = vcos\phi \\ \dot{y} = vsin\phi \\ \dot{\phi} = \omega_{robot} \end{cases} \tag{2.2}$$

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

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

2.2 Visual Units

Camera
The chosen camera for this project was the Point Grey Flea3 GigE, model GE-28S4C-C (Figure 2.3). It is a color camera with 2.8MP and it is equipped with a Sony sensor: ICX687 CCD, 1/1.8",9.69μm; Resolution 1928 x 1448 pixels at 15 FPS. [4]
Power supplied with 12VDC.

Figure 2.3: Flea3 GigE, model GE-28S4C-C. Adapted from [4].

This camera already had been used in another project, so it was available in the laboratory.

Pan and Tilt Unit [8]
Servo controlled pan and tilt unit, model PTU-D46-17 by FLIR (Figure 2.4). Its pan

and tilt axis have 180°/111° positioning range respectively, and both support speeds of 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 be used to ensure a constant camera orientation.

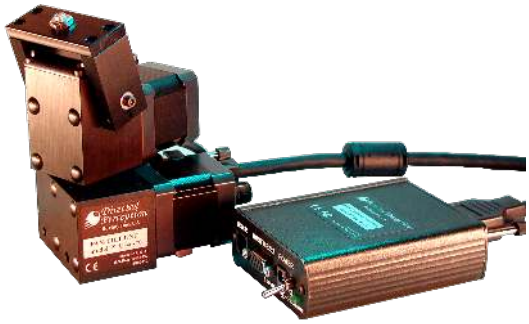


Figure 2.4: Pan-Tilt Unit FLIR PTU-D46-17. Adapted from [8].

2.3 ROS

ROS [5], 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 because of 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 2.5 shows an example of this topology.

In the diagram from Figure 2.5, the "**Node 1**", which is responsible for the image acquisition, publishes the data (in the 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).

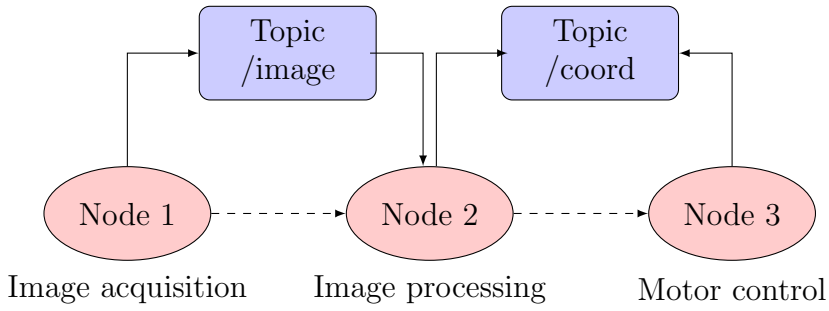


Figure 2.5: ROS topology example with three nodes exchanging messages on two topics.

Despite not being represented in the diagram, each topic can be accessed by different nodes (this is valid both for publish and subscribe actions).

2.4 OpenCV

OpenCV stands for Open Source Computer Vision Library and as its name suggests, it is a free library, both for academic and commercial use, of programming functions mainly aimed to real-time computer vision.

It has different languages interfaces, such as C++, Python, C and Java. It was written in C/C++ and can take advantage od multi-core processing. Regarding the Operating Systems (OS), it supports Windows, Linux, Mac OS, iOs and Android [6].

Chapter 3

Solution Development

3.1 System Diagram

The diagram in Figure 3.1 illustrates all the main blocks and explains how they interact among them, giving an overview of the full system. 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 to the desired navigation. These motors are actuated trough a PWM signal on the DC drives and can feedback its velocity with encoders. This signal is processed by CPU3 and sent to CPU2 closing the feedback 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.

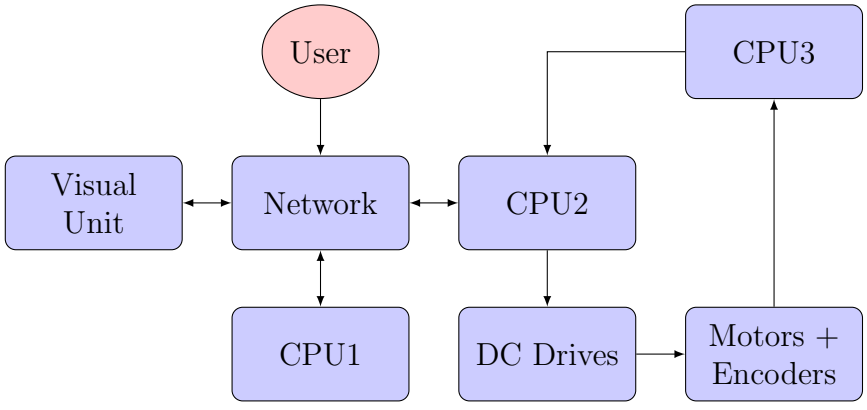


Figure 3.1: Generic system diagram for the platform's global architecture.

Table 3.1: Identification of the blocks from Figure 3.1

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

3.2 Camera Setup

3.2.1 Camera Fix

In order to use the camera, it was designed an aluminum support, appendix A. 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 due to the presented constraints that was not possible without re-arranging the components on the top of it. So, using the original designed support, the camera was placed in the left side of the platform, as shown in Figure 3.2.

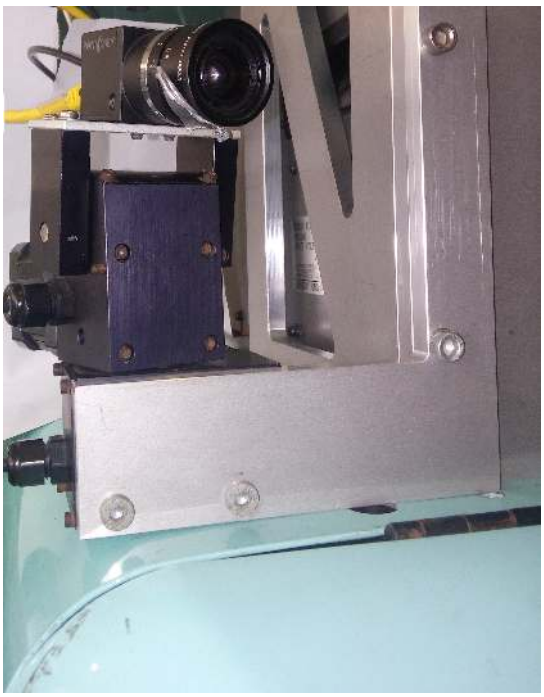


Figure 3.2: First camera setup used for navigation. Detailed view of the camera mounted on the PTU.

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). Due to this, using the same support (only inverted) it was possible to place the camera near the center, minimizing the effect that the perspective introduced to the image. This change can be seen in Figure 3.3.

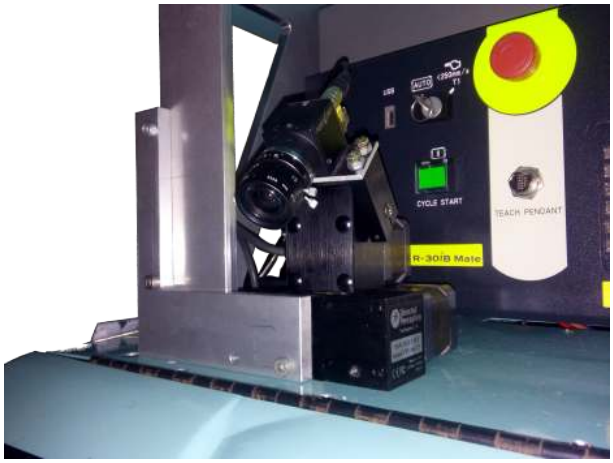
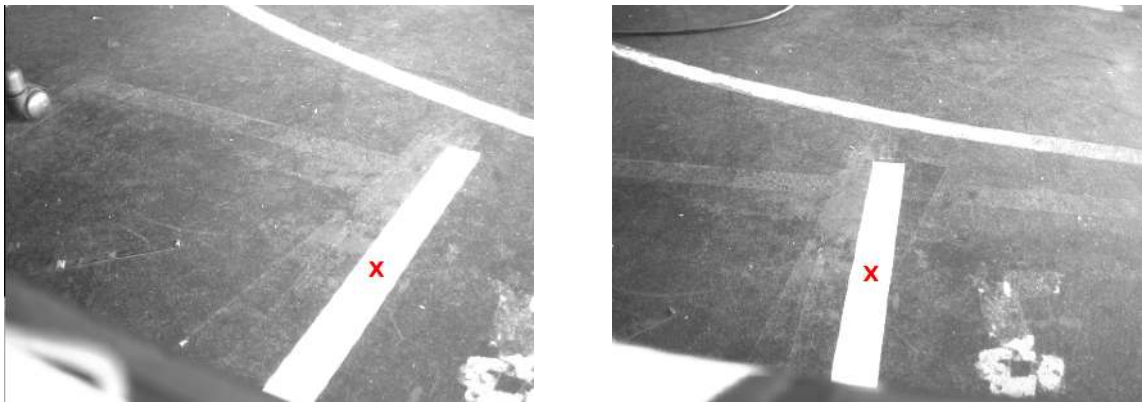


Figure 3.3: Final camera setup used to implement the navigation algorithm. Detailed view of the camera mounted on the PTU.

Comparison between the acquired raw images (MONO8 format) can be observed in Figure 3.4 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 3.4: Raw images acquired from the first setup (a) and second setup (b). The navigation line is marked with a red 'X'.

3.2.2 Camera Configuration

As stated previously in section 2.2, 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"** and the platform's network in the range **"192.168.0.XXX"**.

In order to change the camera 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 3.5 illustrates this process, where it was necessary to write in the hexadecimal registers to configure the desired IP address.

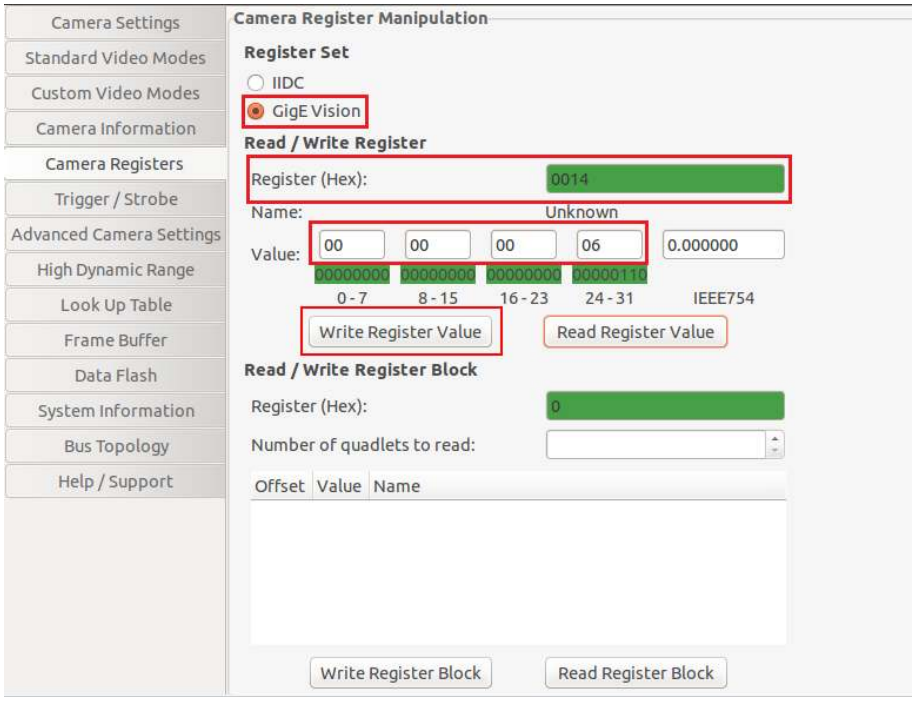


Figure 3.5: Setting the camera's IP range to the predefined in the Robonuc's platform. Illustration adapted from the FLIR documentation [7].

3.2.3 Camera Orientation

To ensure an important condition on the system — constant camera orientation, a servo controlled pan and tilt unit (model PTU-D46-17 by FLIR [8]) 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, *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: **PP150** and **TP-750**.

3.3 ROS nodes

All the developed code had to be implemented in the already existing architecture. This section presents the necessary ROS nodes to perform navigation guided by a line. Some of the nodes had to be developed, while others were already operational and had to be integrated in the developed work. The image acquisition node and the node that establishes communication with CPU2 were already implemented.

3.3.1 Image acquisition

To acquire image the "**r_pointgrey_FL3_28S4**" node, already installed on the platform and originally developed by Marcelo Pereira [9] and David Silva [10], was used. This node is responsible to establish 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.

3.3.2 Image processing

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

The first approach was to find a system reference in order to determine the angle correspondent to a straight line, i.e., to navigate forward. As explained before, this was necessary because of the perspective's effect due to the camera not being aligned. On Figure 3.6 are two reference lines, with the first camera setup (in red) and the used one (green). To obtain these references, in both cases, it was placed a straight line, aligned with the platform, that was located on the center of it (in the middle point of the motored wheels' distance).

After determine the system reference, the generic method to follow the line started to be designed. The ultimate objective is to determine de angle variation, $\Delta\theta$, from its reference. This difference translates the rate of steering, where:

- $\Delta\theta < 0$ — system turns left;
- $\Delta\theta > 0$ — system turns right;
- $\Delta\theta = 0$ — system moves forward, zero angular velocity.

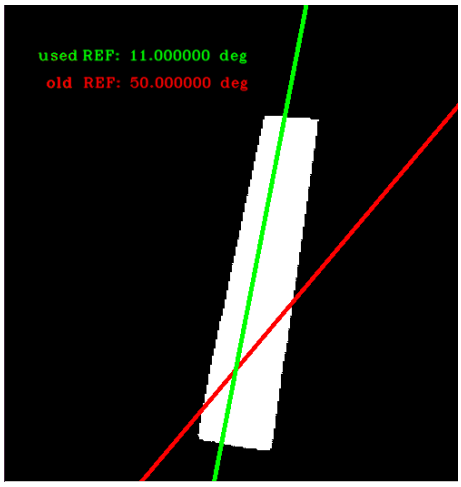


Figure 3.6: Printed lines with the angle references from the first setup (in red) and the second (green).

As consequence of identifying this important objective, the algorithm was designed having it in mind. Diagram from Figure 3.7 contains the major steps of the implemented computer vision algorithm to obtain $\Delta\theta$.

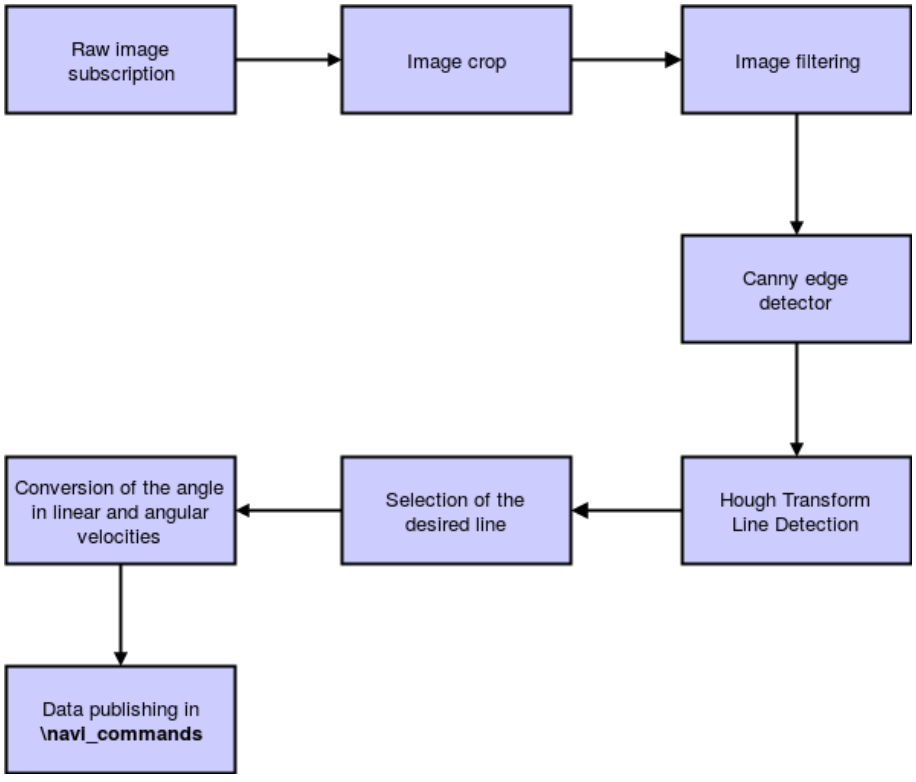


Figure 3.7: Computer vision algorithm applied to detect line and determine $\Delta\theta$.

When a image is loaded, the first operation is to crop it. This action 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.

Afterwards, it was applied an equalization of the histogram, in order to sharpen the image for the canny edge detector be applied. This algorithm outputs a binary image with white edges. With that image, the Hough transform was used to find the best line segment that fit the detected edges. When this point is reached, it is only necessary to retrieve the selected line angle to compare with the reference value.

3.3.3 Velocity conversion

Since the output velocity of the image processing node is given in linear and angular terms, these values have to be converted in motor commands, section 2.1.2. Node "**vel_decompose**" is the publisher and subscriber node responsible for that particular task. The velocities are subscribed from the topic "**\navi_commands**" to perform calculations, equations (2.3) and (2.4).

After the conversion, each wheel velocity is determined to correspond to the desired linear and angular velocity required by the image processing node. This relative velocity is obtained in radians/second, but as stated in subsection 2.1.1, the input motor commands are given in pulses. Equation (3.1) 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). The result, p_r, represents the necessary number of pulses to be counted in $\frac{1}{F}$ of a second. Despite the equation refers to the right wheel, the same relation is valid for the left one.

$$p_r = \frac{V_r P_W}{2\pi F} \tag{3.1}$$

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_node**".

3.3.4 Motor's communication

In diagram from Figure 3.1 is represented the communication between CPU1 and CPU2 (which controls the motors). This communication is established via Ethernet and it is crucial, once 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. This node was already implemented on the platform and 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 "**client_node**", that contains the messages to be transferred, at a rate of 20Hz, to the CPU2. The type of messages are presented on subsection 2.1.1.

3.3.5 Vision system overview

In order to launch all the created nodes, it was written a launch file (from *roslaunch*). 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 [11]. Also, a file with parameters was created, in order to edit some code constants in the test phase without having to compile the code.

Figure 3.8 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.

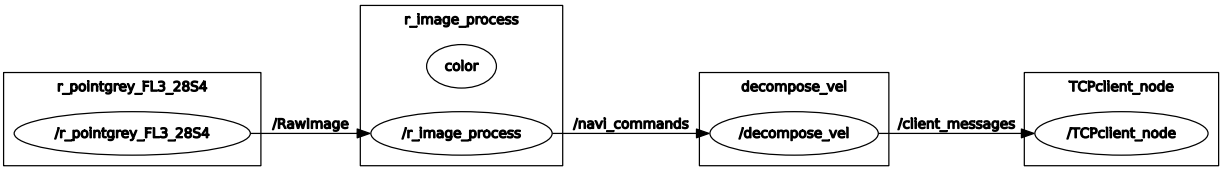


Figure 3.8: System nodes diagram obtained by **rqt_graph** where the inter-node communication, in topics, is represented.

Chapter 4

Experimental Results

4.1 Test conditions

The conditions were limited due to the laboratory constraints. Despite a more complex setup could be arranged, it was used a simple line to test the navigation algorithm (Figure 4.1). Also, for safety reasons and because the platform is still under development, the velocity used to perform the tests was slow — about 5 cm/s.

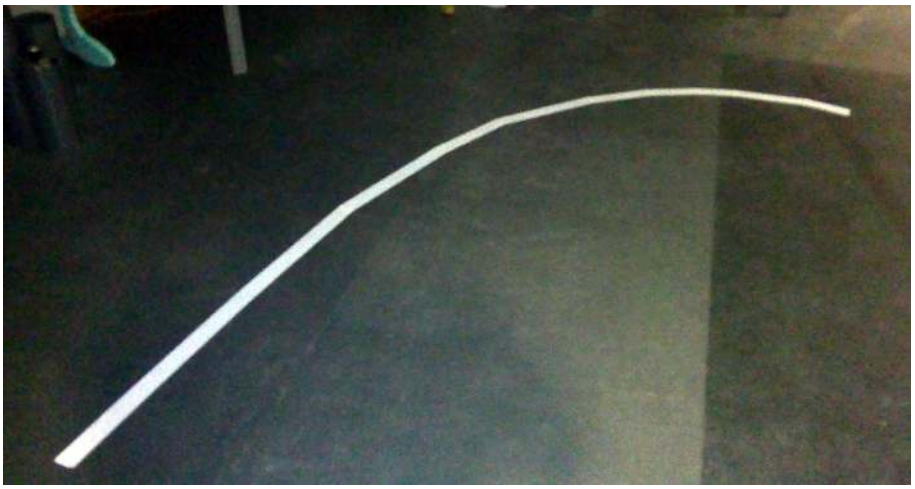


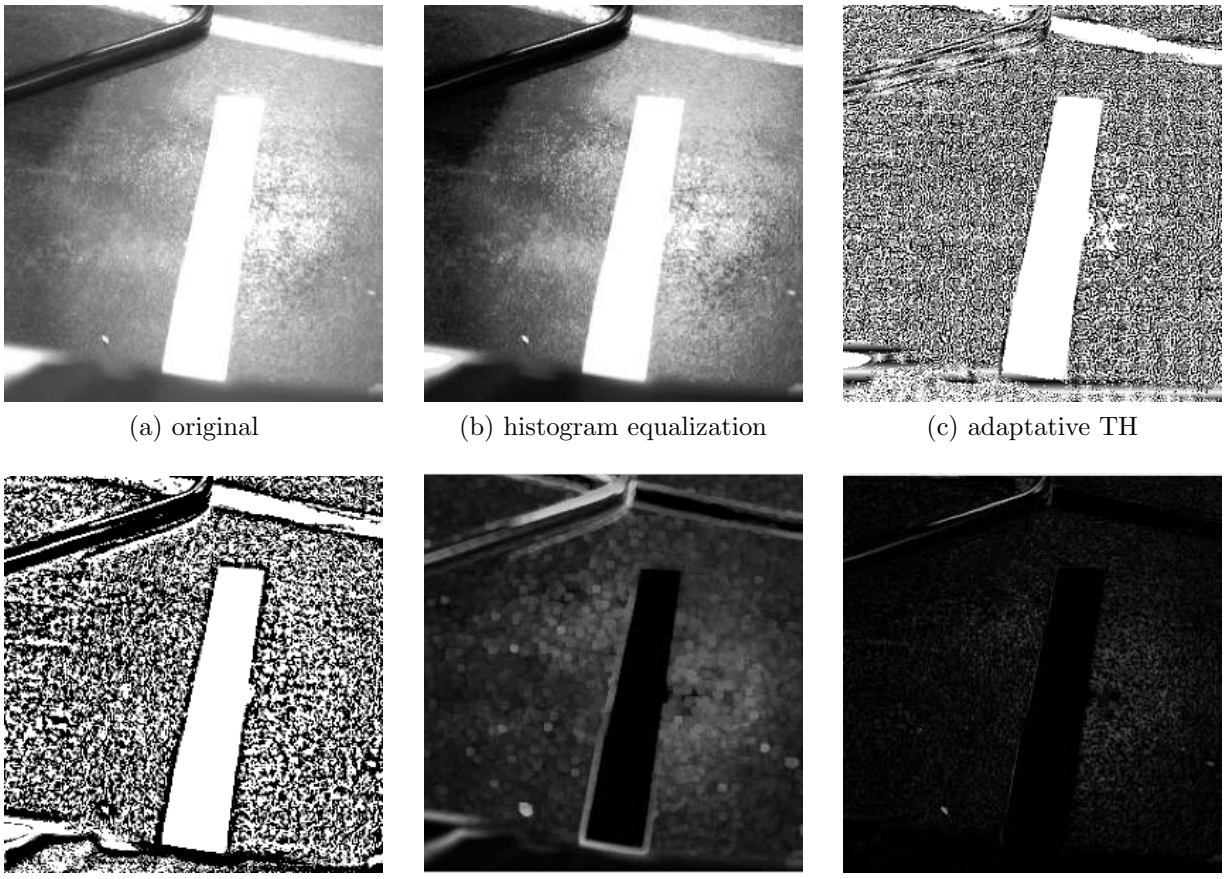
Figure 4.1: Used path to test the navigation algorithm.

4.2 Obtained images

4.2.1 Pre-Processors

Before applying the Canny algorithm it was necessary to apply pre-processing to the image. Figures 4.2, 4.3 and 4.4 illustrate a comparison made with some of the tested pre-processors under different light conditions. On the captions, the acronym **TH** stands

for threshold.



(a) original (b) histogram equalization (c) adaptative TH

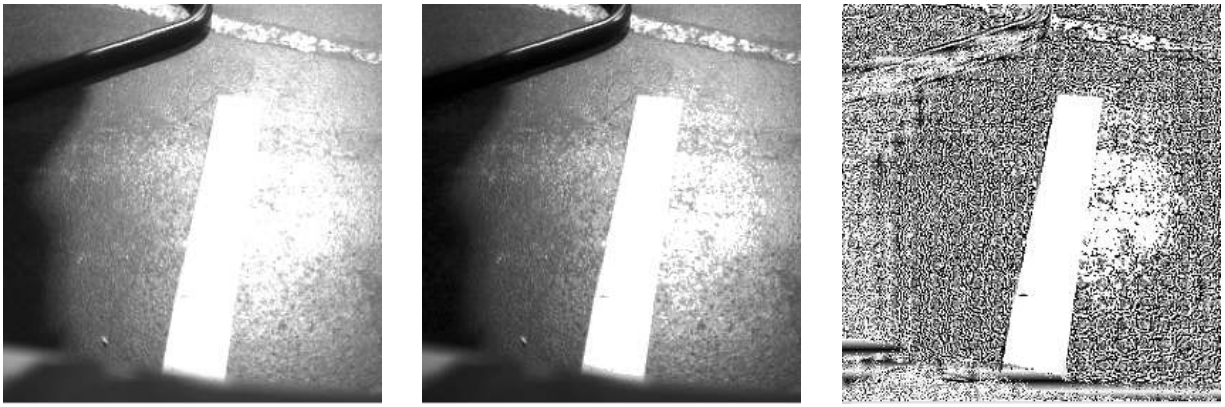
(d) adaptative TH from (b) (e) morphology gradient (f) morphology top-hat

Figure 4.2: Pre-processing algorithms with top incident light.

The corresponding images after the Canny algorithm are in Figures 4.5, 4.6 and 4.7. These figures are arranged in the same way, i.e., the letter designation is correspondent to the pre-processors' results previous sown. It was necessary to perform this comparison to understand which image could have the best response with Hough lines transforms.

4.2.2 Line segment determination

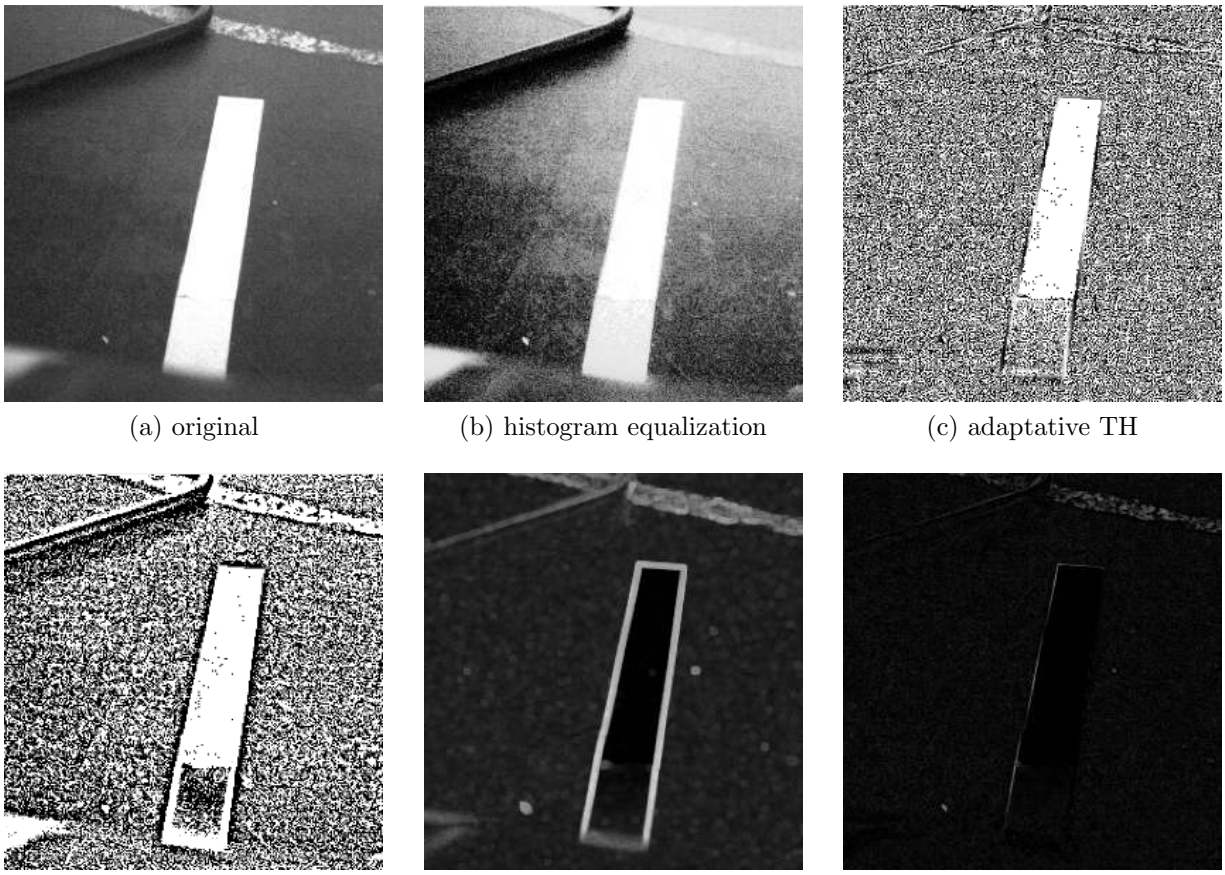
From all of the scenarios it was chosen the method (d) because ir presented better contours. The Hough transform was applied with two methods: the standard transform and the probabilistic, Figure 4.8. Despite better results could be obtained with the probabilistic method, this figure demonstrates that for this particular case is better to use the standard method, because it interpolates the line, avoiding the creation of multiple line segments, which would be more difficult to handle.



(a) original (b) histogram equalization (c) adaptative TH

(d) adaptative TH from (b) (e) morphology gradient (f) morphology top-hat

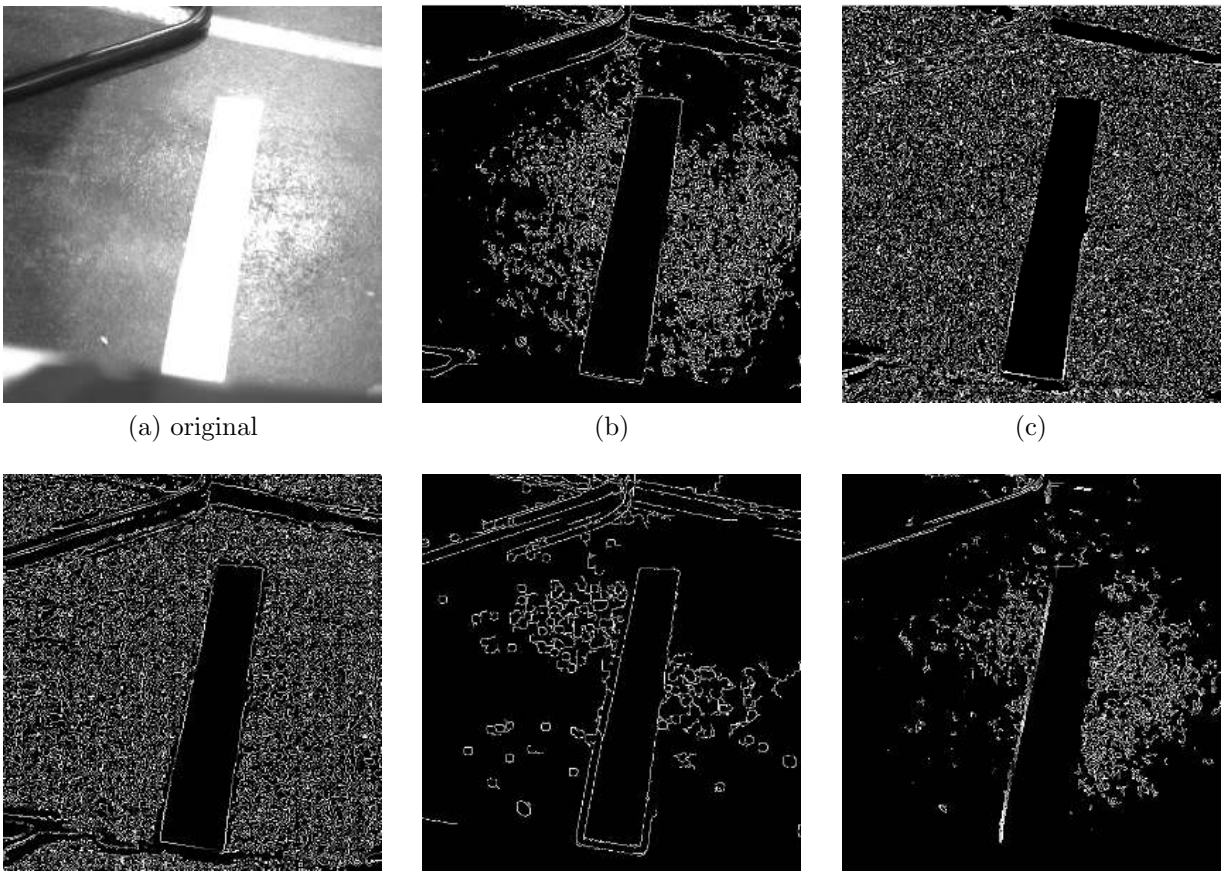
Figure 4.3: Pre-processing algorithms with side light (from the left side).



(a) original (b) histogram equalization (c) adaptative TH

(d) adaptative TH from (b) (e) morphology gradient (f) morphology top-hat

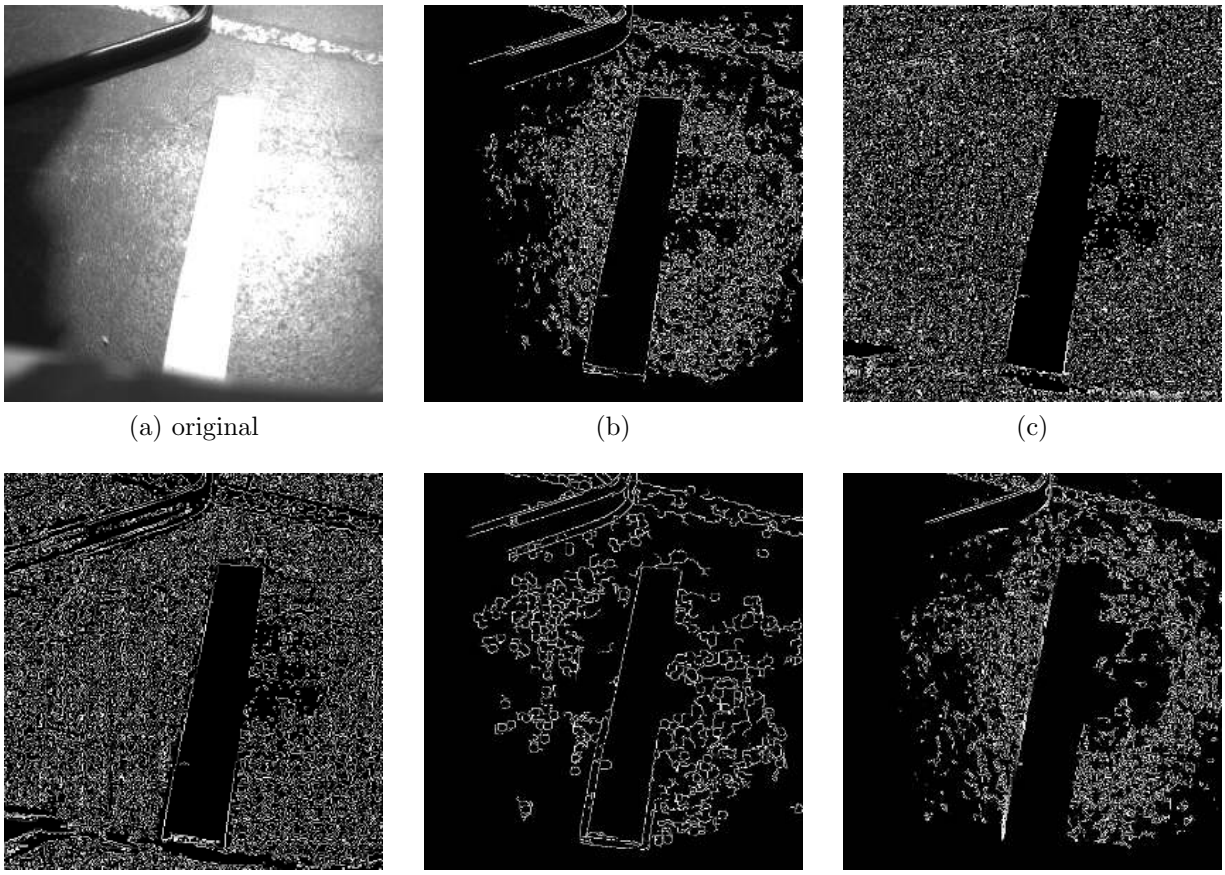
Figure 4.4: Pre-processing algorithms with low light conditions.



(a) original (b) (c)

(d) (e) (f)

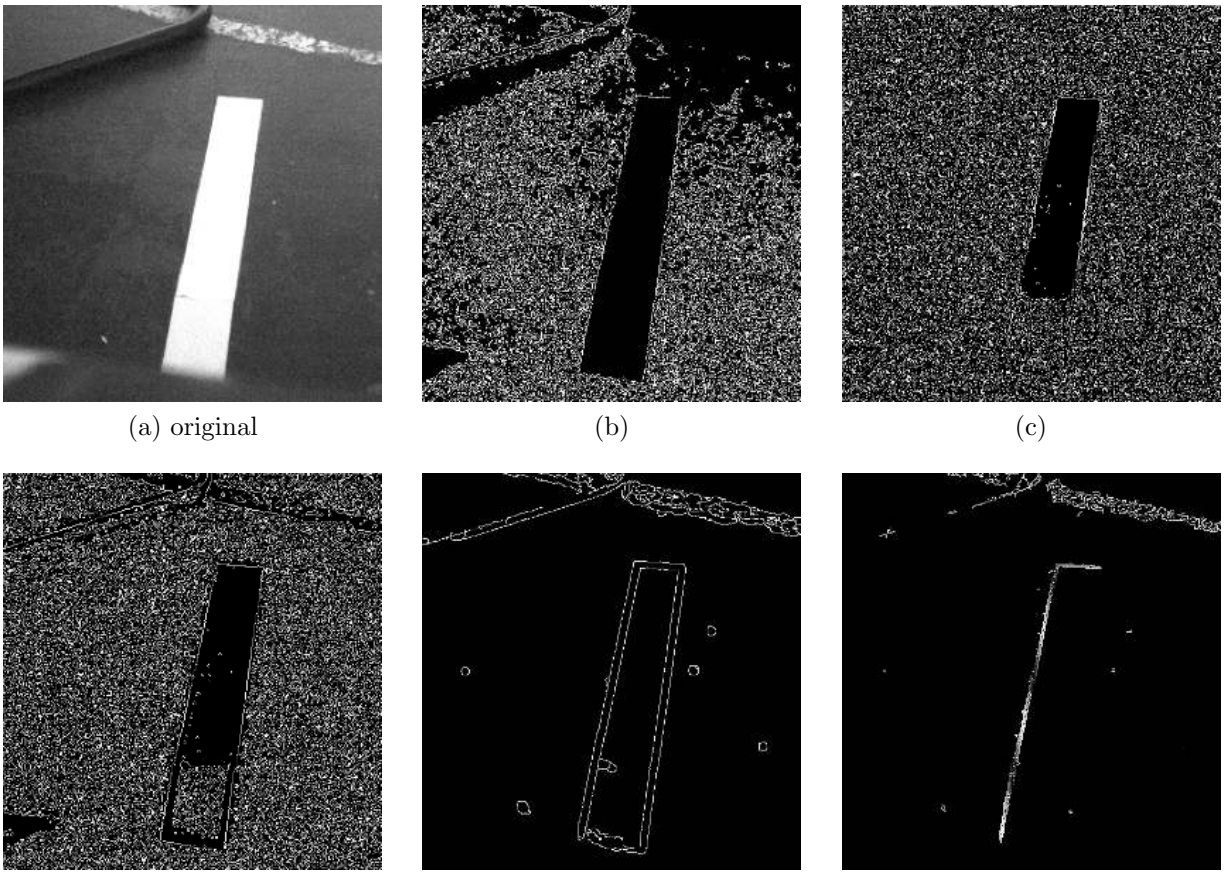
Figure 4.5: Canny edge detection after pre-processing with top incident light.



(a) original (b) (c)

(d) (e) (f)

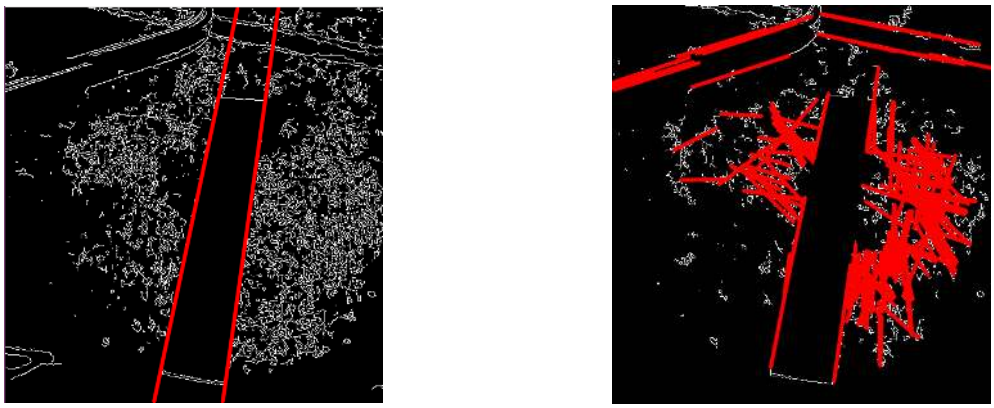
Figure 4.6: Canny edge detection after pre-processing with side light (from the left side).



(a) original (b) (c)

(d) (e) (f)

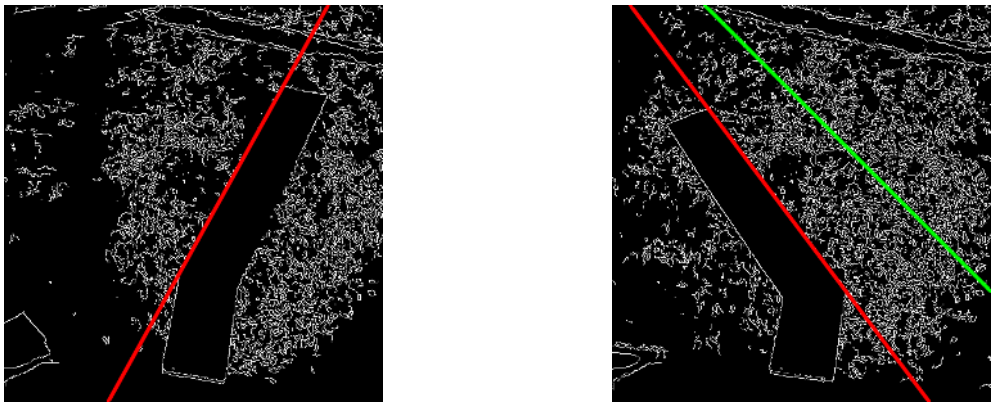
Figure 4.7: Canny edge detection after pre-processing with low light conditions.



(a) (b)

Figure 4.8: Results with standard Hough transform (a) Probabilistic Hough Transform (b).

On Figure 4.9 is shown the same approach but with slopes. On subfigure (b), although two lines were detected, the green one is not used to perform calculations.



(a) (b)

Figure 4.9: Hough transform with right slope (a) and left slope (b).

Besides the presented results, another used approach was the blob detection. It was also possible to identify the line, selecting the biggest area blob, and get his rotation angle through the minimum area rectangle. However, these results are not presented here due to the Hough transforms method was adopted.

Chapter 5

Conclusions

A visual system capable of detect and follow a line was achieved. The velocity rates are set trough two constants which dictates the linear speed and the turning rate.

Since constant and relatively slow velocities were used, it was more simple to design the system, since its response could be based on actuation instead of prediction (which is a more complex approach).

Considering that the laboratory's floor already had several white marks on the floor, the first approach was to use a colored line to perform the detection. This approach was dropped due to the fact that lighting conditions heavily affect the color representation. Since the most common color detection algorithm is based on filters (given an interval to each color, in RGB color system, or space parameters, in HSV), it was verified that it was not a robust technique.

The image processing revealed to be more difficult than expected. Although the available algorithms in the libraries are able to detect a line, the camera is sensitive to the environment conditions, specifically light conditions. Throughout the day, and in the used laboratory, at some point was observed camera saturation, and sometimes, even with the human-eye is difficult to distinguish these areas from the guidance lines. Another factor was the floor's reflection.

5.1 Future Work

While changing the camera fixing place it was detected that, with some platform rearrangements, it is possible to obtain a centered image. In order to do it, it is suggested to fix the camera and PTU inverted, below the front's laser, with an aluminum piece that can be easily designed.

The navigation algorithm can be improved in order to add more features to it, i.e., simple object recognition. Adding this feature would introduce the capacity of making

more complex algorithms, where the platform could take path decisions and set a different velocity rate depending on the recognized objects.

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

Bibliography

[1] Joseph L. Jones and Anita M. Flynn. *Mobile Robots: Inspiration to Implementation, Second Edition*. Natick, MA, USA: A. K. Peters, Ltd., 1998.

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

[3] 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).

[4] *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).

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

[6] *OpenCV library*. URL: http://opencv.org/ (visited on 06/19/2017).

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

[8] 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).

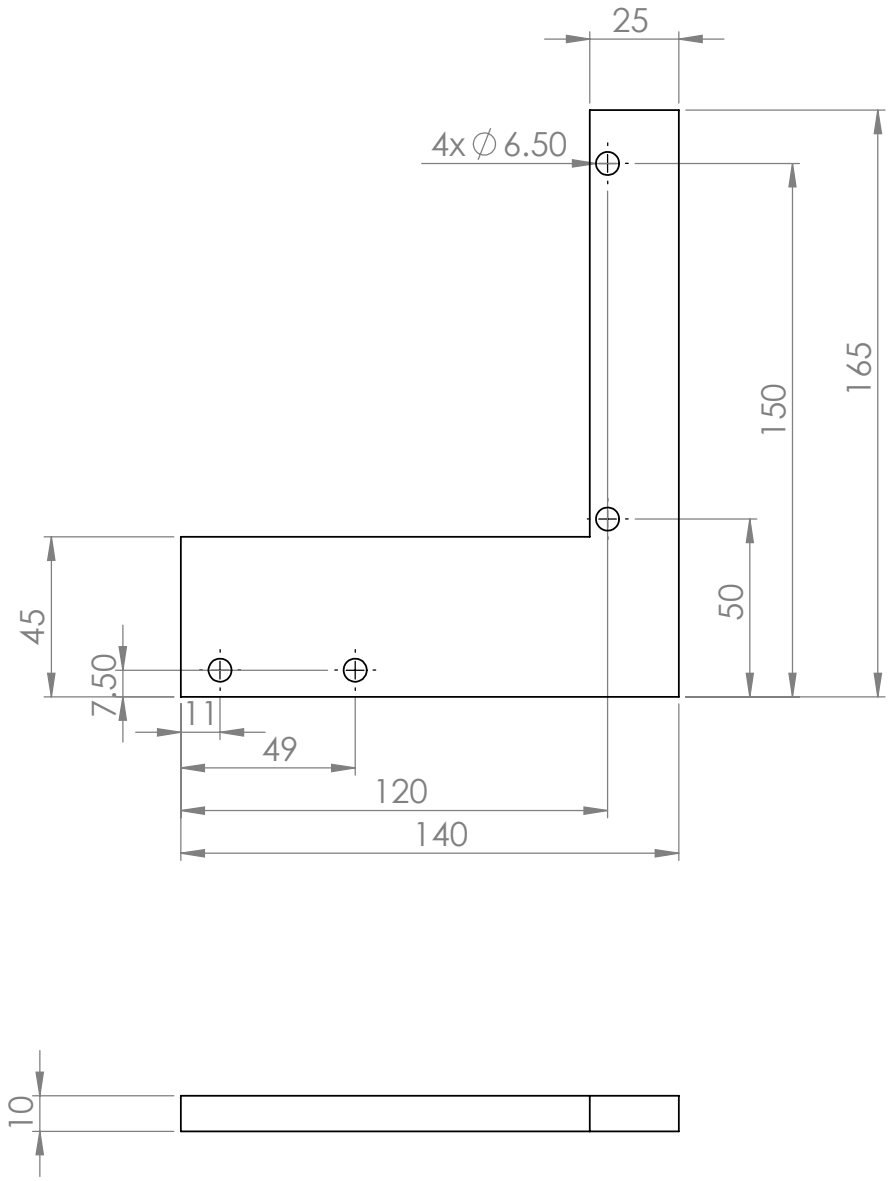
[9] Marcelo Silva Pereira. "Automated calibration of multiple LIDARs and cameras using a moving sphere". MA thesis. Universidade de Aveiro, 2015.

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

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

Appendix A

Technical Drawing for Camera Support



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

	NAME	SIGNATURE	DATE
DRAWN			
CHK'D			
APPV'D			
MFG			
Q.A			
		WEIGHT:	

TITLE:

Author: Bruno Vieira
Tutor: Dr. Vitor Santos

SOLIDWORKS Student Edition.
For Academic Use Only.

DWG NO. PTU support

A4

SCALE:1:2 SHEET 1 OF 1