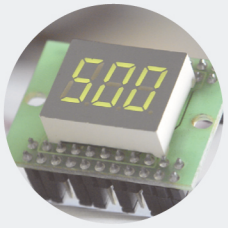
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






DEVELOPMENT OF ADVANCED HUMAN ROBOT INTERACTION SYSTEMS

ALEXANDRE FILIPE MARQUES ABREU
Outubro de 2015

POLITÉCNICO
DO PORTO

DEVELOPMENT OF ADVANCED HUMAN ROBOT INTERACTION SYSTEMS

Alexandre Filipe Marques Abreu

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Automação e Sistemas
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
2015

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Alexandre Filipe Marques Abreu, Nº 1120979, 1120979@isep.ipp.pt
Orientação: André Miguel Pinheiro Dias, ISEP, apd@isep.ipp.pt
Coorientação: Germano Veiga, INESC-TEC - ROBIS, germano.veiga@inescporto.pt

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Automação e Sistemas
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
28 de outubro de 2015

Dedicated to my son, Gonçalo.

Acknowledgements

I thank Professor Manuel Silva and Germano Veiga for all the support provided, their orientation and availability. Also let my thanks to Luís Rocha for his support, availability and advice.

I thank my family, especially my wife, for their patience and their support.

Finally, I would like to thank colleagues at INESC TEC and SARKKIS, who provided me assistance in times of need.

Resumo

O trabalho aqui apresentado é a Dissertação da minha Tese do curso de Mestrado em Engenharia Eletrotécnica e de Computadores do ISEP, realizada em parceria com o INESC TEC. O trabalho consiste no desenvolvimento de um sistema avançado de interação entre homem-robô, usando ferramentas de *software* livres e de domínio público e *hardware* pouco dispendioso e facilmente acessível.

Pretende-se que o sistema desenvolvido possa ser adotado por pequenas ou micro empresas, daí a restrição monetária. Este tipo de empresas tem, por norma, uma capacidade de investimento pequena, e ficam impossibilitadas de aceder a este tipo de sistemas automatizados se estes forem caros. No entanto, o robô continua a ser um componente fundamental, sendo dispendioso.

Os trabalhos realizados pelos sistemas robóticos podem por um lado, ser repetitivos sem necessidade de grandes ajustes; por outro lado, o trabalho a realizar pode ser bastante diverso, sendo necessários bastantes ajustes com (possivelmente) programação do robô. As empresas podem não ter disponível mão-de-obra qualificada para realização da programação do robô. Pretende-se então um sistema de "ensino" que seja simples e rápido. Este trabalho pretende satisfazer as necessidades de um sistema de interação homem-robô intuitivo mesmo para operadores que não estejam familiarizados com a robótica.

Para simplificar a transferência de informação da tarefa a desempenhar pelo sistema robótico é usado um sistema de infravermelhos para delinear a operação a desempenhar, neste caso concreto uma operação de soldadura. O operador usa um apontador com marcadores, a posição destes marcadores é detetada usando duas câmaras para permitir o posicionamento tridimensional no espaço. As câmaras possuem filtros infravermelhos para separar o espectro de luz.

Para o controlo do sistema e interface com o robô é usado um computador de baixos recursos computacionais e energéticos, e também de baixo custo. O sistema desenvolvido é portanto computacionalmente leve para poder ser executado neste computador.

Palavras-Chave

Interfaces Homem-Robô, visão artificial, sistemas de posicionamento, robótica.

Abstract

The work presented here is the report of my Master's thesis in Electrical and Computers Engineering from ISEP, developed in partnership with INESC TEC. The goal is to develop an advanced interaction system between man and robot, using public domain free software tools and inexpensive hardware easily accessible.

It is intended that the developed system can be adopted by small or micro businesses, hence the monetary restriction. Such companies normally have a small investment capacity, and are unable to access this type of automated systems if they are costly. However, the robot continues to be a fundamental component, being expensive.

The work carried out by robotic systems can be, on one hand, repeated without the need for major adjustments; on the other hand, the work involved can be quite diverse and plenty of necessary adjustments needed, possibly involving robot programming. Companies may not have available skilled workers to perform the robot programming. So it is pretended a "teaching" system, that is simple and fast. This work aims to meet the needs of an intuitive human-robot interaction system even if the operator is not familiar with robotics.

To simplify the task of information transfer to be performed by the robotic system is used an infrared system to delineate the operation to play, in this case a welding operation. The operator uses a pointer with markers, and the position of these markers is detected using two cameras to enable determining the three dimensional position in space. The cameras have infrared filters to separate the light spectrum.

For system control and interface with the robot is used a low computational and low power computer, but also low-cost. The developed system is therefore light to be executed on this computer.

Keywords

Human-Robot interfaces, computer vision, tracking systems, robotics.

Table of Contents

Table of Figures

Index of Tables

Acronyms

1-G	-	One Gaussian
2D	-	Two Dimensional
3D	-	Tridimensional
API	-	Application Programming Interface
ARM	-	Advanced RISC Machine
ASIC	-	Application Specific Integrated Circuits
CLARISSA	-	Cooperative DuaL-Arm Robot for Structural Steel fAbrication
CPU	-	Central Processing Unit
DC	-	Direct Current
DSP	-	Digital Signal Processors
FET	-	Field Effect Transistor
FIFO	-	First In, First Out
FPGA	-	Field Programmable Gate Array
GFLOPS	-	Giga FLoating-point Operations Per Second
GigE	-	Gigabit Ethernet
GND	-	GrouND
GNU	-	GNU's Not Unix
GPIO	-	General Purpose Input/Output

GPU	-	Graphics Processing Unit
HD	-	High-Definition
HDMI	-	High-Definition Multimedia Interface
I/O	-	Input/Output
I2C	-	Inter-Integrated Circuit
ICSP	-	In-Circuit Serial Programming
INESC TEC	-	INstituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência
IoT	-	Internet of Things
IP	-	Internet Protocol
IPC	-	Inter-Process Communication
IR	-	InfraRed
LED	-	Light Emitting Diode
OpenCV	-	Open source Computer Vision library
PbD	-	Programming by Demonstration
RAM	-	Random Access Memory
RCA	-	Radio Corporation of America
RGB	-	Red, Green, Blue
ROBIS	-	Unidade de ROBótica e sistemas InteligenteS
SCL	-	Serial Clock Line
SD	-	Secure Digital

SDA	-	Serial DAta line
SME	-	Small and Medium Enterprises
SoC	-	System on a Chip
SPI	-	Serial Peripheral Interface
SS	-	Slave Select
TCP/IP	-	Transmission Control Protocol/Internet Protocol
TIY	-	Track It Yourself
UART	-	Universal Asynchronous Receiver/Transmitter
UCD	-	User Centered Design
USB	-	Universal Serial Bus
VGA	-	Video Graphics Array

1. INTRODUCTION

Currently, industrial robots are not frequently used in small and medium-sized companies because they are expensive and require skilled labor to do their programming. There is, therefore, a need to develop more affordable robotic systems, with simpler programming methods, for this type of companies.

In addition, the hardware costs have decreased in recent years, but robots and some other industrial equipment, continue to cost large amounts of money. Another problem with the robotic systems is the need of skilled labor for their programming. To maximize business productivity and increase the ease of programming or reprogramming robots, it is necessary to adopt innovative methods using the technology available as aid. One way to do this, is using augmented reality and new interfaces for human-machine interaction [1].

If the production of a company consists of small batches and short production cycles, typically there are not used robotic systems due to their cost and the possibility of no skilled operators available. If the cost of robotic systems is decreased and/or it is easier to operate the robotic systems with reduced setup times, then it is possible to use this type of automation to maximize and standardize production.

The robot programming consisting on more conventional interfaces can consume too much time, it is difficult and error prone. In the case of using more user-friendly interfaces, such as the case of the proposed interface in this work, it is achieved a more versatile and broad robotic system. The human-robot interface developed in this project allows programming a robot for various tasks using an intuitive method, using only a pointer to the robot teaching.

Being the robot teaching made with a pointing device, it is used a computer vision system as input in the project. It is also used a computer in which data is processed by software that obtains the pointer position in the three-dimensional space. This data is then used to create trajectories of the robotic arm, his gripper, welding torch or other end-effector tool.

1.1. CONTEXTUALIZATION

This project arose from a thesis proposal provided by Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência (INESC TEC). The project covers areas such as robotics, computer vision systems, microcontrollers, and programming, some interesting areas that are very useful for both the manufacturing industry and other miscellaneous areas. The project itself is an excellent final project for the Master Course in Electrical and Computer Engineering, expertise in automation and systems. As my career has been in the area of industrial automation, this project is an added value in my personal, academic and professional development.

This project is developed in partnership with the SME Robotics Consortium [2], the European robotics initiative whose main goal is to strengthen the competitiveness of Small and Medium Enterprises (SME) in the area of manufacturing using cognitive robotics. INESC TEC is coordinating the development of the Cooperative DuaL-Arm Robot for Structural Steel fAbrication (CLARISSA) (Figure 1), where this thesis is included, in which the main objective is to develop a dual arm robotic manipulator for welding metal structures. There are more two Portuguese companies on the project, SARKKIS Robotics [3] and NORFERSteel [4].

My experience in robotics, microcontrollers and programming is quite reasonable. However, I only had basic knowledge in machine vision, so this project allowed me to deepen my knowledge in areas already known and learn new things, which enrich my training.

This project also arises from the need of providing an alternative to conventional robotic production systems, allowing a simpler and more widespread use of robots for industrial production.

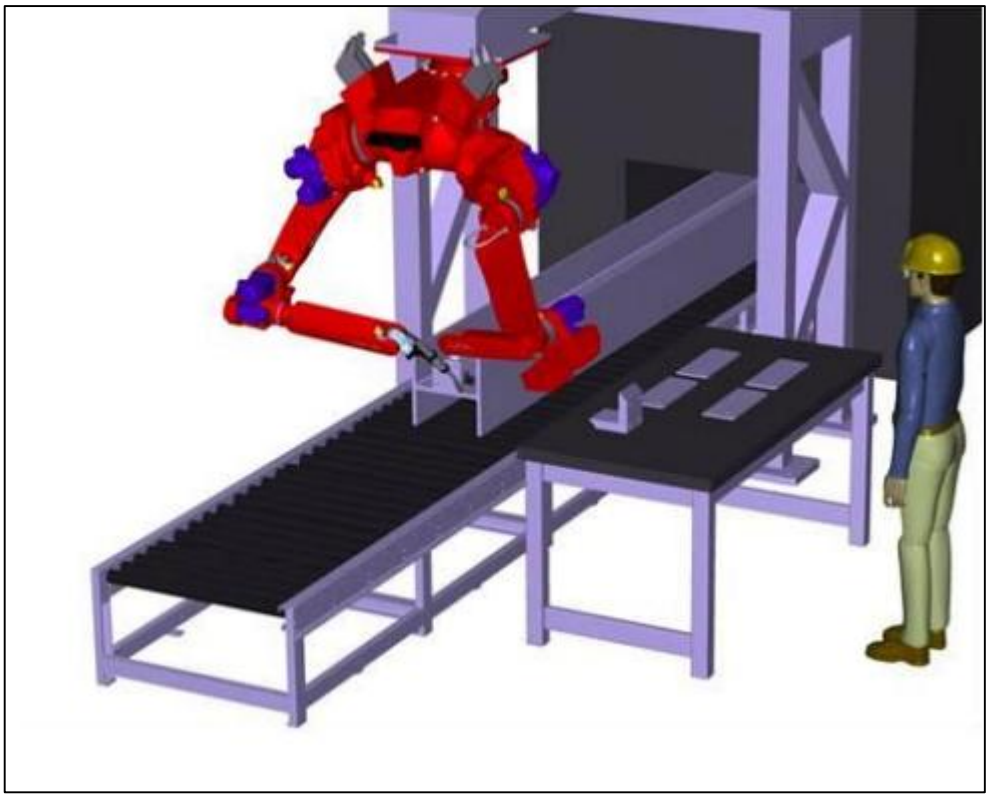


Figure 1 CLARISSA: Cooperative DuaL-Arm Robot for Structural Steel fAbrication.

1.2. GOALS

The goal of this project is to develop a form of interaction between a human operator and a robot that allows fast and easy programming of the robotic system.

The main task was to develop an infrared teaching system for part positioning and correction.

Initially it was selected the software and hardware required for the project, given that it was required to develop a marker tracking system using computer vision. In the sequel is calculated a trajectory for the robot from the marker position and its movement, during the teaching.

Two cameras are used to allow for sensing the marker three-dimensional position, being necessary to calibrate the cameras. The next step is to develop an application that collects data from the cameras.

The information from the cameras is then made available to the application responsible for making the triangulation and getting the object tridimensional (3D) position.

Finally is required an application that provides data to the robot controller to determine the desired trajectory.

1.3. PROJECT SCHEDULE

The need to organize this project in a timeline led to the schedule presented in Table 1. Upon acceptance of the project, an initial research on the subject is required, namely on human-robot interfaces, computer vision, and tracking systems. It is also advisable to analyze other types of similar, or alternative, projects. After the initial research concluded, it is required to analyze the available software and hardware in the market and make its selection.

After the technologies to use had been chosen, it was started the development of the application of information acquisition from the cameras, and their calibration thereof. Next, was performed the information processing and provided the required data to the application that calculates the object 3D position. After the software development, it was tested its operation.

Finally were implemented the infrared filters in the cameras and tested an infrared marker used by a human operator to describe the robot trajectories. It was also required to develop an application that provides the useful data to the robot controller. Lastly were performed functional tests of the final system and its validation.

1.4. REPORT ORGANIZATION

This Dissertation is organized in eight chapters and four annexes.

Chapter 1 is an introduction to the developed project. Chapter 2 is devoted to human-robot interfaces, while in Chapter 3 are analyzed the tracking systems, and in Chapter 4 is presented some information relating to computer vision systems. Chapter 5 is devoted to the system architecture, where the software and hardware technologies used in this project are presented. Chapter 6 is devoted to the project implementation, describing the various steps made. Chapter 7 is dedicated to performed tests and the system validation and, finally, in Chapter 8 are presented the main conclusions drawn from this project and some ideas for future developments.

Table 1 Project timeline.

2. HUMAN-ROBOT INTERFACE

In this chapter are presented some ideas to develop efficient human-robot interfaces, and some issues that may arise in their development. It is also presented the Programing by Demonstration (PbD) methodology that fits to the project, taken into account that the human-robot interface should be as simple and user-friendly as possible.

2.1. DEVELOPING A HUMAN-ROBOT INTERFACES

In recent years, there has been performed much research regarding human-robot interfaces in an attempt to make these interfaces effective, efficient and usable resorting to the inclusion of a perspective view by the user throughout the design and development process.

Sometimes interfaces are developed later in the design and development process, supporting an incomplete user interaction. As a result, these developed interfaces, tends not to meet all the requirements or the users are unwilling to accept the technology.

Then are listed some principles that should be followed in the creation of human-robot interfaces [5]:

- focus on users and their tasks and not the technology;

- consider the functionality first and the presentation after;
- adapting the interface to the user's view of the task at hand;
- do not complicate the user task;
- promote learning;
- provide information, not just data;
- create a responsive interface;
- perform tests with end users and correct any faults or inaccuracies.

The incorporation of the user in the design process has been known in recent years as User Centered Design (UCD) (Figure 2). In addition to this user centered design approach, the research related to human-machine interfaces used in air traffic control systems, cockpits, or industrial systems, originated many theories and results that can be used for development of human-robot interfaces [5].

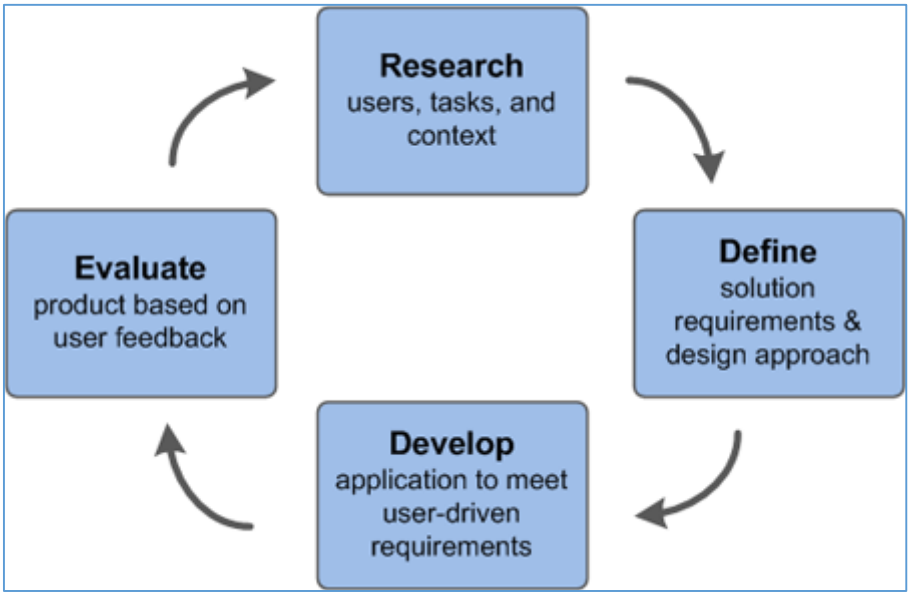


Figure 2 User Centered Design [6].

One goal of the developers is to create a user-friendly interface, that is, that satisfies user needs and consider their weaknesses. The programmer needs to understand the functioning of the machines and the needs of users.

2.1.1. USER CENTERED DESIGN

The user centered design is a philosophy and a process. It is a philosophy that puts a person (not a thing) in the interface recipient; is a process that focuses on cognitive factors, such as perception, memory, learning, troubleshooting, etc., with regard to the interaction between people and things.

The purpose of user centered design is to understand the user, and the tasks and goals that need to be met by the robotic system. This knowledge is then applied to the design and development of the interface that will serve the user to work with the system.

Next are presented some questions that serve to guide the programmer in the interface development process [5]:

- who are the users of this system?
- what are the tasks and goals of the users?
- what is the users experience level on the system?
- what are the features that the user needs of the system?
- how users think that the system should work?
- how can the interface facilitate cognitive user process?

The main principle of the user centered design is that the user plays an integral role during all phases of specification, design, development, and testing. There are many ways to integrate the user in the development cycle. The first step is to identify who exactly are the users. If these users are a well-defined group, such as a set of electronics engineering students with extensive knowledge of computer tools, then the task to include these users in the development process is beneficial. On the other hand, if users do not form a well-defined group, then the task of identifying and including this group of users in the development process is much more complicated because of the diversity and disparity in characteristics of different users of the group. In the case of human-robot interfaces, programmers must have a good knowledge of the user group to which the system is intended.

A principle for the development of an effective, efficient and implementable interface is to comply with the needs and requirements of users. The initial consideration of user centered design is to examine these needs and requirements.

There are some simple considerations that the development team can observe before working with end users. One of these considerations are environmental constraints and how the environment will restrict the design of the interface. Environmental restrictions are, for instance, the location, noise level, temperature or accessibility. In case of a voice control interface, it becomes difficult or even impossible to use it in a very noisy environment.

After environmental and other preliminary considerations are understood, the user centered design initially consists in collecting user information, focusing on the analysis of this representative group, and analysis of the tasks that the system must implement.

The participants, group of users and development team, should discuss the potential system to be developed and characteristics of interface, while analyzing the opinions and identify potential problems. Participants must follow a free and unstructured discussion as a moderator follows a schedule and organizes the topics of discussion.

The initial data collection should also check how users perform tasks at present in their environments, with existing tools and techniques. This study serves to identify how users get around the problems and as a way to provide a reference for the desired features in the system to be implemented.

The analysis of the tasks allows checking how users perform a specific task. The collected data can be obtained by interviews, asking users to do their daily tasks, user observation, or using contextual search. The result of the task analysis is a representation of all the tasks that must be performed to carry out their work, including the information needed to complete the tasks, the steps needed to complete each task, the interdependence between tasks and its steps, and the expected result when the task is completed. This data serves to aid the system design.

The system design should include detailed information about the user interface design. The developers sometimes fail in the representation of the target user group, and the design of the user interface rarely matches the needs and requirements of the users.

The development of human-robot interfaces prototypes should be made early during the design phase so that validation tests can be done to check the interface design. Validation tests should be made in each version of the design whenever possible. The interface should be implemented only after being assured that the design meets the requirements. After the design phase, the tests should be done throughout the development cycle [5].

2.1.2. IMPORTANCE OF THE USER INTERFACE DESIGN

The design of the human-robot interface can directly affect the ability and willingness of the operator to perform the necessary task using the robotic system. The interface design also affects the operator's ability to understand the state of the situation, make decisions, and to supervise and submit high-level commands to the robotic system. One can spend a certain amount of time discussing and experiencing different interaction techniques, but there is plenty of research with regard to human factors that can affect the human-robot interfaces. This research is related to the ability of decision making by humans, knowledge of the situation, monitoring, workload levels and human error. All these aspects must be considered when developing a human-robot interface [5].

2.1.3. HUMAN DECISION MAKING

The area of human decision making appears to be somewhat unexplored. Humans do hundreds, or even thousands, of decisions every day. These decisions are made quickly in dynamic environments, with changing conditions. Depending on the current task at hand, these decisions can have serious consequences if they are determined incorrectly. There must be an understanding of the human decision making process, and this information must be incorporated into the interface design.

2.1.4. VIGILANCE

Vigilance can be understood by the task on which an operator is required to detect signals for a longer period of time, being these signals intermittent and unpredictable infrequent. Other characteristics of vigilance are sustained attention, detection, be alert, be able to detect goals to achieve, and maintain performance over time. The activity of vigilance can be affected by several factors. If the information given by sensors and/or operator tasks is infrequent and/or intermittent, the operator may be bored by the lack of stimulation and excitement. This can lead to mental disengagement or mental drowsiness. If the operator’s

vigilance level is low, it is also possible that control of the situation by the operator is reduced, affecting the capacity of making decisions. There is also the possibility of arising a lot of information and/or many tasks, leading the operator to work hard in order to stay updated.

While the operator is working on a particular task, it is possible that it does not keep watch over the entire system. In this case, the operator's ability to make decisions may be adversely affected.

In addition to the tasks and data provided by the sensors, the operator's vigilance can be affected by other factors. The case of lack of sleep, or circadian rhythms, adversely affects surveillance.

2.1.5. WORKLOAD OF THE OPERATOR

The workload may refer to the mental and cognitive work and also to physical labor. In general, a high workload can lead to reduced vigilance since the operator attempts to maintain the accuracy and the judgment under the pressure of time and the received information. Low workload can lead the operator to lower the vigilance, due to low stimulation.

The mental or cognitive workload is the amount of information provided to the operator, during the execution of tasks, and every human being has a certain mental processing capacity. This mental capacity may be affected by the lack of tasks to be performed, stress, lack of sleep, environmental conditions, and missing information. A high workload may cause the operator to take wrong decisions that could cause disastrous situations such as accidents. Low workload can cause the operator fail to provide the required attention to the system and he may lose control of the situation. It can be said that the more pressure there is on operator in a given situation, greater will be the workload [5].

In the design of human-robot interface, special attention should be given to the interaction between the operator and the robotic system. The mental workload should not be considered insufficient or in excessive amount.

2.1.6. SITUATION AWARENESS

Another important aspect in the development of human-robot interfaces is to maintain awareness of the situation by the operator. Awareness of the situation is the operator's ability to understand the activities performed by the robotic system in the current environment, that is, the perception of the elements in the environment involved in a volume of time and space, the comprehension of their meaning and the projection of its status in the near future [7].

The operator remains informed from his perception of the situation and from the data received from the robotic system. If the received data does not completely describe the operation of the robotic system, and the operator is not present in the robot workspace, awareness of the situation is adversely affected. The same is true if the data provided to the operator is too much. Another problem is that if several operators are required for controlling the robotic system, in this special case, care is needed in developing the interface to allow a good perception of the situation by the operators.

The design of the human-robot interface should attempt to provide the appropriate information to certain situations, distribute tasks among different operators and be aware of operators' capacities [5].

2.1.7. HUMAN ERROR

Human error is the main cause of many accidents and can be defined as a decision, or inappropriate or undesirable human behavior that reduces, or has the potential to reduce the effectiveness, safety, or system performance.

Scientific research discusses the use of robots in cooperation with humans. In many cases the robots work with humans as in the case of rescue situations or military use. Human error is a concern in these cases. Initially, the investigations were focused more on the dangers that robots accounted for humans, but the case of failures caused in robotic systems due to human error must also be considered [8].

2.2. PROGRAMMING BY DEMONSTRATION

In the robotics area, Programming by Demonstration (PbD) allows teaching the robotic system by demonstrating how to perform the task, instead of programming this task using

the commands of programming languages (Figure 3). The main argument in favor of PbD is to enable accelerated learning by providing an optimal solution.

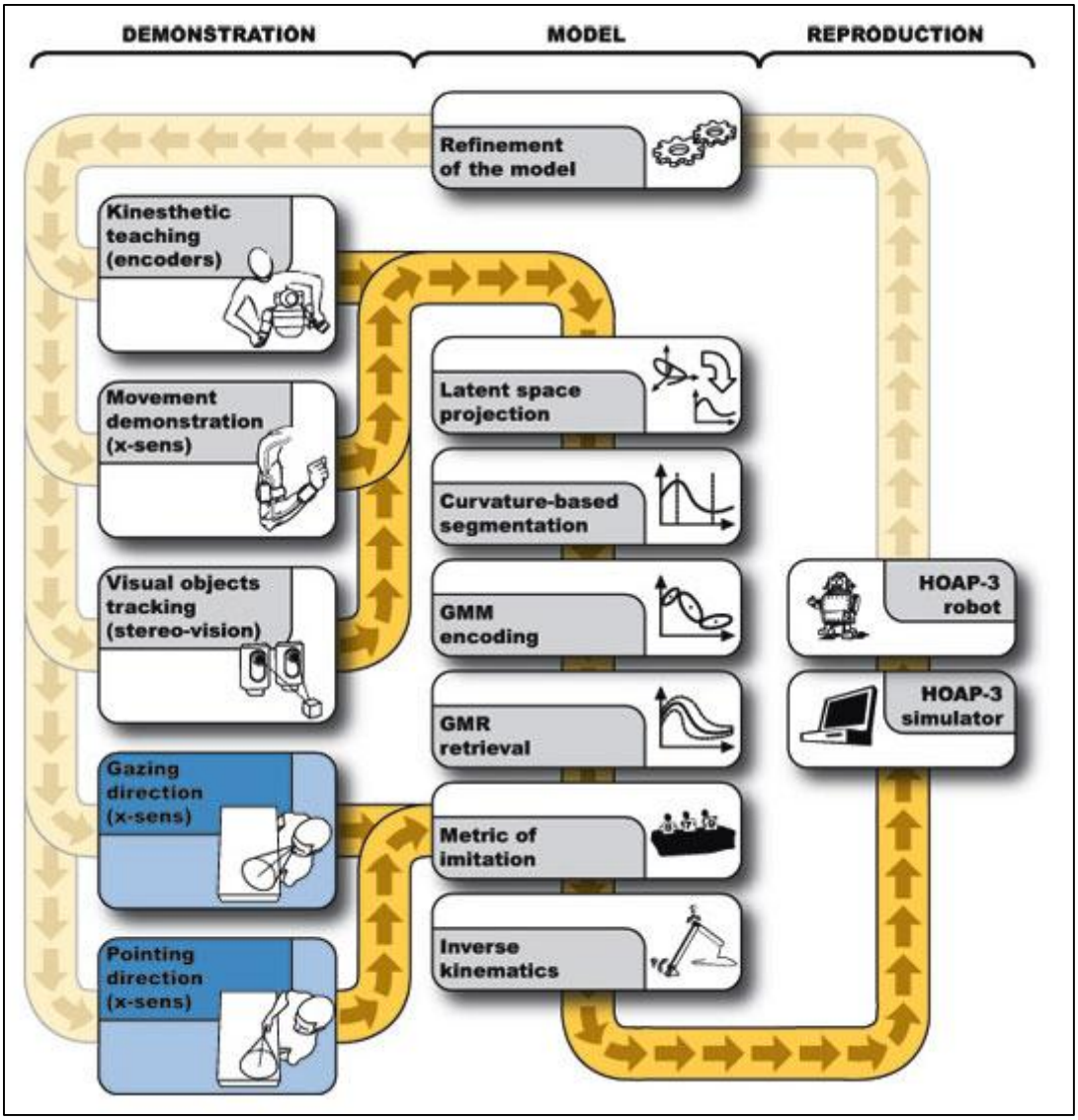


Figure 3 PbD example overview [9].

With this methodology, an operator may do robot programming without having to learn a programming language.

In the case of this project, the operator performs a set of actions (showing a path to the robot describing a welding path, or other operation to perform) that are collected from the implemented tracking system, using computer vision, and provided to the robot. The robot learns the task and then can perform the desired operations autonomously.

Using PbD techniques, the costs of development and maintenance of programs can be reduced. The programming of robots can be performed by operators who know how to

perform the desired task but do not need themselves to have knowledge of robotics or programming languages.

The PbD appeared in the early '80s, frequently used with teach-in, guiding, or play-back methods. The position of the handler and the forces applied were stored with the positions and orientations of obstacles and targets [10]

The latest advances in the PbD area occurred mainly in robots teaching interfaces. More traditional forms of guiding and teleoperating robots are being replaced by more user-friendly interfaces, where are used computer vision systems, data gloves, laser range finders, and also the manual guiding of the robot arms, if it permits it.

2.2.1. IMITATION METHOD

A form of PbD is the imitation, in which case, a demonstrator (human operator) performs a certain movements set and an imitator (robotic system) learns these movements. The demonstrator uses a particular pointer that represents the intent of this, and the imitator watches and learns. The imitation method is oriented to the goal, that is, the actions are designed to fulfill a specific purpose that is the goal and intention of the demonstrator.

The imitation method may not be restricted to simply follow a path and replication of the same, being currently already possible to implement systems that try to understand the objectives of a particular action, so it is not only limited to watch and play.

Learning by imitation depends very much on the ability of the imitator to infer the demonstrator's intentions. There may be errors or ambiguities in the demonstration, the demonstrator may not be able to express what he wants, or the imitator cannot properly infer the pretended. It is therefore required, if there is a possibility of such problems, to implement complementary methods, to permit complete the desired solution.

Another problem identical to that described above, is that there may be several different ways to achieve the desired goal. Thus, the method followed by the demonstrator may not be the ideal to be followed by the imitator. Likewise, there must exist complementary human-robot interface methods provided to solve these problems.

2.3. CHAPTER CONCLUSION

As can be seen, studies have been done and technologies have been implemented, that allows to perform human-robot interfaces, so that they can be used to automate systems, and can be operated in a simple way even by human operators without robotic expertise.

3. TRACKING SYSTEMS

In this chapter, it is given a general overview on the tracking systems and is described the used tracking system in this project.

3.1. TRACKING SYSTEMS

Tracking systems are commonly used to observe moving objects and provide its location in a timely fashion. Some examples of tracking systems are video tracking, sonar tracking, the air traffic control tracking, vehicle tracking, among others.

Various types of sensors can be used to perform tracking, which can be listed some examples [11] [12] [13]:

- sonars: to get the pose and orientation of an object, one must use at least three sonars using the triangulation method for tracking. They normally have a big range, but are susceptible to noise;

- inertial sensors: sensors such as accelerometers or gyroscopes are available on the market in the form of integrated circuits and can be easily integrated with electronic components. They measure accelerations, from which can be calculated the position and orientation of the object. They have low latency and are robust with respect to environmental noise, but may not provide great accuracy;

- mechanical movement sensors: encoders or potentiometers can be used to measure rotation, for example a rotation of a motor, or the positioning of an object;

- optical sensors: the tracking systems based on optics consist of one or more light sources and one or more optical sensors. The light source can be ambient light or artificially generated light. A common type of optical sensors are the cameras that can have dedicated filters to capture a particular spectrum of light, like the case of ultraviolet light.

In this project, it is performed 3D tracking of one or more objects, more precisely its position and orientation tracking, using video based object tracking.

The 3D tracking can serve various applications such as human-computer interaction, robot learning by demonstration, or recognition of human activity. It can be used markers to perform tracking, or not use markers at all.

The video tracking systems that do not use markers can be divided into two main groups: model-based and appearance-based. The model-based systems continuously provide solutions but are computationally expensive and depend on good visual information (typically provided by multi-camera systems). Appearance-based systems require a lower computational power and hardware complexity but offer only a limited and discrete number of positions of the detected object [14].

3.1.1. VIDEO BASED OBJECT TRACKING

Typically, the video based object tracking has as input, a continuous stream of non-stationary images that is changing over time, been widely used for objects tracking in real time.

Multiple objects can be detected in the analyzed scene, or one may want to detect only one object among many others. For tracking purposes, it is required to analyze video sequences to find (track) the objects in each of the frames.

One common application on video based object tracking, is in robotics and industry, being used to detect moving objects in an unconstrained environment. The video based tracking allows more flexibility than other methods that only rely on common presence or position sensors like the ones described earlier [15].

3.1.2. ALGORITHMS

Next, the main algorithms used in this project are presented.

Background subtraction algorithm

The background subtraction algorithm (Figure 4) is based on the background of a scene at rest, where will be later detected the intended objects. This background is known for background image or background model. Normally, the camera is static and the identification of objects is done by detecting areas on frames sequence that are changing their position over time, being calculated the difference between the image background and the current frame [15].

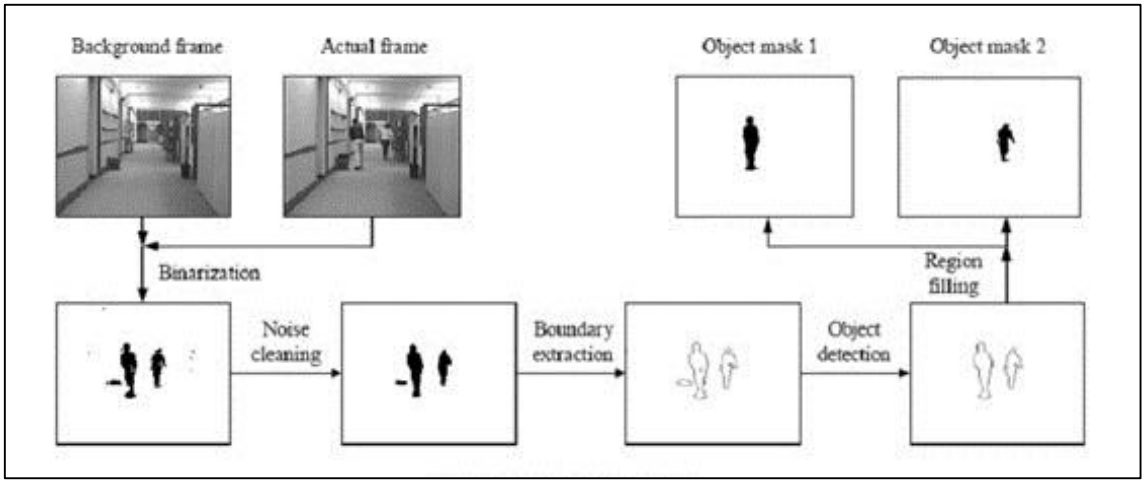


Figure 4 Background subtraction algorithm example [16].

In the background subtraction algorithm it is required to consider some situations:

- choosing the size of the feature (a pixel, a block or cluster);
- choose the type of feature (feature color, the feature corners, stereo features, the feature movement, and feature texture).

The method for getting the background is known as background modeling or representation, and describes the type of model used in the image background. The simplest method of background modeling is to acquire an image that does not include any moving object. There are, however, some restrictions or problems. In some environments the background may not be available or may be changed in a dramatic way due to changes in brightness, objects being inserted or removed from the scene, or reflections from mirrored surfaces, etc. [17].

After getting the background, it is done the background initialization that initializes the model.

During operation of the tracking algorithm, it is appropriate to make the maintenance of the background which consists in adapting the model in relation to changes in the scene during the time. The maintenance can be subdivided into the following topics [17]:

- maintenance rules that are used to adapt the image background;
- learning rates that determine the speed of adjustment in relation to changes occurred in the scene;
- adaptation rates that determine the frequency of maintenance.

The foreground detection consists in the classification of pixels into which they may belong to the background or foreground, i.e., it consists in detecting objects in the scene. There are many foreground detection algorithms, among them: basic motion detection, one Gaussian (1-G), MinMax inter-frame difference, Gaussian mixture model, kernel density estimation or the codebook [18].

Tracking based on color

Tracking based on color lets one isolate a particular color in a frame and return its position and orientation. It is one of the simplest implementations of the background subtraction algorithm, having low computational costs and being suitable for more modest systems in terms of processing power.

All colors have a certain level of red, green and blue (three channels) that form a single color. That is, to perform tracking it is required to define minimum and maximum values for each of the channels, taking into account that the brightness is not completely uniform or even the color of an object, and can therefore be a small change in color perception by the tracking system [19].

Once defined the color (or colors) to detect, various algorithms can be used to process frames and return the detected objects. A simple algorithm for doing this is the image scanning, pixel by pixel, detecting the pixels with the desired color, being these pixels grouped in case they belong to an object with more than one pixel in size. Finally the position of the object

is returned - in the Pixy camera case, used in this project, it is returned the center position of the object. Noise filters may also be used to improve the performance of the algorithms.

3.2. TRACK IT YOURSELF

The Track It Yourself (TIY) is a tool that allows the easily development of a 3D tracking system. The TIY is open-source, runs on Windows and Linux platforms, and can detect multiple objects simultaneously. With TIY, one can mount a low cost computer vision system since it works with a variety of cameras, from simple webcams to industrial cameras. It contains ready for use examples, without the need for additional hardware, and an acceptable operating cycle of approximately 10 ms, available with the original setup. To build a simple computer vision system with TIY, are only needed two cameras, with infrared detection capability, and an object with markers to detect.

The object 3D tracking is achieved by using triangulation from the feedback of the two cameras. To detect an object 3D position it is necessary to place multiple markers in it, and save their initially disposal as a template. During operation of TIY, the software compares the current position of the markers for each instant time, to the arrangement previously stored in the template. With this comparison, it is determined the position and orientation of the object in three-dimensional space.

Once calibrated the cameras, and also the objects, then the software takes care to provide objects 3D positioning. The purpose of using TIY in this project is to provide a simple way to collect the 3D positioning of a marker [20] [21].

Some of the more important features of TIY are as follows [20]:

- support for Open source Computer Vision (OpenCV) and Aravis Gigabit Ethernet (GigE) cameras (these last only supported on Linux);
- ability of sending data over a network to multiple computers;
- ability to log data, capture images and record videos;
- ability to use videos or two dimensional (2D) points located in a file as input, instead of using cameras. In this project, the data is provided to TIY by files with 2D points of an object.

To install and configure TIY, more information is presented on the online documentation [20] [21] as well on Annex A.

3.3. CHAPTER CONCLUSION

In this chapter, it was made a very lightweight approach to tracking systems. In the specialized literature, one can find other methodologies and algorithms oriented to tracking, and in the market there are several systems for tracking.

4. COMPUTER VISION

In this chapter will be addressed some computer vision topics. Computer vision is a comprehensive and flexible form of perception that allows detecting the presence of objects as well as differentiate among them. It is the appropriate form of perception to implement tracking systems, as in the case of this project.

With the advent of computers, there has been a technological revolution, enabling several applications of this equipment to meet the needs that were emerging. One of the applications that came up was computer vision, consisting of the acquisition, processing and analysis of real-world pictures or video to further generate information for use by digital systems. One of the computer vision goals, is to reproduce the view of humans, through the perception and understanding of images, using electronic devices such as cameras. This perception consists in the decomposition of symbolic information of images, using models built using the geometry, physics, statistics, and theory of learning, among others. Computer vision is usually used, for example, on event detection, tracking, object recognition, learning, motion estimation and image restoration.

The field of computer vision is constantly evolving, and, today, this technology is being driven mainly by its use in artificial intelligence applications, industrial control, robotics, or for recreational purposes. In the industry, computer vision is being used to increase

productivity, while reducing costs, increasing quality and providing more safety and comfort for human employees.

One can divide the computer vision systems into two different types of implementation: using image processing software on a computer, or using special hardware such as Digital Signal Processors (DSP), Field Programmable Gate Array (FPGA) or Application Specific Integrated Circuits (ASIC). The greatest advantage of the use of dedicated hardware systems is the lower processing time and computing power, but has the disadvantage of having little versatility in the case of the DSP or ASIC [22].

Computer vision is used in various fields (Figure 5), but in the robotics field it can be used by mobile robots for planning paths and making decisions autonomously, making possible for the mobile robot to move autonomously in a given space. A detailed knowledge of the surrounding environment, normally provided by computer vision systems, together with other types of sensors that gather information of the environment, is given to the robot, enabling it to take decisions from the received data.

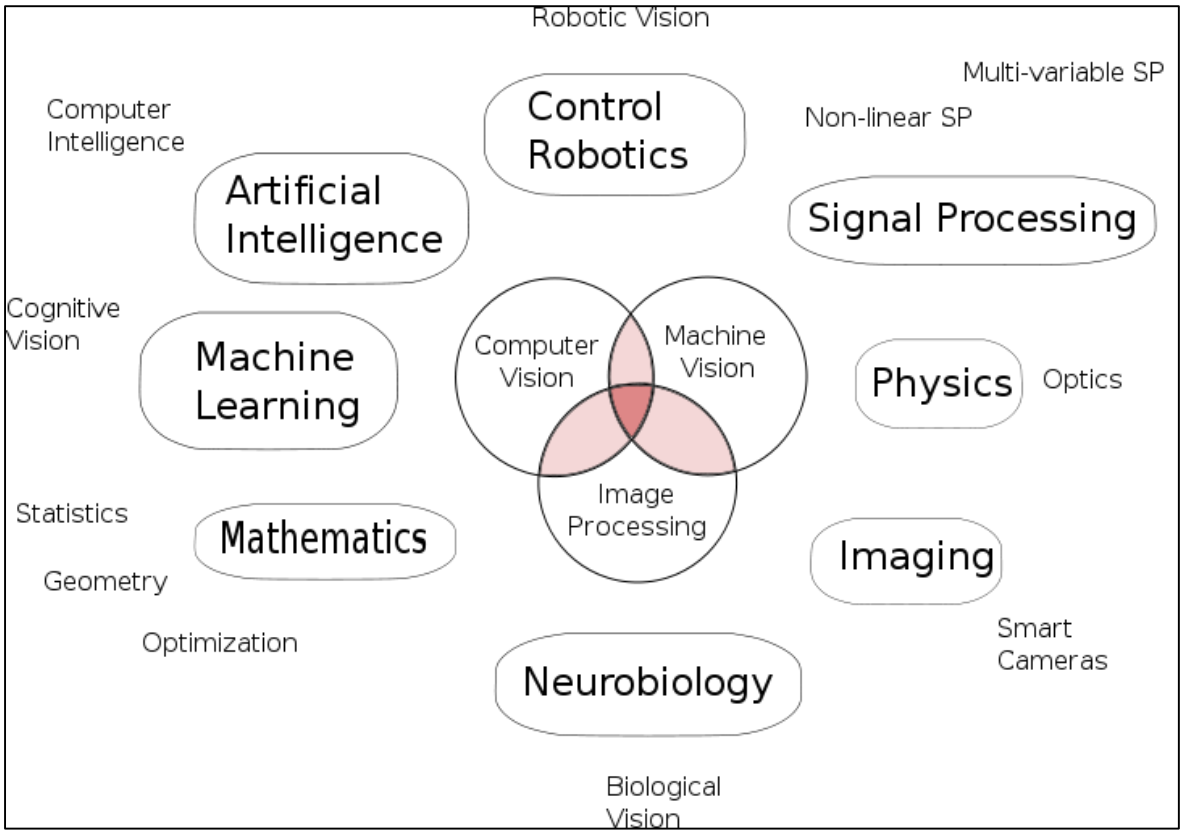


Figure 5 Applications of computer vision on several fields [23].

The computer vision also serves to learning tasks and pattern recognition, being easily integrated in artificial intelligence and in the area of computer science in general.

In industry, computer vision is widely used for location and material selection, automatic inspection, and modeling of objects. In the case of industrial robots, grants them advanced sensory capabilities.

One of the main applications of computer vision is in the recognition and classification of objects. One or more objects, or groups of objects, can be recognized and its position can also be supplied in the two-dimensional or three-dimensional space. It can be used, for example, to estimate the position of objects on a moving conveyor belt that supplies a robot that is performing their packaging, or may be a more simple system that only reads bar codes present on objects.

4.1. MACHINE VISION

When applied in an industrial environment, computer vision is typically known as machine vision and is applied mainly in process and quality control.

4.1.1. OVERVIEW OF MACHINE VISION

Machine vision consists in the automatic interpretation of real images in order to control or monitor machines or industrial processes. The images may be obtained from visible light, X-ray, infrared, or information of ultrasonic sensors.

In some cases, humans inspection can no longer follow the requirements of the current industries, mainly in aspects such as the inspection speed and the desired quality of products inspected. Humans cannot always keep the same attention or also apply the same selection criteria in different periods, and that selection criteria can be quite different from one operator to another. In extreme cases, due to environmental restrictions, the presence of a human operator is not possible.

Machine vision, however, can perform the analysis with constant precision and usually with greater speed and uniformity. Thus, in the industry have been extensively chosen and applied test and control systems which use machine vision, except in some cases where they still failed to implement machine vision systems that can replace humans in a satisfactory manner.

4.1.2. MACHINE VISION OPERATION

An industrial machine vision system normally consists of one or more cameras for image acquisition, lenses for cameras, a lighting system that may be substituted by ambient light in some applications, an interface system for communication between the cameras and the image processor or computer, an image processor (or computer) that may be replaced by cameras that include image processing, and, finally, it is usually also available an interface implemented using software to provide information and receive commands from the outside (Figure 6).

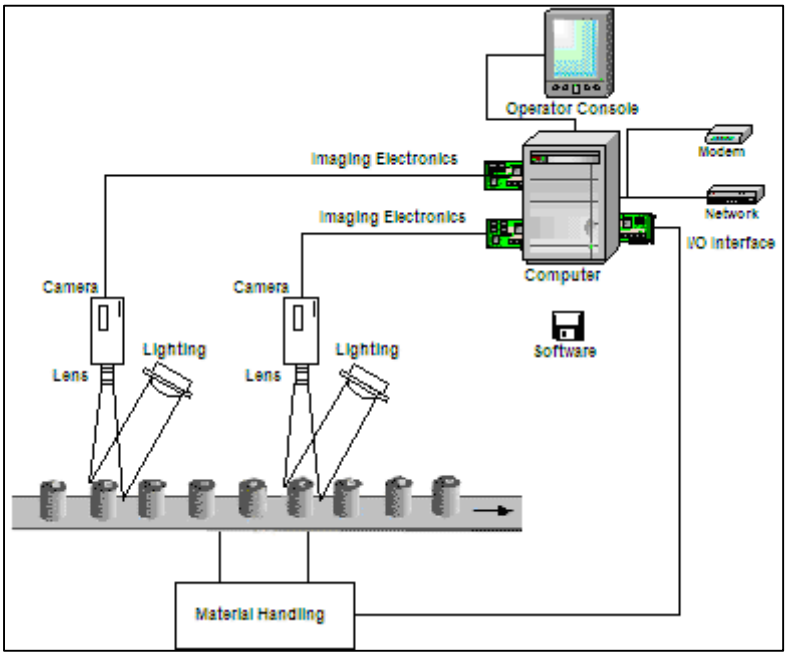


Figure 6 Overview of an industrial machine learning system [24].

The image captured by the camera may be represented by a two-dimensional array for representing energy levels which may vary within a range. These individual elements of the images are known by pixels, and rows and columns of pixels forms images. The pixel is the basic element of the image and cannot be subdivided. A pixel can have only one energy level, a monochrome image has a gray level from white to black, and a polychromatic image has levels of different colors to compose the image. In case of a system that uses primary additive colors it has three different color levels to compose the image: levels for red, green, and blue (RGB).

The amount of energy captured by the camera to each pixel must be scanned, and this task can be performed internally by the camera. Analogic color levels obtained with continuous

variation are converted to levels with finite number of states (digital). In the case of monochrome images it is common to use 8 bits for each pixel, to give 256 gray levels. In the case of polychromatic images, typically 24 bits are used for each pixel, which in the case of RGB allows providing about 16 million colors, but more or fewer bits can be used to represent each pixel.

Image processing is done essentially in two steps: segmentation and analysis. Segmentation is the decision of which parts of the image are relevant and which may be discarded, depending on each application. After the image has been segmented satisfactorily, the image processing step makes the analysis consisting of several tasks and measures of the objects of interest [25].

4.1.3. MACHINE VISION APPLICATIONS

Machine vision systems can be applied in various functions [25]:

- inspection and measurement - compliance check, detection of failures or problems, color confirmation, one, two or three-dimensional measuring (based on stereo vision techniques or time of flight);
- recognition - recognition of parts or objects, recognition of characters, etc.;
- guidance - predetermined guidance or continuous guidance;
- special systems for use in industrial applications.

The compliance check usually consists of a two-dimensional measurement to check the possible displacement of the desired object position and also makes a pixel by pixel scan to check compliance with the reference model. Fault detection is a particular case of compliance check.

The triangulation measurement using methods such as structured light consists of a light beam which is directed to a three-dimensional surface. When visualized by a two-dimensional camera, there is an angle between the light beam and receiving camera. The beam received by the camera may be used to obtain the surface shape where that beam focused. By passing the beam across the surface, it can be generated a three dimensional

map of the surface. It is usually used a laser light as the source of the beam due to its simple handling.

The three-dimensional measurement can also be performed using two or more cameras, method known as stereo vision. Another method for three-dimensional view is by using time of flight cameras that measure the time it takes light to reach a certain object, being that light emitted by a source placed next to the camera.

Character recognition can be based on simple comparison techniques without specialized knowledge of how the individual characters are formed, or may use more advanced algorithms that can, for example, differentiate between a misrepresented ‘8’ and a letter 'S'. The vision system may also be used to read bar codes with better results than when using laser scanners.

Object recognition can be used for compliance testing or for process control, for example, where different detected objects need different treatment by the control system.

The predetermined guidance can be exemplified by the situation in which a camera attached to a robot takes a photograph of the scene and the vision system instructs the robot where to pick and drop a certain object. During the movement of the robot the system is “blind”.

The continuous guidance is similar to the predetermined guidance described above, but the vision system is continuously checking the scene, and the robot path is continually corrected by the vision system [25].

4.1.4. FINANCIAL JUSTIFICATION OF MACHINE VISION USE

There are several reasons that financially justify the use of machine vision systems. For example, reduced scrap due to the use of inspection systems using machine vision results in a relatively short pay-off. Automatic inspection systems allows to check the quality parameters which are deviating from the ideal, being possible to take corrective action before these parameters pass the limit, and the products being inspected are considered scrap.

Other financial justification for acceptance of machine vision systems is to reduce the cost with human resources. Many tasks performed by human operators can be replaced by machine vision systems, and these savings are greater when the company operates in continuous working with multiple shifts. The savings are even greater if it is considered the

costs of recruitment, benefits and increases in human employees. However, those involved in working with machine vision systems are usually most qualified and better paid.

Costs relating to product quality, or their lack, are also an important aspect of how machine vision systems can allow savings. Machine vision systems provide inspection methods that are more objective, reliable and consistent than those made available by human inspectors. There should also be considered the savings by optimizing the use of materials, monitoring the quality of materials from suppliers and ensuring the quality of produced products. The cost of repair work, because of the added guarantees, can be reduced, and the trust of customers is increased due to the higher quality and uniformity of products produced leading to a larger number of orders [25].

In summary, the following benefits of machine vision adoption can be stated [26]:

- higher products quality with machine vision systems doing inspection, measurement, gauging and assembly verification;
- increased productivity replacing repetitive tasks done manually;
- production flexibility on measurement and gauging, robot guidance and operation verification;
- more complete information and tighter process control due to computer data feedback provided by machine vision systems;
- lower capital equipment costs by adding vision to a machine, because it improves its performance and avoids its obsolescence;
- lower production costs because the vision system can detect flaws early in the process;
- scrap rate reduction;
- inventory control due to tracking and identification;
- reduced floor space in most cases, taking into account the vision system versus the operator.

4.2. IMAGE CAPTURE

The captured images by the human vision allow a perception of the surrounding space. In the case of computer vision applied to robotics and other industrial systems, the goal is the same - to provide a perception of the surrounding space. From the images it is possible to deduce the size, shape, color, texture and position of the various objects located in the field of view.

As in humans, in which is formed an image on the eye retina, in a digital camera, a plastic or glass lens forms an image on the integrated circuit surface, who has an array of light sensitive devices that convert this light into a digital image.

The process of forming an image, whether in the human eye or in a digital camera, consists of projecting a three-dimensional space on a two-dimensional surface, and the information about the depth is lost. The transformation of 3 to 2 dimensions is known as perspective projection or transformation.

4.2.1. PERSPECTIVE TRANSFORMATION

There is a relationship between the size of the lens and the brightness level of the captured image. The bigger the lens the higher is the brightness; the smaller the lens more darkened is the captured image. The use of a convex lens also increases the brightness of the captured image.

The elementary aspects of image creation are shown in Figure 7.

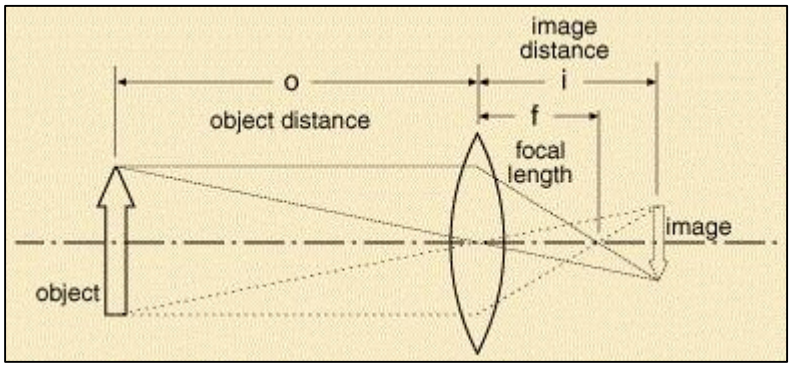


Figure 7 Image formation using a convex lens [27].

There is a geometric relationship between the focal length of a lens (*f*), the distance from the lens to the object (*o*), and the distance of the lens to the projected image (*i*). This relationship is shown by the following equation:

$$\frac{1}{\mathrm{o}}+\frac{1}{i}=\frac{1}{\mathrm{f}} \tag{1}$$

In computer vision, it is normal to use central perspective imaging model (Figure 8). The rays converge at the origin of the camera frame (*C*) and a non-inverted image is projected onto the image plane. A point in world coordinates (*P = (X, Y, Z)*) is projected onto the image plane (*p = (x, y)*) by the following expressions [29]:

$$x = f\frac{X}{Z}, \qquad y = f\frac{Y}{Z} \tag{2}$$

It is a projective transformation or perspective projection, of the real world to the image plane.

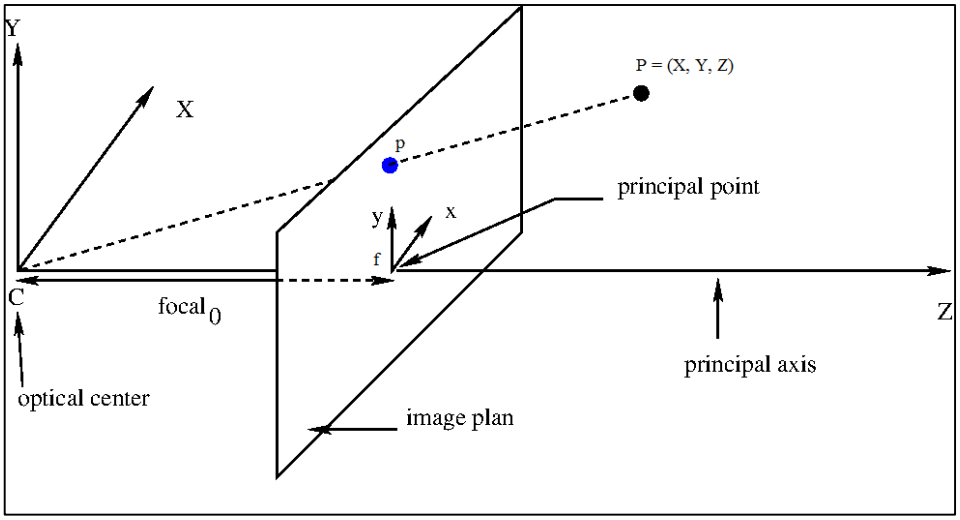


Figure 8 The central projection model [28].

In the case of the point on the image plane in homogeneous form $\tilde{p} = (x', y', z')$ where:

$$x' = \frac{fX}{z'}, \qquad y' = \frac{fY}{z'}, \qquad z' = Z \tag{3}$$

or in compact matrix form:

$$\tilde{p} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{4}$$

The point coordinates in the image plane on the homogeneous form are:

$$x = \frac{x'}{y'}, \qquad y = \frac{y'}{z'} \tag{5}$$

These are usually referred to as the retinal image plane coordinates. In cases where $f = 1$ the coordinates are referred to as the normalized or canonical image plane coordinates.

If world coordinates are written in homogeneous form with ${}^{C}\tilde{P} = (X, Y, Z, 1)^{T}$, then the perspective projection can be written in linear form:

$$\tilde{p} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} {}^{C}\tilde{P} \tag{6}$$

or

$$\tilde{p} = C\, {}^{C}\tilde{P} \tag{7}$$

The **C** matrix of size 3 × 4 is known by camera matrix. The ${}^{C}\tilde{P}$ is the point coordinate relative to the camera frame $\{C\}$. The tilde indicates that these are homogeneous quantities. If the camera matrix is factorized:

$$\tilde{p} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} {}^{C}\tilde{P} \tag{8}$$

wherein the second matrix is the projection matrix.

Taking into account the arbitrary pose ε_C of the camera relative to the world coordinate frame, as shown in the Figure 9, the point position respectively to the camera is:

$${}^{C}P = (\ominus\, \varepsilon_C) \cdot {}^{0}P \tag{9}$$

or in homogeneous coordinates:

$${}^{C}P = T_C^{-1}\, {}^{0}P \tag{10}$$

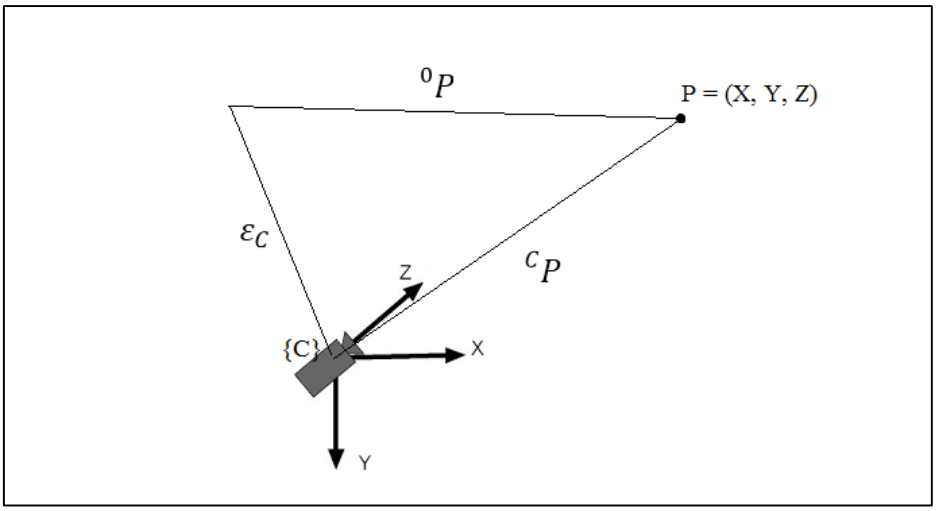


Figure 9 Camera with arbitrary pose.

The image plane can be divided by its constituent pixels. Pixels are vectors (u, v) with integer coordinates, uniform in size, and related to the image plane as follows:

$$u = \frac{x}{\rho_w} + u_0, \quad v = \frac{y}{\rho_h} + v_0 \tag{11}$$

where ρ_w is the width and ρ_h is the height of each pixel, and the point (u_0, v_0) is the main point, *i.e.*, the point where the optical axis intersects the image plane.

Rewriting the perspective projection as follows, by inserting a matrix **K** with the camera parameters:

$$\tilde{p} = \begin{pmatrix} \frac{1}{\rho_w} & 0 & u_0 \\ 0 & \frac{1}{\rho_h} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} {}^C\tilde{P} \tag{12}$$

and combining the point position respectively to the camera with the previous expression, it is obtained:

$$\begin{aligned} \tilde{p} &= \begin{pmatrix} \frac{f}{\rho_w} & 0 & u_0 \\ 0 & \frac{f}{\rho_h} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} (T_C^{-1})\tilde{P} \\ &= KP_0T_C^{-1}\tilde{P} \\ &= C\tilde{P} \end{aligned} \tag{13}$$

All terms can be inserted in the camera matrix (**C**). This matrix is known as projection matrix or camera calibration matrix.

4.3. CAMERA CALIBRATION

The calibration of cameras is required if one are to get an accurate 3D positioning from two 2D views. This calibration can be understood as a relationship between the cameras used. Typically, the calibration is done using multiple images of a checkboard pattern type for each of the cameras (Figure 10).

Figure 10 Checkboard used for calibration. Image captured by a Pixy camera where there is easily seen the camera distortion.

Once captured the images required in each of the cameras, it is needed to estimate the cameras intrinsic parameters (that depend on the characteristics of the cameras), and the extrinsic parameters that depend on their positioning.

There are five intrinsic parameters: the focal length (*f*), the pixel size in *x* and *y* direction (s_x and s_y) and the principal point (o_x and o_y). There is a particular and common case when the pixels are square. In this case, $s_x = s_y = s$ and, therefore, the number of intrinsic parameters is reduced to 4, being the focal length $f_x = f/s_x$, $f_y = f/s_y$ and principal point (o_x and o_y) [29].

Extrinsic parameters consist of a camera rotation matrix and a camera translation vector ([**R** | **T**]). They represent a relation of the camera's position relative to a known frame.

Relating the above parameters, it is obtained a 3 by 4 matrix known as projection matrix (**P** = [**R** | **T**] **K**).

Next are shown the equations relating to image projection:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \mathbf{M_{int}}\mathbf{M_{ext}} \begin{pmatrix} X_w \\ Y_W \\ Z_W \\ 1 \end{pmatrix} \qquad \begin{pmatrix} x_{im} \\ y_{im} \end{pmatrix} = \begin{pmatrix} u/w \\ v/w \end{pmatrix} \tag{14}$$

The $\mathbf{M_{int}}$ matrix containing the intrinsic parameters, the camera parameters in relation to the frame, is given by:

$$\mathbf{M_{int}} = \begin{bmatrix} -f_x & 0 & o_x \\ 0 & -f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \tag{15}$$

The $\mathbf{M_{ext}}$ matrix which contains the extrinsic parameters, the camera parameters in relation to the surrounding space, is given by:

$$\mathbf{M_{ext}} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \end{bmatrix} \tag{16}$$

The focal length may not be accurate, and it is only correct if the lens is focused at infinity. In the case of lens replacement, and focus or aperture adjustment, the intrinsic parameters should be recalculated because they may change [30].

The usual calibration techniques are based on sets of world points where their relative coordinates are known as well their corresponding coordinates in the image plane.

According to Zhang [31], the calibration techniques can be classified into two categories:

- photogrammetric calibration: the camera calibration is performed by observing an object whose three-dimensional geometry is well known;
- self-calibration: the techniques of this category do not use a calibration object. They consists of moving the camera on a static scene. This approach is flexible but has many parameters to estimate.

Calibration of cameras used in this project is based on Zhang's calibration technique [31]. This is a widely used technique and is positioned between the photogrammetric calibration and self-calibration. One main difference regarding to the photogrammetric calibration technique is that, in this case, a 2D pattern is used instead of a three-dimensional object.

4.4. CHAPTER CONCLUSION

In this chapter, it was introduced computer vision and, more specifically, machine vision. It was given an overview, justification of its use, and some areas where these technologies can be implemented.

It has also been approached the camera as the model, where calibration of the cameras was also described.

5. SYSTEM ARCHITECTURE

The main goal of this project is the development of advanced human robot interaction systems with mixed initiative and intuitive even for the worker unfamiliar with robotics.

This project is motivated by the participation of the Unidade de Robótica e Sistemas Inteligentes (ROBIS) unit from INESC TEC in the European Project SMERobotics, CLARiSSA demonstrator. The CLARiSSA demonstrator is focused on a collaborative dual arm robot for assembly applications in steel fabrication, consisting of one robot arm that handles the parts, another robot arm that welds the parts, and a working table.

It is used a "low-cost" solution to perform the interaction system, that consists on an infrared teaching system for part positioning and correction. The selected main components are:

- Raspberry Pi computer;
- two Pixy cameras;
- TIY software (with modifications) to perform the object tracking.

In Figure 11 it is presented how the system is intended to work. In the working area represented with the dual-arm robotic system, a worker using an infrared pointing device can easily teach the task to do. The two cameras equipped with infrared light filters, track

the infrared pointing device and send positioning information to the computer. The Raspberry Pi computer runs a special modified version of TIY developed for this project, which transform the 2D data from the two cameras into the 3D tracking device position. The 3D position is then outputted from the system, and is intended to provide a simple and efficient method of human-robot interaction that can be integrated in the client's facilities.

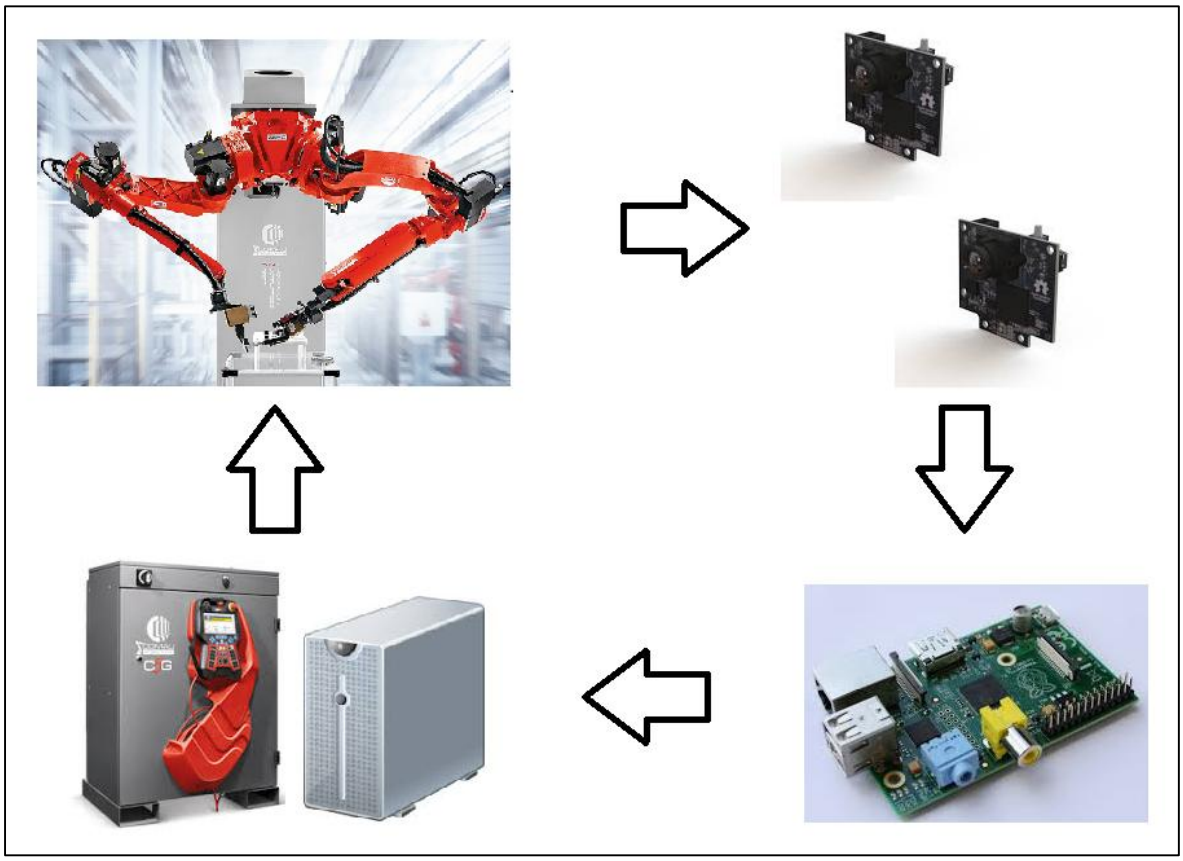


Figure 11 Human-robot interaction system main components.

5.1. DETERMINATION OF THE BEST POSITION FOR THE TRACKING SYSTEM CAMERAS

RobotStudio is a software that allows developers to make simulation and offline programming of ABB robots. This software was chosen because it allows the simulation of artificial vision systems in robotic cells. The main objective of its use in this context was to find the best position to place the cameras of the tracking system.

The model built consists of two articulated robots with six axes each, mounted upside down, with a conveyor beneath where will be located the beams to weld the parts. One of the robots performs the picking of parts from the table and places them in the beam, while the other robot welds the part to the beam (Figure 12).

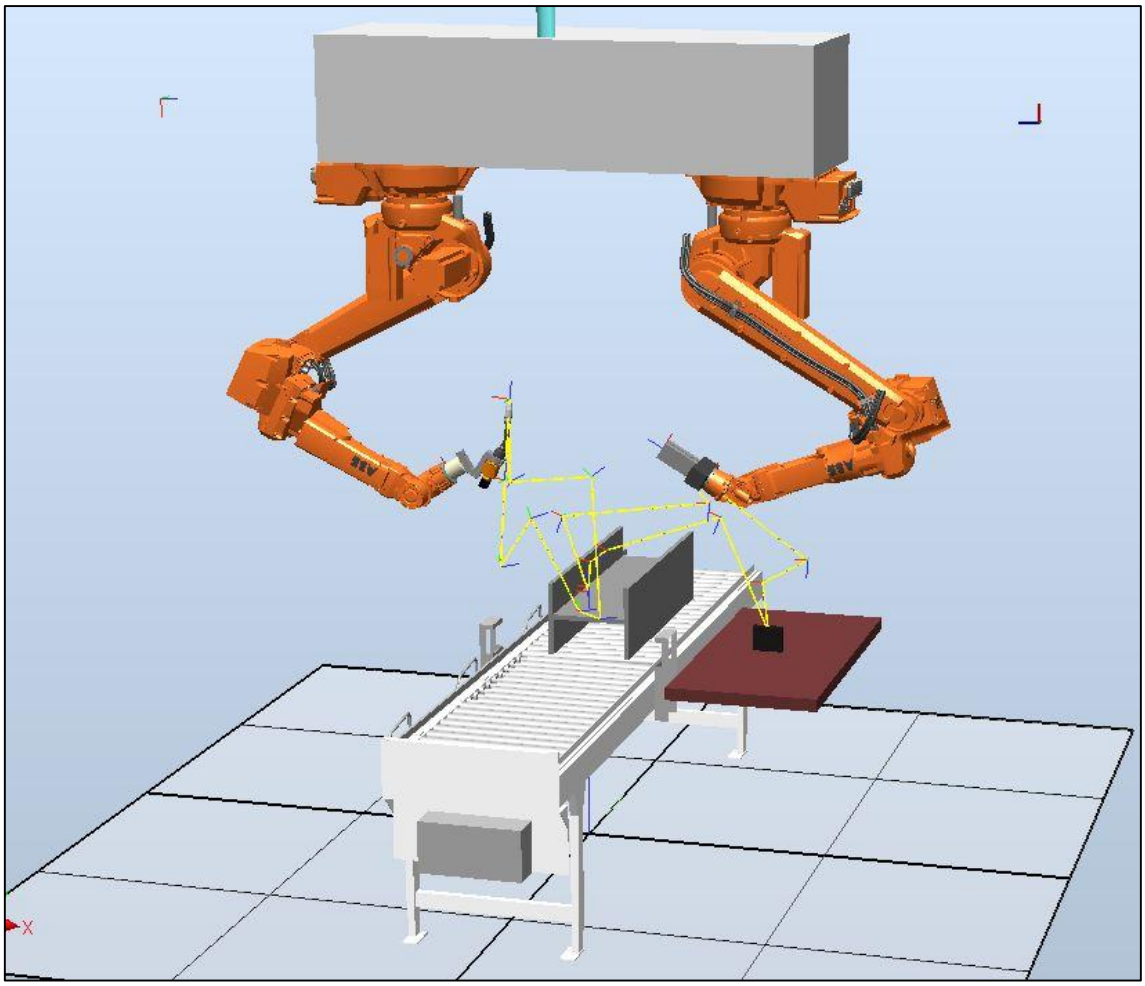


Figure 12 Overview of the robotic system.

As previously seen, to localize objects in three-dimensional space (3D stereo vision) are required at least two 2D cameras that may be positioned close to each other, getting less precision, or further apart between them, getting more precision.

5.1.1. DESCRIBING CAMERA LOCATION

A camera mounted in front of the robots provides a good field of view of the entire system, without blind spots to areas of interest, as can be seen from Figure 13, Figure 14 and Figure 15. It allows seeing both parts on the workbench and on the beam in the welding position. A support for fixing the cameras in this advanced position relative to the base of the robots may be needed.

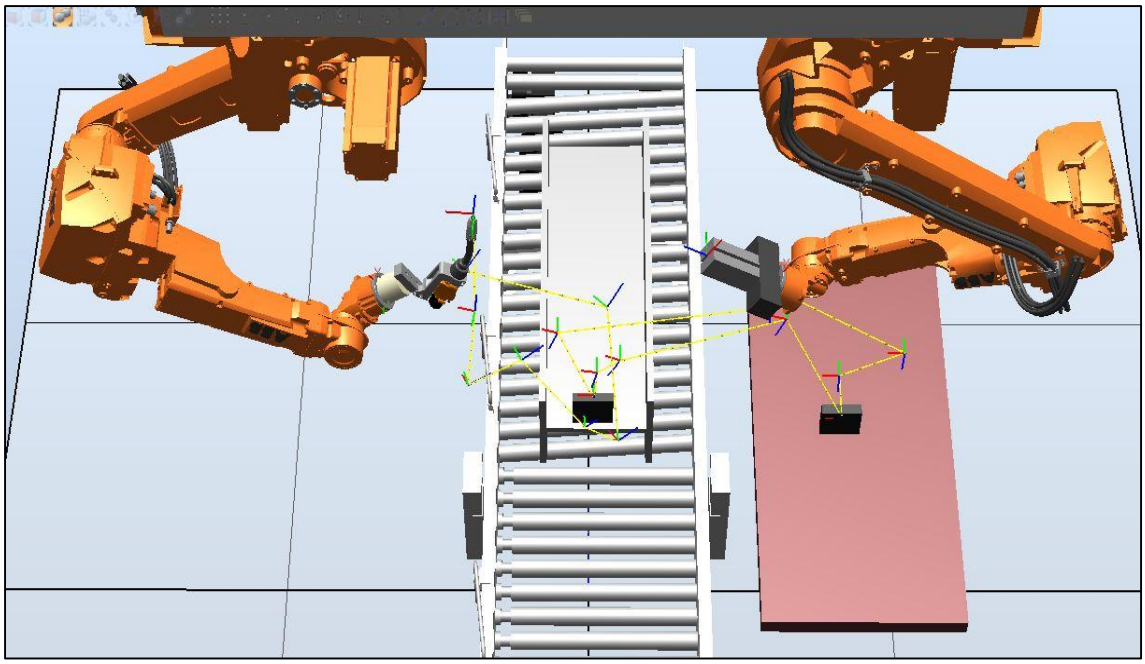


Figure 13 Camera mounted in front of robots. Picker and welder robots in home position.

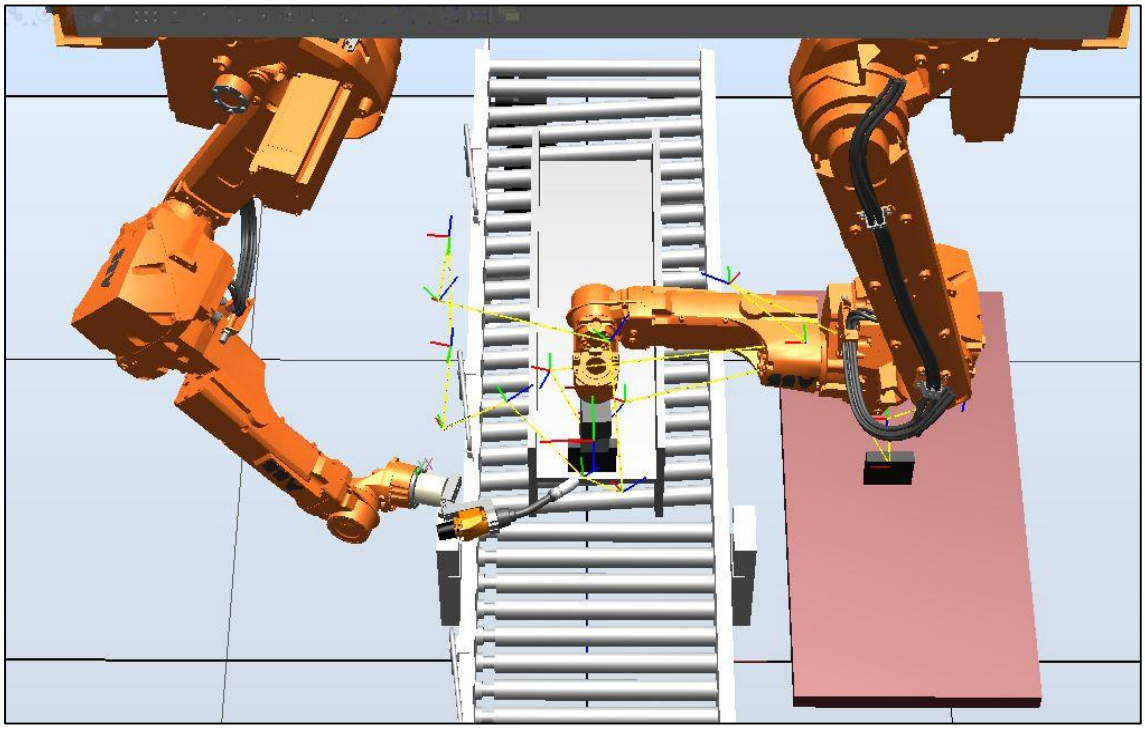


Figure 14 Camera mounted in front of robots. Picker and welder robots in welding position.

It is verified that even with the robots on working position, while welding the part on the beam, the cameras mounted in this position allows a good view of the entire working area.

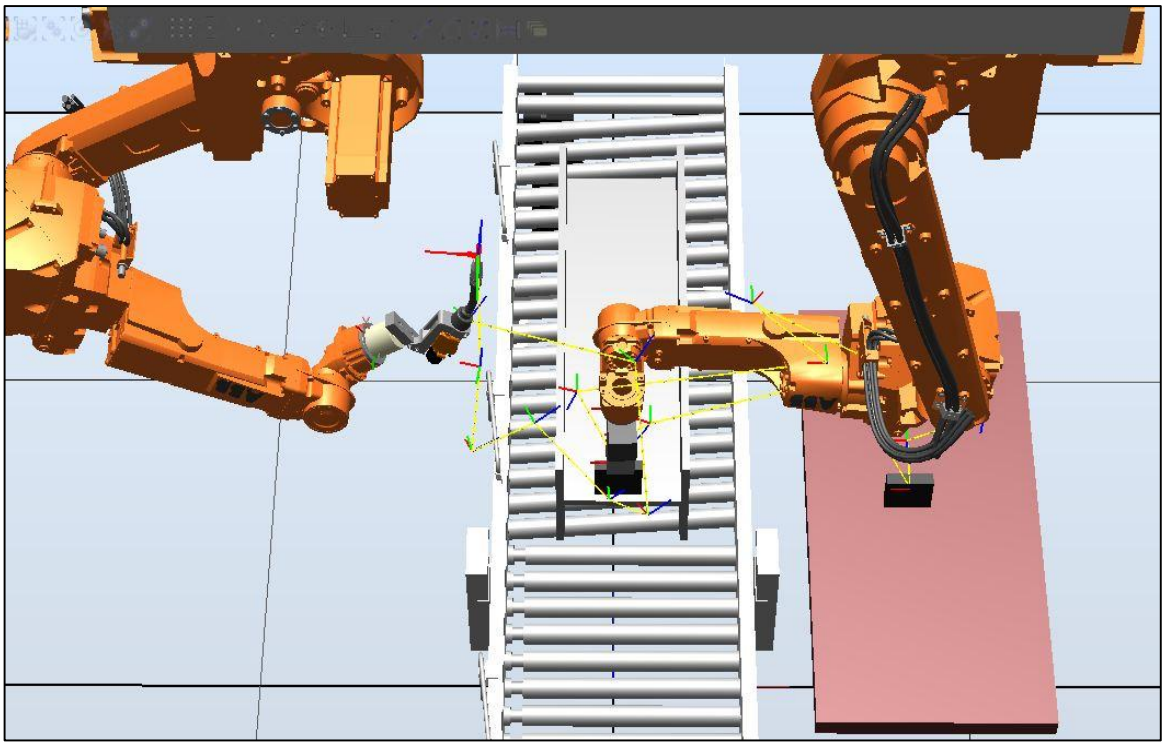


Figure 15 Camera mounted in front of robots. Picker robot in welding position. Welder robot in home position.

The assembly of the camera in the welding robot arm allows a good field of view, if the beam does not obstruct the part in the welding position, or the table where are placed the parts to be welded. This approach is not as flexible as the previous one, in which the camera is mounted in front of the working area, but offers greater accuracy in the detection of objects, since it closer to the parts, as can be seen from Figure 16 and Figure 17.

The assembly of the camera at the top, and centered, above the robots base, provides a poor field of view, with many blind spots, as can be seen from Figure 18 and Figure 19. The space outside the central conveyor is almost totally blocked and when the picker robot is to put the part in the beam, the view is obstructed.

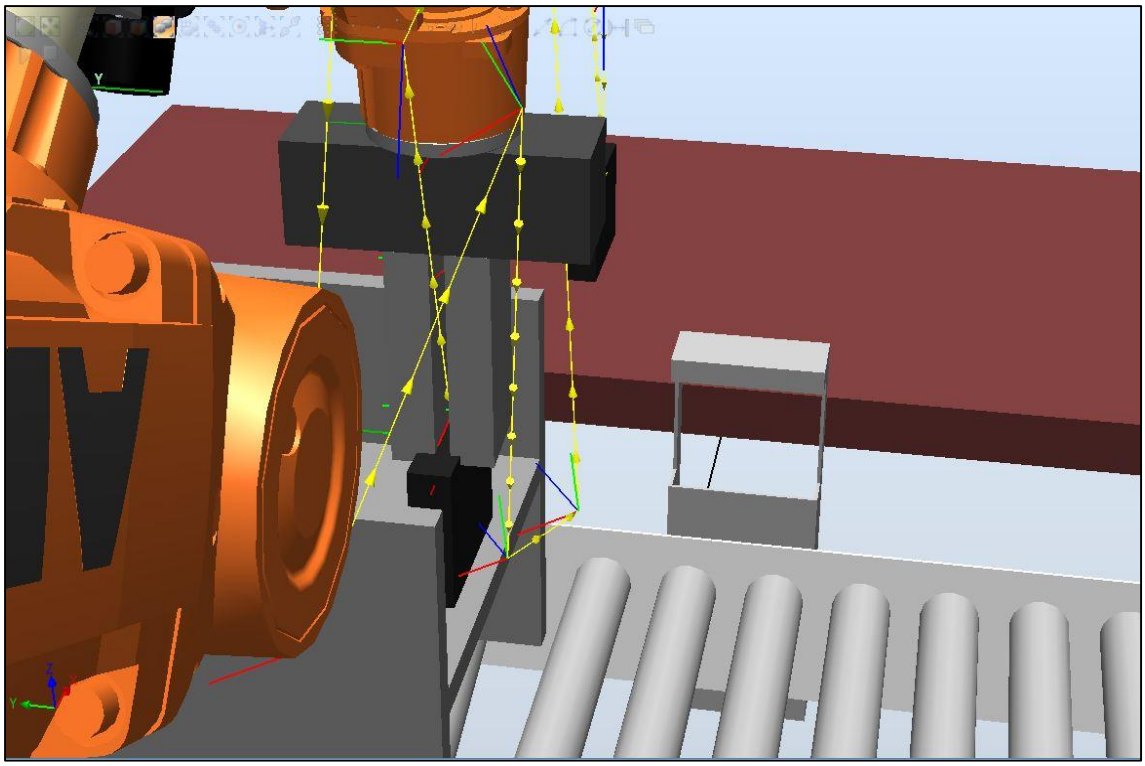


Figure 16 Camera mounted on welding robot. Picker robot in welding position. Welder robot in home position.

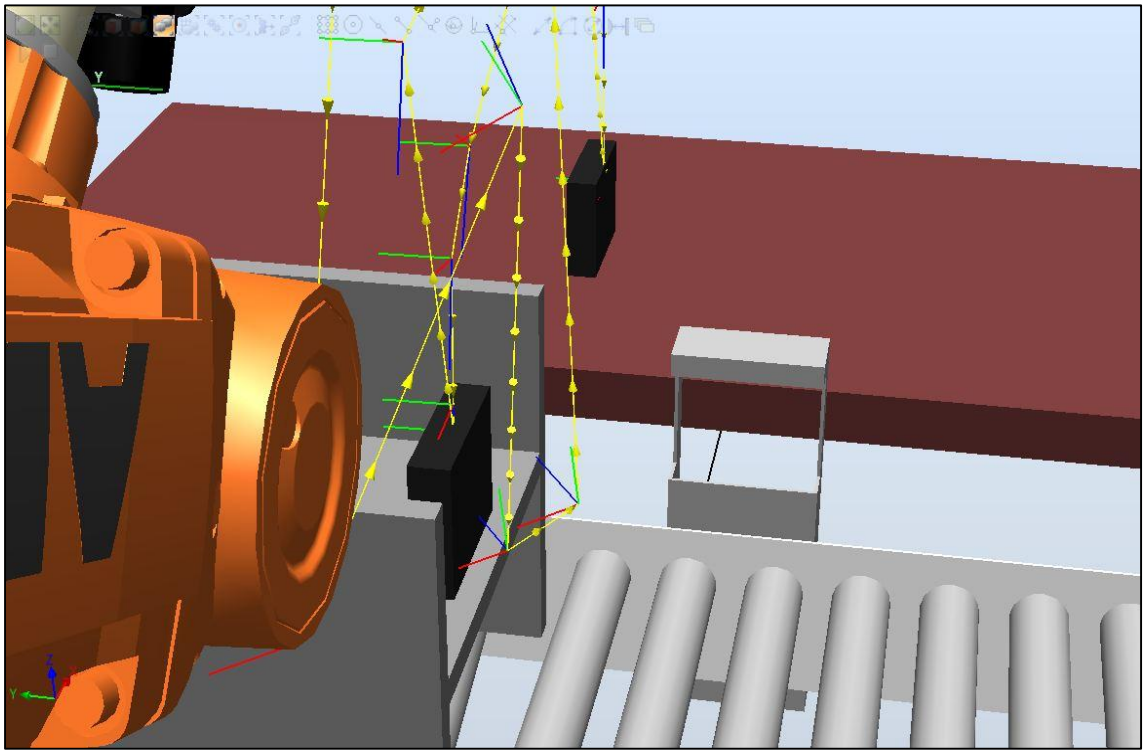


Figure 17 Camera mounted on welding robot. Picker and welder robot in home position.

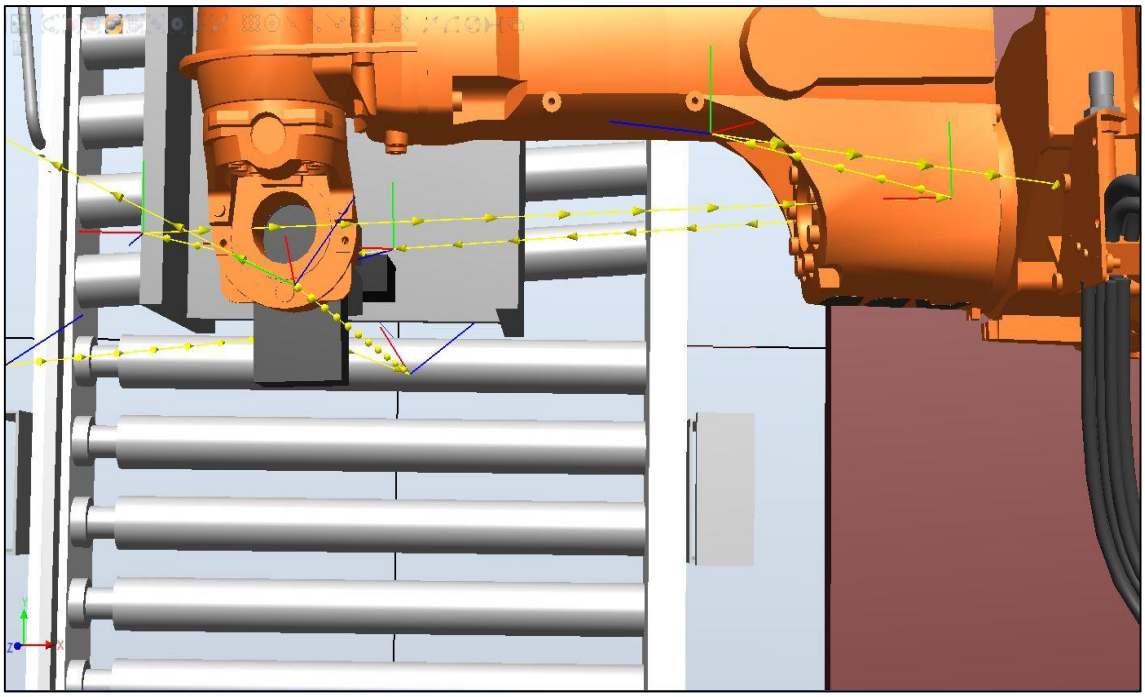


Figure 18 Camera mounted on top and centered above both robots base. Picker robot in welding position. Welder robot in home position.

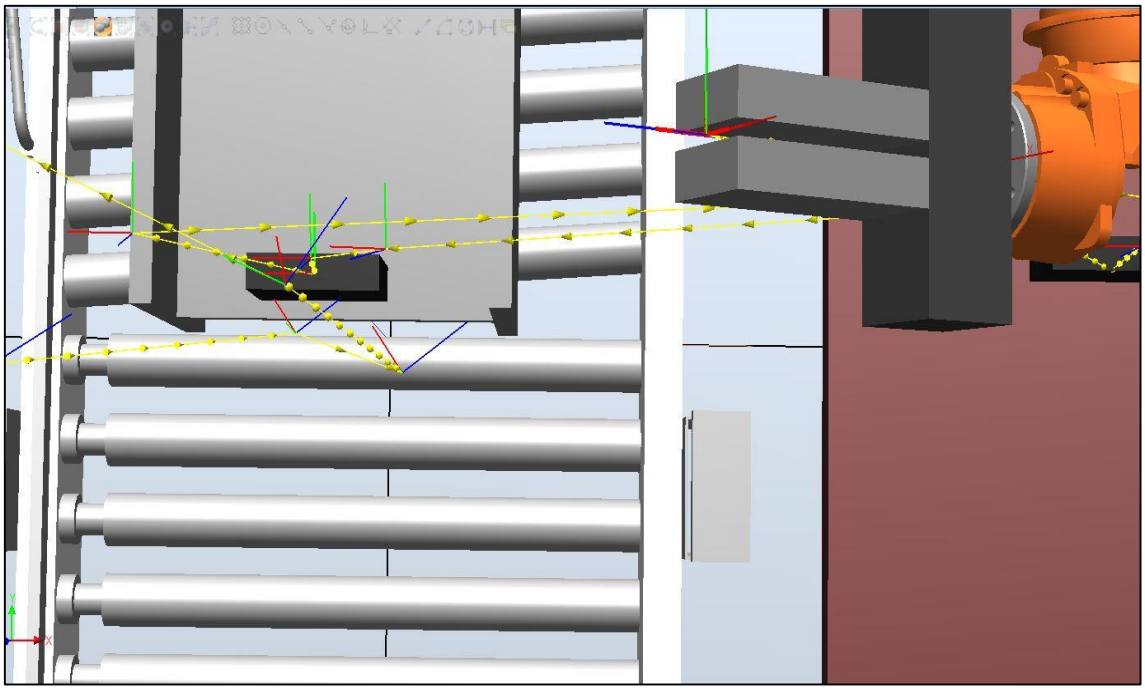


Figure 19 Camera mounted on top and centered above both robots base. Picker and welder robot in home position.

From the three studied situations, it is concluded that in the case of mounting the cameras on the robots base, the vision system cannot be developed properly. For the other two cases,

the assembly of the camera in front of the workstation provides a better field of view, while mounting the cameras in the welding robot arm provides more accuracy due to the cameras being closer to identify the parts. Thus, if the used cameras have sufficient definition, it is preferable to mount them in a position in front of the robots.

5.2. OPEN SOURCE COMPUTER VISION LIBRARY (OPENCV)

OpenCV is an open-source tool dedicated to computer vision, and developed in the C and C++ languages, that runs on Linux, Windows, and MacOS X. This tool was initially developed by Intel and officially launched in 1999, being made available to the public in the following year. There are interfaces developed for the OpenCV in Python, Ruby, Java, Matlab, C, C ++, among other programming languages. The OpenCV was developed with a view to computational efficiency, in order to be used in real time applications, and take advantage of multicore processors.

The main objective of OpenCV is to provide a simple way to implement computer vision systems and contains more than 500 functions that satisfy the requirements of many of the most common applications, such as industrial product inspection, medical imaging, security, user interfaces, camera calibration, stereo vision, and robotics. Because the computer vision and machine learning share many common features, OpenCV provides libraries for this purpose.

OpenCV played an important role in the growth of artificial vision, by allowing thousands of people to do their work in the area of computer vision using a free and open source tool. Furthermore, OpenCV made available to all people interested in computer vision a powerful and versatile tool for the development of their work.

Since its availability to the public, OpenCV has been used in various applications, products and research. These applications include, for example, joining multiple satellite images to generate a map, image alignment, noise reduction in medical diagnostic imaging, analysis of objects, security systems and intrusion detection, camera calibration, military applications and autonomous vehicles.

OpenCV has a structure divided, essentially, into five main components, being four of them shown in Figure 20. The CV component contains basic image processing and high level computer vision algorithms. The MLL component contains the machine learning library,

which includes many statistical classifiers and clustering tools. The HighGUI component contains Input/Output (I/O) routines and functions for data storage and uploading of images and videos. The CXCore component contains the basic data structures and the core of the system [32].

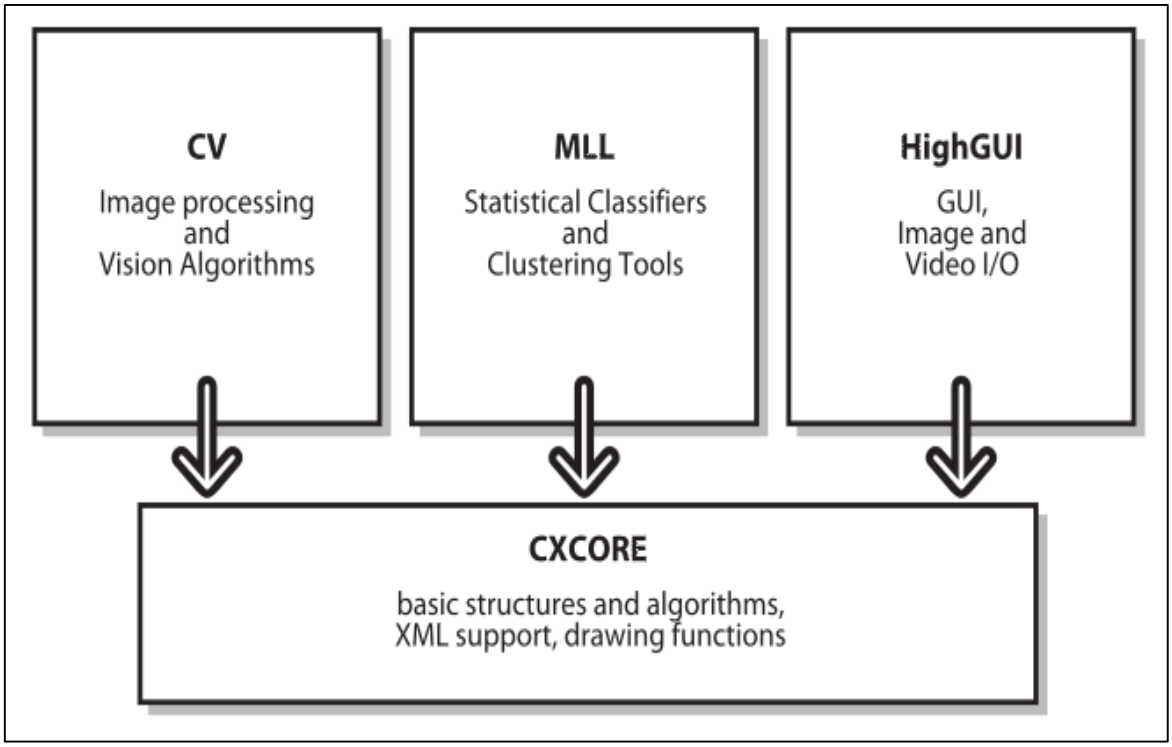


Figure 20 OpenCV basic structure [32].

Figure 20 does not include the CvAux component containing extinct areas of OpenCV and experimental algorithms.

OpenCV provides an Application Programming Interface (API) in C ++, from version 2.0 forward, that is of interest for the development of this project. It was used to treat the images captured by the cameras. OpenCV has a modular structure offering different libraries. Some of the most important modules are the following ones [33]:

- core: module which defines the basic structures, including the multidimensional table Mat and other basic functions used by other modules;
- mmgproc: image processing module which includes linear and nonlinear filtering of images, geometric transformations of images, among others;

- video: video analysis module that includes motion estimation, background subtraction, and objects tracking algorithms;
- calib3d: basic multi-view geometries algorithms, stereo and single cameras calibration, objects positioning estimation, stereo matching algorithms, and 3D reconstruction;
- features2d: feature detectors, descriptors, and descriptor matchers;
- objdetect: detection of objects and instances of the predefined classes;
- ...

More information can be found in the online documentation of this tool [34].

5.3. BOOST

Boost is a set of libraries to the C++ programming language that provides functionality for areas such as image processing, linear algebra, random number generation, multithreading, among others. It comprises over 100 individual libraries and can be used in free projects or proprietary designs, because of its license. Some of its libraries are even integrated into Technical Report 1 and C++ 11 standards. These libraries are platform independent and are supported by many of the common operating systems, including Windows and Linux [35].

The use of this set of libraries in this project is due to the dependence of Track It Yourself in Boost, as well as some available features that are useful in the rest of the project development. The libraries are used, for example, to timestamp the data captured from the cameras.

5.4. CMUCAM5 PIXY

The cameras used in this project are the CMUcam5 Pixy. They result from a partnership between the Carnegie Mellon Robotics Institute and Charmed Labs. They are an evolution of other CMUcams cameras and came onto the market in March 2014. Due to its low cost and its ease of use are often used in small projects related to robotics. Some of their key features are the following [36]:

- small, easy to use, and inexpensive, allowing a readily-available vision system;

- ability to teach objects that later will be detected by the camera;
- provides 50 times per second data from the detected objects;
- has libraries for common controllers, such as the Arduino or the Raspberry Pi;
- it supports the C/C++ and Python programming languages;
- multiple communication interfaces: Universal Serial Bus (USB), Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), Universal Asynchronous Receiver/Transmitter (UART) and digital/analog outputs;
- configuration tool that runs on Windows, MacOS and Linux;
- all software and firmware is open-source GNU (GNU's Not Unix) licensed.

One of the Pixy cameras particularities is to allow tracking objects with their own hardware and firmware, performing all the necessary processing, and providing the output position of the object. The Pixy with integrated tracking are ideal for vision systems that have low processing power since they take the image processing task and object detection from the main external controller. In the camera, apart from the image sensor, is also present a processor to perform the tracking and provide other features.

For the objects detection, Pixy uses a filtering algorithm based on color. These algorithms are popular because they are fast, efficient and relatively robust. The camera calculates the color and saturation of each RGB pixel from the captured image and uses these parameters to filter the content of the image. This type of filtering has the advantage that the color remains relatively unchanged with variations in light and exposure.

The Pixy can store up to seven different color signatures, which means that can be detected up to seven different objects, with different colors signatures. If it is necessary to detect more than seven different objects, color codes are available to allow this.

The camera can detect multiple objects at the same time, determining where an object ends and another begins. Then the camera compiles the size and location for each object and sends the results by the selected communication interface.

Besides the sensor having a higher resolution, the image processing is done in the resolution of 640 × 400 pixels every 20 milliseconds (50 times per second) and the results are outputted with a resolution of 320 × 200 pixels. This downscaling is done by an embedded controller due to processing constraints. This sampling rate is acceptable, even to detect moving objects, but the resolution may be small for some projects.

One advantage of these cameras is to provide the user a relatively simple way of teaching. There is a dedicated button present in the camera for this purpose: it is enough to put the object to detect in front of the camera and press the button (indicated in Figure 21), being generated a statistical model with the contained colors in the object, which will later be used to detect it.

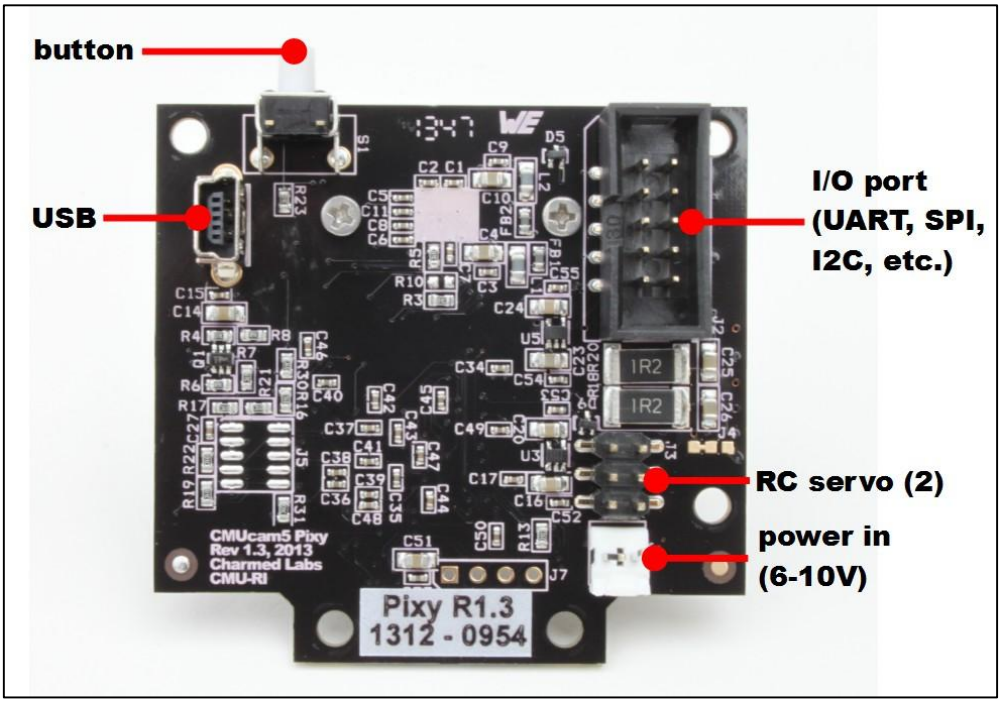


Figure 21 Pixy camera layout [36].

Another way to teach the object is using the supplied software, the PixyMon. This software also allows viewing what the camera is capturing in raw mode or processed video mode. Another software feature is the configuration of the cameras such as the communication interface and color signatures. The PixyMon communicates with the cameras using a standard mini USB cable, and it inclusively allows receiving the video signal while the interface is sending data to the external controller, being this a useful characteristic for debugging purposes.

Some of the key features of the camera are [36]:

- processor: NXP LPC4330, 204 MHz, dual core;
- image sensor: Omnivision OV9715, 1/4", 1280 × 800;
- lens field-of-view: 75 degrees horizontal, 47 degrees vertical;
- lens type: standard M12 (several different types available);
- power consumption: 140 mA typical;
- power input: USB input (5 V) or unregulated input (6 V to 10 V);
- Random Access Memory (RAM): 264 Kbyte;
- Flash Memory: 1 Mbytes;
- available data outputs: UART serial, SPI, I2C, USB, digital, analog;
- dimensions: 2.1" × 2.0" × 1.4;
- weight: 27 grams.

5.5. RASPBERRY PI OVERVIEW

The Raspberry Pi is a computer of the size of a credit card, developed in the United Kingdom by the Raspberry Pi Foundation, which, besides having little processing power relative to a conventional computer, is quite affordable and has certain features that make it suitable for small electronics projects. The Raspberry Pi can be found at various online stores.

Among the many small computers in the market, like the BeagleBoard [37] or the CubieBoard [38], the Raspberry Pi was chosen mainly due to this being one of the most accessible, having the largest community of enthusiasts, and also to my previous experience with the Raspberry Pi.

The original Raspberry Pi is based on the System on a Chip (SoC) Broadcom BCM2835, that includes an ARM1176JZF-S processor with a 700 MHz clock speed, one VideoCore IV Graphics Processing Unit (GPU), and 256 MB of RAM, later upgraded to 512 MB of RAM

in model B, which is the version used in this project and will be the version described in the sequel.

This minicomputer supports various Linux systems and provides some programming tools for the Python, C, C ++, Java, Perl and Ruby programming languages [39].

5.5.1. PROCESSOR

The included processor is an ARM1176JZF-S with only one core operating at 700 MHz, but its speed can be increased up to 1000 MHz, without loss of warranty using the configuration system available within the operating system used in this project - the Raspbian. To make overclocking in the Raspberry, it is possible to run the `sudo raspi-config` command and select the desired speed. Even if a sink is not used in the SoC, the temperature is maintained within acceptable levels as overclocking is disabled if a high temperature is detected on the processor [39].

With the 700 MHz clock, the Raspberry Pi Central Processing Unit (CPU) presents a performance of 0.041 Giga FLoating-point Operations Per Second (GFLOPS) [40] to the main processor and a performance of 24 GFLOPS to the GPU [41].

5.5.2. PERIPHERALS

The Raspberry Pi Model B has a 10/100 Mbps Ethernet port and two USB 2.0 ports, but doesn’t have wireless communications unless it is added. In Figure 22 it is depicted the Raspberry Pi model B board.

Video outputs support the most common interface protocols, such as Video Graphics Array (VGA) or Full High-Definition (HD), with a maximum resolution available of 1920 × 1200.The video outputs have a High-Definition Multimedia Interface (HDMI) connector or a Radio Corporation of America (RCA) connector. The board has a video input from a 15-pin connector that allows a camera connection. There is also a 3.5 mm jack connector for audio output.

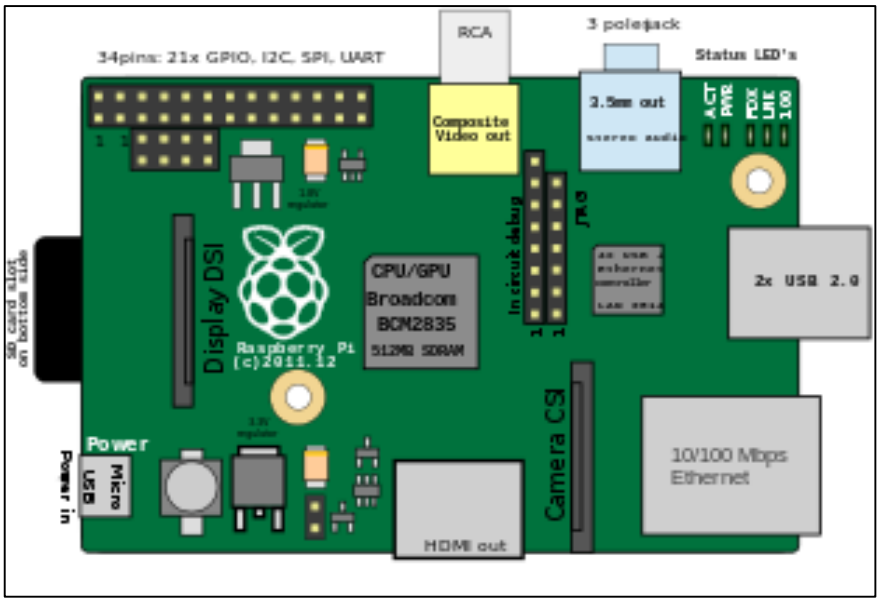


Figure 22 Raspberry Pi model B layout [42].

There is not provided a real-time clock but it can be added one from the General Purpose Input/Output (GPIO) pins or, alternatively, the clock may be synchronized using the network.

The board also features GPIO that allow digital and analog inputs/outputs, as well as 5 V and 3.3 V power pins. From the GPIO pin ports, are available the UART, I2C and SPI interfaces. The board has a 26 pins connector arranged in the 2 × 13 form. The pins voltage level is 3.3 V and have no protection against overvoltage.

The computer power may be supplied by the GPIO pins related to 5 V and GrouND (GND), but a micro USB connector is provided solely to power the device. It can use a common 5 V mobile phone charger with micro USB connector, and must comply with the minimum power consumption, which in the case of model B is approximately 700 mA (according to the manufacturer [39]).

The Raspberry has no hard disk or flash memory to store the operating system and other applications but has a Secure Digital (SD) card connector. Thus, a memory card is required to house the operating system and the rest of the system. The fact that are used common SD memory cards as storage device has some advantages, namely: the cards are inexpensive and is quick and easy to switch between cards that may have different systems, with multiple purposes. However, it has the disadvantage that SD cards are usually slower than the flash

memory commonly found in such devices (for example that present in mobile phones or tablets).

5.5.3. OPERATING SYSTEMS

The Raspberry supports several operating systems based on Linux. Taking into account that the model B processor is based on Advanced RISC Machine (ARM) version 6, some of the most popular systems like Ubuntu are not available. However the latest version of Raspberry Pi, the Pi 2, already supports more systems including Windows 10 Internet of Things (IoT).

Some of the supported systems range from OpenELEC or Raspbmc, allowing the Raspberry to behave as a media center, to the general-purpose systems such as Arch Linux ARM or Raspbian.

In this project it is used Raspbian, being recommended by computer makers, and is the one that has more available support. This system is based on Debian ARM hard-float system and is optimized for this minicomputer hardware. To house the Raspbian it is needed a SD card with a minimum capacity of 4 GB.

5.6. RASPBERRY PI 2 OVERVIEW

The Raspberry Pi 2 was released in February 2015, by the same developers of the previous models (Figure 23). The main advantage of using the Raspberry Pi 2 is that it can provide greater processing power in its main processor. As the cost of Raspberry Pi 2 is low (about the same as the Model B: ± 35$), there are advantages in the use of this, instead of the Model B. In the case of this project, the refresh rate in the tracking system can be increased with a greater processing capacity, to give better results if the detected objects have a very fast movement.

The Raspberry Pi 2 has roughly six times more processing power in the main processor relating to Model B. Unlike the ARM Cortex A6 processor 700 MHz (BCM2835), and 512 MB of RAM, present in the Raspberry Pi Model B, the Pi 2 has a quad-core processor ARM Cortex A7 900 MHz (BCM2836), with 1GB of RAM [43].

The GPU remains the same, and the computer power consumption is relatively equal, but the GPIO pins pass from 26 to 40, and the USB ports pass from 2 to 4. The remaining

peripheral and features of both Raspberry models remain identical [43]. The GPIO pins used in the project are the same for both Raspberry models.

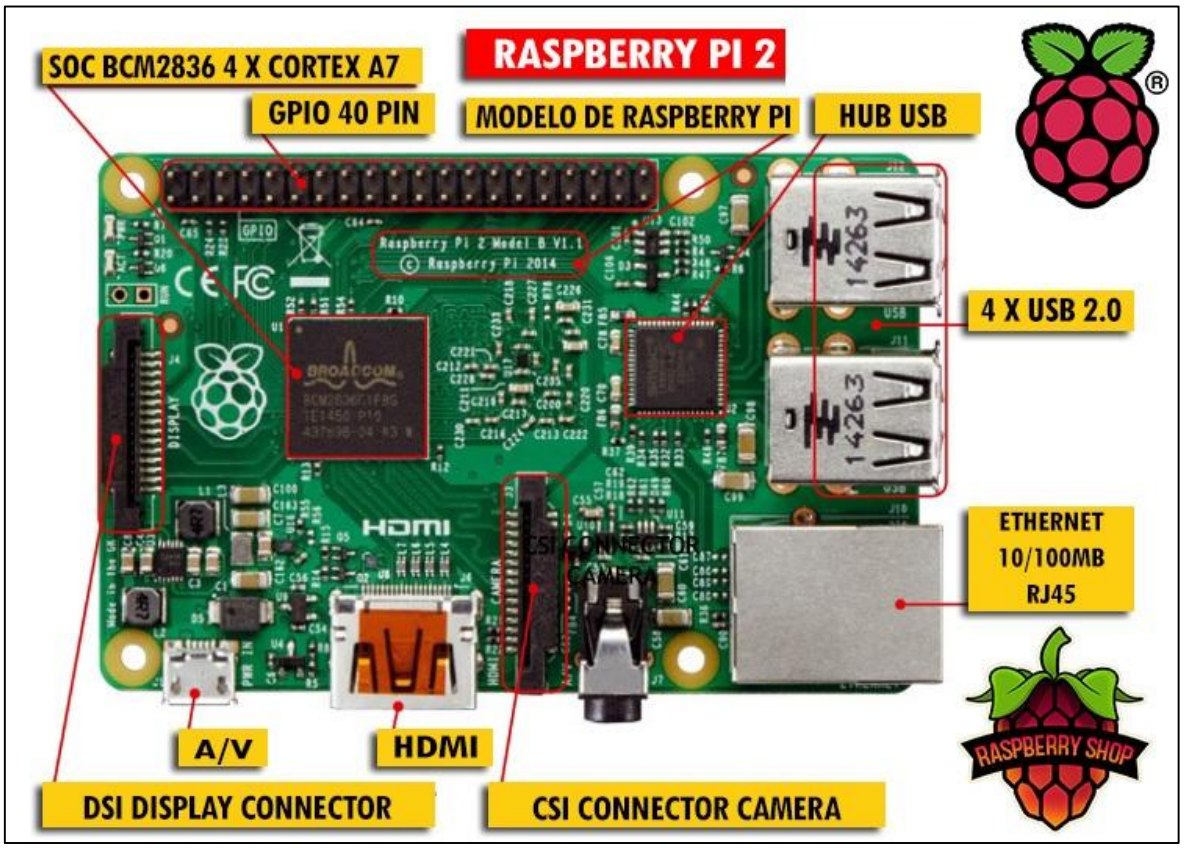


Figure 23 Raspberry Pi 2 layout [44].

5.7. CHAPTER CONCLUSION

In this chapter, it was first introduced a system overview, and the positioning of the cameras using RobotStudio. Next were presented some software tools that support the development of the project as OpenCV or Boost. Finally, were presented the main hardware devices used, the Pixy cameras and Raspberry Pi mini-computer.

6. WORK IMPLEMENTATION

This section describes the development of the project. Therefore, are described the electrical connections, the software developed and the integration of all components to meet the project goal. In Figure 24, is presented the project implementation data flow. In Annex B, is presented the electric circuit, including the computer, the level converter and the cameras.

6.1. PIXY - POWER AND COMMUNICATION

6.1.1. POWER CONNECTION

There are three ways to supply power to the Pixy:

1. supply through the USB connector (5 V regulated);
2. supply through the I/O connector (5 V regulated);
3. supply through the power connector (6 V to 10 V unregulated).

The camera can be powered simultaneously by the USB port and the I/O connector, without resulting problems. When the Pixy is connected to the computer via USB it is powered by the regulated 5 V output provided by the computer.

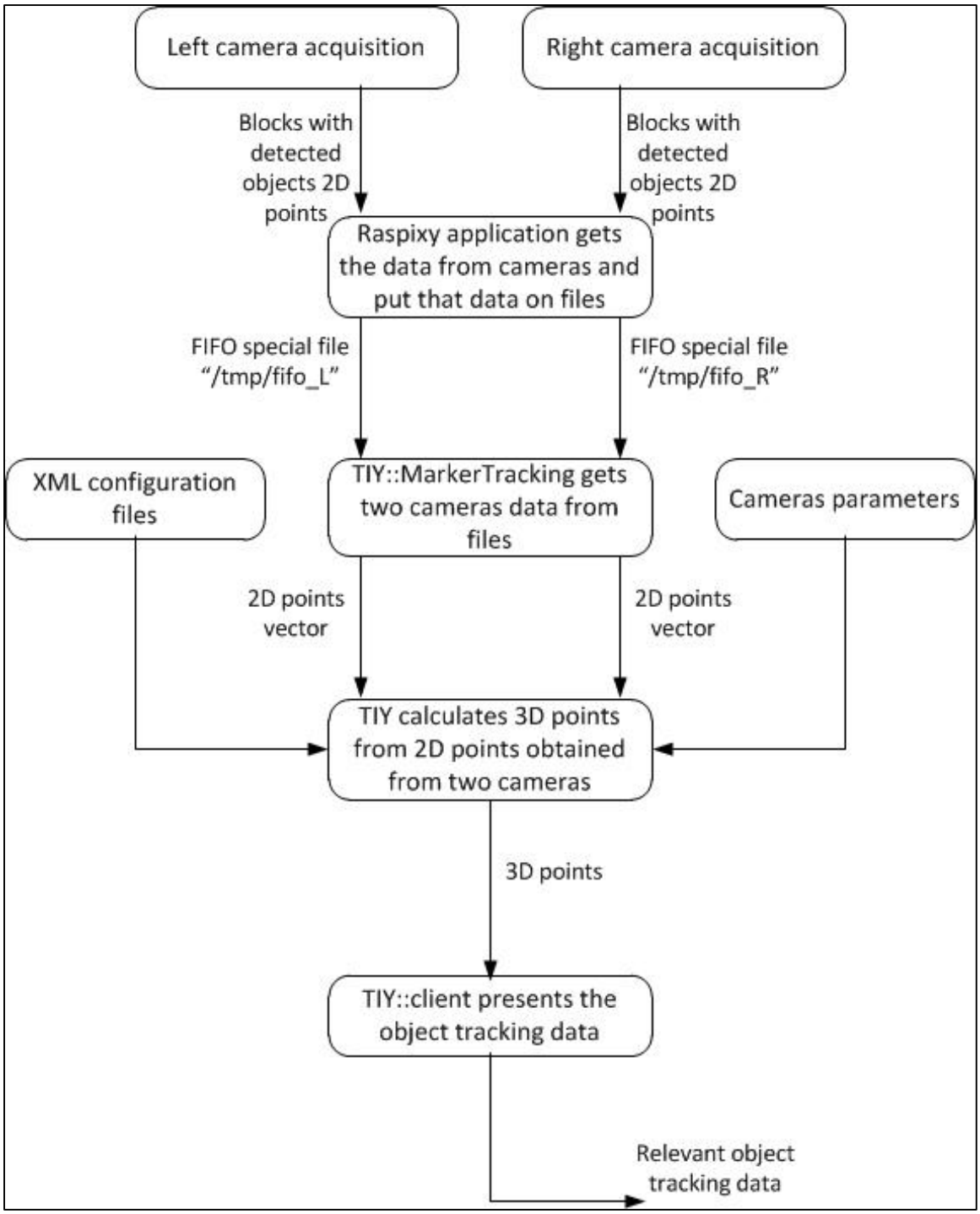


Figure 24 Overview of data flow in the whole project.

The power supply through the I/O connector accepts 5 V, with pin 2 being the input or output of +5 V Direct Current (DC) and pins 6, 8 and 10 being the ground pins. It is needed to pay special attention to the power supply wiring connection from the I/O connector, since there is no polarity reversal protection (Figure 25).

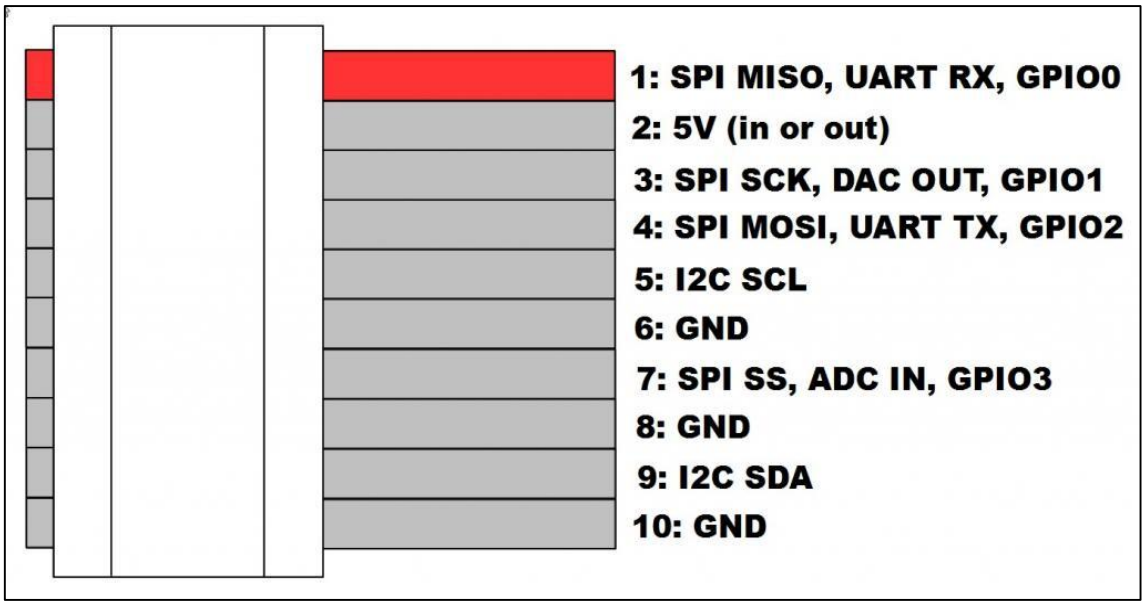


Figure 25 The I/O connector [45].

The third option, the supply of power from the power connector, allows providing more power to the camera with a 1.5 A limit, and it can feed the pan/tilt servomotors of the camera system, leaving still enough current to power a microcontroller from the I/O connector. Unlike the I/O connector, this one has protection against reverse polarity [45].

6.1.2. COMMUNICATION

Pixy has five possible different ways to communicate with other devices: from the USB port or using the I/O connector interface supporting SPI, I2C or UART. It can also communicate with the outside using the analog and digital outputs. The USB interface is the fastest of all interfaces, and allows transferring images smoothly between the pixy and a computer. Of the three serial interfaces provided by the I/O connector, the faster is the SPI interface, followed by the I2C interface. The UART interface is the slowest interface, and should only be used if it cannot be used one of the others interfaces mentioned above. If the controller does not support any of the available interfaces, the digital outputs and the analog output may be used instead.

The communication interface configuration can be set in the PixyMon software, being possible to select what type of interface to use, set the address for the camera in the I2C interface and set the communication speed on the UART interface (Figure 26). If the mouse pointer is hovered over the "Data out port" text, a help string will be displayed, describing which value corresponds to which type of port [46].

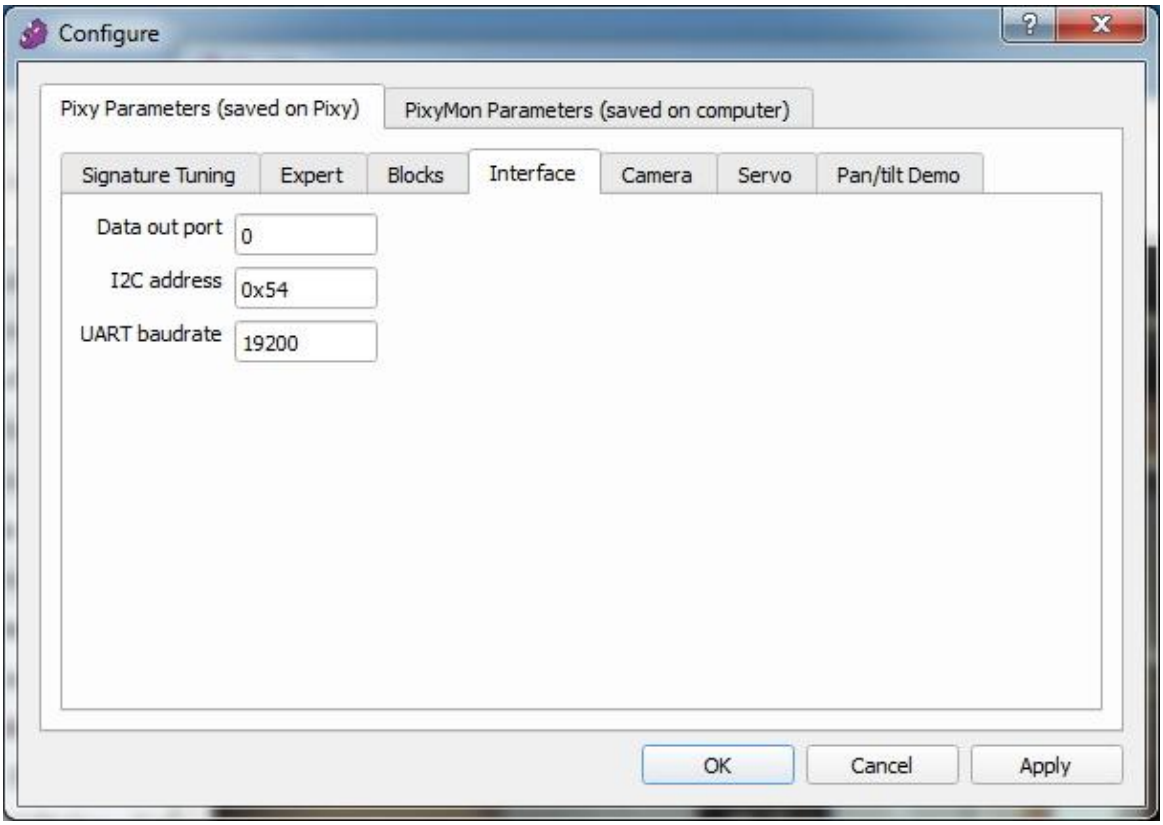


Figure 26 Pixy parameters configuration using PixyMon [46].

The different options available for interface selection are the following:

- In-Circuit Serial Programming (ICSP) SPI - this is the standard interface that uses pins 1, 3 and 4 of the I/O connector, and has the particularity of not using the slave select signal;

- SPI Slave Select (SS) - interface identical to the previous but with the particularity of being used pin 7 of the I/O connector to the slave select signal;

- I2C - interface that uses only two wires (pins 5 and 9 of the I/O connector) and allowing a master to communicate with up to 127 slaves. The "`I2C address`" allows setting Pixy's address;

- UART - a very common interface using pins 1 and 4 of the I/O connector. The camera receives data at pin 1 and transmits from the pin 4. The baudrate can be set in the parameter "`UART baudrate`";

- analog/digital x - this output provides the *x* value of the larger object detected, as an analog output (with a value between 0 and 3.3 V) in pin 3. It also indicates whether an object was detected or not from a digital output (pin 1 of the I/O connector);

- analog/digital y - this output provides the *y* value of the larger object detected, as an analog output (with a value between 0 and 3.3 V) in pin 3. It also indicates whether an object was detected or not from a digital output (pin 1 of the I/O connector).

The I2C interface was selected for the communication between the Pixy cameras and the Raspberry Pi controller. This interface provided by the Pixy cameras has the following features:

- the cameras work in slave mode and need polling;

- there are two resistors for pull-up to 5 V with 4.7 kΩ value, linked to the Serial Data (SDA) and Serial Clock (SCL) signals;

- the signals operate with voltages up to 5 V;

- the I2C address is configurable in the configuration parameters of each camera.

Regardless of the serial interface, the protocol used by the Pixy is always the same:

- the objects in each frame are sorted by size, with larger objects being sent first;

- it can be set the maximum number of objects to be sent by each frame ("`Max blocks`" parameter);

- SPI and I2C interfaces work in slave mode and require polling;

- when there are no objects detected, the camera sends zeros if the chosen interface is SPI or I2C;

- each object is sent within a block (see Table 2);

- all values within a block have a size of 16bit, being first sent the least significant byte (little endian).

The block format for each object is presented in Table 2.

Table 2 Object block format [46].

Bytes	Description
0, 1	Sync: 0xaa55 = normal object, 0xaa56 = color code object
2, 3	Checksum (sum of all 16bit words 2-6)
4, 5	Signature number
6, 7	*X* center of object
8, 9	*Y* center of object
10, 11	Width of object
12, 13	Height of object

An extra sync word is added to separate the frames (0xaa55). This means that a new image is indicated by the 0xaa55, 0xaa55 sequence, or the 0xaa55, 0xaa56 sequence [46].

Control information for Pixy

Data can be sent to control certain features of Pixy, like controlling the servo system of the pan/tilt, adjusting the brightness of the camera and control the Light Emitting Diode (LED) present in the camera, as described in Table 3, Table 4 and Table 5. Each word is 16 bit little endian (least significant byte is sent first) [46].

Table 3 Pan/tilt servomotor control [46].

Bytes	16 bit word	Description
0, 1	Yes	Servo sync (0xff00)
2, 3	Yes	Servo 0 (pan) position, between 0 and 1000
4, 5	Yes	Servo 1 (tilt) position, between 0 and 1000

Table 4 Camera brightness (exposure) control [46].

Bytes	16 bit word	Description
0, 1	Yes	Camera brightness sync (0xfd00)
2	No	Brightness value

Table 5 LED control [46].

Bytes	16 bit word	Description
0, 1	Yes	LED sync (0xfd00)
2	No	Red value
3	No	Green value
4	No	Blue value

6.2. RASPBERRY PI - POWER AND COMMUNICATION

6.2.1. POWER CONNECTION

The Raspberry Pi has a 5 V DC supply voltage. The power connector is a micro USB, commonly used by current mobile phones. In this case, is used a common charger for a mobile phone or tablet that has 5 V and 1000 mA output, and USB micro connector. The board has protections against polarity reversal and excessive power consumption.

The model used is the Raspberry Pi Model B (Figure 27), where the recommendation is a charger that provides between 700 mA and 1500 mA. The consumption of this minicomputer is approximately 500 mA, and it can deliver 500 mA through the USB ports and 50 mA through the GPIO [47]. The used cameras require 140 mA each, being fed by the computer power source. Given this, at least 780 mA is required (500 mA + 140 mA + 140 mA). In case of the Raspberry Pi 2, the power consumption is practically the same.

In addition to the micro USB connector, the computer may be powered by the GPIO pins, however, these pins do not have protections. Another way to power the computer is through one of the two USB ports, being possible to use an USB hub for this effect. However, it must be considered that according to the USB 2.0 standard, a USB port can only provide 500 mA, and this generally is the power supplied by the hub in each of the USB ports [48].

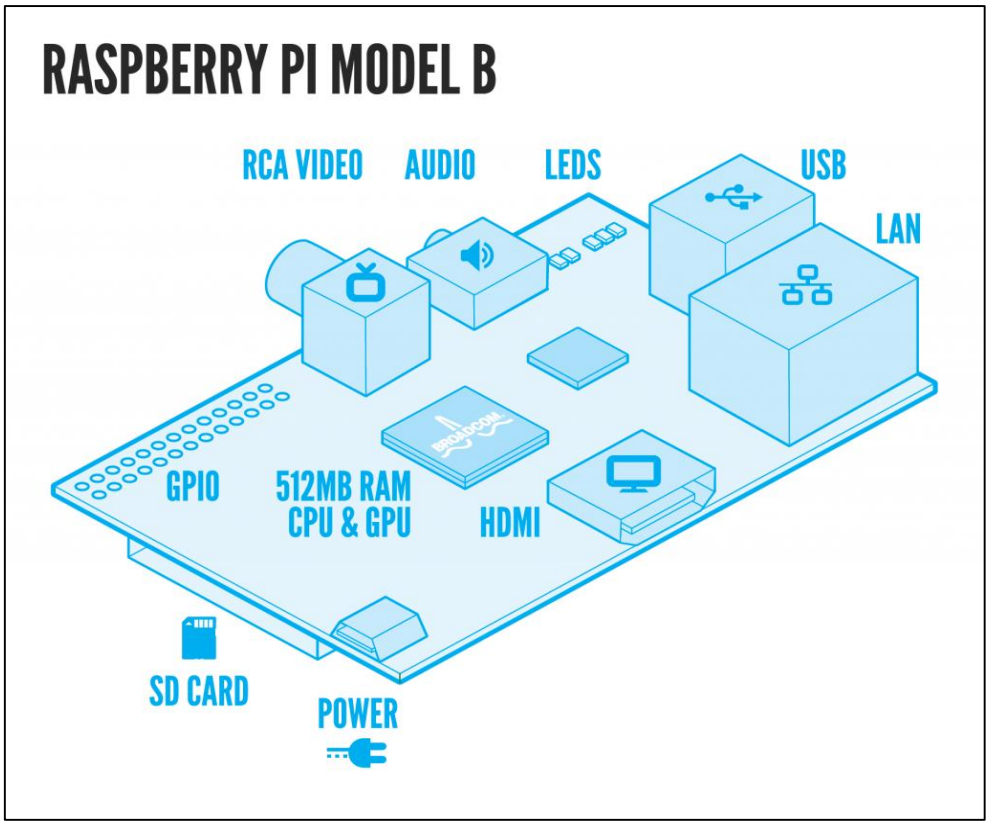


Figure 27 Raspberry Pi Model B [47].

6.2.2. COMMUNICATION

The Raspberry Pi Model B board provides various forms of communication with the outside environment. It has two USB2.0 ports, one 10/100 Ethernet port (with a RJ45 connector), and 26 GPIO that can be configured to support the SPI, I2C and UART protocols. The Ethernet port, for example, can be used to remotely access the system. The Raspberry Pi 2 is identical but it has 4 USB ports and the GPIO has 40 pins, with the first 26 having the same ports in relation to the Model B version.

There is a connector in the expansion header, identified as P1 and shown in Figure 28, having 26 pins for Pi Model B and 40 pins for Pi 2, spaced between them by 2.54 mm, and arranged as a 2 × 13 grid for Pi Model B and a 2 × 20 grid for Pi 2. In addition to providing GPIO, it also presents pins that provide +3.3 V, +5 V and GND. Viewed from the front, this connector has pin 1 in the first column of the bottom line. Since communication between the computer and the cameras is made through I2C, and the cameras are powered by the computer, the most relevant pins are (Figure 28):

- Pin 1: output voltage +3.3 V DC;

- Pin 2: output voltage + 5 V DC;
- Pin 3: I2C Serial DAta line (SDA) with 1.8 kΩ pull-up resistance;
- Pin 4: output voltage +5 V DC;
- Pin 5: I2C Serial Clock Line (SCL) with 1.8 kΩ pull-up resistance;
- Pin 6: GND.

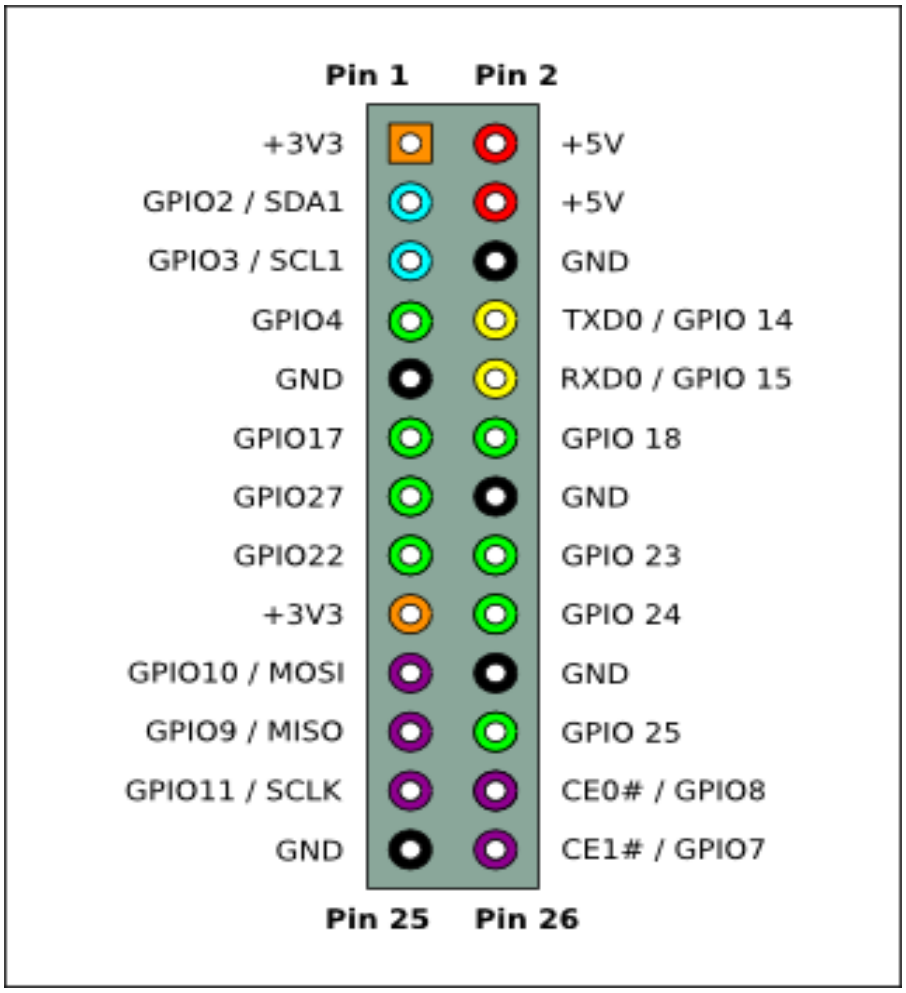


Figure 28 Raspberry Pi Model B GPIO pinout [49].

6.2.3. LEVEL CONVERTER

As the GPIO of the two cameras works with 5 V and the Raspberry Pi GPIO runs at 3.3 V, a level converter is required to meet the I2C communication between devices. Since the most common level converters do not behave as expected with the I2C [50], a suitable converter has been chosen, namely the 4-channel I2C safe bi-directional Logic Level Converter - BSS138 developed by Adafruit (Figure 29).

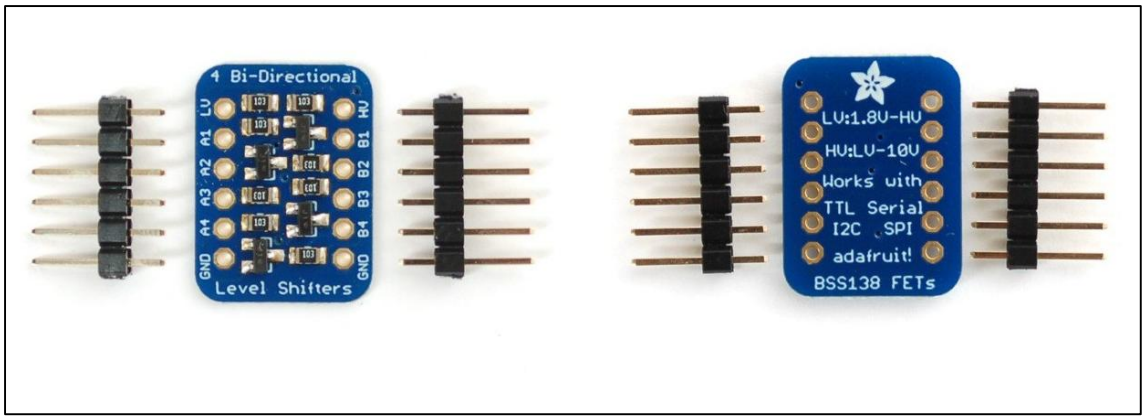


Figure 29 4-channel I2C-safe Bi-directional Logic Level Converter – BSS138 [52].

For signal level conversion are used 4 BSS138 Field Effect Transistors (FET), with 10 kΩ pull-up resistors. On the low voltage side can be used voltages down to 1.8 V and on the high level side can be used voltages up to 10 V [51].

6.3. INTER-PROCESS COMMUNICATION

The communication between processes, normally known for Inter-Process Communication (IPC), enables the transfer of data between different processes that can be on the same physical machine, or on different machines. The types of communication between processes more commonly used are:

- shared memory: allows processes to communicate by simply reading and writing in a specific memory space;
- mapped memory: similar to shared memory, with the difference that is used a system file for data sharing;
- pipes: allow sequential communication from one process to another related process;
- First In, First Out (FIFO) IPC: identical to pipes, however unrelated processes can communicate with each other because it is given a name to the pipe in the file system;
- sockets: supports communication between unrelated processes, even between different machines.

Note that some of these methods can be used to perform communication within the same process.

These types of IPC differ among them according to the following criteria:

- restricted communication between related processes (processes with a common ancestor), communication between unrelated processes that share the same system, or communication between processes that are on different systems and communicate over a network;
- ability of a process to only write or read data, or reading and writing ability by the process;
- the number of processes that can communicate using the type of IPC;
- if it is the IPC that controls the timing of communication.

6.3.1. SHARED MEMORY

Shared memory is one of the simplest methods of IPC. It allows two or more processes to access the same portion of memory just as if each one of the processes executes a request for memory to the system and receives a pointer to the same portion of memory. When a process changes data in that memory portion, all other processes have access to that change.

Since all processes involved share the same portion of memory, shared memory is the fastest form of IPC. System calls, entries in the kernel and copying data are avoided.

Because the kernel does not synchronize accesses to shared memory, it must be provided a mechanism that deals with the synchronization of processes. Processes should not be able to read data before that data is written, and two processes should not be able to write data in the same memory location at the same time. The normally used strategy to solve this synchronization problem is using semaphores.

Shared memory allows quickly two-way communication between multiple processes. Each process can read and write, but precautions should be taken with regard to timing between processes [53].

6.3.2. MAPPED MEMORY

Mapped memory allows different processes to communicate with each other using a shared file. It is given a name to that file, which creates an association between the file and the

process memory. For Linux, the files are loaded into the virtual memory system in order to be accessed from the address space of the processes. Thus, the processes can access the contents of the file with a memory access, making access to files rather quickly.

Mapped memory creates a buffer to store the file content in the system memory. Reads and writes are performed in this buffer. The system updates the file if the buffer content changes [53].

6.3.3. PIPES

A pipe allows unidirectional communication. Data written in the writing side is read in the reading side. The pipes are serial devices, *i.e.*, the data is always read in the same order it was written. Normally, pipes are used to communicate between threads within a single process, or between a parent process and a child process.

The ability of a pipe to store data is limited. If the writing on pipe is faster than reading the data and the pipe cannot accommodate more data, the writing process is blocked until something is read and space released. If a process attempts to read the pipe and it has no data, the process blocks until data is written to the pipe that can be consumed. Thus, pipes automatically synchronize the processes.

The creation of a pipe causes a file descriptor which is valid only for this process or to one of the child processes. The pipes cannot be used by processes that are not related [53].

6.3.4. FIFO

A FIFO file is a pipe that has a name in the file system. Any process can open or close a FIFO, and the processes that are linked to this type of pipe need not be related. FIFO are also known as named pipes.

To create a FIFO it must be specified a path to it and the permission level that will allow the processes to write and read in the related file. Access to the FIFO can be done as if it was a normal file. Communication using FIFO requires a process to open it for writing and another open it for reading. The common system I/O functions, and functions provided by the C programming language libraries can be used to open, write, read and close the FIFO.

A FIFO can have multiple processes writing also as multiple processes reading data. As with the pipes, the FIFO has limited capacity which is system dependent. There are also differences in the behavior of FIFO in different systems; in Linux the FIFO are unidirectional, while on Windows, the FIFO behave similarly to sockets - can serve as a mean of communication between processes of different machines and enable two-way communication [53].

6.3.5. SOCKETS

A socket allows bidirectional communication between processes that may be running on the same machine, as well as on different machines. Sockets are widely used by programs running on the Internet due to its facilities.

When a socket is created, three parameters must be specified: communication style, namespace, and the protocol.

The communication style specifies how the socket will process the data. When data is sent via sockets, it is grouped into packets. The communication style determines how these packets are handled and how they are addressed from the sender to the receiver.

The socket namespace specifies how the addresses are written. One address uniquely identifies a socket connection of each one of the sides. A local address to a socket needs only to specify a file name. In the case of an established connection on the Internet, it must be specified the Internet Protocol (IP) of the machine and the port number. The port number distinguishes a socket from several others on the same machine.

The protocol determines how data is transmitted. The best known is the Transmission Control Protocol/Internet Protocol (TCP/IP), but there are others, like the local communication protocol of UNIX [53].

6.3.6. USED METHOD

For the communication between the process that performs the Pixy cameras data acquisition and the process responsible to obtain the 3D positioning (TIY), are used two named pipes (FIFO), one for each camera.

Both processes run on the same machine, communication is one-way, and the process that obtains data from the cameras creates two FIFO, one for each camera: `/tmp/fifo_L` for the left camera data and `/tmp/fifo_R` for the right camera data. The blobs written by the data acquisition process are then read by the process that handles the calculations for obtaining the 3D positioning, respectively for each of the two cameras FIFO.

6.4. GETTING BLOBS FROM CAMERAS (RASPIXY PROCESS)

One of this project parts is to get blobs with the detected objects position. This operation is done by getting the data from the cameras and sending the required information for TIY, which subsequently performs the 3D positioning, from 2D data of the two cameras. This process is done by the `raspixy` application developed in the C++ programming language (which is described below), and that depends on some developed libraries (Annex C).

Initially the I2C addresses for each of the cameras must be set. These addresses must match the addresses assigned to each of the cameras, using the Pixymon software. It is used the hexadecimal address 10 to the right camera and 20 to the left camera, according to:

```
Pixy pixy_R(0x10); // Pixy address of right Pixy
camera
Pixy pixy_L(0x20); // Pixy address of left Pixy
camera
```

In this project were chosen the named pipes for inter-process communication. These pipes are created in the folder intended for temporary files storage by the Linux system, using the `mkfifo` command, with read and write access. Then are declared and initialized two file descriptors, one for each camera, which are used for handling the pipes, as follows:

```
const char* fifo_R = "/tmp/fifo_R";
const char* fifo_L = "/tmp/fifo_L";
mkfifo(fifo_R, 0666);
mkfifo(fifo_L, 0666);
int fd_R = open(fifo_R, O_WRONLY|O_NONBLOCK);
int fd_L = open(fifo_L, O_WRONLY|O_NONBLOCK);
```

The pipes are open only for writing and, using the O_NONBLOCK flag, an error is only returned if there is no process to read the pipe.

After the initial settings, this application is limited to running an infinite loop that runs until the reading process stops.

In each cycle are read the data of each of the cameras, according to:

```
// get block from Pixy cameras
blocks_R = pixy_R.getBlocks();
blocks_L = pixy_L.getBlocks();
```

Due to the processing limitations of the used computer, and also for debugging purposes, a delay at the beginning of each cycle may be used.

With the used configuration, the Pixy cameras can return data from multiple colors signatures, *i.e.*, several points. To make the 3D positioning processing, the same number of points must be used for both cameras, so it is used the lowest number of detected points between the two cameras:

```
// get the number of detected objects (camera
with the minimum detected)
if (blocks_R < blocks_L)
  num_blocks = blocks_R;
else
  num_blocks = blocks_L;
```

Possibly it may not exists an object detected by the cameras, due, for example, to be no object to detect in the cameras field of view. If signatures are detected for one or more objects, these are written in the respective pipes. The next code takes into account the particularities referred to in this paragraph:

```
if (blocks_R && blocks_L)     // print the
detected objects
{
  if ((i%1)==0)  // Pixy outputs data 50 times
per second
  {
        i = 1;

        std::cout << "Left number of blocks: " <<
blocks_L << std::endl;
        std::cout << "Right number of blocks: "
<< blocks_R << std::endl;

        std::cout << "Number of detected objects:
" << num_blocks << std::endl;
        std::cout << "R: ";
        for(int i = 0; i < num_blocks; i++)
        {
              blob_temp << pixy_R.blocks[i].x <<
"\t" << pixy_R.blocks[i].y << std::endl;
              blob_R = blob_temp.str();
              write(fd_R, blob_R.c_str(),
(unsigned)strlen(blob_R.c_str()));
              blob_temp.clear();
              blob_temp.str("");
```

```
                std::cout << blob_R.c_str() <<
std::endl;
        }

        std::cout << "L: ";
        for(int i = 0; i < num_blocks; i++)
        {
                blob_temp << pixy_L.blocks[i].x <<
"\t" << pixy_L.blocks[i].y << std::endl;
                blob_L = blob_temp.str();
                write(fd_L, blob_L.c_str(),
(unsigned)strlen(blob_L.c_str()));
                blob_temp.clear();
                blob_temp.str("");
                std::cout << blob_L.c_str() <<
std::endl;
        }
   }
   i++;
}
else
{
   std::cout << "No blocks detected" << std::endl;
}
```

Finally, if an exit occurs in the infinite loop, the file descriptors for the created named pipes are closed and the process stops:

```
close(fd_R);
close(fd_L);
```

6.5. USING TIY TO CALCULATE THE 3D MARKER POSITION

The TIY can implement a simple form of tracking system, providing the source code freely, in order to be adapted.

It was required to make some changes to the TIY source code. The TIY is designed to be used primarily with video from two cameras, as input. In this project, data input to the TIY is made from two text files, so are eliminated some code sections that were used only for the case of video signal as data input.

In order to simplify the remaining code, parts of the code that are not needed were deleted, and additional code was developed and added to meet the requirements of the project.

The TIY is divided by several source files. Some of the most important and that need of special attention are described in the sequel.

6.5.1. SERVER

The `server` source code file, as the name implies, is the TIY server component. Initially it deals with some settings, which can be edited in the xml files, available in TIY main folder. Then it calls the functions responsible for the data acquisition from cameras, video files, images, or point files. In the case of this project, as pipes are used for inter-process communications, data acquisition is performed from two text files with 2D points.

Essentially, there are created two points vectors, one for each of the cameras:

```
std::vector< cv::Point2f > points_2D_left,
points_2D_right;
```

Next it is called a function to load the vectors with the 2D points from the text files (one call to each of the cameras):

```
m_track.get2DPointsFromFile('l',
&points_2D_left);
m_track.get2DPointsFromFile('r',
&points_2D_right);
```

After the return from the calls to the last presented functions, the 3D points are calculated from the 2D points, using the following function:

```
cv::Mat points_3D =
m_track.get3DPointsFrom2DPoints(points_2D_left,
points_2D_right);
```

With the 3D points calculated, the server application outputs the desired data. The behavior of the output of this application is affected by editable xml files.

6.5.2. MARKERTRACKING

The source code file `MarkerTracking` has some of the most important functions for the operation of the TIY application.

One of the first steps is to create two file descriptors, which are directed to named pipes created to hold the 2D points of the two cameras described before.

```
// FIFO between Pixy's and TIY
#define MAX_BUF 256
const char* fifo_R = "/tmp/fifo_R";
const char* fifo_L = "/tmp/fifo_L";
char buf_R[MAX_BUF];
char buf_L[MAX_BUF];

// open the FIFO
int fd_R = open(fifo_R, O_RDONLY);
```

```
int fd_L = open(fifo_L, O_RDONLY);
```

These file descriptors only allow reading the 2D points.

A major function is concerned with reading the 2D points from the named pipes. For this, it is used the `get2DPointsFromFile` function, that takes a character to identify the right or left camera and a reference to the vector location of the 2D points defined in the file server, a vector to the left camera and a vector for the right camera. Initially, the 2D points are read from named pipes to a buffer, which are then transferred to a `stringstream` facilitating data read operations. Points are read into temporary variables and then inserted in the 2D points vector of the respective camera. Next, is presented the code for the left camera. The process for the right camera is analogous.

```
MarkerTracking::get2DPointsFromFile(const char
camera, std::vector< cv::Point2f > *points_2D)
{
  cv::Point2f L_2D_point, R_2D_point;
  points_2D->clear();

  if(camera == 'l')
  {
        read(fd_L, buf_L, MAX_BUF);
        std::stringstream linestream_L(buf_L);
        while (linestream_L >> L_2D_point.x)
        {
              if (linestream_L >> L_2D_point.y)
              {
                    points_2D-
>push_back(L_2D_point);
              }
              else
                    break;
        }
  }
}
```

Another equally important function is the one that allows generating 3D points from two 2D points vectors, one for each of the two cameras. The `get3DPointsFrom2DPoints` function takes two points, one for the left camera and another for the right camera, and returns an array with the 3D points, calculated from the 2D points using correspondence optimization and triangulation.

After obtained the markers 3D points, it is calculated the detected object position in the three-dimensional space, using the `fit3DPointsToObjectTemplate` function.

6.5.3. CLIENT

The TIY client application was provided by the developers of TIY in order to allow extracting the desired information from the 3D tracking software. In the case of this project it is used to communicate with the rest of the system.

The client application consists on an infinite cycle, which outputs the pointing device positioning. It may be outputted other data if desired, and the data format may also be altered to meet the project requirements.

6.6. CHAPTER CONCLUSION

In this chapter was made available information about the implementation of the project, in terms of both hardware and software.

The chapter begins with the hardware description, which is primarily focused on the integration and communication of the various components involved.

Next it was described the software, starting with an overview of the inter-processes communication, crucial to allow the best possible performance for this project, since it uses a computer with low processing power. Are also described changes to the TIY that allow performing 3D tracking from the 2D points provided by the Pixy cameras. An application has been developed that takes care of getting the data from the cameras and provide them to TIY, which is also described in this chapter.

7. IMPLEMENTATION TESTING

This chapter presents the testing results of the tracking system, developed in the previous chapter.

7.1. FIRST IMPLEMENTATION TESTING

The first system tests were made with the Pixy cameras original lenses. These lenses have a horizontal field of view of 75º, a vertical field of view of 47º, with M12 mounting and focal length of 2.8 mm. The used sensor is an Omnivision OV9715, with the size of 1/4" and a resolution of 1280 × 800 pixels [54] [55].

Bearing in mind that the resolution provided by Pixy for data output is 320 × 200 pixels, and the distance at which the cameras are mounted to the work area is approximately 1.5 meters, with the original lens, the results obtained are not satisfactory.

The initial tests were made with Pixy tracking small blue circumferences with a 3 cm diameter. With a 1.5 m distance from the cameras to the workspace, it is not possible to detect the circumferences, using the hardware described above, and the software provided by Pixy camera that uses the tracking based on color.

As can be seen in the Figure 30, circumferences placed on a white sheet, even visible to the human eye, are hardly noticeable in both its shape and in its color, and are not properly detected by the installed setup, due to low detection capability of the used cameras.

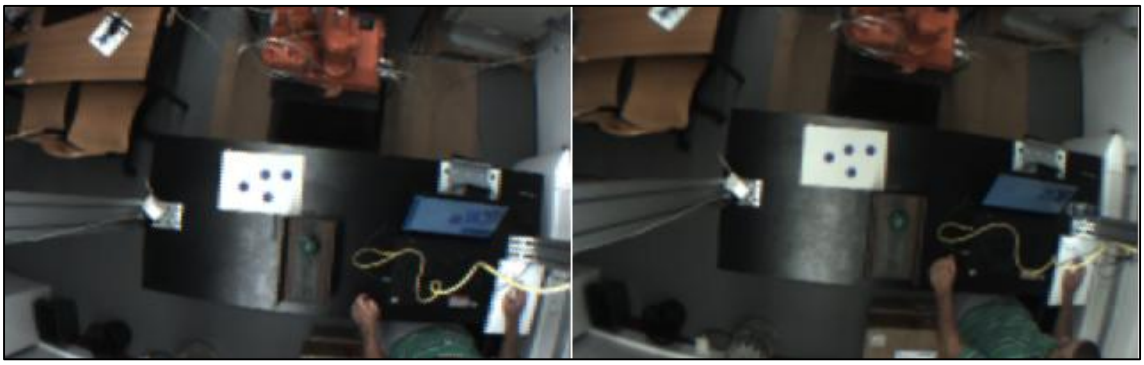


Figure 30 Left Pixy camera snapshot on left, and right Pixy camera snapshot on right, using the default hardware.

To make the circumferences being detected by the cameras, it is necessary to approximate the cameras to the workspace. From about a meter away, the 3 cm circumferences begin to be detected by the tracking system, and with a distance of 50 cm away, the circumferences are perfectly detected with no false negatives.

Due to the bad results of the initial tests, and to improve the detection of objects in the workspace without changing the distance at which the cameras are mounted, the camera lenses were replaced with better ones.

7.2. LENSES SELECTION

Defining an 800 × 600 mm field of view with a distance of 1.5 m from the position of the cameras to the workspace, and with a 1/4"sensor size, the lenses should have approximately 6 mm of focal length, as shown in Figure 31, or by using the following expression [56] to calculate the ideal lens characteristics:

$$focal\ length \cong \frac{sensor\ size \times working\ distance}{field\ of\ view + sensor\ size} \quad (17)$$

Taking into account that a 1/4" sensor has an approximate size of 3.6 mm width and 2.7 mm height, with a 4.5 mm diagonal, [56] gives:

$$focal\ length \cong \frac{3.6 \times 1500}{800 + 3.6} \cong 6.7 \quad (18)$$

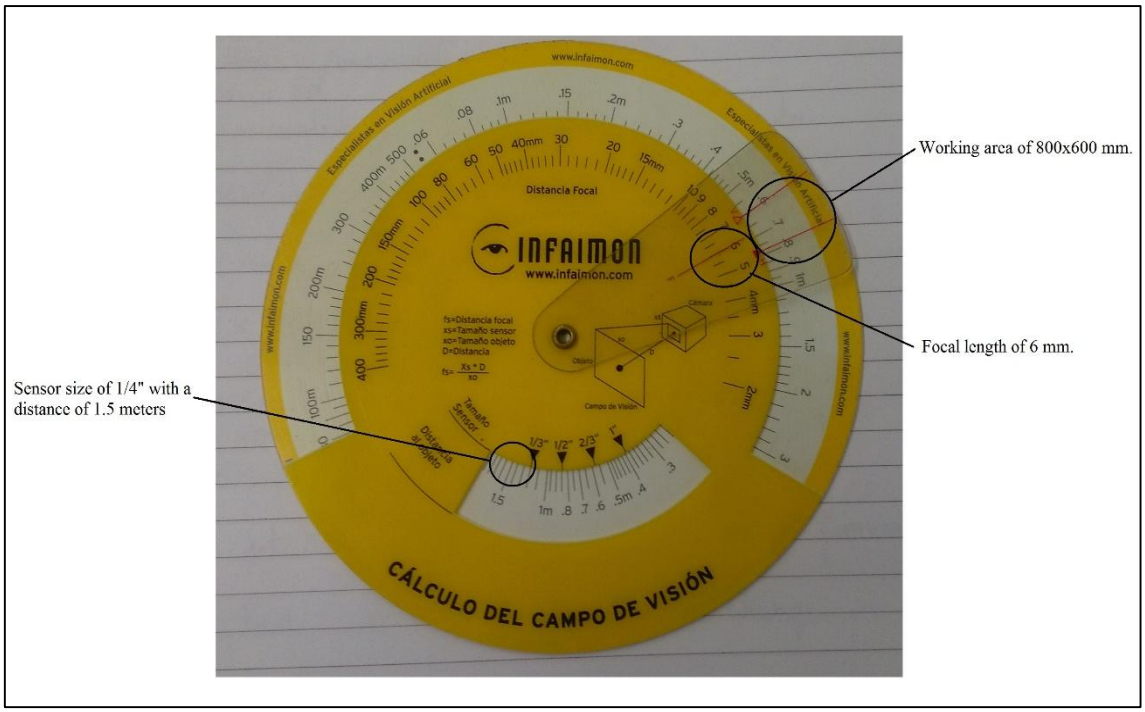


Figure 31 Focal length selection.

In the market, the offering of lenses for 1/4" sensors is practically nonexistent, but it is easy to find out lenses designed for 1/3" sensors and larger sizes. A lens intended for a sensor size, can however be used in a smaller sensor. Therefore, a lens for a 1/3" sensor was chosen, because of its good availability, being necessary to make the correction consisting, in this case, of dividing the focal length by 1.33 [56]:

$$6.7 \div 1.33 = 5.03 \tag{19}$$

Doing some rounds and taking into account the lenses available on the market, the lens chosen, is a 6 mm lens for the presented values.

The lenses ordered to replace the original lens of Pixy, were the M12*0.5 lens, with a focal length of 6 mm and a viewing angle of 51º [57].

7.3. LENSES CHANGING AND IR FILTER PLACING

Not having immediately available the optimal lenses, other filters were mounted in the two cameras that allow the passage of Infrared (IR) light, and an effective detection of the IR pointing device. The filters are part of a kit which is available on the market, IR-LOCK Filter Kit for Pixy [58], which is also composed of a M12 lens with a focal length of 3.6 mm.

Therefore, the original lens of the Pixy camera, with focal length of 2.8 mm were replaced by 3.6 mm lenses that allow better focusing of the IR pointing device. With the original 2.8 mm lens, the field of view width is approximately 1929 mm, according to the following formula:

$$x \cong \frac{3.6 \times 1500}{2.8} \cong 1929\, mm \tag{20}$$

while with the lenses with focal length of 3.6 mm, the field of view width is approximately 1500 mm, as shown:

$$x \cong \frac{3.6 \times 1500}{3.6} \cong 1500\, mm \tag{21}$$

The field of view achieved with the new lens is still large, considering that the working space is quite smaller and the resolution of the cameras is very low.

The kit provides the assembly instructions of the filter [59]. The filter assembly consists essentially of loosening the lens support that is over the sensor, and place the filter fixed between this support and the sensor.

To replace the lens, it must be removed the one which is present in the Pixy, rotating the lens counterclockwise, and fixing the new performing the reverse movement. The screw on the lens, beyond the mounting purposes, serves to perform the focusing.

The project that provided the IR-LOCK kit [58] also provides a modified firmware for the cameras. This firmware allows the cameras capturing the IR light. The installed firmware version was the `firmware_IRLOCKpixy_1.0.1.hex`, that is present in the IR-LOCK repository [60], and was also used a version of Pixymon that supports this firmware (`pixymon_windows-1.0.2beta.exe`), also present in the IR-LOCK repository.

After the installation of the new filters and lenses in the cameras, it was necessary to make a new camera calibration. The calibration method is described in Annex D. The results of the left camera calibration are shown below:

```
Intrinsic parameters:
Focal Length:          fc = [ 650.49356
651.31588 ] ± [ 8.11176   7.73068 ]
Principal point:       cc = [ 315.49340
225.95000 ] ± [ 7.60872   7.25265 ]
Skew:             alpha_c = [ 0.00000 ] ± [
0.00000  ]   => angle of pixel axes = 90.00000 ±
0.00000 degrees
```

```
Distortion:            kc = [ -0.44858   0.23859
-0.00404   -0.00040  0.00000 ] ± [ 0.03039
0.12578   0.00287   0.00192  0.00000 ]
Pixel error:          err = [ 0.75598   0.66664 ]

Extrinsic parameters:
Translation vector: Tc_ext = [ -89.335736      -
65.894073     769.247541 ]
Rotation vector:   omc_ext = [ -2.088204       -
2.021539      0.168527 ]
Rotation matrix:    Rc_ext = [ 0.041793
0.969761      -0.240453
                               0.996189   -
0.022004      0.084403
                               0.076560   -
0.243064      -0.966984 ]
Pixel error:           err = [ 0.88853   1.04480
]
```

and the results for the right camera are shown in the following:

```
Intrinsic parameters:
Focal Length:          fc = [ 643.51450
643.66794 ] ± [ 6.72942   6.78550 ]
Principal point:       cc = [ 315.81863
160.84134 ] ± [ 9.54332   8.00484 ]
Skew:             alpha_c = [ 0.00000 ] ± [
0.00000  ]   => angle of pixel axes = 90.00000 ±
0.00000 degrees
Distortion:            kc = [ -0.44881   0.28018
0.00413   0.00034  0.00000 ] ± [ 0.03210
0.13265   0.00247   0.00191  0.00000 ]
Pixel error:          err = [ 0.85006   0.73812 ]

Extrinsic parameters:
Translation vector: Tc_ext = [ -92.378224
14.935854     723.326198 ]
Rotation vector:   omc_ext = [ 2.148095
2.028876      0.106836 ]
Rotation matrix:    Rc_ext = [ 0.063705
0.981929      0.178208
                               0.995214   -
0.049250      -0.084398
                               -0.074096
0.182732      -0.980367 ]
Pixel error:           err = [ 0.86054   0.75437
]
```

7.4. SETUPS USED TO TEST THE TRACKING SYSTEM

Two different setups were used to test the tracking system: one where the cameras are mounted above the working area, and one where the cameras are coupled to the robotic arm.

It was used a structure in aluminum rail for the mounting above the working area, where the cameras are mounted, about 1.5 m away from the working space (Figure 32). In the other case, in which the cameras are on the robotic arm, it was used a sealed box inside of which were accommodated the cameras and the computer.

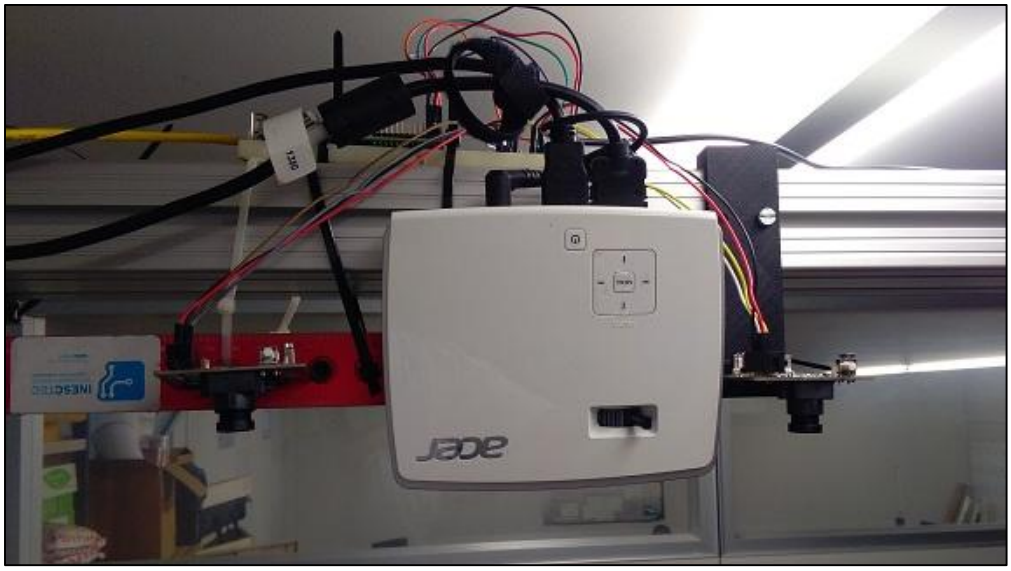


Figure 32 Setup where the cameras are mounted above the working area. It can be seen the left and the right camera and a projector in middle. The computer is above the rail.

7.5. POINTING DEVICE

The object to be detected by the tracking system, needs to emit IR light or have reflectors to reflect the light emitted by an IR external source. By imposition of TIY, the object needs at least four IR LED or four reflectors for an accurate tracking [61].

For testing purposes, it was developed a setup with 4 IR LED, arranged in a breadboard to simulate the pointing device (Figure 33).

After teaching the Pixy cameras to detect the position of the IR LED, using the select signature in PixyMon software, it can be performed the pointing device tracking. In Figure 34, can be verified the detection of the 4 signatures from Pixymon, representing the 4 LED of the pointing device, detected by one of the two cameras.

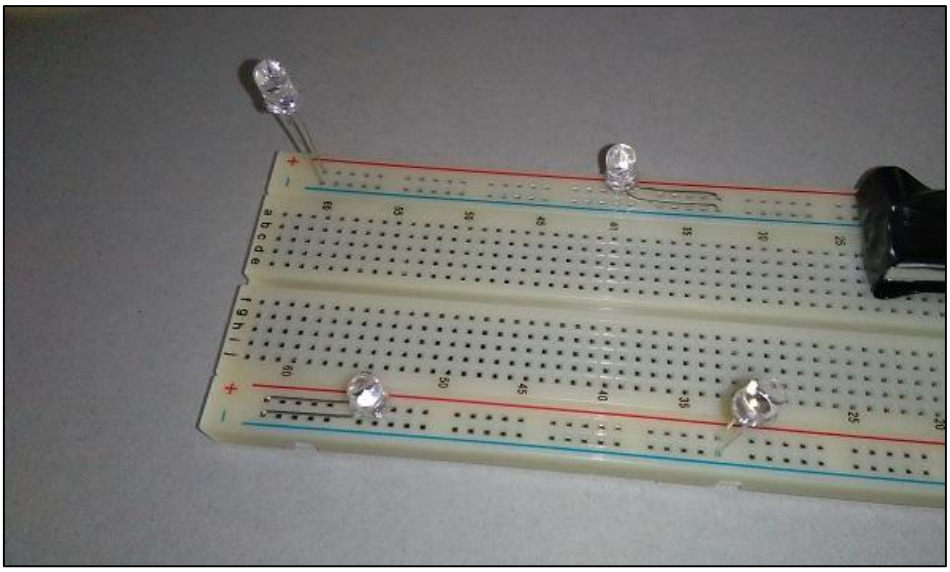


Figure 33 Setup with 4 IR LED to simulate the pointing device.

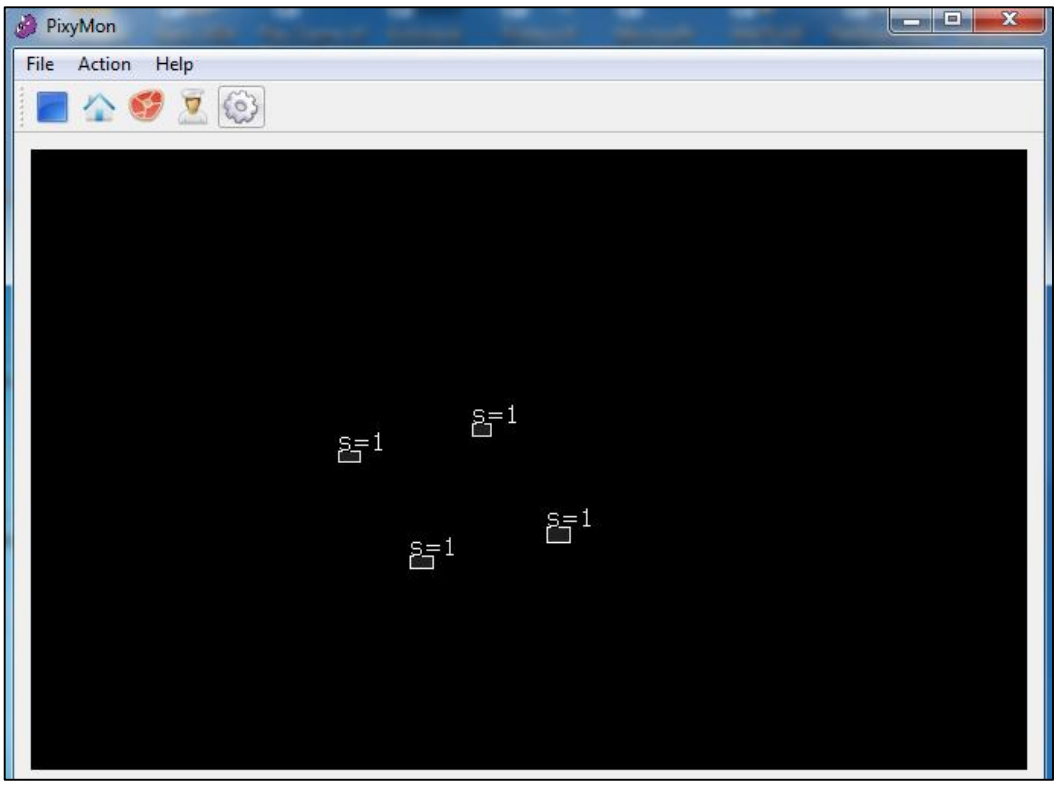


Figure 34 Detection of the 4 signatures representing the 4 LED's of the pointing device.

7.5.1. POINTING DEVICE CALIBRATION

For proper pointing device tracking, it is required for the TIY to know the exact arrangement of the markers (reflectors or LED) that are coupled to it. The TIY provides a simple method

for performing this operation, which consists in a calibration of the pointing device, and the procedure is as follows:

- it is necessary to log the 3D points of at least 10 different positions and orientations of the markers attached to the pointing device. To do this, and if the used cameras are identified by OpenCV, it must be enabled the `<do_log_3D>` option in `config_run_parameters.xml` configuration file and pressing the `SPACE` key (or the left mouse button, if this is the chosen input method) for each pose. The log file, `<log_points_3D>`, is saved in the `tiy_log` folder. The log file must be renamed to `capture_template_1.dat`;

- the cameras used in the project are not recognized by OpenCV, so it is required another method to obtain the 3D points. One simple way, is output the 3D points to a text file, during the execution of TIY. This text file must be edited and renamed to `capture_template_1.dat`;

- move the log file to `tiy_calibration/marker_object_calibration` folder;

- replace the `config_object.xml` file there with the project file located in TIY folder. Run the script `make_templates_from_3dpoints.m` to place the calibration information in `config_object.xml` file (if missing the `write_XML.m` and the `change_XML.m` files in `fcn` folder, it is necessary to copy the files located in `camera_calibration/fcn` folder). Finally, copy the changed configuration file for TIY folder;

- if it is also needed to define a virtual point (a point fixed to the object without a marker, that is desired to be tracked; e.g. the peak of the pointing device, repeat all the steps, except for turning the object around the fixed (in position) virtual point while logging and using the `capture_virt_point_1.dat` log file and the `make_virtual_point.m` script.

7.6. TRACKING SYSTEM TESTS

Running the tracking system in the two used computers, in other words, starting the server TIY and raspixy application developed to obtain the 2D object points provided by the cameras, tests were made to validate its operation, and determine its performance.

In the case of a tracking system it is required to determine the rate at which the object can be detected, bearing in mind that the object may be in motion. With a low refresh rate it may not be possible to correctly detect the position of an object that moves at high speed. In this project, being required to track a pointing device to be handled manually by a human operator, it is appropriate to use the highest possible rate, with the limit of 50 Hz (the maximum rate provided by the Pixy cameras).

After making several tests with both computers, it was possible to find the maximum rate at which the tracking system can function properly. For the Raspberry Pi Model B with the CPU speed increased to 900 MHz, the maximum rate is 5 Hz, or 5 samples per second – above this value, the tracking system cannot keep up with the rate at which the data from the cameras are provided. In the case of Raspberry Pi 2 with the CPU speed at 900 MHz, the maximum recommended rate is 16.67 Hz, or 16.67 samples per second – if increased the cameras sampling rate, the tracking system can no longer keep the rate of the provided data.

Table 6 depicts the computational times to perform the main TIY tasks, in the Raspberry Pi Model B with the CPU overclocked to 900 MHz.

Table 6 Raspberry Pi Model B (900 MHz) working at 5 Hz.

Sample	Lens Undistortion (us)	3D correspondence candidates (us)	3D triangulation (us)	Total time (us)
1	299	2879	12878	16056
2	299	1132	2172	3603
3	296	996	23040	24332
4	301	1001	9542	10844
5	297	1596	15251	17144
Mean	298	1521	12577	14396

Table 7 presents the computational times to perform the main TIY tasks, in the Raspberry Pi 2 with the CPU working at 900 MHz.

Table 7 Raspberry Pi 2 working at 16.67 Hz.

Sample	Lens Undistortion (us)	3D correspondence candidates (us)	3D triangulation (us)	Total time (us)
1	263	1307	2131	3701
2	356	2386	3786	6528
3	217	2341	3358	5916
4	250	2466	3001	5717
5	212	1059	1730	3001
Mean	260	1912	2801	4973

7.7. CHAPTER CONCLUSION

As shown by several tests, the implemented tracking system works, although in the case of the Raspberry Pi Model B the maximum data rate refresh is on the order of 5 Hz. As the Raspberry Pi 2 is on the market for the same price that was the Model B, it is preferable to use the Pi 2 due to having a greater processing power, providing a maximum data rate refresh in the order of 17 Hz.

8. CONCLUSIONS

The work presented in this Dissertation was developed to offer a solution for the proposed thesis provided by INESC TEC, the CLARISSA demonstrator. The goal of the CLARISSA demonstrator is to provide robotic solutions that can be integrated and used in small and medium enterprises, in which the robotic systems should have a relatively low cost and must be simple to program and operate. The main task was to develop an infrared teaching system for part positioning and correction.

The IR teaching system proposed for part positioning and correction consists of a stereo vision system, using two cameras, and software able to calculate the 3D position of a pointing device. The used cameras allow perform objects 2D tracking using detection algorithms based on color differences, making the output of the coordinates of detected objects. With objects coordinates, is calculated and provided the 3D positioning of those objects, using for it the developed software.

After the tests, it is clear that the tracking system works well being limited by the processing power of the used computer, and the low resolution of the Pixy cameras.

8.1. FUTURE WORK

The main goal, developing a tracking system, was achieved. However there are some limitations and some aspects that could be improved in the future.

The used cameras, process and make the output of the data with a 320 × 200 pixels resolution. Since this project was developed to provide an industrial solution, this resolution becomes insufficient. The problem of the low resolution also arises because the field of view may need to cover a working area that can be too big for the available resolution.

The Raspberry Pi 2 computer provides enough processing power to implement the tracking system in real time, but it is necessary to test the system in a real implementation. If the tracking system cannot properly track the movement of the pointing device, it will be required the use of a computer with greater processing power, or the improvement of the developed software to be computationally lighter.

The CLARISSA demonstrator, beyond the IR teaching system for part positioning and correction, also has other components such as the robotic system, the augmented reality to assist the operator, and the interconnection between these components should provide a simple human-robot interface, that allows fast setup and operation of the whole robot system. It is therefore necessary to incorporate and test the tracking system in the CLARISSA demonstrator as a whole.

As a personal note, in the development of this project it was possible to deepen knowledge in embedded systems and programming and acquire new knowledge in the field of computer vision and robotic systems.

References

[1] Lewis, John. *Introduction to Machine Vision*. Cognex Corporation, 2014.

[2] SMErobotics. *The European Robotics Initiative for Strengthening the Competitiveness of SMEs in Manufacturing by integrating aspects of cognitive systems*. Published in http://www.smerobotics.org/. Consulted in September 2015.

[3] SARKKIS. *SARKKIS offcially joins SMErobotics*. Published in http://www.sarkkis.com/mechatronics/?p=2354. Consulted in September 2015.

[4] NORFER. Published in http://www.norfer.com/. Consulted in September 2015.

[5] Adams, Julie A. *Critical Considerations for Human-Robot Interface Development*. From AAAI Technical Report FS-02-03. 2002.

[6] Intel. *Enabling Great User Experiences with User-Centered Design*. Published in https://software.intel.com/en-us/articles/enabling-great-user-experiences-with-user-centered-design. Consulted in November 2014.

[7] Endsley, M.R. *Design and Evaluation for Situation Awareness Enhancement*. In Proceedings of the Human Factors Society 32nd Annual Meeting, 97-101. Santa Monica, CA: Human Factors Society. 1988.

[8] Nokata, M., Ikuta, K., and Ishii, H. *Safety Optimizing Method of Human-care Robot Design and Control.* In the proceedings of the 2002 IEEE International Conference on Robotics and Automation, 1997 – 2002. Washington, DC.

[9] Calinon, Sylvain. *Research on robot learning by imitation and exploration*. Published in http://www.calinon.ch/research.php. Consulted in August 2015.

[10] Aude Billard and Sylvain Calinon, Ruediger Dillmann, Stefan Schaal. *Handbook of Robotics Chapter 59: Robot Programming by Demonstration*. September 2007.

[11] G. Welch and E. Foxlin. *Motion tracking: No silver bullet, but a respectable arsenal*. IEEE Computer Graphics and Applications, 2002.

[12] Bishop, B.D.A.G., Welch, G. and Allen, BD. *Tracking: Beyond 15 minutes of thought: Siggraph 2001 course 11*. In Computer Graphics and Interactive Techniques. ACM Press, 2001.

[13] Gaschler, Andre. *Real-Time Marker-Based Motion Tracking: Application to Kinematic Model Estimation of a Humanoid Robot*. February 2011.

[14] Oikonomidis, Iason, Kyriazis, Nikolaos, Argyros, Antonis. *Efficient Model-based 3D Tracking of Hand Articulations using Kinect*. 2011.

[15] Hyung-Bok, Kim, Kwee-Bo, Sim. *A Particular Object Tracking in an Environment of Multiple Moving Objects*. International Conference on Control, Automation and Systems 2010.

[16] ISE Lab. *Recent Developments in Human Motion Analysis*. Published in http://iselab.cvc.uab.es/tutorials_ise/PPTs/survey/. Consulted in September 2015.

[17] Bouwmans, Thierry. *Background Subtraction*. Published in https://sites.google.com/site/backgroundsubtraction/. Consulted in September 2015.

[18] Benezeth, Yannick, Jodoin, Pierre-Marc, Emile, Bruno, Laurent, Hélène, Rosenberger, Christophe. *Comparative study of background subtraction algorithms*. Journal of Electronic Imaging, Society of Photo-optical Instrumentation Engineers (SPIE). 2010.

[19] CMUcam: Open Source Programmable Embedded Color Vision Sensors. *Color-tracking Explanation*. Published in http://www.cmucam.org/projects/cmucam4/wiki/Color-tracking_Explanation. Consulted in September 2015.

[20] Track-It-Yourself (TIY). *3D Marker Tracking Software*. Published in http://code.google.com/p/tiy/. Consulted in July of 2015.

[21] Track-It-Yourself (TIY). *3D Marker Tracking Software*. Published in https://github.com/gaschler/tiy/. Consulted in July of 2015.

[22] Aguilar-Torres, Michel A., Argüelles-Cruz, Amadeo J., Yánez-Márquez, Cornelio. *A real time artificial vision implementation for quality inspection of industrial products*. Centro de Investigación en Computación, Instituto Politécnico Nacional Col. Nueva Industrial Vallejo, C.P. 07738, México D.F. - Electronics, Robotics and Automotive Mechanics Conference 2008.

[23] Wikipedia. *Computer Vision*. Published in http://en.wikipedia.org/wiki/Computer_vision#mediaviewer/File:CVoverview2.svg. Consulted in January 2015.

[24] HIS Engineering 360. *Vision Sensors Information*. Published in http://www.globalspec.com/learnmore/video_imaging_equipment/machine_vision_inspection_equipment/vision_sensors. Consulted in February 2015.

[25] Www.ukiva.org. *Machine Vision Handbook*, third version, summer 2007.

[26] Lewis, John. *Introduction to Machine Vision*. Cognex Corporation, 2014.

[27] HyperPhysics. *Image Formation by Lenses and the Eye*. Published in http://hyperphysics.phy-astr.gsu.edu/hbase/class/phscilab/imagei.html. Consulted in August 2015.

[28] SPIE - Journal of Electronic Imaging. *Magnification-continuous static calibration model of a scanning-electron microscope*. Published in http://electronicimaging.spiedigitallibrary.org/article.aspx?articleid=1367829. Consulted in August 2015.

[29] Roth, Gerhard. *Camera Calibration*. Published in http://people.scs.carleton.ca/~c_shu/Courses/comp4900d/notes/camera_calibration.pdf. Consulted in August 2015.

[30] Corke, Peter. *Robotics, Vision and control – Fundamental Algorithms in Matlab,* 2011.

[31] Z. Zhang, *A flexible new technique for camera calibration*. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2000.

[32] Bradski, Gary, Kaehler, Adrian. *Learning OpenCV*. From O'Reilly. 2008.

[33] OpenCV. *Introduction - OpenCV 2.4.11.0 documentation*. Published in http://docs.opencv.org/modules/core/doc/intro.html. Consulted in July 2015.

[34] OpenCV. *Documentation - OpenCV*. Published in http://opencv.org/documentation.html. Consulted in July 2015.

[35] Schäling, Boris. *The Boost C++ Libraries*. Published in http://theboostcpplibraries.com/. Consulted in July of 2015.

[36] CMUcam: Open Source Programmable Embedded Color Vision Sensors. *CMUcam5 Pixy overview*. Published in http://www.cmucam.org/projects/cmucam5. Consulted in August 2015.

[37] BeagleBoard.org. *BeagleBone credit-card-sized Linux computer*. Published in http://beagleboard.org/bone. Consulted in September 2015.

[38] CubieBoard. *CubieBoard - A series of open source hardware*. Published in http://cubieboard.org/. Consulted in September 2015.

[39] Elinux.org. *Raspberry Pi Wiki/Hub*. Published in http://elinux.org/RPi_Hub. Consulted in August 2015.

[40] Elinux.org. *Raspberry Pi Performance*. Published in http://elinux.org/RPi_Performance. Consulted in August 2015.

[41] Elinux.org. *Rpi Datasheet 201 Raspberry Pi Computer*. Published in http://elinux.org/Rpi_Datasheet_201_Raspberry_Pi_Computer. Consulted in August 2015.

[42] Wikipedia. *Drawing of Raspberry Pi model B rev2.svg*. Published in https://en.wikipedia.org/wiki/File:Drawing_of_Raspberry_Pi_model_B_rev2.svg. Consulted in August 2015.

[43] The official Raspberry Pi magazine. *The MagPi*. Issue 31. March 2015.

[44] Raspberry Shop. *Raspberry Pi 2*. Published in http://www.raspberryshop.es/. Consulted in October 2015.

[45] CMUcam: Open Source Programmable Embedded Color Vision Sensors. *Powering Pixy*. Published in http://www.cmucam.org/projects/cmucam5/wiki/Powering_Pixy. Consulted in April 2015.

[46] CMUcam: Open Source Programmable Embedded Color Vision Sensors. *Porting Guide*. Published in http://www.cmucam.org/projects/cmucam5/wiki/Porting_Guide. Consulted in April 2015.

[47] Raspberrypi.org. *Raspberry Pi FAQs - Frequently Asked Questions*. Published in https://www.raspberrypi.org/help/faqs/. Consulted in July 2015.

[48] Elinux.org. *Raspberry Pi Hardware*. Published in http://elinux.org/RPi_Hardware. Consulted in July 2015

[49] Elinux.org. *Raspberry Pi Low-level peripherals*. Published in http://elinux.org/RPi_Low-level_peripherals. Consulted in July 2015.

[50] NXP Semiconductors. *Level shifting techniques in I2C-bus design*. Published in http://www.adafruit.com/datasheets/AN10441.pdf. Consulted in July 2015.

[51] Adafruit. *4-channel I2C-safe Bi-directional Logic Level Converter*. Published in https://www.adafruit.com/products/757. Consulted in July 2015.

[52] Inmotion. *4-channel I2C-safe Bi-directional Logic Level Converter BSS138.* Published in http://www.inmotion.pt/pt/adafruit/842-4-channel-i2c-safe-bi-directional-logic-level-converter-bss138.html. Consulted in July 2015.

[53] Mitchell, Mark, Oldham, Jeffrey and Samuel, Alex. *Advanced Linux Programming*. From New Riders Publishing, 2001.

[54] CMUcam: Open Source Programmable Embedded Color Vision Sensors. *Pixy Lens Specs: especially the focal length*. Published in http://www.cmucam.org/boards/8/topics/4279. Consulted in September 2015.

[55] CMUcam: Open Source Programmable Embedded Color Vision Sensors. *CMUcam5 Pixy General Overview*. Published in http://www.cmucam.org/projects/cmucam5. Consulted in September 2015.

[56] Point Grey. *Selecting a Lens for your Camera*. Published in https://www.ptgrey.com/tan/10694. Consulted in September 2015.

[57] M12 Lenses. *PT-0620 6.0mm, F2.0 Board Lens*. Published in http://www.m12lenses.com/6-0mm-F2-0-Board-Lens-p/pt-0620.htm. Consulted in September 2015.

[58] IR-LOCK. *IR-LOCK Filter Kit for Pixy*. Published in http://irlock.com/collections/frontpage/products/ir-lock-filter-for-pixy. Consulted in September 2015.

[59] IR-LOCK. *Installing the IR-LOCK Filter*. Published in http://irlock.com/pages/installing-ir-lock-filter-onto-pixy. Consulted in September 2015.

[60] IR-LOCK. *Archived Files*. Published in http://irlock.com/pages/archived-files. Consulted in September 2015.

[61] Track-It-Yourself (TIY). *Object Set Up*. Published in https://code.google.com/p/tiy/wiki/ObjectSetUp. Consulted in October 2015.

[62] Bouguet, Jean-Yves. *Camera Calibration Toolbox for Matlab.* Published in http://www.vision.caltech.edu/bouguetj/calib_doc/. Consulted in July 2015.

[63] Track-It-Yourself (TIY). *Camera Set Up*. Published in http://code.google.com/p/tiy/wiki/CameraSetUp. Consulted in July of 2015.

[64] Bouguet, Jean-Yves. *Camera Calibration Toolbox for Matlab: First calibration example - Corner extraction, calibration, additional tools*. Published in http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example.html. Consulted in July 2015.

[65] Wiring Pi. *GPIO Library for the Raspberry Pi*. Published in http://wiringpi.com/. Consulted in August 2015.

[66] Wiring Pi. *GPIO Library for the Raspberry Pi: Download and Install*. Published in http://wiringpi.com/download-and-install/. Consulted in August 2015.

[67] Wiring Pi. *GPIO Library for the Raspberry Pi: I2C Library*. Published in http://wiringpi.com/reference/i2c-library/. Consulted in August 2015.

[68] McDuffie, James. *Pixy Camera Raspberry Pi Interface*. Published in https://github.com/omwah/pixy_rpi. Consulted in August 2015.

Annex A TIY Installation and Configuration

The instructions in this Annex refer to the TIY installation and configuration on a Linux system. For installation on a Windows system is recommended to consult the online documentation of TIY [20] [21].

Initially the dependencies must be installed:

- OpenCV(≥ 2.2):

```
sudo apt-get install libopencv-dev
```

- Boost(≥ 1.46):

```
sudo apt-get install libboost-dev libboost-
filesystem-dev libboost-system-dev libboost-date-
time-dev libboost-thread-dev
```

- `CMake` (necessary because the TIY is built in this project, optional otherwise):

```
sudo apt-get install cmake build-essential cmake-
curses-gui
```

There are two ways to install the TIY on system: using a pre-build package with one single installation file and no building necessary, and the way used in this project, building from source (based on `CMake`) allowing further customization. The steps of the used method are the following:

1. Download and unzip the newest `tiy-X.X.zip` file from the Downloads section [20] [21].

2. If building a production version of TIY go into Release folder. Otherwise, if building a test version with debugging capabilities go into Debug folder. Use `CMake` to build the make file:

```
ccmake ../src
```

3. Press the C key to change the building options:

```
BUILD_client ON: the client will also be build.
BUILD_server ON: the server will also be build.
CMAKE_BUILD_TYPE Release or Debug: if a
production or a test version respectively.
USE_ARAVIS OFF: we don't use Aravis cameras.
```

4. Press C key a few times and then the G key to generate the `makefile`.

5. Make the project with the following command:

```
make
```

6. And finally install the TIY:

```
make install
```

After these steps the TIY is installed in the Linux system and ready to be used.

Annex B Electric circuit schematic

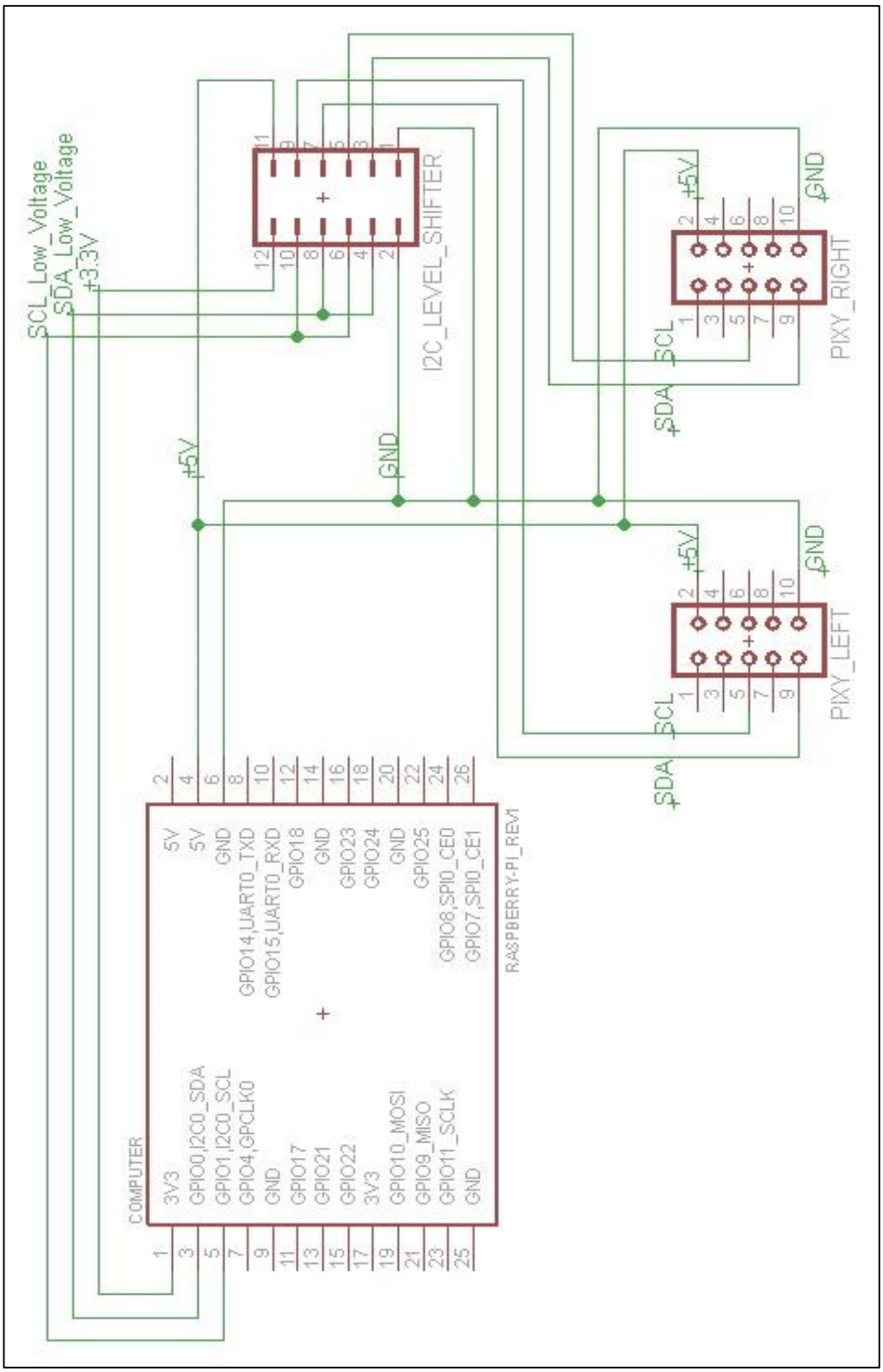


Figure 35 Schematic of the electric circuit project.

Annex C WiringPi and Pixy library

The `WiringPi` library simplifies the use of the Raspberry Pi GPIO. It is developed in the C programming language, with GNU Lesser Public License version 3 (LGPLv3). It is based on the Arduino `Wiring` library, which is responsible for processing the inputs and outputs of the device. The `WiringPi` library is used in this project to facilitate the use of I2C in the Raspberry Pi.

This library has an executable that can be run from the command line, allowing the programming and setup of GPIO pins, as well as reading and writing of inputs and outputs. The command in question is the `gpio` command.

Since the Raspberry Pi allows adding expansion ports, `WiringPi` can also be extended by modules to support external GPIO interfaces. One of these modules is `Devlib`, which includes a set of routines that support some of the more common peripherals, such as some types of displays [65].

C.1. Installation Instructions

The `WiringPi` is kept in the `Git` repository. The following instructions apply to the library installation in the Raspbian operating system [66].

If `Git` is not installed on the system, it is required to install it to enable the download of `WiringPi`:

```
sudo apt-get install git-core
sudo apt-get update
sudo apt-get upgrade
```

Then, download `WiringPi` from `Git`:

```
git clone git: //git.drogon.net/wiringPi
```

After obtaining the library set, it is time to make the build and install on the operating system using the available script:

```
cd wiringPi
./build
```

Finally, one can use the `gpio` command to test whether the installation ran properly:

```
gpio -v
gpio readall
```

C.2. I2C Library

To use `WiringPi`, the header file must be included in the source code of the program to develop:

```
#include <wiringPi.h>
```

Also, using ordinary compilers on the Linux command line, the following parameter must be included in the command line to instruct the compiler to load the library:

```
-lwiringPi
```

as in the following example:

```
g++ raspixy.cpp -ggdb -lwiringPi -I ./ -o raspixy
```

Before using the I2C interface, it is required to load the I2C drivers in the kernel [67]:

```
gpio load i2c
```

In the previous command, the default speed of 100 kbps is selected, and if another speed is required this can be selected using the previous command, followed by the desired speed. For example, for a transmission speed of 500 kbps the following command should be used:

```
gpio load i2c 500
```

To use the I2C library in the program, it must include the following statement in the code to develop:

```
#include <wiringPiI2C.h>
```

C.3. Pixy and TPixy Libraries

The `Pixy.h` library allows performing some basic operations on Pixy cameras. It depends on `TPixy.h` and `wiringPiI2C.h` libraries. The `Pixy.h` and `TPixy.h` libraries are created using as a basis the information provided by Pixy developers [9], and a similar project that uses the Pixy camera with SPI communication [68].

The available operations allow initialize a Pixy camera (`void init (uint8_t addr)`) where an I2C address must be assigned. There is also an operation to return a data block of 16 bit length (`uint16_t getWord ()`) from the camera, and an operation to return a data block with 8 bit size (`uint8_t getByte ()`). Finally, there is an operation for sending a data block with 8 bit size (`int8_t send (uint8_t *date, uint8_t len)`), which can be used to send commands to the camera.

The `TPixy.h` library allows to obtain a block of data, wherein each of these blocks contains information about each detected object. The relevant information contained in each data block is related to the two-dimensional position (`x, y`) of each of the detected objects (Table 2).

Annex D Pixy Camera Calibration using Matlab

To obtain a stereo vision system using two cameras, are needed both the intrinsic and extrinsic camera parameters, meaning that it is needed a camera calibration. For this task, it should be used the TIY and a Matlab camera calibration toolbox (http://www.vision.caltech.edu/bouguetj/calib_doc/) [62]. This is the recommend calibration toolbox by Track It Yourself authors [63].

To perform the camera calibration using the two last mentioned tools, it is first needed to download and unzip the file `tiy_calibration.zip` from the TIY downloads section [20] or [21]. Furthermore, the following steps need to be performed:

- download and unzip the Matlab toolbox from the Vision Caltech site downloads area [62];
- put all MAT-files from the extracted toolbox into the extracted folder `tiy_calibration/camera_calibration/toolbox_calib`;
- it is also needed a pattern to perform the camera calibration. There is a pattern sample in the file `tiy_calibration/camera/calibration/pattern.pdf`, that can be printed, as large as possible, and fixed to a completely flat plate;
- next are needed, at least, 15 snapshots of the checkerboard pattern, filling the screen in different angles for the left and right cameras, and stereo snapshots that can be taken by the TIY `server` example, when activating the `<do_log_frame>` option in the `config_run_parameters.xml` configuration file, and pressing the `SPACE` key for each snapshot. The files are stored in the `tiy_log` folder, in the TIY `bin` or `home` directory. Therefore, it is made one stereo snapshot of the checkerboard pattern lying preferably flat on a defined reproducible position in the sight of both cameras.

D.1. Main Calibration Procedure

With the snapshots from both cameras and the stereo snapshot, it is possible to start the calibration procedure [64]. From within Matlab, select the example folder `camera_calibration/toolbox_calib`, containing the images from one of the

cameras (initially with the right camera snapshots, which must be copied to this folder). Perform the steps:

- run `calib_gui.m` script;
- select the first choice, `Standard`;
- click on the `Images names` button, in the `Camera Calibration Toolbox` window. Enter the base name of the calibration images (`image`) and the image format (`jpeg`). All the images are then loaded in memory (through the command `Read images` that is automatically executed) (Figure 36);

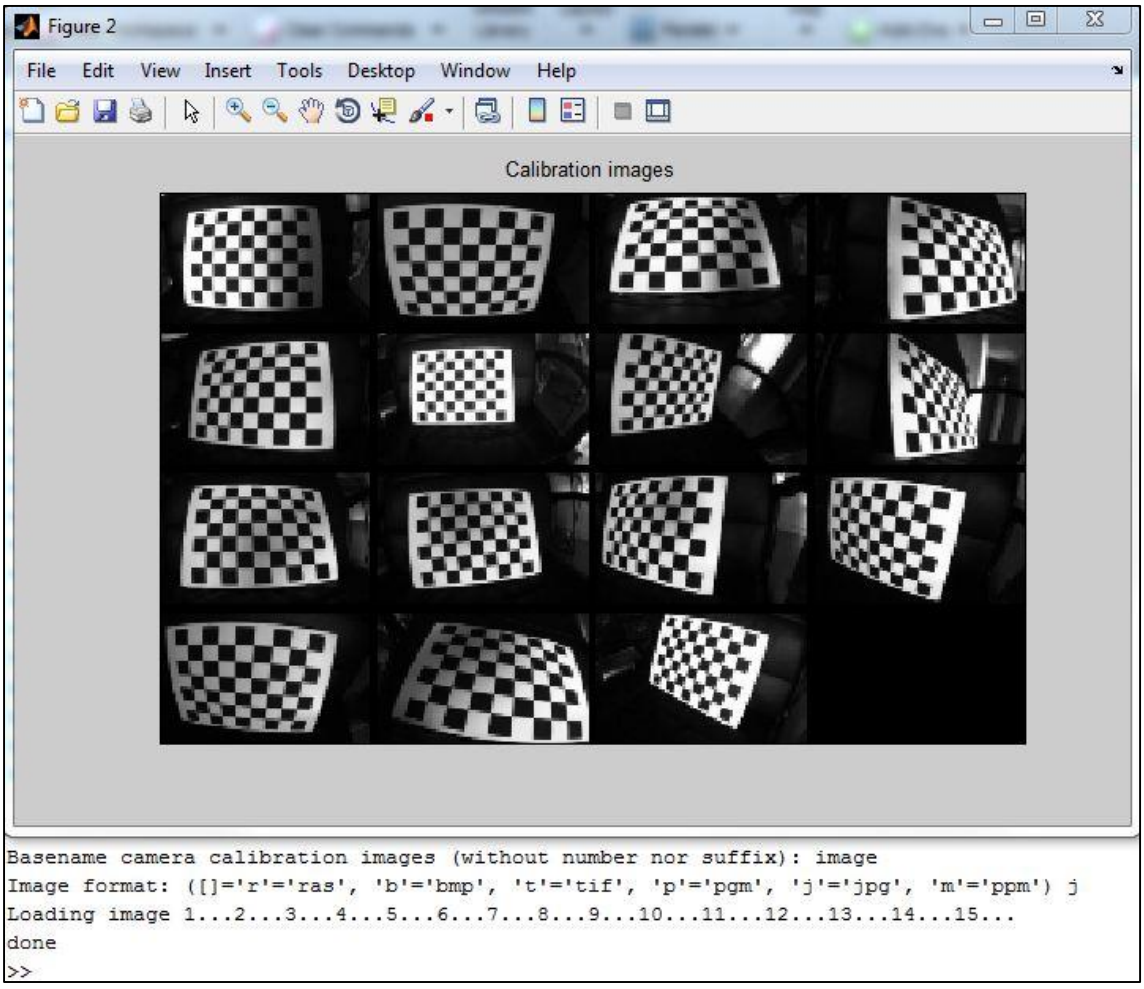


Figure 36 Snapshot from the calibration toolbox image loading.

- click on the `Extract grid corners` button, in the `Camera calibration tool` window;

- press the `ENTER` key (with an empty argument) to select all the images. Then, select the default window size of the corner finder: `wintx = winty = 5` by pressing the `ENTER` key with argument `5` to the `wintx` and `winty` questions. This leads to an effective window of size 11×11 pixels;

- in next question, it must be chosen to manually enter the number of squares due to the Pixy camera low resolution (320×200 pixels) and the difficulty of the toolbox in finding the squares automatically, so it must be entered `1` (Figure 37);

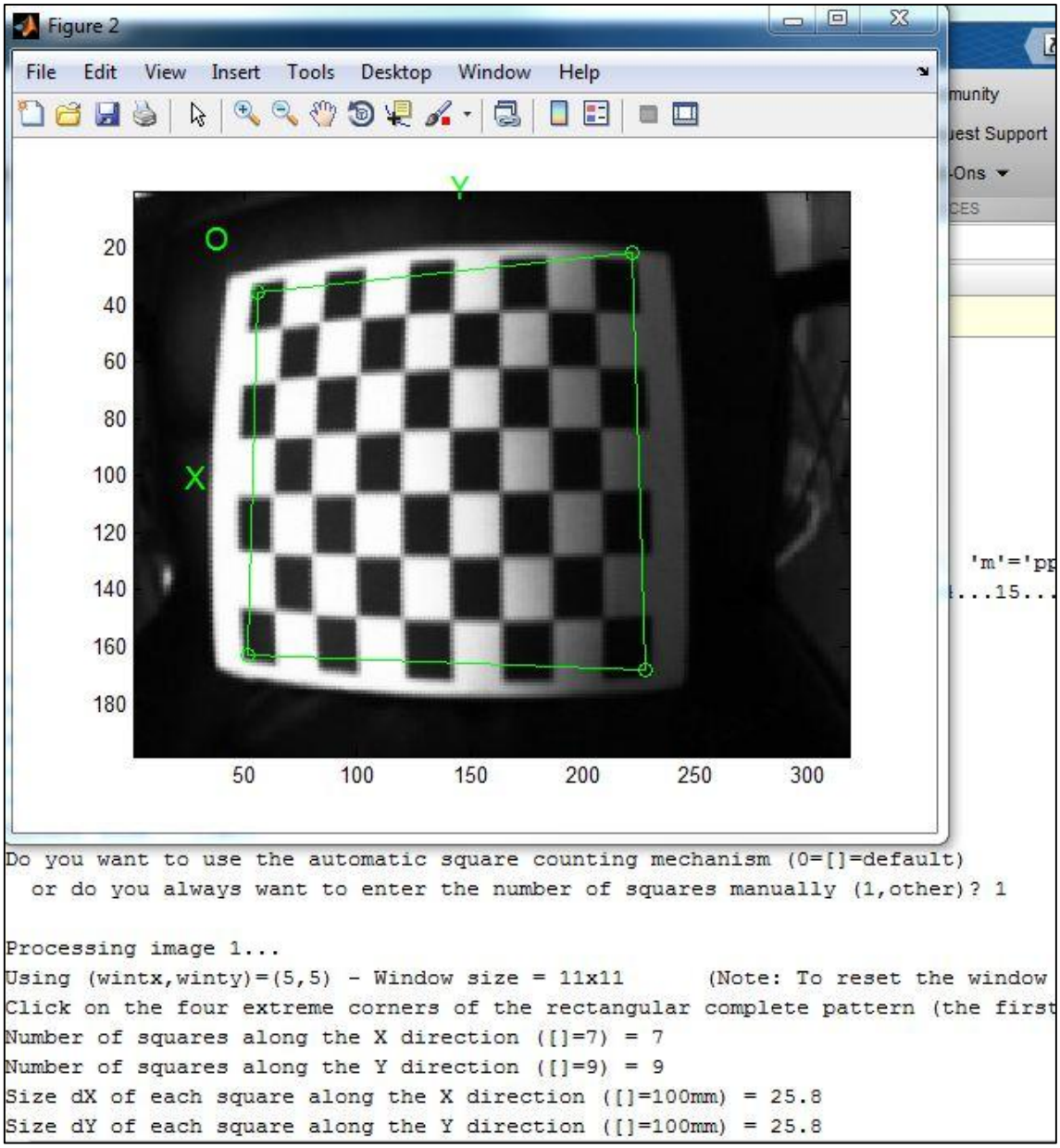


Figure 37 Number and size of squares in the checkerboard pattern.

- give the number of squares in the checkerboard pattern: 7 squares in the *X* direction and 9 squares in the *Y* direction. The size of each side of the squares is 25.8 mm (Figure 37);

- now the corners of the external squares must be selected and they must obey a clicking rule. In this project the first corner selected is the left up corner, followed by the right up corner, next the right down corner and, finally, the left down corner (Figure 37);

- after selecting the corners, one can try to reduce the images distortion, adjusting the radial distortion factor `kc`. The Pixy cameras have high distortion and low resolution so it is necessary to adjust `kc` for all images, trying to obtain the best result. It is advised to try an initial guess for `kc` for each image, and next try to refine it to the best corner extraction. In Figure 38 can be seen the extracted corners for Image 1 of right camera after adjusting the distortion;

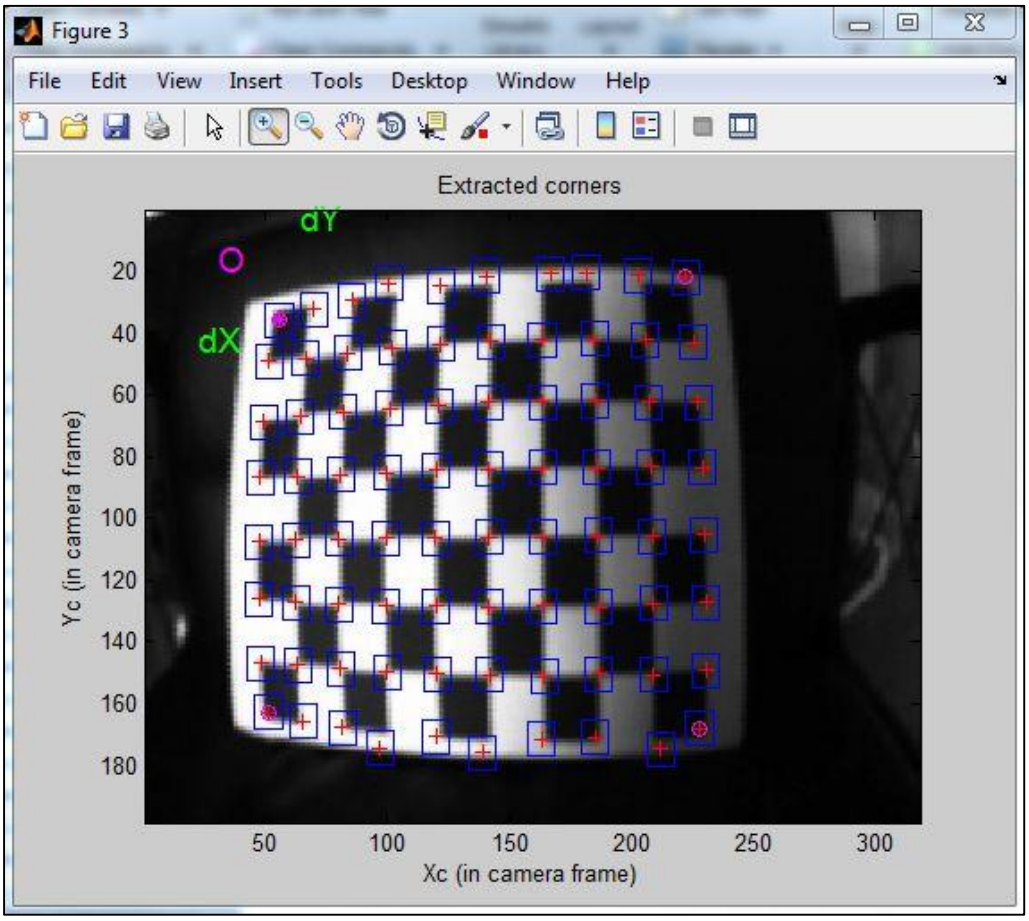


Figure 38 Extracted corners for Image 1 of right camera.

- following corner extraction, the Matlab data file `calib_data.mat` is automatically generated. This file contains all the information gathered throughout the corner extraction stage (image coordinates, corresponding 3D grid coordinates, grid sizes, etc.);

- next, click on the button `Calibration` of the `Camera calibration toolbox` to run the main camera calibration procedure. Calibration is done in two steps: first initialization, and then nonlinear optimization. After optimization, are outputted the calibration results:

```
Calibration results after optimization (with
uncertainties):

Focal Length:          fc = [ 245.08432
248.51762 ] ± [ 5.40520   5.64040 ]
Principal point:       cc = [ 167.19811
99.41153 ] ± [ 6.08342   5.45645 ]
Skew:             alpha_c = [ 0.00000 ] ± [
0.00000  ]   => angle of pixel axes = 90.00000 ±
0.00000 degrees
Distortion:            kc = [ -0.41574   0.14812
0.00666   -0.00492  0.00000 ] ± [ 0.03576
0.04649   0.00419   0.00375  0.00000 ]
Pixel error:          err = [ 1.40865   1.09947 ]
```

- to show the reprojections of the grids onto the original images click on `Reproject on images` in the `Camera calibration toolbox`. These projections are computed based on the current intrinsic and extrinsic parameters (Figure 39). It is also show the reprojection error (Figure 40);

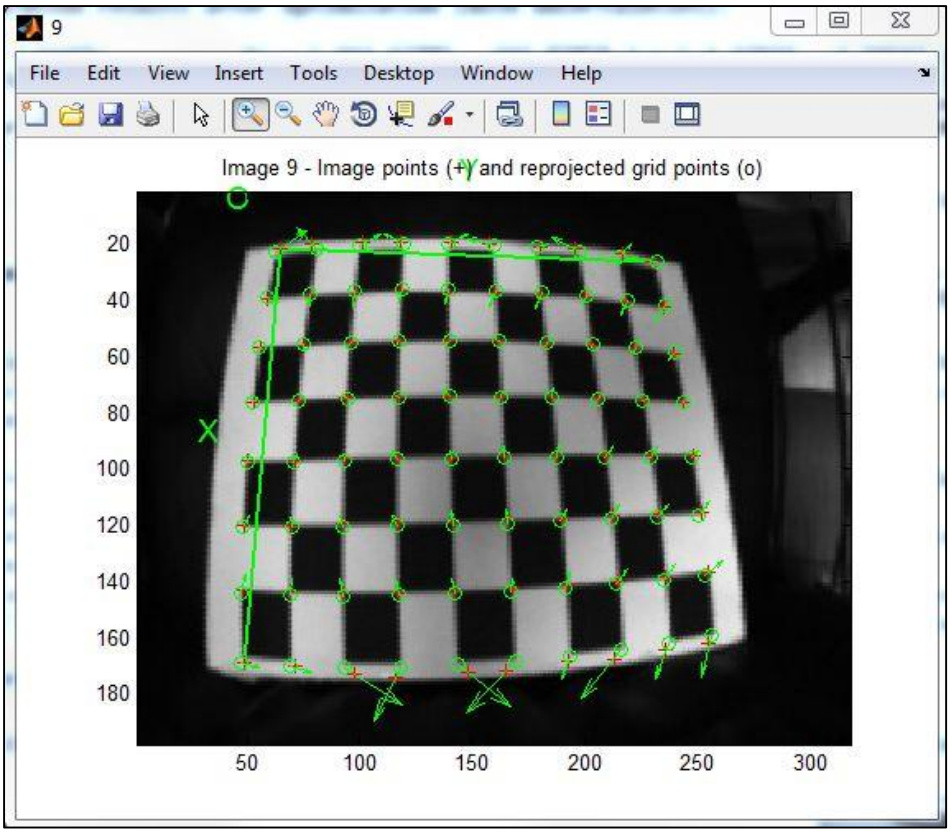


Figure 39 Reproject on images applied to Image 9.

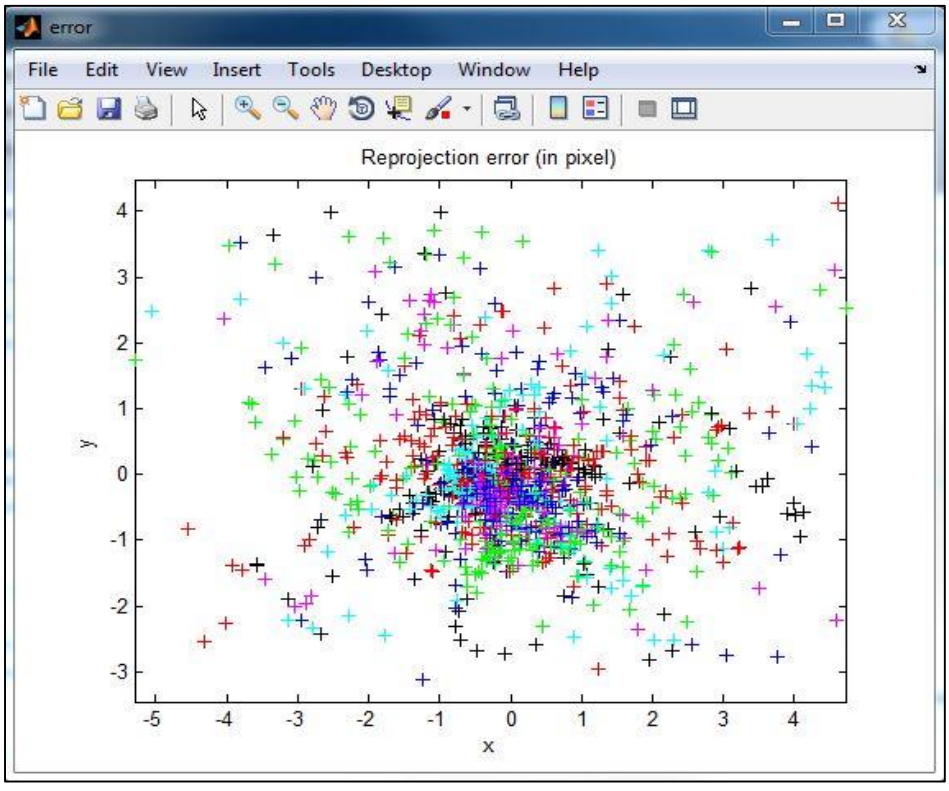


Figure 40 Reprojection error.

- to view the extrinsic parameters click on `Show Extrinsic`, in the `Camera calibration toolbox`. The extrinsic parameters (relative positions of the grids with respect to the camera) are then shown in a form of a 3D plot presented in Figure 41;

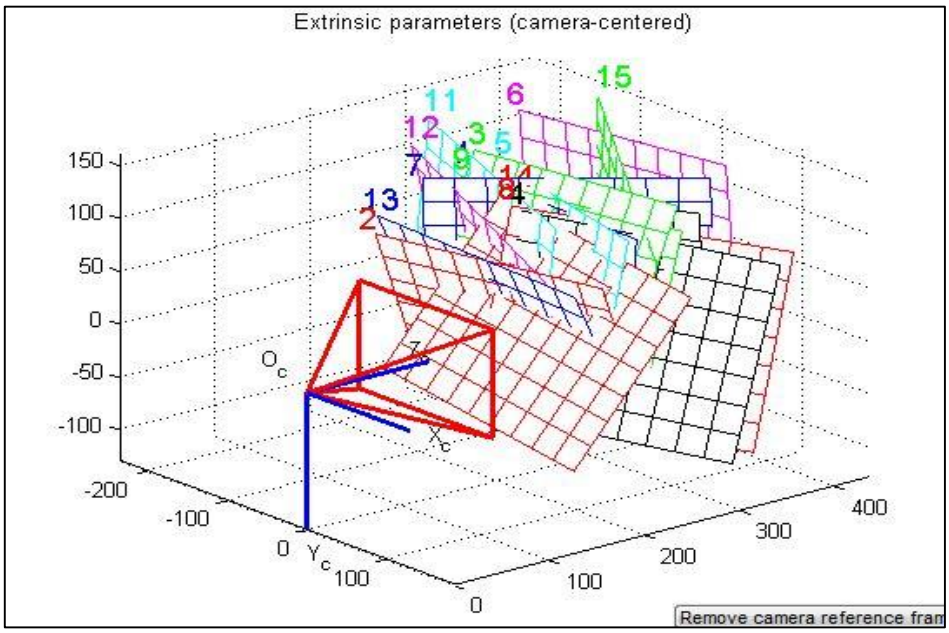


Figure 41 Extrinsic parameters with centered camera.

- looking back to Figure 40, it is possible to see the reprojection error with a sparse area due to high distorted images. The used toolbox gives a way to reduce the reprojection error, recomputing the image corners on all images automatically. To do so, press the `Recomp. corners` in the main `Camera Calibration Toolbox` and select a corner finder window size of `wintx = winty = 5`, letting the other options unaltered. After that, it is necessary to perform another calibration clicking in the `Calibration` button. After the calibration done, the results are the following:

```
Calibration results after optimization (with
uncertainties):

Focal Length:          fc = [ 249.49723
254.64851 ] ± [ 4.52995   4.79900 ]
Principal point:       cc = [ 169.10465
101.84428 ] ± [ 5.18265   4.61071 ]
Skew:             alpha_c = [ 0.00000 ] ± [
0.00000  ]   => angle of pixel axes = 90.00000 ±
0.00000 degrees
```

```
Distortion:             kc = [ -0.44932   0.18418
0.00341   -0.00528  0.00000 ] ± [ 0.03001
0.03940   0.00365   0.00329  0.00000 ]
Pixel error:          err = [ 1.22761   0.88622 ]
```

- after optimization, click on the `Save` button to save the calibration results (intrinsic and extrinsic) in the Matlab file `Calib_Results.mat`;

- once again, click on `Reproject on images` button, to reproject the grids onto the original calibration images. The four first images are illustrated in Figure 42;

- click on `Analyze error` button to view the new reprojection error (observe that the error is smaller than before), as depicted in Figure 43.

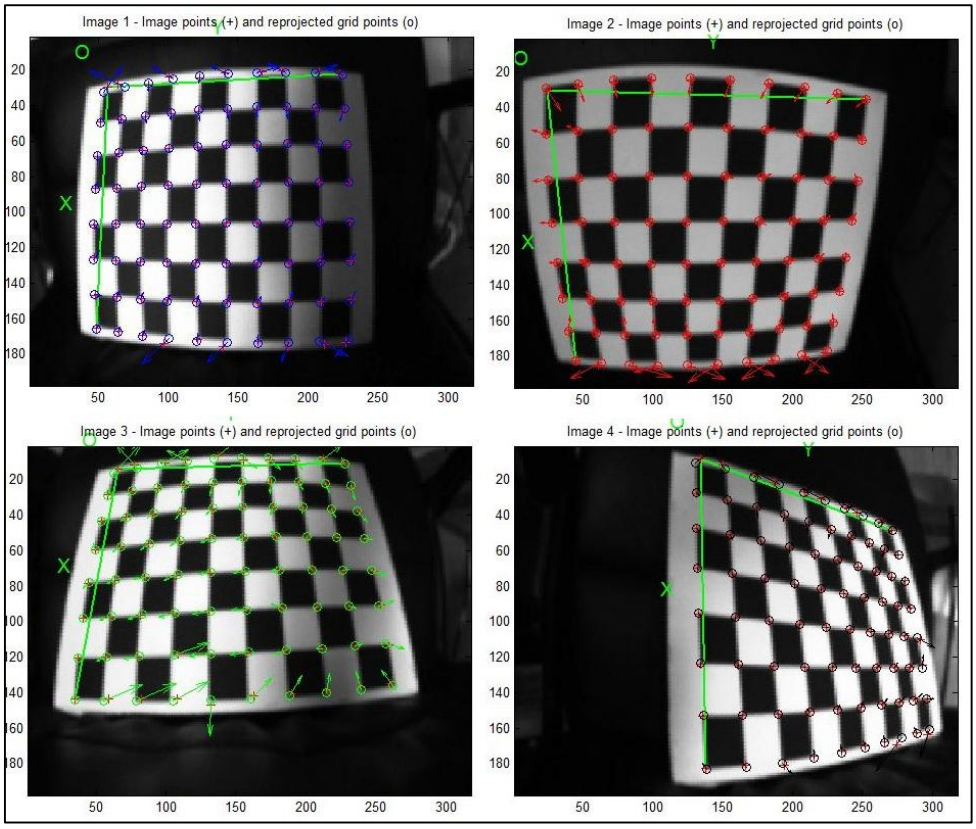


Figure 42 Reprojection of the grids onto the first four original calibration images.

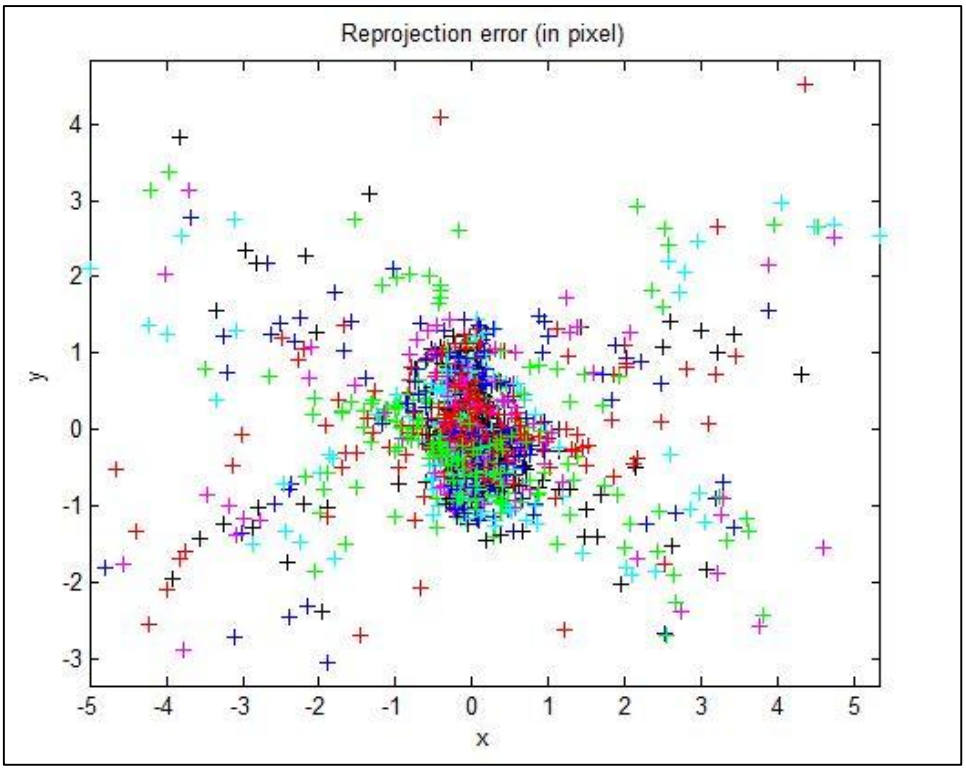


Figure 43 Reprojection error after optimization.

For the left camera calibration, it is required to repeat the same steps described in this procedure.

D.2. Computation of Extrinsic Parameters

For the computation of the extrinsic parameters it is needed an additional image of the same calibration grid, that has not been in main calibration procedure.

The goal of this procedure is to compute the extrinsic parameters attached to this image, given the intrinsic camera parameters previously computed [64].

Click on `Comp. Extrinsic` in the `Camera calibration toolbox`, and successively enter the image name (without extension), the image type, and extract the grid corners (following the same procedure as previously presented, but remembering that the first clicked point is the origin of the pattern reference frame). The extrinsic parameters (3D location of the grid in the camera reference frame) are then computed.

In the case of this project, the left camera results are the following:

```
Extrinsic parameters:
```

```
Translation vector: Tc_ext = [ -133.265269     -
108.933383    277.620898 ]
Rotation vector:   omc_ext = [ 1.765153
1.810689      0.220403 ]
Rotation matrix:    Rc_ext = [ 0.058350
0.855302      0.514834
                              0.953831
0.104433      -0.281601
                              -0.294620
0.507496      -0.809720 ]
Pixel error:           err = [ 4.22583    3.91175
]
```

and ones for the right camera are:

```
Extrinsic parameters:
Translation vector: Tc_ext = [ -123.864459     -
103.137622    334.797860 ]
Rotation vector:   omc_ext = [ 1.899626
2.166396      -0.776641 ]
Rotation matrix:    Rc_ext = [ -0.182182
0.959369      -0.215455
                              0.877745
0.059924      -0.475366
                              -0.443140       -
0.275718      -0.852999 ]
Pixel error:           err = [ 0.94125    0.94903
]
```

D.3. Last Calibration Steps

Finally, it is required to update the actual camera parameters with the calibration results. The steps that must be performed are the following:

- after each intrinsic calibration, replace the `Calib_Results.m` and `Calib_Results.mat` files, in the specific `tiy_calibration/camera_calibration/left` or `right` folder. At the end of the extrinsic calibrations, replace the data in the `Calib_Results_extrinsic.m` files;

- replace the `config_camera.xml` file, in the `tiy_calibration/camera_calibration` folder, with the file from the `TIY` folder. Run the `make_camera_parameters.m` script, to write the calibration data into the `config_camera.xml` file and copy the changed configuration file back to the `TIY` folder.