



**Universidad de Valladolid**



**ESCUELA DE INGENIERÍAS  
INDUSTRIALES**

**UNIVERSIDAD DE VALLADOLID**

**ESCUELA DE INGENIERIAS INDUSTRIALES**

**Grado en Ingeniería Electrónica Industrial y Automática**

# **El controlador KUKA youBot**

**Autor:**

**Justo Lobato, Alberto**

**Responsable de Intercambio en la UVa**

**Herráez Sánchez, Marta**

**Universidad de destino**

**Vilnius Gediminas Technical University**

**Valladolid, Junio 2019.**



## TFG REALIZADO EN PROGRAMA DE INTERCAMBIO

---

TÍTULO: The KUKA youBot controller  
ALUMNO: Alberto Justo Lobato  
FECHA: 05/06/2019  
CENTRO: Electronics Faculty. Vilnius Gediminas Technical University  
TUTOR: Dainius Udris

### Cinco palabras claves que describen el TFG:

---

Robótica, Morfología, PCB, ROS, Octave

### Abstract en español (máximo 150 palabras):

---

En nuestros días, muchas placas base y PCBs presentan fallos debido al fin de su vida útil o a errores en el proceso de producción. En muchos de estos casos, su clasificación se realiza mediante operaciones manuales, llevando a los empleados a una posible exposición frente a elementos tóxicos. Debido a este motivo, en este proyecto se propone detectar dichos fallos con el procesamiento de imágenes tomadas por una cámara Siemens MV440 mediante el programa GNU Octave, así como la clasificación de las mismas gracias a la programación de un robot-brazo KUKA youBot dentro del entorno Linux conocido Robot Operating System o ROS. Previa a la programación de dicho robot, se simulará su comportamiento dentro del programa Gazebo.



VILNIUS GEDIMINAS TECHNICAL UNIVERSITY  
FACULTY OF ELECTRONICS  
DEPARTMENT OF ELECTRICAL ENGINEERING

Alberto Justo Lobato

**THE KUKA YUBOT CONTROLLER  
KUKA YUBOT ROBOTO VALDIKLIS**

Bachelor Thesis

Automation study programme, State Code 612H62002  
Automatic Control of Technologies specialization  
Electrical Engineering Study Area

Vilnius, 2019



VILNIUS GEDIMINAS TECHNICAL UNIVERSITY  
FACULTY OF ELECTRONICS  
DEPARTMENT OF ELECTRICAL ENGINEERING

CONFIRMS

Head of Department

(signature)

Assoc. Prof. Dr. S. Tolvaišienė

2019 06 04, 2019

Alberto Justo Lobato

**THE KUKA YUBOT CONTROLLER**  
**KUKA YUBOT ROBOTO VALDIKLIS**

Bachelor Thesis

Automation study programme, State Code 612H62002

Automatic Control of Technologies specialization

Electrical Engineering Study Area

**Supervisor** Assoc. Prof. Dr. Dainius Udris

(scientific title and degree, name, surname)

(signature)

2019-06-04

(date)

Vilnius, 2019





Alberto Justo

The KUKA youBot controller. Bachelor Thesis for degree. Vilnius Gediminas Technical University. Vilnius, 2019.

Abstract:

Nowadays, many mother boards and PCBs have errors due to their life cycle end or mistakes in production process. In most of these cases, their classification is done manually, meaning an exposition to toxic elements for the employees. Thus, in this project we propose a solution based on computer vision and robotics. On one hand, error detection is made with a Siemens MV440 camera and images taken by it are processed on Octave. On the other hand, PCB classification is done by robot KUKA youBot, which is programmed on Linux supported software Robot Operating System or ROS. Previously, we simulate our robot workspace on Gazebo program.

Key words:

ROS, Octave, PCB, image processing, error detection.

# INDEX OF CONTENTS

1. INTRODUCTION	9
1.1. Motivation of the project.....	10
1.2. Main goals.....	10
2. ANALYTICAL PART	12
2.1. ROS: architecture and characteristics .....	12
2.1.1. Architecture .....	12
2.1.1.1. Filesystem level.....	12
2.1.1.2. Computation graph level .....	13
2.1.2. First steps on ROS .....	15
2.1.2.1. Creating a new workspace.....	15
2.2. KUKA youBot.....	16
2.2.1. Overview of kinematic structure.....	16
2.2.1.1. How to move KUKA youBot arm to its home position .....	16
2.2.2. Robot kinematics theory .....	18
2.2.3. Denavit-Hartenberg parameters .....	19
2.2.4. Inverse kinematics .....	21
2.2.5. Technical specifications.....	25
2.2.6. Connections .....	26
2.3. Camera .....	26
2.4. Computer vision on Octave.....	29
2.4.1. Programming on Octave. Artificial vision most used commands ...	29
2.4.2. Computer vision techniques for error detection in PCBs .....	32
2.4.3. PCBs. Production and use errors. ....	36
3. DESIGN PART	38
3.1. Real workspace: settings and preparation. ....	38
3.1.1. Camera connections.....	38
3.1.2. Platform instalation for camera.....	40
3.2. Robot simulation. ....	42
3.2.1. Forward and inverse kinematics equations obtained on Robotics Toolbox.....	42

3.2.2. Inverse Kinematics programming for KUKA youBot.....	45
3.2.3. Robot programming on C++.....	46
3.2.4. Simulation on Gazebo.....	58
3.3. Computer vision programming. ....	61
3.3.1. License Automation Manager from Siemens. ....	61
3.3.2. Image acquisition mode for Siemens MV440 camera.....	63
3.3.3. PCBs error detection programming. ....	64
3.3.4. Communication between camera and ROS .....	72
4. RESULTS AND CONCLUSIONS	76
4.1. Computer vision part results .....	76
4.2. KUKA youBot part results .....	80
4.3. Final conclusions.....	82
5. ACKNOWLEDGEMENTS	83
6. REFERENCES	84
7. ANNEXES	86
7.1. ROS C++ CODE.....	86
7.2. OCTAVE COMPUTER VISION CODE .....	99
7.3. MATLAB FORWARD AND INVERSE KINEMATICS CODE.....	106
8. APPENDIXES	111
8.1. KUKA YOUBOT ARM SPECIFICATIONS.....	111
8.2. MAXON JOINT MOTORS .....	115
8.3. SIEMENS MV440 CAMERA SPECIFICATIONS .....	116



# INDEX OF ILUSTRATIONS

Figure 1. Scheme of final thesis .....	11
Figure 2. Filesystem level folders[2] .....	13
Figure 3. Computation graph level elements[2] .....	14
Figure 4. Communication between nodes[3].....	14
Figure 5. Creating a new workspace on ROS.....	15
Figure 6. Packages included in created workspace[3].....	16
Figure 7. KUKA youBot .....	16
Figure 8. Robot's arm movement to its home position[6] .....	17
Figure 9. KUKA youBot home position.....	18
Figure 10. Kinematics model for a 2 DOF robot[8] .....	18
Figure 11. D-H link twist and length[8] .....	20
Figure 12. Example of D-H parameters obtaining[8].....	20
Figure 13. Example of inverse kinematics[18].....	21
Figure 14. Inverse kinematics of 2R arm[18].....	22
Figure 15. How to calculate $q_1$ and $q_2$ for 2R arm[18].....	22
Figure 16. KUKA youBot model for obtaining inverse kinematics[20] .....	23
Figure 17. KUKA youBot manipulator scheme in XYZ axes[20] .....	24
Figure 18. Joints motors[8.2].....	25
Figure 19. Robot's angles and height[8.1] .....	25
Figure 20. KUKA youBot connections .....	26
Figure 21. Siemens MV440 camera[4].....	27
Figure 22. MV440 camera connections and components[4] .....	28
Figure 23. MATLAB supported vision cameras .....	29
Figure 24. Robot's arm image with Imshow[9] .....	31
Figure 25. PCB image with Imshow[9].....	31
Figure 26. PCB image converted to greyscale with rgb2gray[9] .....	31
Figure 27. Graphical example of dilation matrix[11].....	33
Figure 28. Erosion graphical example[10] .....	33
Figure 29. Erosion practical example, before and after. [10].....	33
Figure 30. Example of filtering[10].....	35
Figure 31. Example of template matching[10] .....	36

Figure 32. PCB ML/XEROX model used for detection.....	36
Figure 33. Bad PCB ML/XEROX model.....	37
Figure 34. Connection between Siemens camera and PC[4] .....	38
Figure 35. Camera connections to voltage[3.1.1].....	39
Figure 36. 24 V DC power supply[3.1.1].....	39
Figure 37. Holes dimensions for screwing camera to platform[8.3].....	40
Figure 38. Template and camera screwed to platform[3.1.2].....	41
Figure 39. Workspace with robot and camera installed [3.1.2].....	41
Figure 40. Test 1 for KUKA youBot on MATLAB .....	42
Figure 41. Test 2 for KUKA youBot on MATLAB .....	43
Figure 42. Test 3 for KUKA youBot on MATLAB .....	43
Figure 43. Instructions for using KUKA youBot with keyboard [3] .....	46
Figure 44. Reference system for KUKA youBot programming [5].....	47
Figure 45. KUKA youBot on Gazebo[3.2.4] .....	58
Figure 46. Measurement of PCBs[3.2.4].....	59
Figure 47. Workspace constructed in Gazebo[3.2.4] .....	59
Figure 48. PC and camera IP connection[3.3.1].....	61
Figure 49. Opening web browser in PRONETA for user interface[3.3.1].....	62
Figure 50. Code reading configuration on web browser[3.3.1].....	62
Figure 51. SIMATIC MV440 HR user program[3.3.2] .....	63
Figure 52. PCB with white background for detection[3.3.3] .....	64
Figure 53. Original and filtered image[3.3.3.1].....	65
Figure 54. Standard image for comparing elements in mymorphology function[3.3.2] .....	67
Figure 55. Testing image in mymorphology function[3.3.2] .....	68
Figure 56. Both images transformed and compared in same plot for mymorphology function[3.3.2] .....	68
Figure 57. Result obtained after applying mymorphology function[3.3.2].....	69
Figure 58. Template selected in standard image[3.3.3.3].....	71
Figure 59. Plotting elements in common between two images with template matching[3.3.3.3].....	72
Figure 60. Rosoct commands[3.3.4].....	73
Figure 61. Contour image with filtering function[4.1].....	76
Figure 62. Standard image number 2 for mymorphology function[4.1] .....	77
Figure 63. Testing image number 2 for mymorphology function[4.1] .....	77
Figure 64. Images after applying dilation[4.1].....	78
Figure 65. Mymorphology function applied to second couple of images. [4.1] .....	78

Figure 66. Jig for second standard image in matching function[4.1] .....	79
Figure 67. Template matching applied on second testing image[4.1].....	79
Figure 68. Robot's movement from home position .....	80
Figure 69. KUKA youBot trying to reach PCB .....	81

## INDEX OF TABLES

Table 1. D-H symbolic parameters for KUKA youBot.....	21
Table 2. Electrical and reading specifications .....	27
Table 3. Test values for MATLAB Robotics Toolbox on KUKA youBot .....	42
Table 4. D-H parameters for KUKA youBot obtained with MATLAB Robotics Toolbox .....	44
Table 5. Values for $Q_i$ angles of KUKA youBot .....	44

## INDEX OF EQUATIONS

Equation 1. Denavit-Hartenberg parameters matrix.....	20
Equation 2. Inverse kinematics for KUKA youBot orientation .....	23
Equation 3. $Q_1$ angle .....	24
Equation 4. $Q_3$ angle .....	24
Equation 5. $Q_2$ angle .....	24
Equation 6. $Q_4$ angle .....	24
Equation 7. Example of dilation matrix .....	32
Equation 8. Correlation coefficient .....	35

## 1. INTRODUCTION

### **E-waste material problem: the end of life cycle of our electronics devices [1]**

Nowadays, there are millions of mobile phones, tablets, PCs and other types of electronic components across the globe. When their life span is over, they are sent to developing or third world countries, like China or Taiwan, implying a big environmental impact on most of their cities, which have been transformed into landfills. On account of bad working conditions, their population have a lot of health issues like respiratory distress, leading into many cancer cases.

Due to this situation, it is necessary to apply automated solutions for managing e-waste material and making a correct classification of it, not only in the end of electronic devices service life, but in production process too. This leads to a good social and economic global effect. On one hand, companies can make more money because their products, like PCBs for example, are better monitored (without ignoring their planned obsolescence politics) with these systems. On the other hand, health and labor problems in the countries mentioned before can be avoided, improving living standard of their habitants.

### **What is ROS?**

ROS letters correspond to “Robotic Operation System”. Basically, it is an open source system that is taking robotics world by storm. It allows to program every robot available in the market, due to be a system support by Linux, concretely Ubuntu, opening the possibility of reusing code for robotics and development. Mainly it is based on C++ or Python code.

It is composed by processes, also known as nodes, which can be designed individually and assembled at runtime, because of ROS’s middleware. How nodes are communicated, with messages or services, will be explained in the following section. These processes are also bundled into packages, making easier the communications between different ROS codes. Furthermore, ROS has a lot of toolboxes to help its users in any application they want to make.

## **What is digital image processing?**

Image processing or computer vision is a technique of pattern recognition in different areas of our world. From license plate recognition to fire detection, it is a helpful tool for taking and analyzing images with the purpose of being processed by a machine.

One simple way to understand this concept is to base on our own senses. Artificial vision tries to imitate our comprehension about the world around us but replacing our eyes for a machine. This is accomplished thanks to image breakdown into small fragments (pixels) and a later studio.

### **1.1. Motivation of the project**

Due to the need of applying an automated system that allows e-waste material identification and classification, a prototype is being proposed, formed by a KUKA youBot, which takes care of grabbing electronics components to classify and take them to one place or other, depending on their conditions, and a camera that will take photos previously of these ones. KUKA youBot is going to be programmed on ROS, establishing communication with Octave at the same time, which is going to be our programming language for artificial vision part.

This project provides a solution for electronics components inspection problem, as well as improving product quality, but avoiding accidents and exposures of dangerous agents too.

### **1.2. Main goals**

First, our webcam will take a photo of the electronic component at hand. Octave will process the image thanks to different computer vision algorithms, concluding if this component is acceptable or not. In short, it will be a homographic transformation, which consists in overlaying two different images of the same element, one good and another

bad, in order to subtract one from the other and seeing the differences between them. This is explained deeply in the section relative to computer vision.

Next step will be communicating Octave with ROS, exchanging information between them. It is necessary to exchange one variable that expresses if our element is right or not.

Finally, KUKA youBot is programmed in ROS (robot workspace, home position and speed must be defined previously), which will grip our component and take it to one place or other, depending on its condition.

A scheme of how we are going to manage this project can be seen in Figure 1.

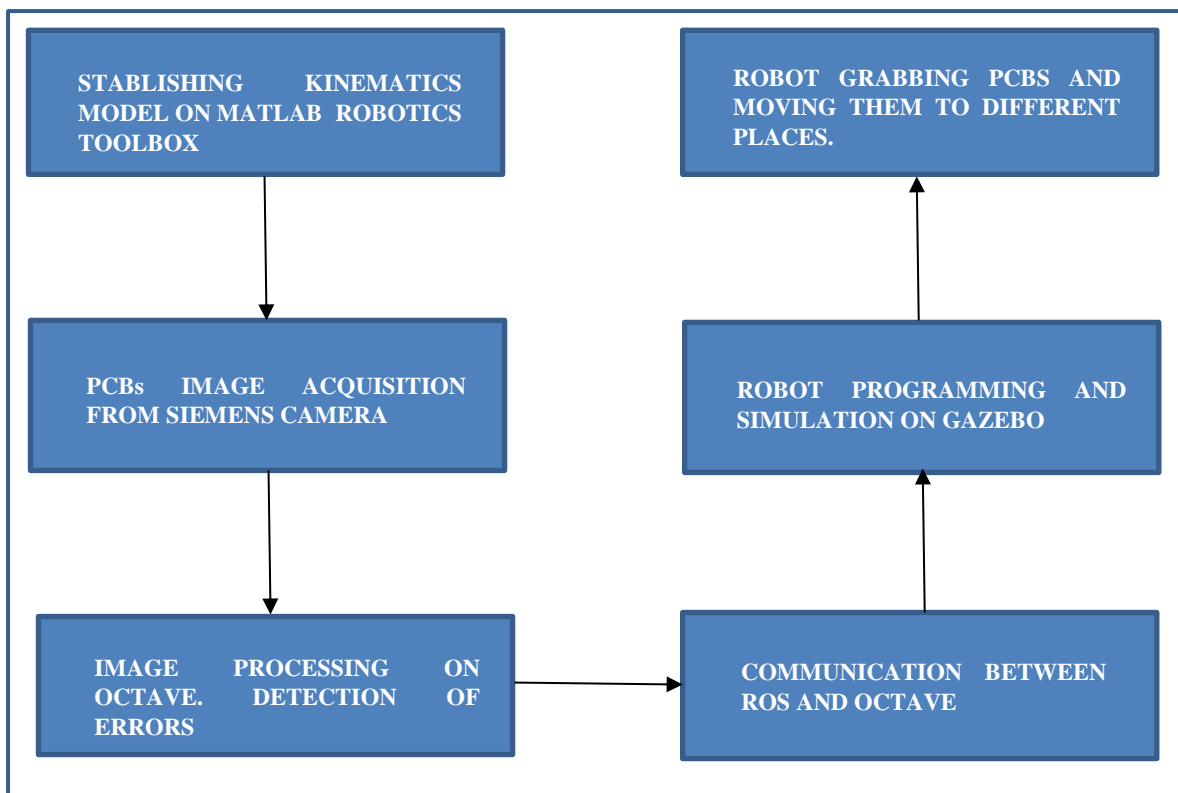


Figure 1. Scheme of final thesis

## 2. ANALYTICAL PART

### 2.1. ROS: architecture and characteristics

#### 2.1.1. Architecture

ROS structure can be divided in three levels of knowledge: filesystem level, computation graph level and community level. We will make a previous introduction of each one of them, to make a deeper analysis afterwards. [2]

**Filesystem level.** This level contains all ROS resources that we can find on disk. Without them, it would not be able to work. It contains packages, metapackages, package manifests, repositories, message types and service types. All these concepts are going to be defined later. Filesystem level's main aim is to group the building process of a project and keep flexibility on each of its parts.

**Computation graph level.** It is the network of ROS processes that are connected all together. This includes a lot of concepts that we should know before starting to program in ROS: nodes, master, parameter server, messages and topics. On this level, any element can interact with other elements, see information sent between them and exchange data.

**Community level.** It is the level in charge of enable different communities in order to transmit information and software. We are not going to focused on this level because it is more oriented to sharing code to the cloud.

##### 2.1.1.1. Filesystem level.

The same way as an operating system like Windows or Linux, ROS programs are formed by folders, which define their functions. We can see this in the Figure 2 attached.



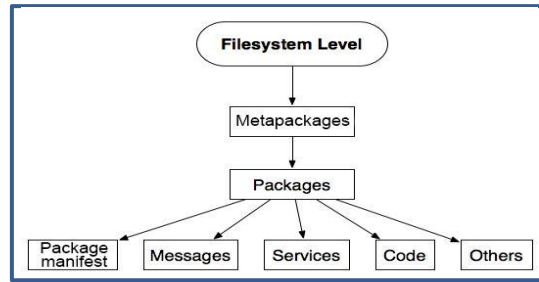


Figure 2. Filesystem level folders[2]

- **Packages.** They are minimum structures necessary for ROS to work properly and can contain configuration files, nodes, etc.
- **Metapackages.** A group of packages received the name of metapackage. An example of metapackage could be the navigation stack.
- **Package manifests.** They are like packages, but with their extension in XML. Package manifests mainly give information to users about packages, licenses, compilation files, and so on.
- **Message types.** It is the kind of information standardized about communication between processes.
- **Service types.** They give information about processes structure in ROS.

### 2.1.1.2. Computation graph level

This level is used by ROS to set up systems, handle all the processes and establish communication with more than one computer only. Now we are going to introduce some concepts that are very useful to understand what computation graph level is about, as shown in Figure 3.

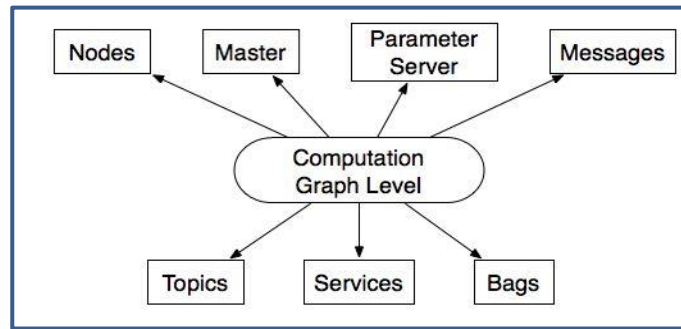


Figure 3. Computation graph level elements[2]

- **Nodes.** They are processes which are communicated to each other. ROS normally uses a combination of nodes in order to manage different functions.
- **Master.** It is the server responsible of the communication between nodes, messages, topics, etc. Regardless of the number of nodes we have, without the master they would be useless because we couldn't send information from one to another. All computers with ROS installed must have a master. Normally we launch ROS master with *roscore* and *roslaunch*.
- **Parameter server.** As its name shows, parameter server provides configuration of nodes parameters.
- **Messages.** They are the way of communication between nodes. In ROS, we can develop our own message types, sending different data to other nodes, such as float, double, integer, etc.
- **Topics.** A standardized way to say that a node is sending information is that it is publishing a topic. Topics are routes used by ROS network to give a name to each message.
- **Services.** In order to get a response from a node, topics are not the answer. Because of this, we need to establish communication previously with services, so every node that has a service can exchange information with the rest, as shown in Figure 3. This is done thanks to request-reply communication, due to distributed systems like ROS do not use client-server or other many-to-many politics.

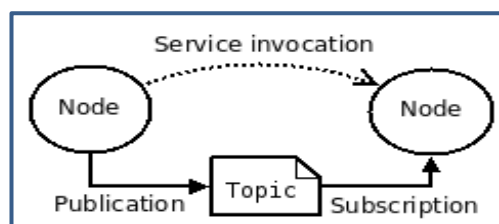


Figure 4. Communication between nodes[3]

- **Bags.** They are used to save data from nodes, such as sensor data, camera data, and so on, to develop new algorithms and programs with it.

## 2.1.2. First steps on ROS

### 2.1.2.1. Creating a new workspace

Before starting to work with nodes and packages, a new workspace must be created to have all the information necessary, such as make files or build files. For this reason, we need to execute the following commands on a Linux terminal, attached in Figure 5.

```
robotas@Youbot:~$ cd beginner_tutorials
robotas@Youbot:~/beginner_tutorials$ cd
(arg: 555) source /opt/ros/~/C
(arg: 5) source /opt/ros/~/C
robotas@Youbot:~$ source /opt/ros/%YOUR_ROS_DISTRO%/setup.bash
bash: /opt/ros/%YOUR_ROS_DISTRO%/setup.bash: No such file or directory
robotas@Youbot:~$ source /opt/ros/beginner_tutorials/setup.bash
bash: /opt/ros/beginner_tutorials/setup.bash: No such file or directory
robotas@Youbot:~$ cd beginner_tutorials
robotas@Youbot:~/beginner_tutorials$ ls
CMakeLists.txt  alberto_pkg  include  package.xml  src
robotas@Youbot:~/beginner_tutorials$ mkdir build
robotas@Youbot:~/beginner_tutorials$ cd build
robotas@Youbot:~/beginner_tutorials/build$ cmake ..
-- The C compiler identification is GNU 6.3.0
-- The CXX compiler identification is GNU 6.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Using CATKIN_DEVEL_PREFIX: /home/robotas/beginner_tutorials/build/devel
-- Using CMAKE_PREFIX_PATH: /opt/ros/melodic
-- This workspace overlays: /opt/ros/melodic
-- Found PythonInterp: /usr/bin/python2 (found suitable version "2.7.13", minimum required is "2")
-- Using PYTHON_EXECUTABLE: /usr/bin/python2
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
```

Figure 5. Creating a new workspace on ROS

Once we have created our workspace, all the packages included in it (and the ones that we are going to include next too) can be seen with the command *ls*, as shown in Figure 6.

[3]

```
robotas@youbot:~/beginner_tutorials/build$ ls
CATKIN_IGNORE  CMakeCache.txt  CMakeFiles  catkin  catkin generated  devel  qtest  test results
```

Figure 6. Packages included in created workspace[3]

## 2.2. KUKA youBot

This robot has been made with the purpose of scientific research and training on an open source platform. In our case, this robot only has an arm with five degrees of freedom as further illustrated in Figure 7, but it normally includes an omnidirectional base. Anyway, we don't need this one for the application explained.



Figure 7. KUKA youBot

We decide to choose this robot for our project because of its small size and flexibility. For grabbing mother bases, it can grip them without making too many movements, in contrast to other types of robots that we can find in the industry. This implies a lot of advantages, but mainly one: our workspace doesn't need to be big to let the robot do its job, as larger ones do.

### 2.2.1. Overview of kinematic structure

#### 2.2.1.1. How to move KUKA youBot arm to its home position

It is crucial to consider that our robot's joints are based on relative coordinates. This means that they have encoders whose position depends on a reference one, so there are not absolute joint angles. The reference position that we mentioned before is also called home position or rest position, which is a starting point to make different actions with the robot. The process of going to this position must be slow and smooth, so we will not break any of robot's joints. [5]

Moreover, a critical factor is knowing that every time robot's movement starts, it must depart from a rest position in order to keep everyone's safety and robot's too. For this motive, we must program our robot to move to home position at the start and at the end of the process. With the purpose of bringing the arm to its rest position, it is necessary to follow these instructions as shown in Figure 8. [6]

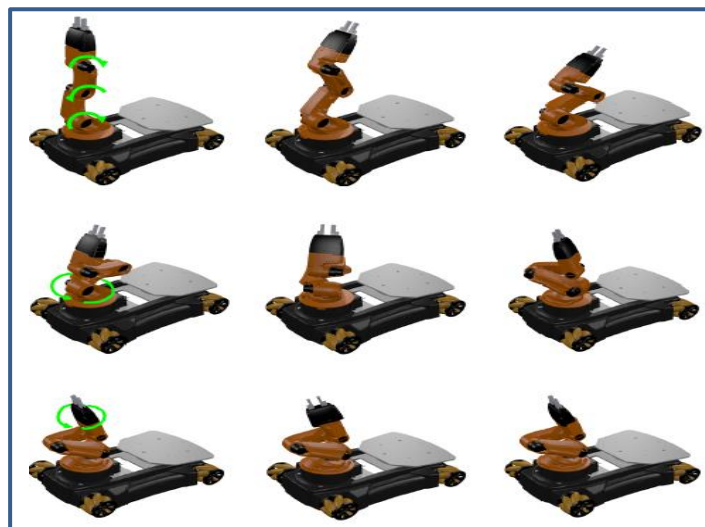


Figure 8. Robot's arm movement to its home position[6]

In a nutshell, this process consists in moving all robot's joints to its mechanical stops. Now we can obtain its Denavit-Hartenberg parameters, but previously an explanation of it is necessary. Following the instructions mentioned before, our robot must be in rest position like it is shown in Figure 9 attached.



Figure 9. KUKA youBot home position

### 2.2.2. Robot kinematics theory

Before introducing D-H parameters and calculating them, there must be an introduction of some robotics and mechanics concepts. [8]

**Robot kinematics.** It is the study of robot's movement compared to a reference system, including an analytical description of it as a time function and a relation between robot's end location and joints values. We can see an example of this explanation in Figure 10, which is a direct kinematic model of 2 D.O.F (Degrees Of Freedom) plane robot. Our joints coordinates are  $q_1$  and  $q_2$ , while  $(x, y)$  are robot's end position grid reference.

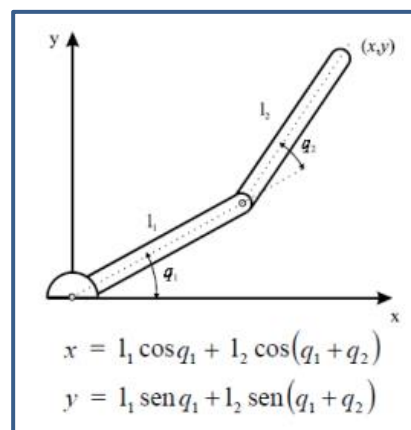


Figure 10. Kinematics model for a 2 DOF robot[8]

**Direct kinematics problem.** Its aim is to get robot's end position and orientation with regard to a baseline coordinates system. Previously, we have to know joints angles and geometrical parameters of robot's elements.

**Differential model.** It is a mathematical model, based on Jacobian matrix which associates joints movement speed with robot end's ones.

To make things easier, we are going to use Denavit-Hartenberg model to calculate kinematics model of KUKA youBot.

**Inverse kinematics problem.** In this case, the issue is the opposite compared to the direct kinematics one. Here, we know the end positions, but our main goal is getting arm positions and angles in order to reach it.

### 2.2.3. Denavit-Hartenberg parameters

J. Denavit and R.S. Hartenberg proposed, back in 1955, a matrix model that gives a reference system in a systematic way, relative to each tier named  $i$  from a joint, so robot's kinematic model can be obtained of it.

According to this representation, choosing a reference system for each tier, we can obtain the next one following 4 basic transformations that only depend on tier's geometrics characteristics. [8] These transformations are based on rotations and translations:

- Rotation around  $z_{i-1}$  axis an angle  $\theta_i$ .
- Translation through  $z_i$  axis a distance  $d_i$ .
- Translation through  $x_i$  axis a distance  $a_i$ .
- Rotation around  $x_i$  axis an angle  $\alpha_i$ .

Finally, we will obtain a matrix like the attached one in Equation 1. Also, an example of how to calculate D-H parameters is shown in Figure 12.

Note:  $C\alpha_i$  and  $S\alpha_i$  mean  $\cos(\alpha_i)$  and  $\sin(\alpha_i)$ , respectively.

$${}^{i-1}A_i = \text{Rot}_z(\theta_i) T(0,0,d_i) T(a_i,0,0) \text{Rot}_x(\alpha_i) = \begin{bmatrix} C\theta_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i & a_i C\theta_i \\ S\theta_i & C\alpha_i C\theta_i & -S\alpha_i C\theta_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 1. Denavit-Hartenberg parameters matrix

Furthermore, we can see how to obtain twists and lengths in Figure 11, continuing with D-H model.

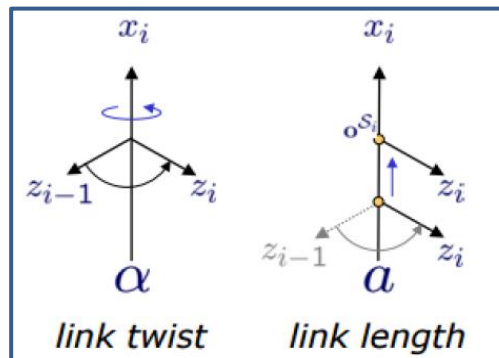


Figure 11. D-H link twist and length[8]

Because this process is really annoying to do, as it requires 16 steps according to D-H model to obtain its parameters for our robot, we are going to use Robotics Toolbox on MATLAB for it.

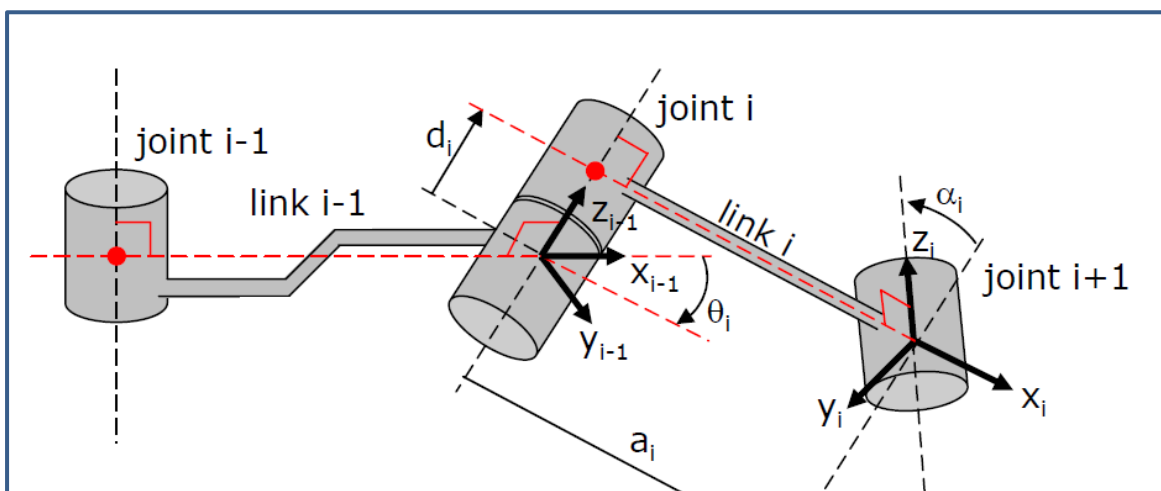


Figure 12. Example of D-H parameters obtaining[8]



In KUKA youBot case (5 DOF), D-H symbolic parameters are included in following Table 1. These parameters were explained at the beginning of this section [2.2.3]

Link	$\alpha_i$	$a_i$	$d_i$	$q_i$
1	$\alpha_1$	$a_1$	$d_1$	$q_1$
2	$\alpha_2$	$a_2$	$d_2$	$q_2$
3	$\alpha_3$	$a_3$	$d_3$	$q_3$
4	$\alpha_4$	$a_4$	$d_4$	$q_4$
5	$\alpha_5$	$a_5$	$d_5$	$q_5$

Table 1. D-H symbolic parameters for KUKA youBot

These parameters will be used in next sections also for obtaining inverse kinematics in our project. [3.2.1]

### 2.2.4. Inverse kinematics

In most of cases, we know the position which we want our robot to go, but it is necessary to get their joints positions and angles for it. [18] Due to this, inverse kinematics concept is introduced, as shown in Figure 13.

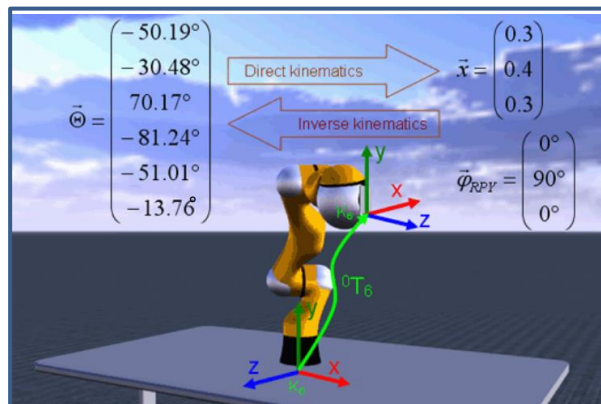


Figure 13. Example of inverse kinematics[18]

Basically, inverse kinematics aim is obtaining joint coordinates values for our robot, in order to its TCP or Tool Center Point gets a specific position and orientation. Moreover, this solution is not systematic, so it means there is not only a unique one, depending on robot's configuration.

We can distinguish between two methods to solve this problem: geometric and homogenic transformation matrix (like the one we saw in Equation 1). One example of calculating inverse kinematics for a 2 DOH robot is attached in Figure 14. Furthermore, it can be seen in that there are two possible options for it.

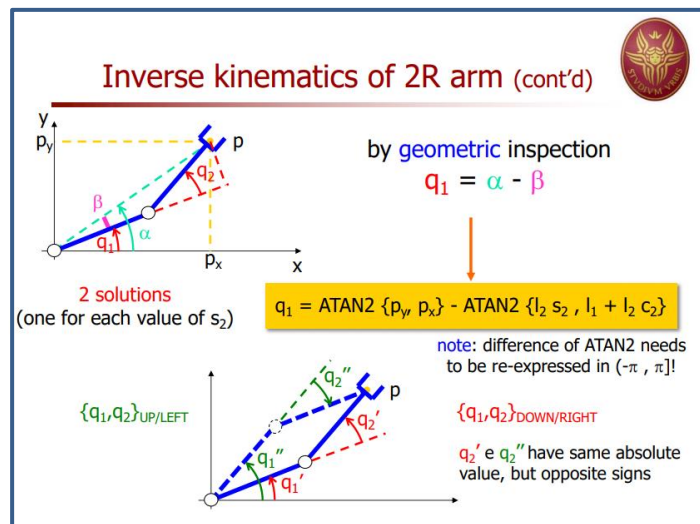


Figure 14. Inverse kinematics of 2R arm[18]

As shown in Figure 15, we know position of arm's end, called P, and with trigonometrical relations we can obtain angles q1 and q2 to reach that specific position.

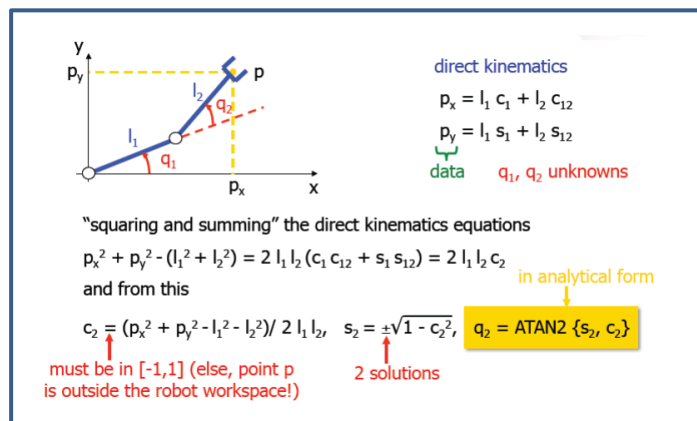


Figure 15. How to calculate q1 and q2 for 2R arm[18]

From KUKA youBot perspective, it is a big problem, where we will reduce robot inverse orientation kinematics problem from 5 D.O.F arm to 3, as shown in Figure 16 . Moreover, it is notable to explain each equation to understand this better in a mathematical way.

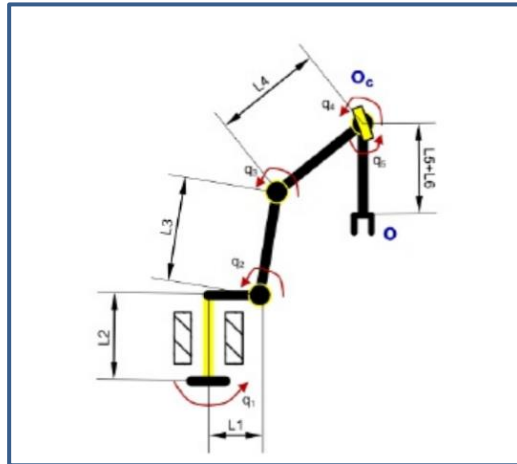


Figure 16. KUKA youBot model for obtaining inverse kinematics[20]

If we have a given point, in this case called  $o$ , and also a  $R$  matrix, we can get point  $oc$ , as Equation 2 shows, where  $R$  is rotation matrix of the robot.

$$o_c^0 = \begin{bmatrix} xc \\ yc \\ zc \end{bmatrix} = o - (L5 + L6)R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} ox - (L5 + L6)r13 \\ oy - (L5 + L6)r23 \\ oz - (L5 + L6)r33 \end{bmatrix}$$

Equation 2. Inverse kinematics for KUKA youBot orientation

In terms of orientation, this problem is solved, but we also need to know its position. That is our next step to achieve. To facilitate equations comprehension for the reader, Figure 17 is attached. Moreover, it should be kept in mind that all these parameters have a range of values (minimum and maximum) which can be seen in Figure 19, provided by the manufacturer.

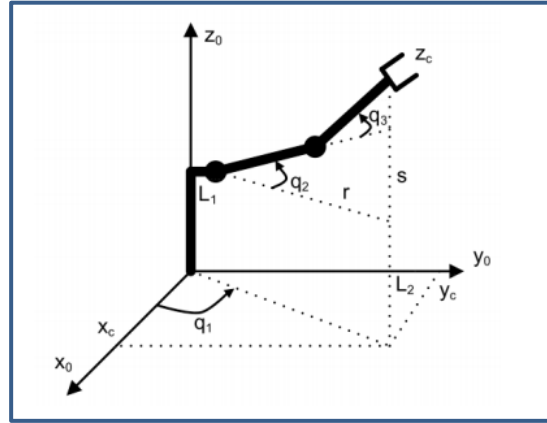


Figure 17. KUKA youBot manipulator scheme in XYZ axes[20]

Thanks to its projection to  $(x_0, y_0)$ ,  $q_1$  angle can be obtained as we can see in Equation 3.

$$q_1 = \tan^{-1}\left(\frac{y_c}{x_c}\right)$$

Equation 3. Q1 angle

In order to obtain  $q_2$  and  $q_3$  angles, we will apply some trigonometric rules like cosine one for triangles, concluding with their expressions shown in Equation 4 and Equation 5, which will be computed on MATLAB, as we will see in next sections. [3.2.1]

$$q_3 = \tan^{-1}\left(\pm \frac{\sqrt{1 - 4R^2}}{2R}\right)$$

Equation 4. Q3 angle

$$q_2 = \tan^{-1}\left(\pm \frac{\sqrt{x_c^2 + y_c^2} - L_1}{z_c - L_2}\right) - \tan^{-1}\left(\frac{L_4 \sin(q_3)}{L_3 + L_4 \cos(q_3)}\right)$$

Equation 5. Q2 angle

$$q_4 = -q_2 - q_3 + \pi/2$$

Equation 6. Q4 angle

## 2.2.5. Technical specifications

For this part, we are going to start with robot's joints first. They are formed by a total of five brushless flat motors EC45 and EC32 models, like the ones we can see in Figure 18, plus gearheads and encoders too.



Figure 18. Joints motors[8.2]

Both motors, despite not having big dimensions (only 32 mm diameter), can provide 15-50 W of power.

About robot's arm dimensions, it is approx. 66 cm long, with a two-finger gripper at the end of the arm, being able to grab objects of 500 g max. We can see this with more detail in Figure 19.

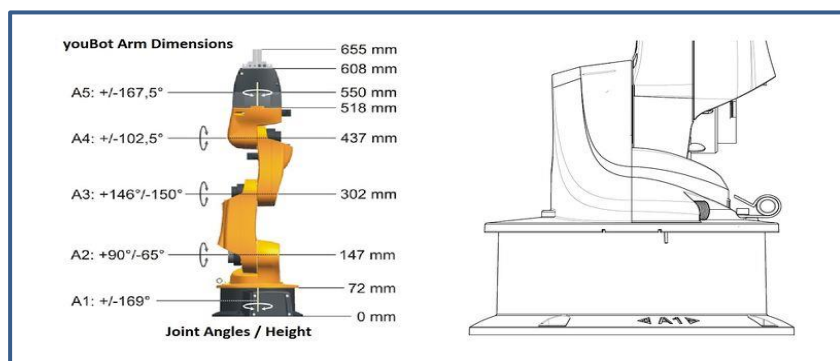


Figure 19. Robot's angles and height[8.1]

### 2.2.6. Connections

For connecting I/O devices, KUKA youBot has USB (keyboard and mouse) and VGA (PC monitor) ports, so the rest of connections, like HDMI or DVI, require appropriate adapters, as can be seen in Figure 20. Joint's motors can be controlled thanks to EtherCAT and Ethernet. Moreover, it has a ON/OFF port obviously to provide power to robot.



Figure 20. KUKA youBot connections

### 2.3. Camera

We must keep in mind that KUKA youBot already has a camera, specifically a Bumblebee Stereo Camera, but its accuracy and range are not very suitable for our application. [4] On account of this, thanks to VGTU's Mechatronics and Robotics department, we acquired a Siemens Simatic Vision Camera, in detail a MV440 model, which is designed for industrial recognition and assessment of multiple pixels. A picture of this camera can be seen in Figure 21.



Figure 21. Siemens MV440 camera[4]

A little review of its most important technical specifications is shown in Table 2. We can see there that, in terms of image recognition, our camera is really fast, capable of watching objects with a speed of 10 m/s. It is a great advantage compared to KUKA's camera because we can detect mother bases in a shorter period of time and, from an industry perspective, this implies a faster production process.

**Electronical and reading specifications:**

Power supply (V)	Image acquisition	Degree of protection	Object speed (m/s)	Number of I/O	Codes read per image acquisition
24	CCD	IP 67	10	4	150

Table 2. Electronical and reading specifications

Furthermore, camera's reader has a resolution of 1024 x 768 pixels, really good for making quality photos with high definition. This is crucial for our application because a high resolution is necessary for detecting mistakes in mother bases, such as components not well welded and so on. An image of our camera with all its connections and components is attached in Figure 22.

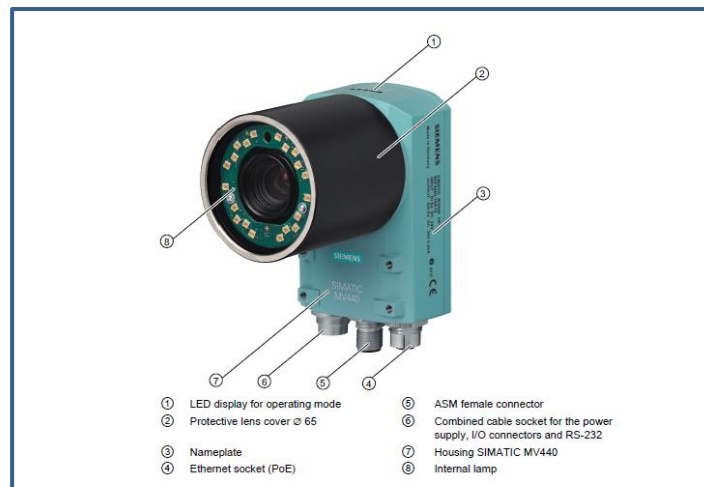


Figure 22. MV440 camera connections and components[4]

In respect of connection ports, our camera works with Ethernet and obviously its power supply port. The Ethernet one is used to control our camera with a PC, so it must require a network connection via TCP/IP. Combined cable socket (number 6 in Figure 11) is used for power supplying, I/O connectors and RS-232 interface.



## 2.4. Computer vision on Octave

Instead of using MATLAB, we are going to develop the part relative to computer vision with Octave, which functions and programming are really similar to MATLAB, but it is an open source IDE, which means that nodes communications between it and ROS will not be a problem in terms of permissions, keys, etc. Octave is a MATLAB compatible software, so every program made in MATLAB can be compiled in Octave and vice versa. Moreover, thanks to its working on Linux softwares, Octave can take any kind of camera for image acquisition, not like MATLAB which only accepts a specific amount of them, as shown in Figure 23.



Figure 23. MATLAB supported vision cameras

### 2.4.1. Programming on Octave. Artificial vision most used commands[11]

We are going to use mainly a combination of these basic commands on Octave, but they are the same as MATLAB:

- *Imshow*. It shows an input image.
- *Imread*. It loads an image from disc on a I matrix, accepting multiple formats as (.jpg, .tif, .bmp, etc)
- *Imwrite*. It saves an I image on disc with a specific format.
- *Imhist*. It calculates an histogram from a I image and it saves it on a variable.
- *Imtool*. Tool for image processing that allows to zoom, measure, visualize and manipulate histograms, contrast ajust and shining.

We can see some examples of code and image processing, made on MATLAB, which work on Octave perfectly. Images attached in Figure 24, Figure 25 and Figure 26.

```
%%  
  
%Shows an image through screen  
  
I=imread('brazo_robot.jpg');  
  
figure  
  
imshow(I)  
  
%%  
  
whos  
  
%%  
  
figure  
  
J=imread('HeroImage_PCB.jpg');  
  
imshow(J)  
  
whos  
  
figure  
  
W=rgb2gray(J); % convert image into greys with rgb2gray  
  
imshow(W)
```

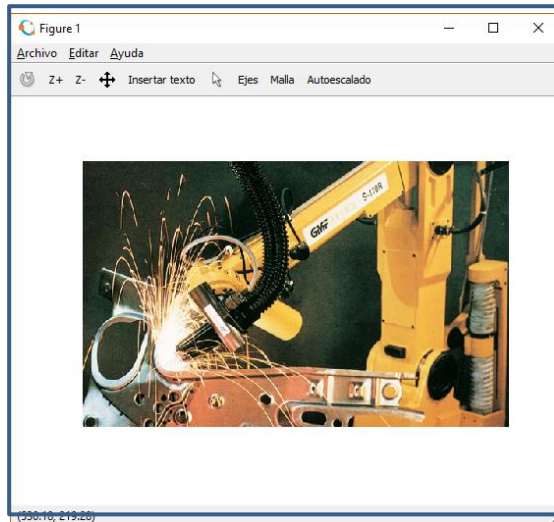


Figure 24. Robot's arm image with imshow[9]

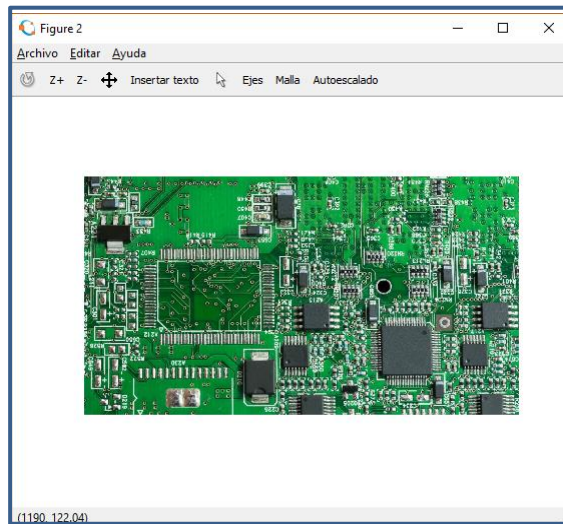


Figure 25. PCB image with imshow[9]

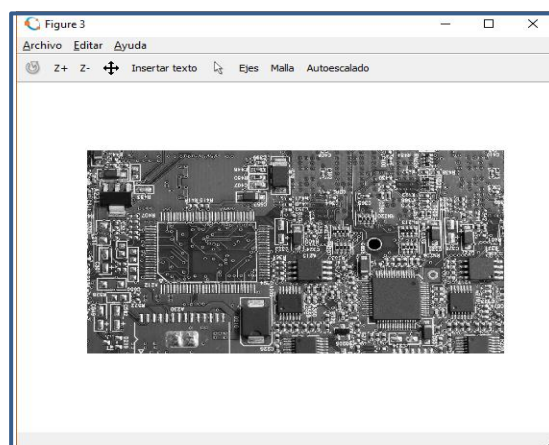


Figure 26. PCB image converted to greyscale with rgb2gray[9]

## 2.4.2. Computer vision techniques for error detection in PCBs [11]

In order to detect components not well welded, corrosion or other type of errors in mother bases, we have to introduce a definition, called morphology, which is going to be applied on Octave.

Morphology is a mathematical concept, introduced by G. Matheron and J. Serra at Mines School of Paris in the 1960s, which is applied for changing and understanding objects structure that appear in images. An important concept in morphology is the structuring element, which is the responsible of defining form and extension of morphological operation. This means that we can apply morphology on a specific zone of our picture without changing the rest, thanks to structuring element.

First of all, we consider that readers have a previous knowledge of Set Theory and its operations such as union, intersection, complement, and so on. For our project, we are going to use two main morphological operations: dilation and erosion.

- Dilation. This operation is similar to the XOR logical port, where we have an image A and structuring element B, as it is shown in attached example in Equation 7.

$$A \text{ XOR } B = \begin{Bmatrix} (0,0) & (0,1) & (0,2) \\ (1,0) & (1,1) & (1,2) \\ (2,0) & (2,1) & (2,2) \end{Bmatrix}$$

Equation 7. Example of dilation matrix

Result of dilation is an image which is a set of all vectors possibles of adding pairs elements of both A and B. A graphical example of this can be seen in Figure 27.

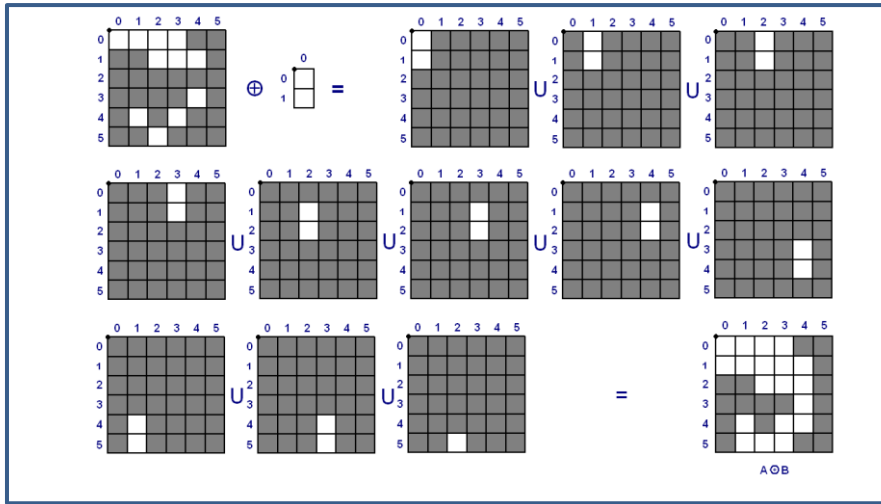


Figure 27. Graphical example of dilation matrix[11]

- Erosion. This operation implies a decrease of active pixels number compared to our original image. Its main application is to eliminate details of small size on a binary image. We can see a graphical example of erosion in Figure 28 and also a practice of it in Figure 29, in order to count the number of screws in attached image.

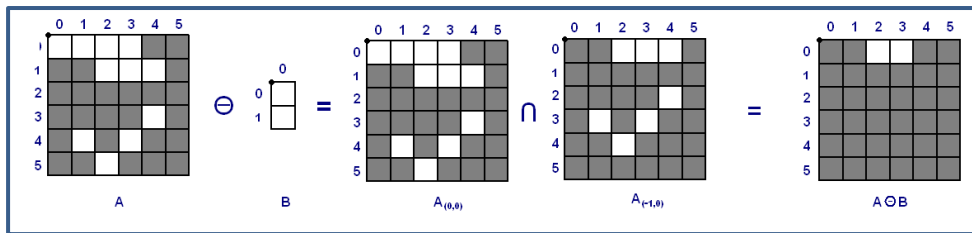


Figure 28. Erosion graphical example[10]

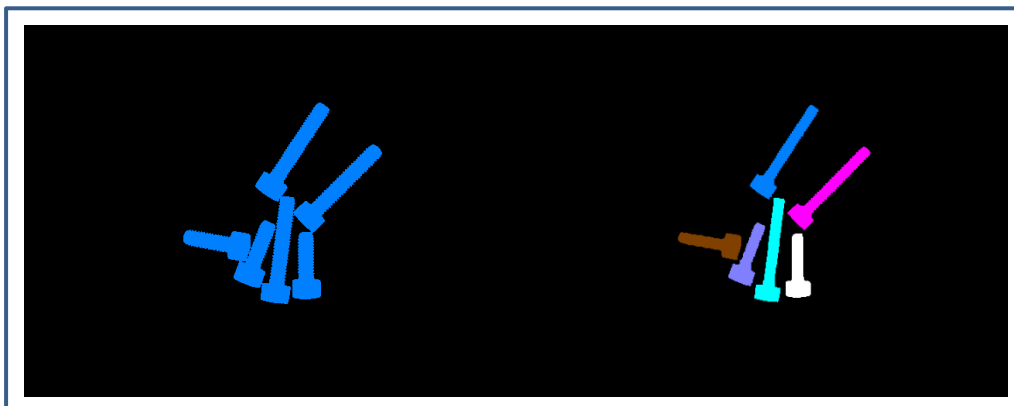


Figure 29. Erosion practical example, before and after. [10]

As we have seen in last picture, erosion is a very useful tool to count pieces or distinguish different elements of them, with a view to apply other morphological operations so we could know their area, labelling, etc.

Main commands on Octave are the following: [11]

- `Se=strel('disk', R)`. Structuring element with size disk of radius R.
- `Se=strel('line', long, grad)`. Lineal structuring element with height long, grad specifies angle in grades on counterclockwise.
- `Se=strel('square', w)`. Square structuring element of w pixels.
- `D=imdilate(BW, se)`. Picture dilation with structuring element se.
- `E=imerode(BW, se)`. Picture erosion with structuring element se.
- `C=imclose(BW,se)`. Morphological closing of BW image with element se.
- `O=imopen(BW, se)`. Morphological opening of BW image with element se.
- `Z=imsubtract(X, Y)`. Subtraction of images X and Y.

Previous to apply morphological operations, we must filter our images to eliminate pixels that can ruin fails detection process. Filtering aim is to modify images in order to make them more proper for later transformations. One thing that should be considered is that filtering can eliminate desired characteristics of our image, so it needs to be applied carefully.

Depending on which mathematical expression has been used, we can distinguish between lineal and non-lineal filters. From lineal filters perspective, there are two main types: average filter and gauss filter. From non-lineal filters view, most important one is median strainer.

In our project we are going to focused on lineal ones, mainly on gauss filter, because they have better applications for filtering out pixels that are less important in our image. Gauss filter is based on mathematical bell of the same name, which downgrades less image at issue than average filter, keeping details and outlines.

An example of filtering is shown in Figure 30, which corresponds to a photo of a one-euro coin. Here first image has noise (specifically a salt and pepper one) and second one is after applying median strainer.



Figure 30. Example of filtering[10]

Moreover, our last computer vision method that will be applied is template matching, which is based on statistical concept of correlation coefficient. Level of association between two variables (in our case it will be pictures) is defined as correlation coefficient, whose ranges change between -1 and 1. This means that if two variables have a correlation of 1, they are identical. Mathematical formula is attached in Equation 8 . However, we are not going to get into this concept deeply, but in image processing one instead.

$$\rho_{x,y} = \frac{E[(X - \mu_x)(Y - \mu_y)]}{\sigma_x \sigma_y}$$

Equation 8. Correlation coefficient[9]

In terms of computer vision, correlation will be translated to matrix correlation, simply searching for elements in common between images. Each pixel of search image with coordinates (xs, ys) has intensity matrix  $I_s(x_s, y_s)$  and another one in the jig with coordinates (xt, yt) has intensity matrix  $I_t(x_t, y_t)$ . Therefore, absolute difference in the pixel intensities is defined as the following:  $\text{Diff}(x_s, y_s, x_t, y_t) = |I_s(x_s, y_s) - I_t(x_t, y_t)|$ .

Lowest differential provides an estimate for best position of template within main picture, being shown in Figure 31, where elements that match are marked by an asterisk.

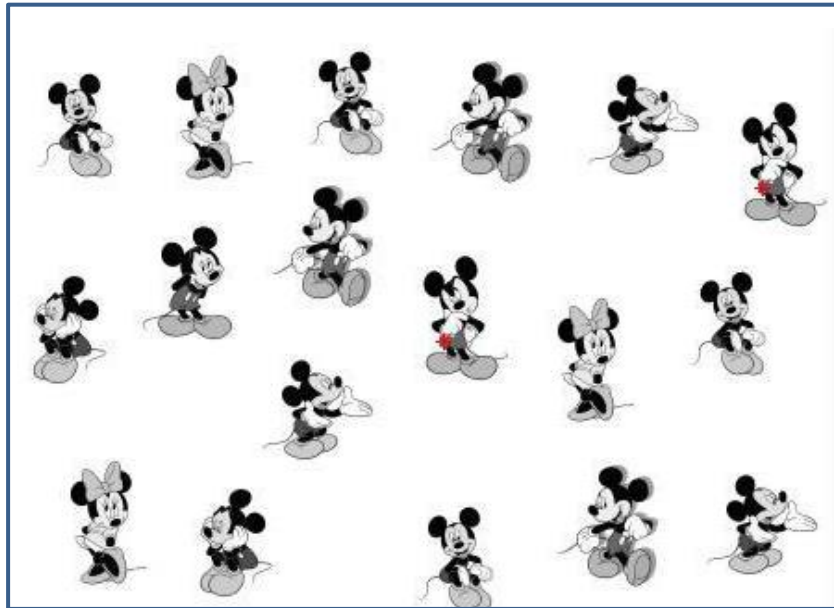


Figure 31. Example of template matching[10]

### 2.4.3. PCBs. Production and use errors.

During production process and also when PCBs end their life cycle, there are some different types of errors that can produce their bad working, such as corrosion, components not well welded or broken, etc. Identification of these errors is purely visual and done manually by workers, but in some occasions, they are difficult to see for human eye.

In our case, we are going to use PCBs from printing toners, with a standard one, as it is shown in Figure 32, being compared with others that have broken pads or other types of errors.

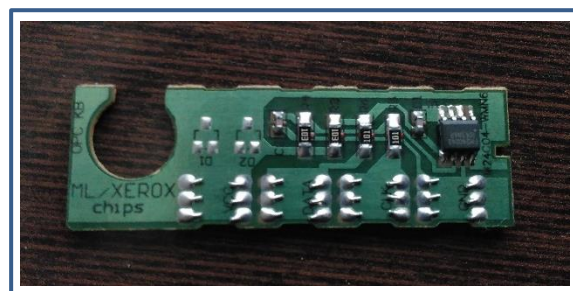


Figure 32. PCB ML/XEROX model used for detection



Thanks to these reference images, we are going to compare them easily, due to the fact that there are big differences between well working PCBs and underwhelming ones, as we can see in the following picture, Figure 33.

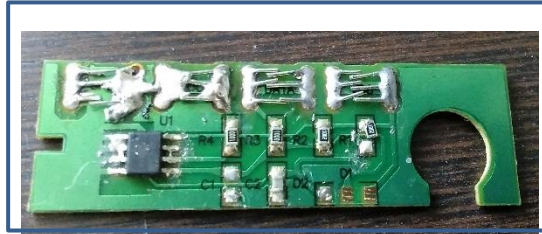


Figure 33. Bad PCB ML/XEROX model

On this image it is quite obvious that there is too much welding in pads and also some components such as capacitors are missing.

To sum up, we are going to do a recognition that can be seen by workers, but in an automated way, as it implies better production process as well as more safety for employees due to exposition of noxious fumes.



For connecting camera to voltage, we needed to shell part of power supply cable given by manufacturer, attached in first picture of Figure 36, where red wire is connected to plus pin and blue one to minus. In order to connect these cables to power supply, we use a hitch as shown in second image of Figure 35.

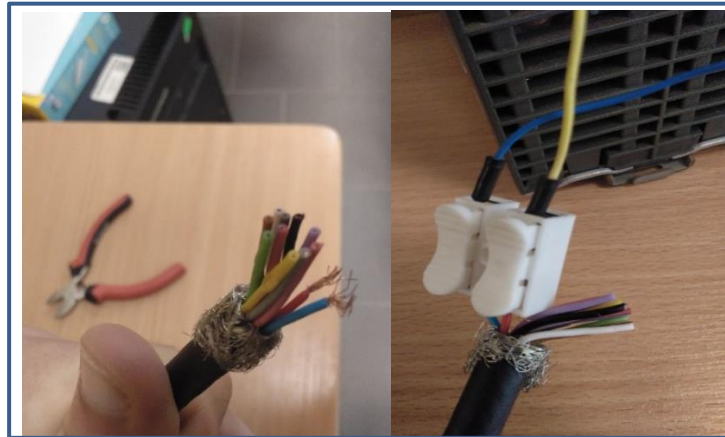


Figure 35. Camera connections to voltage[3.1.1]

Our power supply provides 24 V DC, enough to make camera work, as we can see in Figure 36, where yellow cable corresponds to plus and blue one to minus, which are connected with camera wires as we saw in previous paragraph.



Figure 36. 24 V DC power supply[3.1.1]

### 3.1.2. Platform instalation for camera.

It is necessary to install a platform, as camera doesn't disturb robot's movement and provides good quality photos for computer vision. Furthermore, camera has different holes in its back part and a template where we can screw it to a proper area, shown in Figure 37.

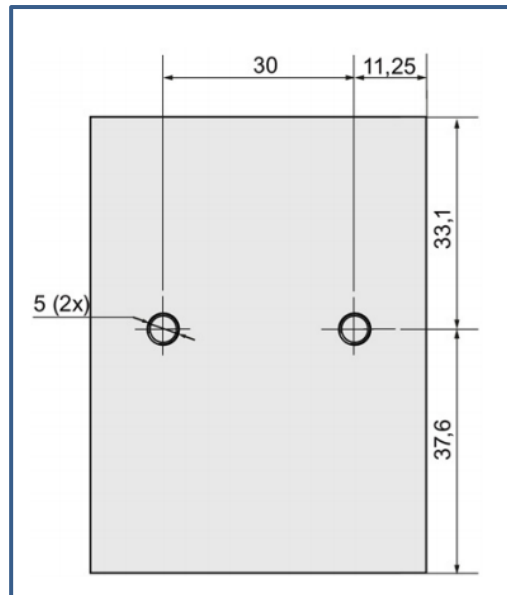


Figure 37. Holes dimensions for screwing camera to platform[8.3]

Included in camera's box, there is a template which we connect to the platform and the camera to it as well. Our camera needs to be in a vertical position in order to take photos of PCBs properly. When we are trying to detect errors in this kind of devices, it is better to do it in this position, as programming will be easier for distinguishing PCBs components of other elements appearing in pictures. Consequently, we screwed the template to the platform as shown in Figure 38.



Figure 38. Template and camera screwed to platform[3.1.2]

As a result, our robot and camera workspace remain as we can see in Figure 39, where they don't bother each other's working.



Figure 39. Workspace with robot and camera installed [3.1.2]

## 3.2. Robot simulation.

### 3.2.1. Forward and inverse kinematics equations obtained on Robotics Toolbox.

To prove our code developed on MATLAB[7.3] and obtain trustworthy values for forward and inverse kinematics equations of KUKA youBot, we are going to test it as shown in Table 3, where we put different values for the angles with the purpose of getting point position. Both Q4 and Q5 angles are introduced by user, regardless of values for the rest. [20]

Test	Q1	Q2	Q3	Q4	Q5
1	0	0	0	0	0
2	$\pi/2$	0	$-\pi/2$	0	0
3	0	$-\pi/2$	0	$\pi/2$	0

Table 3. Test values for MATLAB Robotics Toolbox on KUKA youBot

Once we apply our algorithm on MATLAB Robotics Toolbox, we can conclude that using this test values and also D-H parameters for KUKA youBot, it works perfectly, as we can see on Figure 40, Figure 41 and Figure 42. In these pictures, we tested our model and obtained D-H parameters for KUKA youBot.

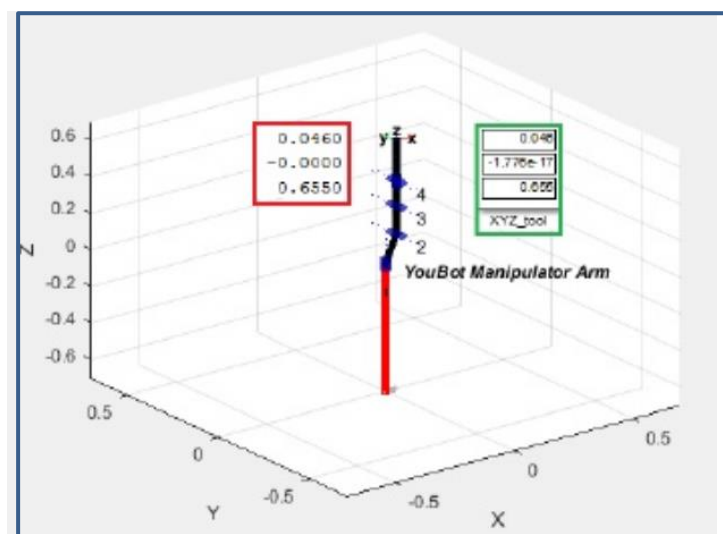


Figure 40. Test 1 for KUKA youBot on MATLAB

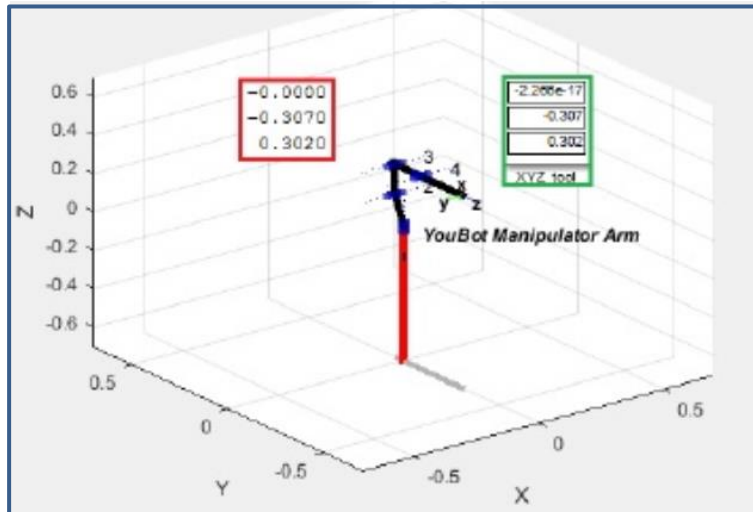


Figure 41. Test 2 for KUKA youBot on MATLAB

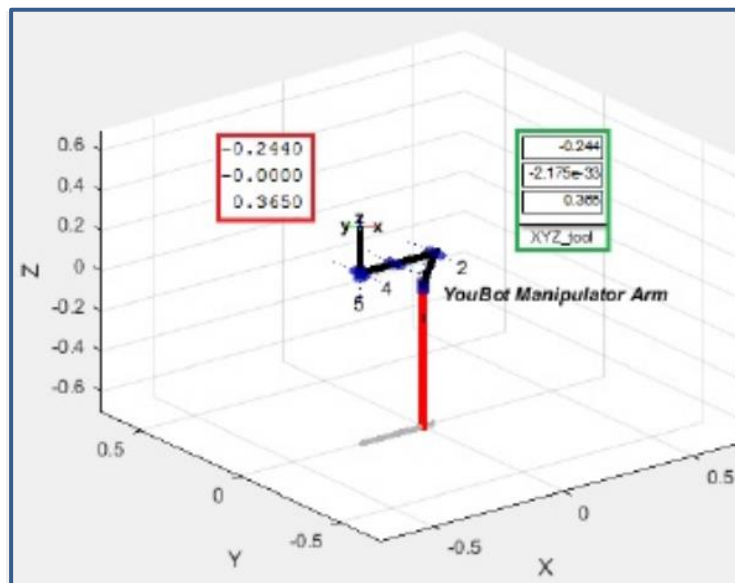


Figure 42. Test 3 for KUKA youBot on MATLAB

As a conclusion, we obtained these values as a prove that our theoretical model, which will be applied on our C++ code, is accurate. [3.2.3] Moreover, this code will follow the equations shown in previous sections. [2.2.4] For the inverse kinematics part, we obtain the values of Q angles thanks to the code attached in [2] and [3].

Therefore, D-H parameters for the KUKA youBot are shown in Table 4, where L1, L2, L3, L4, L5 and L6 are known values from our robot specifications[Figure 19] and  $q_i$  angles are obtained on inverse kinematics code.

Link	$\alpha_i$	$a_i$	$d_i$	$q_i$
1	$-\pi/2$	L1	L2	$q_1$
2	0	L3	0	$q_2$
3	0	L4	0	$q_3$
4	$\pi/2$	0	0	$q_4$
5	0	0	L5+L6	$q_5$

Table 4. D-H parameters for KUKA youBot obtained with MATLAB Robotics Toolbox

Finally, these angles values can be seen in the following Table 5, which will be applied in our C++ code afterwards.

LINK	$q_i$
1	0
2	-1,9526
3	-2,0192
4	1,73184
5	3,14

Table 5. Values for  $Q_i$  angles of KUKA youBot



### **3.2.2. Inverse Kinematics programming for KUKA youBot.**

First of all, KUKA youBot model needs to be initialized. Then, we implement our model with `youbot_driver` package, downloaded from GitHub webpage. After that, we need to test robot's movement, as its gripper can be calibrated in order to take our PCBs in a proper way. This is done thanks to `hello_world_demo` program, included in the package mentioned before, where the robot starts to move and its gripper as well, stopping in its rest position. In terms of our program, we are going to modify this one and some inverse kinematics code that we can find on the Internet, as well as adding our communication with computer vision part. [17]

Next, we need to program inverse kinematics of our robot, so we will know each notable angle and position for our joints. Then, with the results obtained by computer vision part, we should configure robot movement to reach the positions we wanted in first touch.

For the inverse kinematics part, we programmed it on C++, where now we are going to explain each part of the code and what it does in next sections. Furthermore, it is remarkable that relevant positions were taken empirically, configuring our KUKA youBot with PC keyboard, following the instructions given inside its user manual, as it can be seen in Figure 35. This code is included in [2]. ROS C++ code, part `keyboard.cpp`.

#### 8.1.4 Using the ROS wrapper for the KUKA youBot API

To start the driver you have to open a terminal window and type:

```
$ roslaunch youbot_oodl youbot_oodl_driver.launch
```

Per default the driver will assume a KUKA youBot base in combination with one arm. In case only the base or only the arm is available the wrapper will print according error messages but continue to work with the components that are found, as long as proper configuration files are provided as described in Section 5.2.2.

In case you want to visualize the robot and its movements e.g. with the *rviz* tool, open another terminal and type (see also Section 8.1.5):

```
$ roslaunch youbot_oodl youbot_joint_state_publisher.launch
```

The `youbot_oodl` package comes with two simple example applications. The first one is a keyboard based teleoperation application for the base. You can start it with

```
$ rosrn youbot_oodl youbot_keyboard_teleop.py
```

The following keys can be used to move the KUKA youBot,

u	i	o
j	k	l
m	,	.

where `<i>` means forward, `<,>` backward, `<j>` left, `<l>` right, `<m>` turn left on spot, `<. >` turn right on spot, `<u>` move along an arc to the left and `<o>` move along an arc to the right. All other keystrokes will stop the KUKA youBot.

**NOTE:** The key bindings work best with a German keyboard layout. In case of a different layout you might consider to adapt the source code.

The second sample application allows to set joint angles for an arm. It asks for values for each arm. When all values are specified, the command will be executed. You can start the sample with:

Figure 43. Instructions for using KUKA youBot with keyboard [3]

### 3.2.3. Robot programming on C++

First, we should remark that for our program we are following the right hand rule, meaning that all robot's movement are going to follow the reference system attached in Figure 44, but in our case without omnidirectional base. Positive X-axis direction is straight outward from the front of our robot. Positive Y direction is on the left side and Z is positive in the upward one.

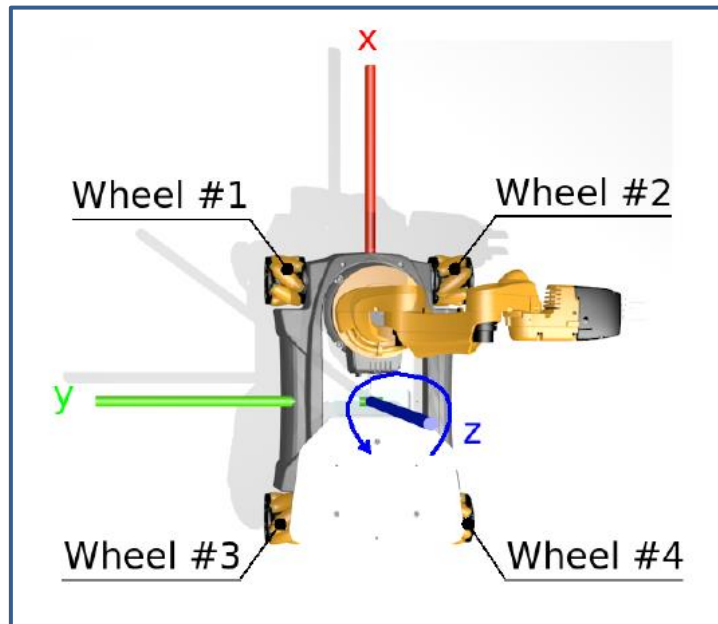


Figure 44. Reference system for KUKA youBot programming [5]

Next step remarkable that we are going to take is explaining each part of code relative to our robot. [1]

```
#include "youbot/YouBotBase.hpp"
#include "youbot/YouBotManipulator.hpp"
#include <math.h>
#include <studio.h>
#include <ImageService.h>

using namespace youbot;

int main() {

double px=0,pz=0,py=0;
double theta_base=0, theta_2=0, theta_3=0, theta_4=0;
double theta_link_1=0 , theta_link_2=0;
double theta_link_1p=0, theta_link_2p=0;

/* configuration flags for different system configuration (f.e. arm without base)*/
```



```
bool youBotHasBase = false;
bool youBotHasArm = false;

/* define speeds */
double translationalVelocity = 0.05; //meter_per_second
double rotationalVelocity = 0.2; //radian_per_second

/* pointers for youBot base and manipulator */
YouBotBase* myYouBotBase = 0;
YouBotManipulator* myYouBotManipulator = 0;

/* obtain response from Octave */
    res=req.create_response_();
```

As we can see here, first variables are configured with value 0 or False in defect, except for translational and rotational speeds. This is because of they will change during each loop that we will apply in a relative form, so it means we are using increments, not absolute coordinates. Also, we put some youBot base related variables due to the fact that in all simulators available for KUKA youBot they have it included, so we will not have problems with it. Finally, we have to receive a response from our image processing program, so we include a data structure res for it.

```
try {
myYouBotBase = new YouBotBase("youbot-base", YOUBOT_CONFIGURATIONS_DIR);
myYouBotBase->doJointCommutation();

youBotHasBase = true;
} catch (std::exception& e) {
LOG(warning) << e.what();
youBotHasBase = false;
}

try {
```

```

myYouBotManipulator = new YouBotManipulator("youbot-manipulator",
YOUBOT_CONFIGURATIONS_DIR);

myYouBotManipulator->doJointCommutation();

myYouBotManipulator->calibrateManipulator();

// calibrate the reference position of the gripper

//myYouBotManipulator->calibrateGripper();

youBotHasArm = true;

} catch (std::exception& e) {

LOG(warning) << e.what();

youBotHasArm = false;

}

```

This part is the relative of proving robot configurations such as gripper's or arm's.

```

/* Variable for the base.

quantity<si::velocity> longitudinalVelocity = 0 * meter_per_second;

quantity<si::velocity> transversalVelocity = 0 * meter_per_second;

quantity<si::angular_velocity> angularVelocity = 0 * radian_per_second;

/* Variable for the arm. */

JointAngleSetpoint desiredJointAngle;

//GripperBarSpacingSetPoint gripperSetPoint;

```

Then, we define robot speeds variables, as well as joints angles.

```

try {

/*

* Simple sequence of commands to the youBot:

```



```
*/

/*if (youBotHasBase) {

/* move forward */

/*longitudinalVelocity = translationalVelocity * meter_per_second;
transversalVelocity = 0 * meter_per_second;
myYouBotBase->setBaseVelocity(longitudinalVelocity, transversalVelocity, angularVelocity);
LOG(info) << "drive forward";
SLEEP_MILLISEC(2000); /*we always put a little delay between moves*/
/* move backwards */
/*longitudinalVelocity = -translationalVelocity * meter_per_second;
transversalVelocity = 0 * meter_per_second;
myYouBotBase->setBaseVelocity(longitudinalVelocity, transversalVelocity, angularVelocity);
LOG(info) << "drive backwards";
SLEEP_MILLISEC(2000);

/* move left */
/*longitudinalVelocity = 0 * meter_per_second;
transversalVelocity = translationalVelocity * meter_per_second;
angularVelocity = 0 * radian_per_second;
myYouBotBase->setBaseVelocity(longitudinalVelocity, transversalVelocity, angularVelocity);
LOG(info) << "drive left";
SLEEP_MILLISEC(2000);

/* move right */
/*longitudinalVelocity = 0 * meter_per_second;
transversalVelocity = -translationalVelocity * meter_per_second;
angularVelocity = 0 * radian_per_second;
myYouBotBase->setBaseVelocity(longitudinalVelocity, transversalVelocity, angularVelocity);
LOG(info) << "drive right";
SLEEP_MILLISEC(2000);
```

```

/* stop base */

longitudinalVelocity = 0 * meter_per_second;

transversalVelocity = 0 * meter_per_second;

angularVelocity = 0 * radian_per_second;

myYouBotBase->setBaseVelocity(longitudinalVelocity, transversalVelocity, angularVelocity);

LOG(info) << "stop base";

} */

```

On this part, we test our robot's base movement, in order to prove if all variables are working properly on Gazebo later.[3.2.4] This means that it will be forward, backwards, left and right with breaks of 2 seconds between them.

```

if (youBotHasArm) {

/* unfold arm

* position were taken empirically using keyboard and KUKA youBot

*/

desiredJointAngle.angle = 2.9624 * radian;

myYouBotManipulator->getArmJoint(1).setData(desiredJointAngle);

SLEEP_MILLISEC(1000)

desiredJointAngle.angle = 2.5988 * radian;

myYouBotManipulator->getArmJoint(2).setData(desiredJointAngle);

desiredJointAngle.angle = -2.4352 * radian;

myYouBotManipulator->getArmJoint(3).setData(desiredJointAngle);

desiredJointAngle.angle = 1.7318 * radian;

myYouBotManipulator->getArmJoint(4).setData(desiredJointAngle);

desiredJointAngle.angle = 2.88 * radian;

myYouBotManipulator->getArmJoint(5).setData(desiredJointAngle);

```



```
/*desiredJointAngle.angle = 3.14 * radian;

myYouBotManipulator->getArmJoint(5).setData(desiredJointAngle);

desiredJointAngle.angle = 1.9526 * radian;

myYouBotManipulator->getArmJoint(2).setData(desiredJointAngle);

desiredJointAngle.angle = -2.0192 * radian;

myYouBotManipulator->getArmJoint(3).setData(desiredJointAngle);

/*desiredJointAngle.angle = 1.73184 * radian;

myYouBotManipulator->getArmJoint(4).setData(desiredJointAngle);

*/LOG(info) << "unfold arm";

SLEEP_MILLISEC(4000);
```

On this section, the meaning of our code is the same as in previous part, but here we test robot's arm, putting the desired joint angle as robot's home position, so it will always start its movement from it every time we compile the code.

```
std::cout << " ----- Co-ordinates of Point Location ----- : " << endl;

if(res.myPCB=0){

/* co-ordinates taken empirically for end positions where we are going to put our PCB */

x_cordinate=-0.1225;

y_cordinate=0.1124;

z_cordinate=0.0239;

std::cout << "PCB in bad conditions! " << endl;

std::cout << "Co-ordinates of end position are : " << endl;

std::cout << " X: " << x_cordinate << endl;

std::cout << " Y: " << y_cordinate << endl;

std::cout << " Z: " << z_cordinate << endl;
```



```

}
if(res.myPCB=1){

x_cordinate=-0.2225;
y_cordinate=0.1124;
z_cordinate=0.0239;

std::cout << "PCB in good conditions! " << endl;

std::cout << "Co-ordinates of end position are : " << endl;
std::cout << " X: " << x_cordinate << endl;
std::cout << " Y: " << y_cordinate << endl;
std::cout << " Z: " << z_cordinate << endl;
}

double px = x_cordinate + px_p;
double py = -y_cordinate + py_p;
double pz = z_cordinate; //0.0432;

```

This part explains how we manage to get Octave's variables from computer vision part, as well as configuring aim points(where PCBs will be left) which were taken empirically.

[3.2.2]

```

////////// THETAS FOR ARM-LINK 1 2 3 //////////

```

```

double l1 = 0.302 - 0.147;

```



```
double l2 = 0.437 - 0.302;
```

```
double l3 = 0.655 - 0.437;
```

```
double xc = sqrt(px*px +py*py);
```

```
double zc = pz;
```

```
double phi_c = 0;
```

```
double d = sqrt (xc*xc + zc*zc);
```

```
std::cout << " d = " << d << std::endl;
```

```
if (d >= 0.500){
```

```
    double theta_link_1 = atan2(zc,xc) ;
```

```
    //break;
```

```
}
```

```
else if (d == 0.508) {
```

```
    double theta_link_1 = atan2(zc,xc) ;
```

```
    //break;
```

```
}
```

```
else if (d > 0.508) {
```

```
    std::cout << " Co-ordinate Points are out of the work space. \n" ;
```

```
    std::cout << "Enter New Co-ordinates Points. \n" ;
```

```
    //break;
```

```
}
```

```
else {
```

```

double xw = xc - l3*cos(phi_c);

double zw = zc - l3*sin(phi_c);

double alpha = atan2 (zw,xw);

double cos_beta = (l1*l1 + l2*l2 -xw*xw -zw*zw)/(2*l1*l2);

double sin_beta = sqrt (abs(1 - (cos_beta*cos_beta)));

double theta_link_2 = 3.1416 - atan2 (sin_beta , cos_beta) ;

double cos_gama = (xw*xw + zw*zw + l1*l1 - l2*l2)/(2*l1*sqrt(xw*xw + zw*zw));

double sin_gama = sqrt (abs (1 - (cos_gama * cos_gama)));

double theta_link_1 = alpha - atan2(sin_gama, cos_gama);

double theta_link_1p = theta_link_1 + 2*atan2 (sin_gama , cos_gama);

double theta_link_2p = - theta_link_2;

double theta_2 = theta_link_1p;

double theta_3 = theta_link_2p;

double theta_4 = (theta_2 + theta_3);

std::cout << " Base Angle " << theta_base << std::endl;

std::cout << " Link-1 Angle " << theta_2 << std::endl;

std::cout << " Link-2 Angle " << theta_3 << std::endl;

std::cout << " Link-3 Angle " << theta_4 << std::endl;

```

Here, we put our equations solutions for inverse and forward kinematics translated to C++ code, so the robot will configure its joints to reach our desired positions.

```

desiredJointAngle.angle = theta_base * radian;

myYouBotManipulator->getArmJoint(1).setData(desiredJointAngle);

```

```
//SLEEP_MILLISEC(1000);

desiredJointAngle.angle = (2.5988 - theta_2) * radian;
myYouBotManipulator->getArmJoint(2).setData(desiredJointAngle);

//SLEEP_MILLISEC(1000);

desiredJointAngle.angle = (-2.4352 - theta_3) * radian;
myYouBotManipulator->getArmJoint(3).setData(desiredJointAngle);

//SLEEP_MILLISEC(1000);

desiredJointAngle.angle = (1.7318 + theta_4) * radian;
myYouBotManipulator->getArmJoint(4).setData(desiredJointAngle);

//SLEEP_MILLISEC(1000);

//desiredJointAngle.angle = -3.3760 * radian;

//myYouBotManipulator->getArmJoint(3).setData(desiredJointAngle);

SLEEP_MILLISEC(5000);

desiredJointAngle.angle = (1.7318-0.4 + theta_4) * radian;
myYouBotManipulator->getArmJoint(4).setData(desiredJointAngle);

SLEEP_MILLISEC(1000)
}

}

}

//desiredJointAngle.angle = 0.0 * radian;

//myYouBotManipulator->getArmJoint(1).setData(desiredJointAngle);

/*/* fold arm (approx. home position) using empirically determined values for the positions */
```

```

desiredJointAngle.angle = 0.1 * radian;

myYouBotManipulator->getArmJoint(1).setData(desiredJointAngle);

desiredJointAngle.angle = 0.011 * radian;

myYouBotManipulator->getArmJoint(2).setData(desiredJointAngle);

desiredJointAngle.angle = -0.1 * radian;

myYouBotManipulator->getArmJoint(3).setData(desiredJointAngle);

desiredJointAngle.angle = 0.1 * radian;

myYouBotManipulator->getArmJoint(4).setData(desiredJointAngle);

LOG(info) << "fold arm";

SLEEP_MILLISEC(4000);

}

} catch (std::exception& e) {

std::cout << e.what() << std::endl;

std::cout << "unhandled exception" << std::endl;

}

/* clean up all variables*/

if (myYouBotBase) {

delete myYouBotBase;

myYouBotBase = 0;

}

if (myYouBotManipulator) {

delete myYouBotManipulator;

myYouBotManipulator = 0;

}

```



```
LOG(info) << "Done.";
```

```
return 0;
```

```
}
```

Finally, we set all the variables to 0, in order to start from the same beginning every time we compile this code.

### 3.2.4. Simulation on Gazebo.

First of all, we have to consider that every package related to Gazebo(and ROS as well) should be installed, if not, we have to reinstall them again. To run simulator, we simply put this command on terminal `roslaunch gazebo_ros gazebo`, opening Gazebo as consequence. [5]

In order to add KUKA youBot to Gazebo workspace, we insert this model in the options that simulator give us, as well as a table similar to the one who is going to be used in reality, as it is shown in Figure 45.

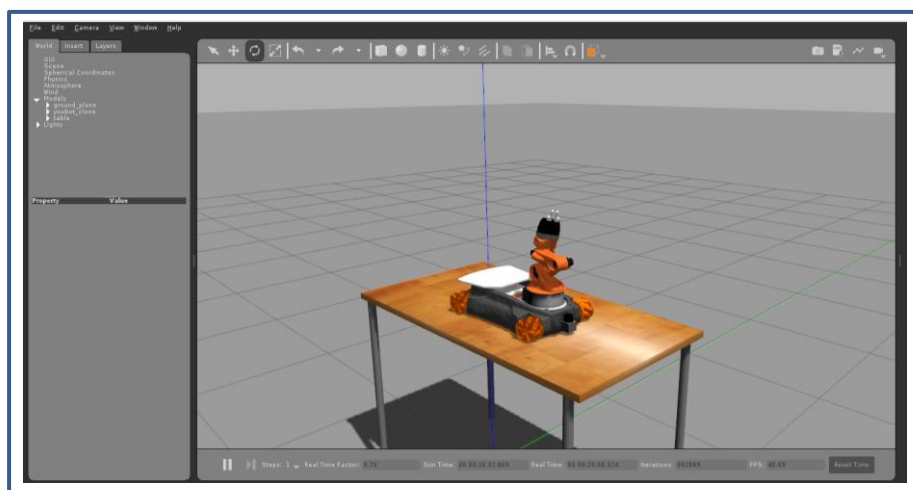


Figure 45. KUKA youBot on Gazebo[3.2.4]

However, we need to try to make simulation model as accurate as possible in relation to real workspace. Due to this, a camera and cube of 5 cm Gazebo objects are inserted to it, basically to correspond in a better way to Siemens MV440 camera and PCBs. As we can see in attached , PCBs size was measured in order to select an object as similar as possible in Gazebo. This is shown in Figure 46.



Figure 46. Measurement of PCBs[3.2.4]

In following Figure 47, we can see that camera has no gravity because it is more comfortable to put it this way than constructing a platform in simulation. Otherwise, if we didn't turn off gravity option, camera would fall down.

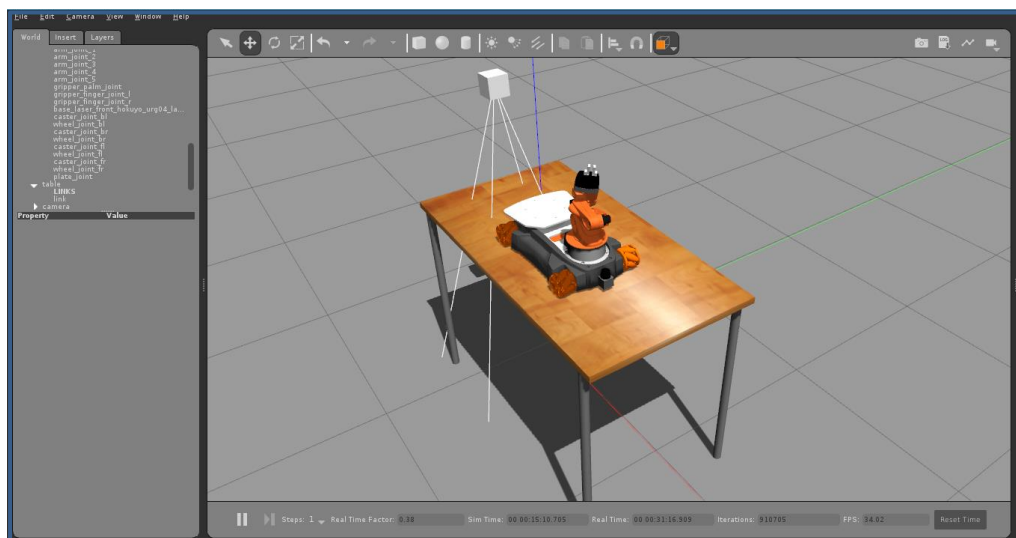


Figure 47. Workspace constructed in Gazebo[3.2.4]



Once we have our workspace built in the simulator, we have to compile our code developed on C++ for our robot and see how it responds. Before doing this, it should be pointed out that simulation doesn't correspond to reality exactly, as we saw in previous paragraphs, where KUKA youBot model provided by Gazebo has both omnidirectional base and arm, despite of our real robot only owns the last one.

Next step is to launch our code developed on C++ with the command `roslaunch mypackage myrobot.launch`. After this, our robot starts to move in simulator workspace, grabbing the cube and putting it in the left side of the table or in the right one, depending on the result given by image processing part, which needs to be compiled previously.



### 3.3. Computer vision programming.

#### 3.3.1. License Automation Manager from Siemens.

Before taking photos with Siemens camera, we need to install its licenses and software in order to work with it properly. [4] Due to this, working with Siemens MV440 camera on Linux is not possible, so it was necessary to install every single package in a second PC with Windows support and establish communication between computers afterwards. Furthermore, we have to take into account that network between these two PCs will be via Internet and another Siemens program that we can download in every platform, but this part will be explained in next sections. [3.3.4]

Primarily, camera has to be connected physically to the PC (this means both power supply and Ethernet wire) before installing the software, mainly because IP camera configuration has to be done to communicate camera and computer on PRONETA (another Siemens software included in installation CD), as we can see in attached Figure 48.

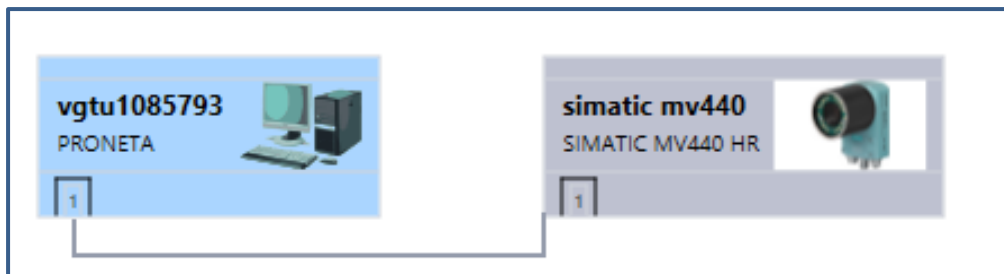


Figure 48. PC and camera IP connection[3.3.1]

After this part, license needs to be installed with License Automation Manager, keeping in mind previously that IP address put on PRONETA must be the same for this program as well. Next step is installing user interface on our computer, with Internet connection needed, as we are going to use web browser, following attached Figure 49.

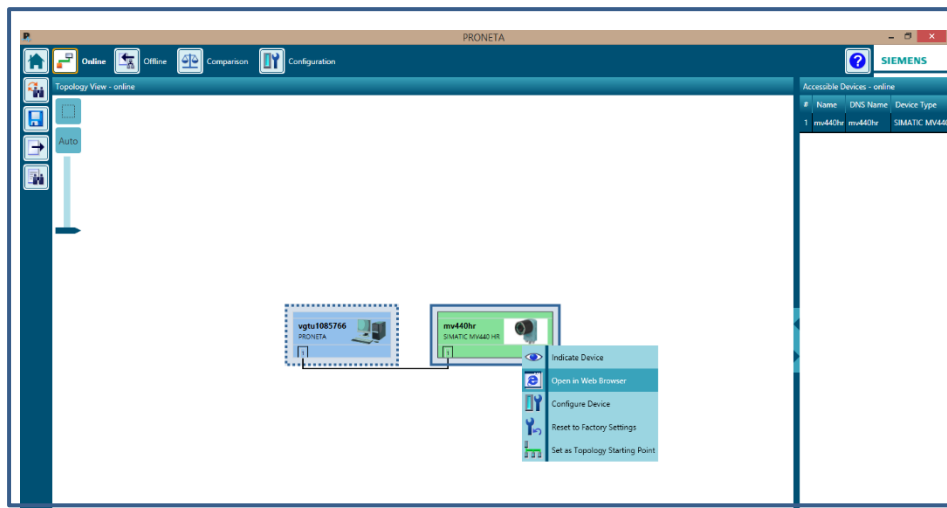


Figure 49. Opening web browser in PRONETA for user interface[3.3.1]

Once we open user program, communication between our two PCs can be established, as in computer which software is Linux, we can put IP address of the reader in a web browser and work with both of them at the same time. An example of configuration on Internet Explorer is attached in Figure 50.

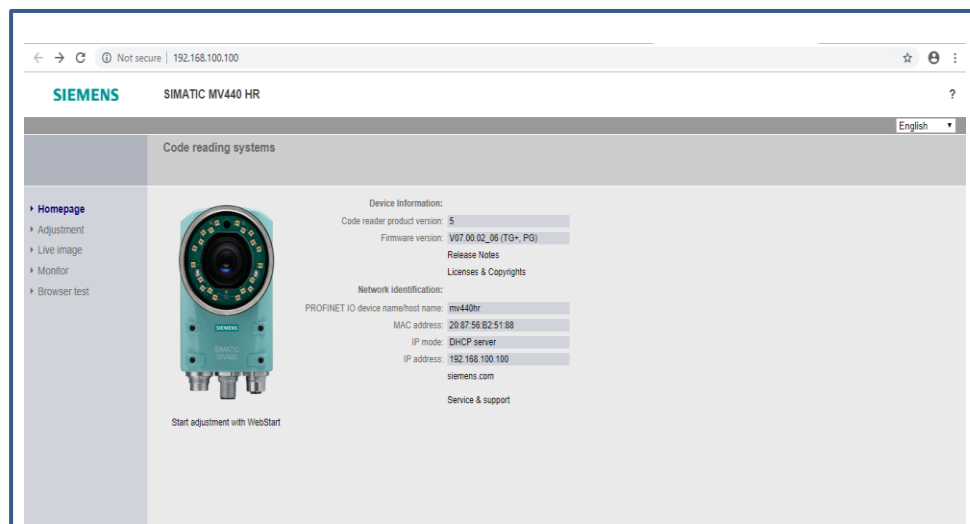


Figure 50. Code reading configuration on web browser[3.3.1]

Next step is to adjust camera settings according to user's will, as its calibration is needed before saving all the pictures for image processing. Tough this part will be explained in next section. [3.3.2]

### 3.3.2. Image acquisition mode for Siemens MV440 camera.

Before putting our pictures taken by the camera on image processing part, it is necessary to configure our reader and trigger mode. Three types of image acquisition can be distinguished, but we are only going to use one of them. This is going to be the triggered mode, where camera takes photos only when we press the refresh button. Therefore, user has to be focused on pulling the trigger every time PCBs are changed for their recognition.

Another reason to use this mode is that we can control the brightness of each picture between each camera shot, so error detection will be easier afterwards. [4]

Finally, we can also adjust camera's flash in order to take better quality photos with more or less resolution. The only part which cannot be configured in real time is image color, as it always be binarize picture, so we need to transform it later when saving to a directory. Before transferring pictures to Octave directory, user has to select which part of the photo wants to be processed. An example of this proceeding is enclosed in Figure 51.

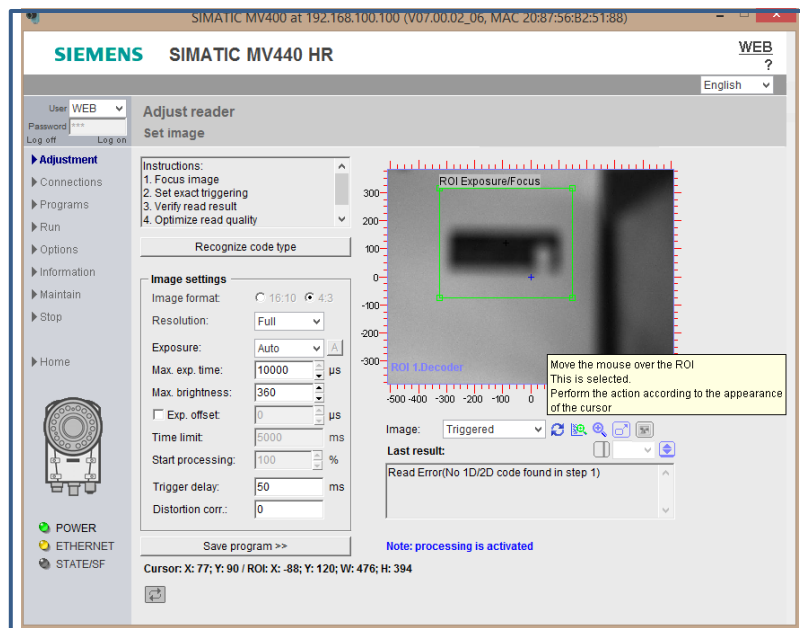


Figure 51. SIMATIC MV440 HR user program[3.3.2]

### 3.3.3. PCBs error detection programming.

In this section we are going to explain each part of code belonging to functions used for error detection programming, adding representative pictures of it as well. However, first it is needed to point out that every photo of PCBs has a white background. This is due to restriction between elements and background is easier when the last one is white in terms of binarizing, filtering, and so on, but the main reason is that pixels after our transformation can only be black or white, so it makes detection more user-friendly, as Figure 52 shows. [11] Also, we should consider illumination, which takes an important part on this section, as if there is too much light or too little, it can end up in recognition mistakes.

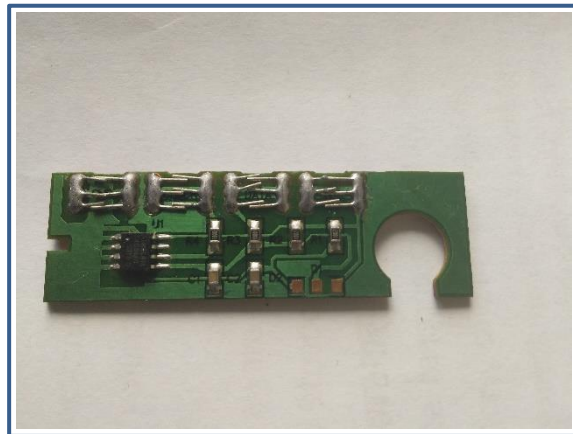


Figure 52. PCB with white background for detection[3.3.3]

#### 3.3.3.1. Filtering function

The main purpose of this function is to filter one of our images taken by camera to eliminate possible noises from them. Before binarizing the image to apply gauss filters, it is necessary to check if it is in color frame range or black and white one. Thus, this process can be applied in an optimum way without mistakes.

```
%Obtain image edge from subtract of two gauss filters
if ndims(I) == 3 %Checking if image is in black and white or not
disp('Image in color')
I=rgb2gray(I);
```

end

After that, we build a zeros matrix where we will add our image transformed. Two matrixes of filtering are built for being applied to the original picture, considering both axes X and Y.

```
[rows, columns]=size(Ig);  
Ig=double(Ig); %passing to double format  
I_contour=zeros(rows, columns);  
Sx=[-1 0 1;-2 0 2;-1 0 1];  
Sy=[-1 -2 -1;0 0 0; 1 2 1];
```

Then, our image is retouched with `imfilter` and the matrixes created previously. Next step is transforming this matrix from mathematics view to gray scale with `mat2gray`, to plot our images with and without filtering finally. [9]

```
Ix=imfilter(Ig, Sx);  
Iy=imfilter(Ig, Sy);  
I_contour=abs(Ix)+abs(Iy);  
S=mat2gray(I_contour);  
figure  
imshow(I), title('original image')  
figure  
imshow(S), title('contour image with mat2gray')
```

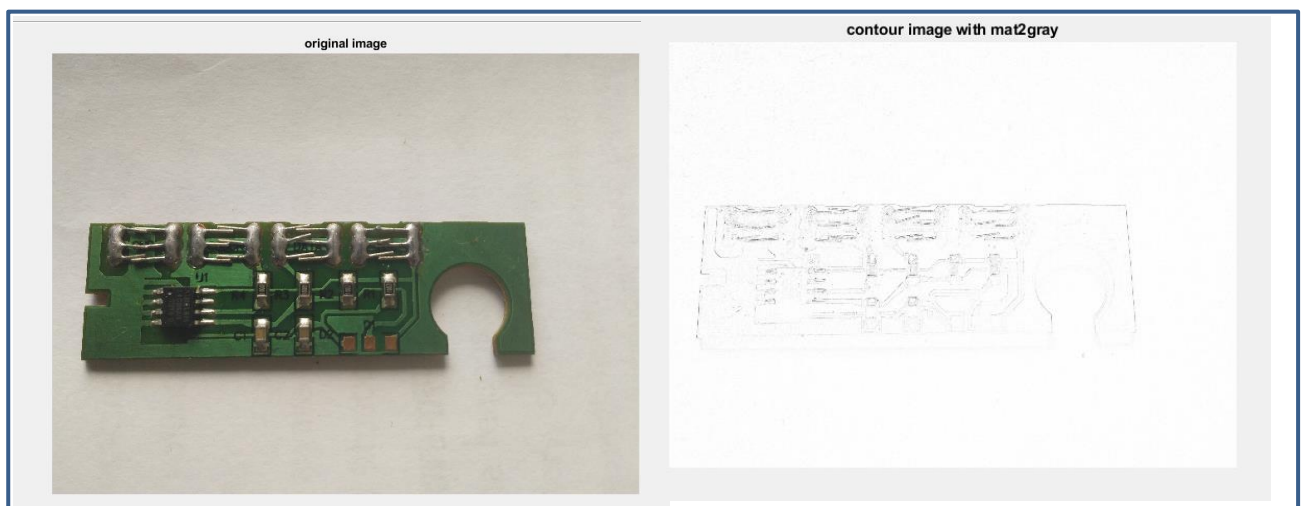


Figure 53. Original and filtered image[3.3.3.1]



As we can see in Figure 53, the contour of image obtained with `mat2gray` command can be distinguished from the background, just as we wanted at the beginning. Both pads and elements included in the PCB taken as an example are seen without any problem, so it is a good signal for later treatments.

### 3.3.3.2. Morphology

This function is the responsible of detecting different elements of pictures taken by Siemens camera, with the purpose of concluding if PCBs are in a good condition or not.

```
function [T]=mymorphology(I,A)

if(islogical(I))
disp('Image is already binarized, so it will not be modified')
T=I;
return
end
repeat=1;
while(repeat==1)

umbral=graythresh(I);
BW=im2bw(I,umbral);
BW2=im2bw(A,umbral);
BW2inv=~BW2;
BWinv=~BW;
radius=input('Introduce radius of structuring element between 1-5: ');
ee=strel('disk',radius);
% We apply dilation for splitting elements and detecting them
B=imerode(BWinv,ee);
B2=imerode(BW2inv,ee);
```

```
L=~B;  
L2=~B2;
```

Subsequently, it is needed to make a loop where users can apply a dilation according to their taste, as if we select the same structuring element in order to do it in an automated way, it can result in a mistake due to camera's illumination between shot and shot. We apply dilation to both images in order to delete elements that are not of our interest.

```
figure  
imshow(I),title('original image');  
figure  
imshow(A),title('testing image');  
figure  
subplot(1,4,1),imshow(BWinv), title('binarized original image')  
subplot(1,4,2),imshow(L), title('original image with dilation')  
subplot(1,4,3),imshow(BW2inv), title('binarized test image')  
subplot(1,4,4),imshow(L2), title('test image with dilation')
```

Next step is plotting all the images used and transformed by this function, as we can see in Figure 54, Figure 55 and Figure 56.

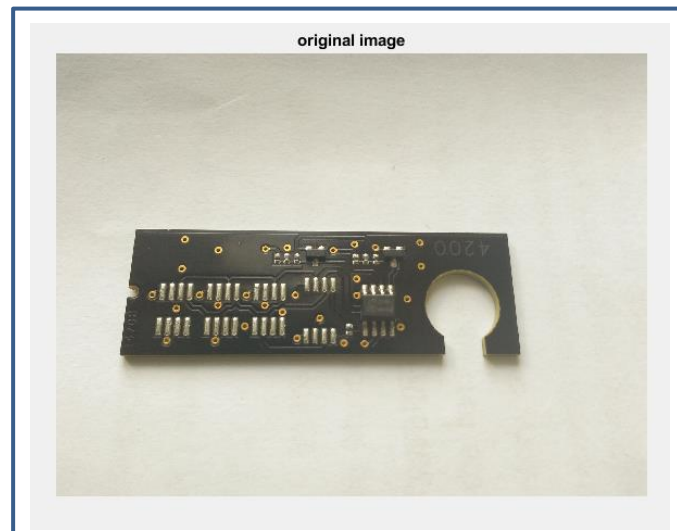


Figure 54. Standard image for comparing elements in mymorphology function[3.3.2]

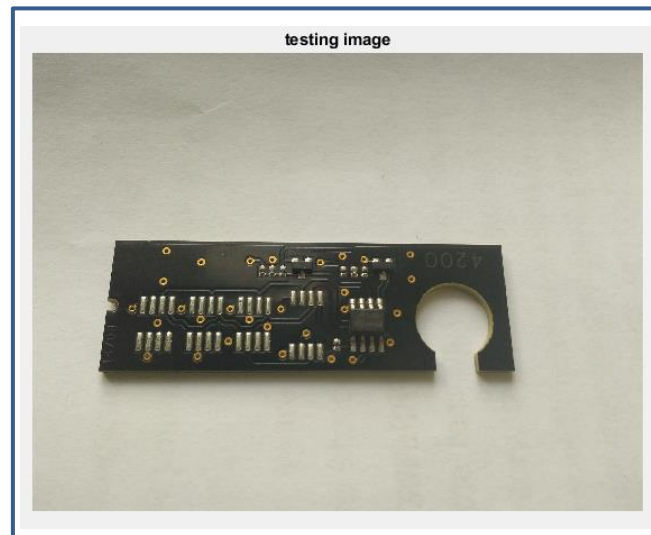


Figure 55. Testing image in mymorphology function[3.3.2]

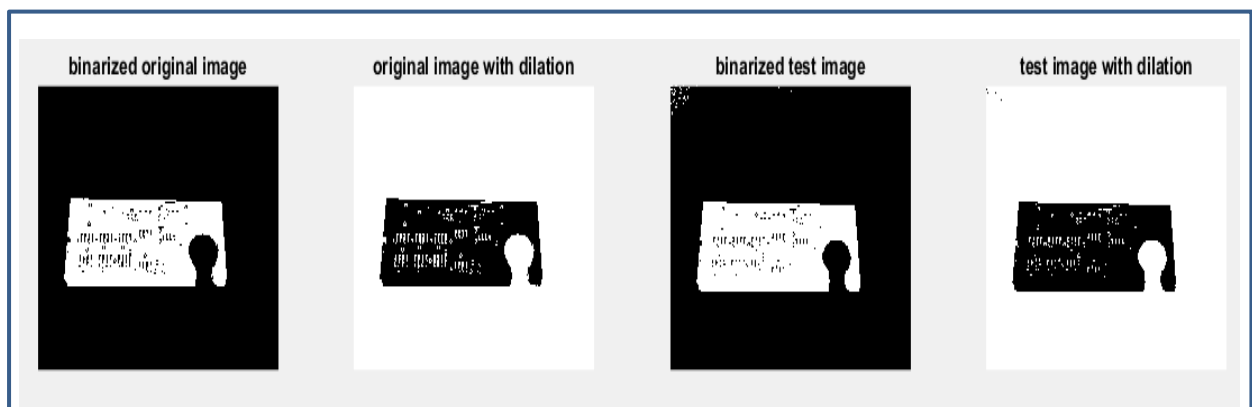


Figure 56. Both images transformed and compared in same plot for mymorphology function[3.3.2]

```

% We see split objects after transformations
cc=bwconncomp(B); % creating two data structures for knowing n°
elements
a=cc.NumObjects
ccs=bwconncomp(B2);
as=cc.NumObjects

if (a~=as)

```



```

disp('Number of elements are really different. Second image has
mistakes');

myPCB=0; %% we create a variable called myPCB which value is 0
when it has mistakes and 1 when it is in good conditions
else

disp('Number of elements are similar in both images. So testing
image is correct');

myPCB=1; %% this variable will be shared with ROS

end

fprintf('\n Is obtained result the one you wanted?');
fprintf('\n t1. Yes (press 0)');
fprintf('\n t2. No (press 1)');
fprintf('\n');
repeat=input(' '); %Asking user about results

end

```

```

>> I=imread('imag5.jpg');
>> A=imread('imag6.jpg');
>> myhomography(I,A)
Introduce radium of structuring element between 1-5: 2

radius =

     2

a =

    54

as =

    54

Number of elements are similar in both images. So testing image is correct

Is obtained result the one you wanted?
t1. Yes (press 0)
t2. No (press 1)
0

```

Figure 57. Result obtained after applying mymorphology function[3.3.2]

Thanks to command `bwconncomp`, we create a data structure with the purpose of knowing number of elements in each PCB and compare them to each other. If the number of

elements in both images are the same, it means that second picture, the one relative to PCB to test, can be considered of good quality. In this example, both pictures taken are from similar PCBs which are in same condition, so our program concludes that there are no mistakes. Finally, we asked users if they are satisfied with the result, as shown in Figure 57.

### 3.3.3.3. Matching

This function is going to be used for detecting elements in a graphical way, so users can see the differences and similarities marked on each image. First of all, as always, it is needed to test if pictures are in gray scale or color.

```
function matching (I,A)

if ndims(I) == 3
disp( 'Image in color')
N=rgb2gray(I);
L=rgb2gray(A);
else
disp( 'Image in B&W')
N=I;
L=A;
end
```

Next step is to select a part of our standard image for comparing it with the second one. It is done thanks to `imcrop` command, where user can select a specific part of a picture and take it as a `jig`. An example of this function usage is shown in Figure 58. [9]

```
jig=imcrop(N);%select jig
```

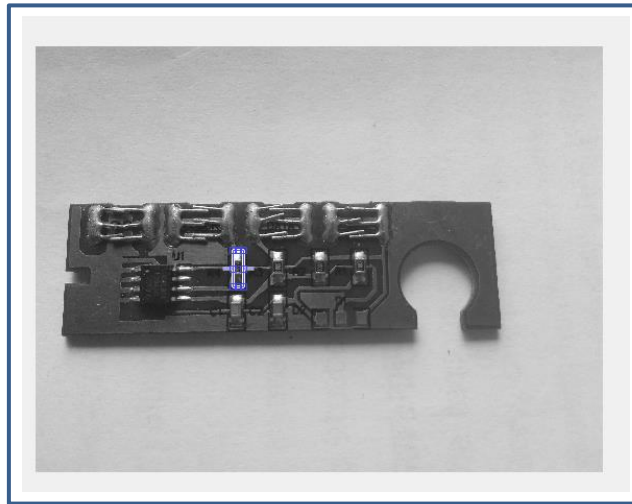


Figure 58. Template selected in standard image[3.3.3.3]

```
[x,y]=size(jig);
C=normxcorr2(jig,L);
B=C>0.9; %Only select points that have correlation bigger than 0.9
cc=bwconncomp(B);
s=regionprops(cc,'Centroid');
num_el=length(s) %number of elements in common
figure
imshow(N);
figure
imshow(L);
```

Once we selected our jig, we use `normxcorr2` command in order to search for correlation between our jig and testing image.[11] In the next line, it is specified that this correlation coefficient between images has to be 0.9 or bigger, meaning that we are only going to select parts that are really similar between each other. Furthermore, a data structure is created to know the number of elements and also to search for their centroids. Finally, thanks to a for loop, we are going to plot red asterisks in all elements that fulfill task mentioned before.

```
hold on
for k=1: num_el
plot((s(k).Centroid(1))-(x)/2, (s(k).Centroid(2))-(y)/2, '*r'); %draw
asterisks in elements in common between images
```

```
end  
hold off  
end
```

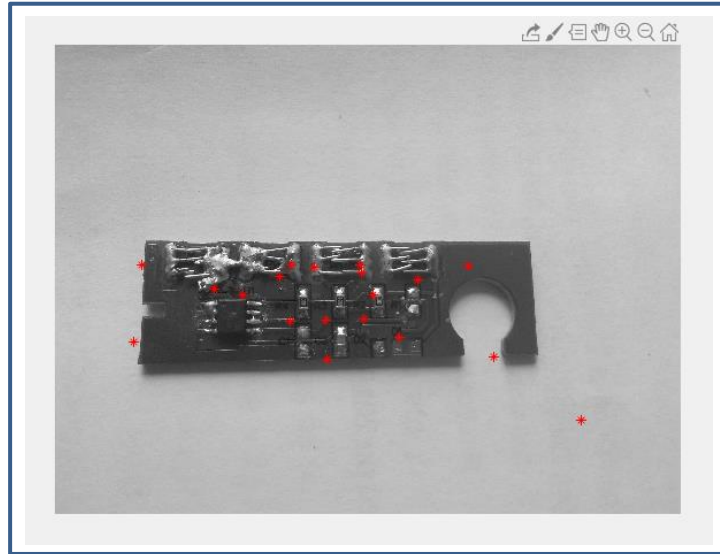


Figure 59. Plotting elements in common between two images with template matching[3.3.3.3]

In a nutshell, template matching is a useful tool to detect elements in common between images. However, it is really inconsistent as we can see in Figure 59, due to some asterisks plotted have no relation with our standard image. This is because template matching is really sensitive to illumination changing and any minimum variation in testing photo can end up in detection mistakes.

### 3.3.4. Communication between camera and ROS

In previous part, pictures were processed on Octave in the second PC, where we were taking images from the camera. However, we need to establish communication between our pictures sent to the PC working on ROS and this system. In order to accomplish that, we are going to use the command `rosock`, which allows to share information with the entire ROS network. Furthermore, it involves a lot of commands which can be related to our C++ program for KUKA youBot. Some examples of that are attached in Figure 60.

- **rosoct(cmd)** - initializes the run-time, must be called before anything else. Possible commands are:
- **shutdown** - disconnect from the ROS network and terminate
- **clear** - clear all resources and topics advertised
- **nohook** - launch rosoct without registering a hook to automatically call **rosoct\_worker**
- **rosoct\_X** - public API calls that wrap the low-level calls and make using Octave simpler
- `success = rosoct_advertise(servname,@msg,queuesize)`
- `success = rosoct_advertise_service(servname,@msg,@callback)`
- `[sessionid, response] = rosoct_create_session(sessionname,request)`
- `success = rosoct_publish(topicname, message)`
- `response = rosoct_service_call(servicename, request)`
- `response = rosoct_session_call(sessionid, servicename, req)`
- `success = rosoct_subscribe(topicname,@msg,@callback,queuesize)`
- **rosoct\_X** - low-level API calls. Some functions are wrapped directly by the public API so are now specified here.
- `paramvalue = rosoct_get_param(paramname)`
- `topics = rosoct_get_topics(type)` - type can be one of 'advertised' or 'published'
- `success = rosoct_msg_unsubscribe(topicname)`
- `rosoct_set_param(paramname,paramvalue)`
- `success = rosoct_terminate_session(sessionid)`
- `success = rosoct_unadvertise(topicname)`
- `success = rosoct_unadvertise_service(servicename)`
- `success = rosoct_wait_for_service(servicename)`
- `numprocessed = rosoct_worker()` - worker function that handles all the pending service requests and message callbacks.

Figure 60. Rosoct commands[3.3.4]

One of the best things that Octave provides us on ROS is that is not mandatory to rewrite its code in C++ or Python. In our case, for communicating our image processing functions with ROS and images saved in ROS directory, we are going to use ImageImageService, which is a service that takes an Image message and gives another one.

```
rosoct_add_msgs('image_msgs');

% rosoct_add_srvs('myimages') %add the package where ImageImageService
resides

sus=rosoct_advertise_service('mymorphology',@ImageImageService,@mymorp
hology)
```

After that, we need to rewrite our computer vision functions, just as the following:

```
function res=matching (req1,req2) % res means response, as req1 and req2
mean request for images

res=req.create_response_();
```



```
if ndims(req1) == 3
disp( 'Image in color')
N=rgb2gray(req1);
L=rgb2gray(req2);

else
disp( 'Image in B&W')
N=req1;
L=req2;
end

jig=imcrop(N);%select jig
[x,y]=size(jig);
C=normxcorr2(jig,L);
B=C>0.9; %Only select points that have correlation bigger than 0.9
cc=bwconncomp(B);
s=regionprops(cc, 'Centroid');
num_el=length(s) %number of elements in common
figure
imshow(N);
figure
res.data=imshow(L); %data that it is going to be shared with ROS
hold on
for k=1: num_el
plot((s(k).Centroid(1))-(x)/2, (s(k).Centroid(2))-(y)/2, '*r'); %draw
asterisks in elements in common between images
end
hold off
end
```

Finally, we also added an error message in each function for the request of images, just in case ROS is not responding. [16]

```
if (strcmp(req.str,'dofail')) %failure condition
    req= [];
else
    res.str= [req.str '_a'];
end
```

## 4. RESULTS AND CONCLUSIONS

### 4.1. Computer vision part results

#### Filtering function

As it concerns to filtering function, we can conclude that it works in a proper way, as we can see in Figure 61, and other examples that will be introduced in project presentation afterwards. In all images introduced to this function, their contour can be distinguished without any problem, as well as their elements such as capacitors, resistors, microprocessors and so on.

To sum up, filtering function is a good way for detecting contour of each element from our PCBs in the first view, so it can help users to see if they already have errors before applying other methods.

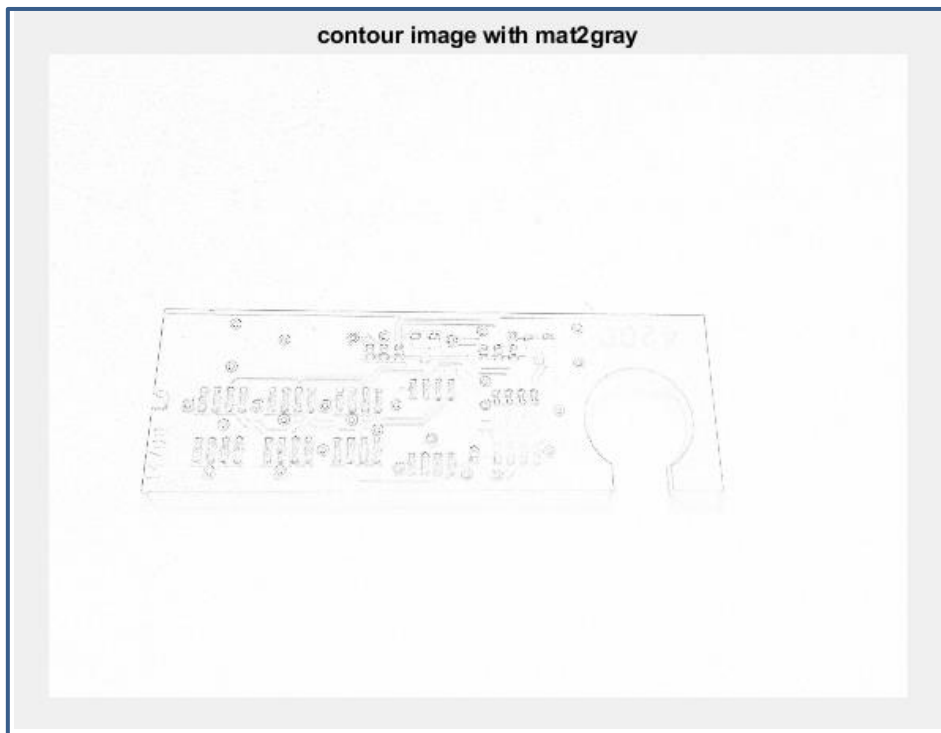


Figure 61. Contour image with filtering function[4.1]



## Mymorphology function

As a result of the mymorphology function, it is possible to detect mistakes and elements missing in testing images, as shown in following pictures. We can value at a glance that testing image, corresponding to Figure 62, has too much welding and also some pads and elements are missing, so this mistakes recognition is going to be easy to make in terms of programming.

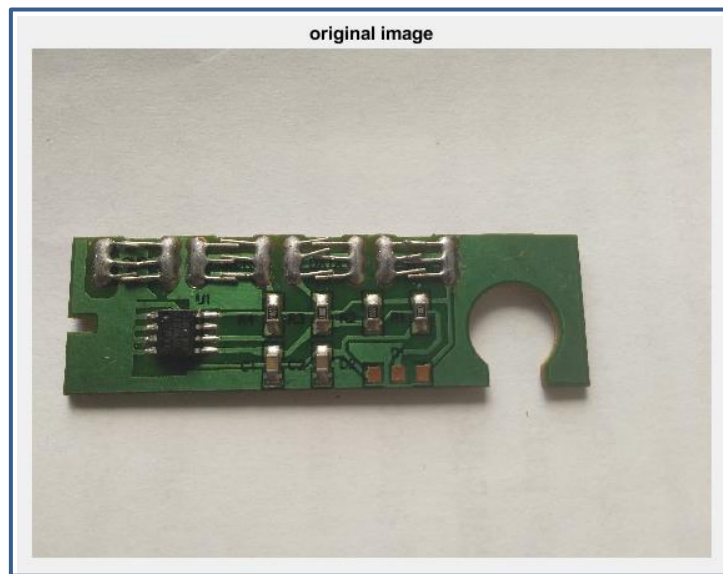


Figure 62. Standard image number 2 for mymorphology function[4.1]

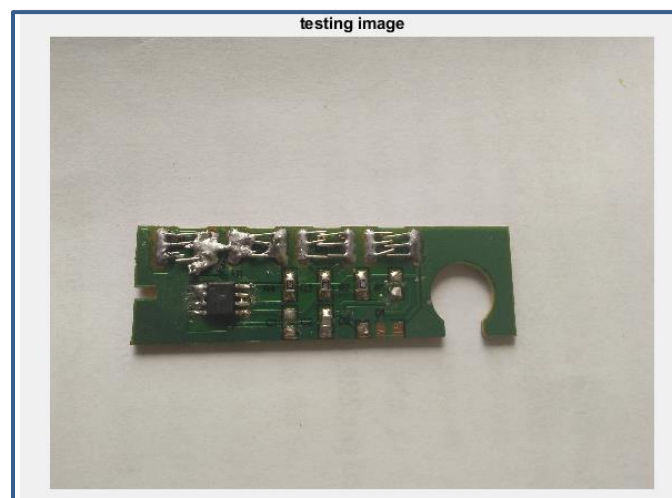


Figure 63. Testing image number 2 for mymorphology function[4.1]

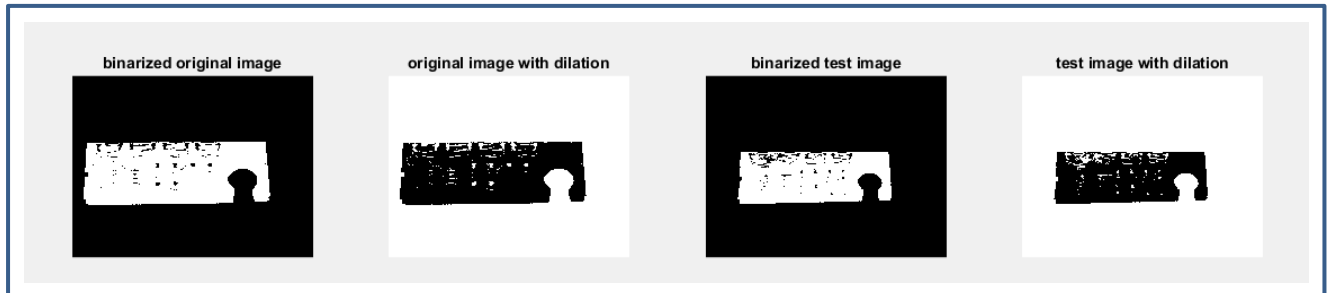


Figure 64. Images after applying dilation[4.1]

In above Figure 64, it can be seen that after binarizing and applying dilation to both images, the difference between each image is remarkable, as later we can check in commands line that number of elements are different, so our function takes the conclusion that second image is not right. This is shown in Figure 65.

```
>> myhomography(S,O)
Introduce radius of structuring element between 1-5: 1

radius =

    1

a =

    209

as =

    199

Number of elements are really different. Second image has mistakes

Is obtained result the one you wanted?
t1. Yes (press 0)
t2. No (press 1)
```

Figure 65. Mymorphology function applied to second couple of images. [4.1]

In short, this function is the most useful one for our project, as it always detects mistakes in testing PCBs with accuracy. Also, it is the responsible of sending to ROS if our testing PCB is okay or not, so the robot will put it in one side or other, depending on the results obtained.

### Matching function

As for matching function, we can say, without any doubt, that is the least helpful of all, as it is very inconsistent depending on photos illumination, ending up in detection

mistakes. Here we have an example of average recognition of elements in Figure 66 and Figure 67.

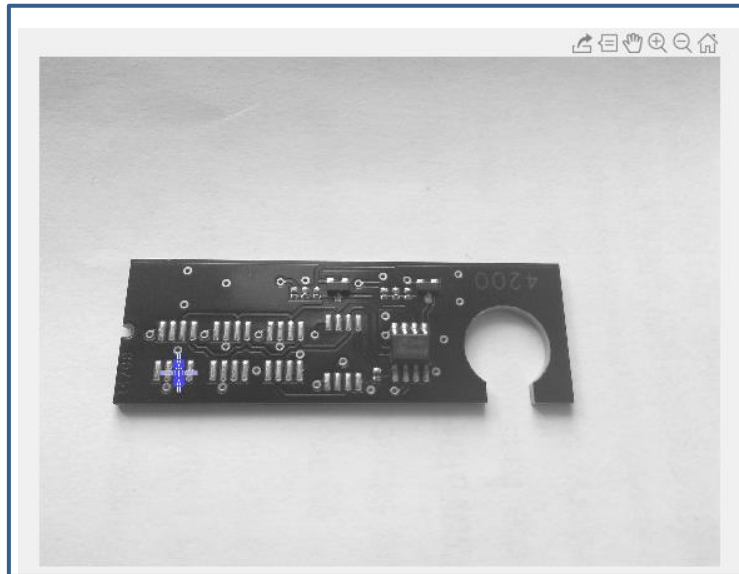


Figure 66. Jig for second standard image in matching function[4.1]

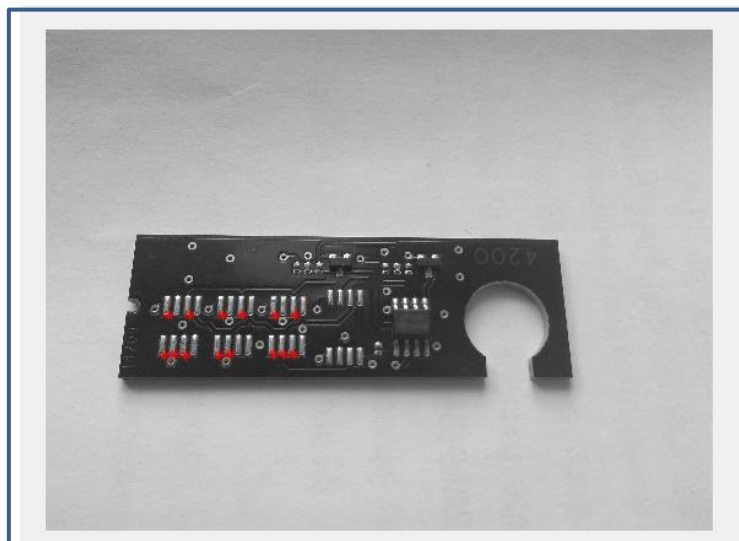


Figure 67. Template matching applied on second testing image[4.1]

## 4.2. KUKA youBot part results

Our first part for robot results is its D-H parameters, as well as kinematic equations obtained by our code on MATLAB and C++ mentioned in previous sections. [3.2.2]

After that, we can see that the robot is starting to move properly, trying to reach PCB position which is shown in Figure 68, but unable to grab it, as we can see in Figure 69 .



Figure 68. Robot's movement from home position

One of the most probable reasons why I didn't work was due to Gazebo simulator doesn't have accurate torques for each arm's joint of KUKA youBot, so it is working on it, but not in the real workspace. Another one can be that, because of this robot normally has its base included, we should add an offset that is not existent in our working model, so it could reach its main goal. Finally, the most probable one is that an universal gripper should be included to grip small objects like the ones we are using in this project.



Figure 69. KUKA youBot trying to reach PCB

### 4.3. Final conclusions

From computer vision standpoint, we can say that most of our functions are working properly. However, they can be improved as a first approach, mainly for making them in a more automated way. On the other hand, Octave is a good tool for image processing, but it is quite slow, as it doesn't work in real time (same for MATLAB) like other IDEs do, such as Python+OpenCV. So a second approach could be programming all this part in this software instead of Octave. In terms of our camera, the Siemens MV440 one, my opinion is that we could use a normal camera instead of this one. Basically, this is due to the difference is not that notable and licenses problems for working on Linux too.

From robotics forward and inverse kinematics perspective, we can conclude that MATLAB provides a really good toolbox for programming and testing our equations, taking different values and parameters as a result. Thanks to it, we can program our code on C++ for our robot later, following these equations and parameters obtained on it.

From robotics programming point of view, our KUKA youBot works perfectly in simulator environment. Nevertheless, when we try to apply our code in real robot, it doesn't move as we want in first touch, mainly because it is not able to grip the PCBs correctly. The conclusion obtained from this issue, due to differences between real robot and simulator one (it has base with wheels, real one does not) and also because of accuracy problems with the gripper, is that an universal gripper is needed to catch every type of object, including small ones like PCBs, and that a new model of KUKA youBot arm should be added on Gazebo in order to fix the offset between these two configurations. Overall, I think this robot may be more useful in terms of working with it on a Windows developed system, like LabView Robotics or something similar. Despite of being a robot that can be programmed on a Linux environment, where all the licenses are free, it implies a lot of problems, especially on calibration and drivers installation, compared to other robots like ABB where you can use their own simulator (Robot Studio) and they are all configured from scratch.

## 5. ACKNOWLEDGEMENTS

From my point of view, working on this project was a great experience, which has developed my knowledges in robotics and programming. I have learnt how ROS works and how to use different libraries already developed for creating my own algorithm. Furthermore, I have learned a new toolbox on MATLAB, such as the robotics one, as well as improving my computer vision programming on this IDE.

This thesis cannot be achieved without my teacher's help, Prof. Dr. Dainius Udrys, who gave me the opportunity to work with him and all his advices and recommendations.

I would like to thank VGTU in general and its Department of Mechatronics and Robotics and head of the department Professor Vytautas Bučinskas personally for providing the equipment and support.

Also, thanks to ESN VGTU, from top to bottom, which has made me have a really good concept of Lithuania overall, way better than it would be without it. Special mention to my mentor, Živilė Latakaitė, who has helped me on adapting to Vilnius and making me feel like home, as well as organizing a lot of events with more Erasmus people, making our friendships even stronger. Now she is not a mentor anymore, but a friend instead. It has been a bless meeting her and all my new friends here in Lithuania.

Finally, thanks to my parents, for giving me all their confidence and support from Spain. Regardless of all the kilometers that separate us, they have always been there when I needed them the most, both in good and bad moments.

## 6. REFERENCES

1. *Short circuit: The lifecycle of our electronic gadgets and the True Cost to Earth*. Author: Philippe Sibaud. Retrieved 2013, from <http://therightsofnature.org/short-circuit-report-the-true-cost-of-our-electronic-gadgets/>
2. *ROS Programming: Building Powerful Robots: Design, build and simulate complex robots*. Authors: Anil Mahtani, Luis Sanchez, Enrique Fernandez, Aaron Martinez, Lentin Joseph. Retrieved 2018, from [www.PacktPub.com](http://www.PacktPub.com)
3. *A Systematic Approach to Learning Robot Programming with ROS*. Author: Wyatt S. Newman. Retrieved 2018, from <http://www.taylorandfrancis.com>.
4. *SIMATIC Ident for MV420/MV440 Code Reader Systems. Operating systems*. Retrieved 2012.
5. *KUKA youBot User Manual*. Retrieved 2012, from [KUKA.com](http://KUKA.com).
6. *A novel kinematics for a 5 DOF manipulator based on KUKA youBot*. Authors: Allan D. Zhang, Xiao Xiao, Yangmin Li. Retrieved 2015, from <https://www.researchgate.net/publication/300416616>.
7. *Robotics toolbox for Matlab*. Author: Peter Corke. Retrieved 2015, from <https://petercorke.com/wordpress/toolboxes/robotics-toolbox>.
8. *Parámetros Denavit-Hartenberg. Cinemática Directa de un Robot Manipulador*. Authors: Alberto Herreros, Juan Carlos Fraile Marinero. Retrieved 2017.
9. *Digital Image Processing in Matlab*. Authors: Rafael C. González, Richard E. Woods, Steve L. Eddins. Retrieved 2011.
10. *Image Processing in Octave*. Author: John W. Eaton. Retrieved 2018, from <https://octave.org/doc/v4.4.0/index.html#Top>.
11. *Visión Artificial Industrial. Procesamiento de imágenes para inspección automática y robotica*. Authors: Eusebio de la Fuente López, Félix Miguel Trespaderne, from Valladolid University. Retrieved 2012.
12. *Simulación, control cinemático y dinámico de robots comerciales usando la herramienta de MATLAB, Robotics Toolbox*. Author: Miguel Ángel Mato San José. Retrieved 2014 from Valladolid University.
13. *Research of robot trajectory control by optical method*. Author: Alejandro Perez San Martín. Retrieved 2017 from



- [https://upcommons.upc.edu/bitstream/handle/2117/114061/Alejandr\\_o\\_Perez\\_Sanmartin.pdf?sequence=1&isAllowed=y](https://upcommons.upc.edu/bitstream/handle/2117/114061/Alejandr_o_Perez_Sanmartin.pdf?sequence=1&isAllowed=y).
14. ROS youBot Inverse Kinematics, from <https://github.com/arifanjum/ROS-YouBot-Inverse-Kinematics>.
  15. ROS tutorials, from [http://wiki.ros.org/ROS/Tutorials#Beginner\\_Level](http://wiki.ros.org/ROS/Tutorials#Beginner_Level)
  16. ROS and Octave tutorial, from <http://wiki.ros.org/rosoct>
  17. ROS-Youbot-Inverse-Kinematics, from <https://github.com/arifanjum/ROS-YouBot-Inverse-Kinematics>
  18. Inverse Kinematics theory, University of Rome, from [http://www.diag.uniroma1.it/~deluca/rob1\\_en/10\\_InverseKinematics.pdf](http://www.diag.uniroma1.it/~deluca/rob1_en/10_InverseKinematics.pdf)
  19. KUKA youBot motor joints, from [https://www.maxonmotor.com/medias/sys\\_master/root/8815566520350/2014-10-story-enUS-kuka-research-robot.pdf?attachment=true](https://www.maxonmotor.com/medias/sys_master/root/8815566520350/2014-10-story-enUS-kuka-research-robot.pdf?attachment=true)
  20. Derivation of Kinematic Equations for KUKA youBot and implementation on Simulink, . From [https://www.academia.edu/23871359/Derivation\\_of\\_Kinematic\\_Equations\\_for\\_the\\_KUKA\\_youBot\\_and\\_implementation\\_of\\_those\\_in\\_Simulink](https://www.academia.edu/23871359/Derivation_of_Kinematic_Equations_for_the_KUKA_youBot_and_implementation_of_those_in_Simulink)

## 7. ANNEXES

### 7.1. ROS C++ CODE

#### 1. Main.cpp

```

/*****
/*****
/*          myrobot program          */
/*          */
/* Author: Alberto Justo Lobato      */
/*          */
/* Bachelor's Graduation Thesis: E-waste material classifier with */
/* KUKA youBot and computer vision  */
/*          */
/* Vilnius Gediminas Technical University, Lithuania */
/*          */
/* Explanation: This code includes all the parts necessary for */
/* moving our robot after image processing. We are using inverse */
/* kinematics, as end positions are known and our goal is to */
/* move our robot to them and find angles and joints positions and */
/* speeds. In terms of compiling, we will launch this program */
/* on Gazebo first and then on the real robot, using the following */
/* command roslaunch myrobot myrobot.launch */
/*          */
/*****
/*****

#include "youbot/YouBotBase.hpp"
#include "youbot/YouBotManipulator.hpp"
#include <math.h>
#include <studio.h>
#include <ImageService.h>

using namespace youbot;

int main() {

```

```

double px=0,pz=0,py=0;
double theta_base=0, theta_2=0, theta_3=0, theta_4=0;
double theta_link_1=0 , theta_link_2=0;
double theta_link_1p=0, theta_link_2p=0;

/* configuration flags for different system configuration (f.e. arm without base)*/
bool youBotHasBase = false;
bool youBotHasArm = false;

/* define speeds */
double translationalVelocity = 0.05; //meter_per_second
double rotationalVelocity = 0.2; //radian_per_second

/* pointers for youBot base and manipulator */
YouBotBase* myYouBotBase = 0;
YouBotManipulator* myYouBotManipulator = 0;

/* obtain response from Octave */
    res=req.create_response_();

try {
myYouBotBase = new YouBotBase("youbot-base", YOUBOT_CONFIGURATIONS_DIR);
myYouBotBase->doJointCommutation();

youBotHasBase = true;
} catch (std::exception& e) {
LOG(warning) << e.what();
youBotHasBase = false;
}

try {
myYouBotManipulator = new YouBotManipulator("youbot-manipulator",
YOUBOT_CONFIGURATIONS_DIR);
myYouBotManipulator->doJointCommutation();
myYouBotManipulator->calibrateManipulator();
// calibrate the reference position of the gripper
//myYouBotManipulator->calibrateGripper();

```



```
youBotHasArm = true;
} catch (std::exception& e) {
LOG(warning) << e.what();
youBotHasArm = false;
}

/*
 * Variable for the base.
 * Users have to put a value and a unit.
 */
quantity<si::velocity> longitudinalVelocity = 0 * meter_per_second;
quantity<si::velocity> transversalVelocity = 0 * meter_per_second;
quantity<si::angular_velocity> angularVelocity = 0 * radian_per_second;

/* Variable for the arm. */
JointAngleSetpoint desiredJointAngle;
//GripperBarSpacingSetPoint gripperSetPoint;

try {
/*
 * Simple sequence of commands to the youBot:
 */

/*if (youBotHasBase) {

/* move forward */
/*longitudinalVelocity = translationalVelocity * meter_per_second;
transversalVelocity = 0 * meter_per_second;
myYouBotBase->setBaseVelocity(longitudinalVelocity, transversalVelocity, angularVelocity);
LOG(info) << "drive forward";
SLEEP_MILLISEC(2000); /*we always put a little delay between moves*/
/* move backwards */
/*longitudinalVelocity = -translationalVelocity * meter_per_second;
transversalVelocity = 0 * meter_per_second;
myYouBotBase->setBaseVelocity(longitudinalVelocity, transversalVelocity, angularVelocity);
LOG(info) << "drive backwards";
SLEEP_MILLISEC(2000);
/* move left */
/*longitudinalVelocity = 0 * meter_per_second;
transversalVelocity = translationalVelocity * meter_per_second;
```

```

angularVelocity = 0 * radian_per_second;
myYouBotBase->setBaseVelocity(longitudinalVelocity, transversalVelocity, angularVelocity);
LOG(info) << "drive left";
SLEEP_MILLISEC(2000);
/* move right */
/*longitudinalVelocity = 0 * meter_per_second;
transversalVelocity = -translationalVelocity * meter_per_second;
angularVelocity = 0 * radian_per_second;
myYouBotBase->setBaseVelocity(longitudinalVelocity, transversalVelocity, angularVelocity);
LOG(info) << "drive right";
SLEEP_MILLISEC(2000);
/* stop base */
longitudinalVelocity = 0 * meter_per_second;
transversalVelocity = 0 * meter_per_second;
angularVelocity = 0 * radian_per_second;
myYouBotBase->setBaseVelocity(longitudinalVelocity, transversalVelocity, angularVelocity);
LOG(info) << "stop base";
} */

if (youBotHasArm) {

/* unfold arm
* position were taken empirically using keyboard and KUKA youBot
*/

desiredJointAngle.angle = 2.9624 * radian;
myYouBotManipulator->getArmJoint(1).setData(desiredJointAngle);
SLEEP_MILLISEC(1000)

desiredJointAngle.angle = 2.5988 * radian;
myYouBotManipulator->getArmJoint(2).setData(desiredJointAngle);

desiredJointAngle.angle = -2.4352 * radian;
myYouBotManipulator->getArmJoint(3).setData(desiredJointAngle);

desiredJointAngle.angle = 1.7318 * radian;
myYouBotManipulator->getArmJoint(4).setData(desiredJointAngle);

```



```
desiredJointAngle.angle = 2.88 * radian;
myYouBotManipulator->getArmJoint(5).setData(desiredJointAngle);

/*desiredJointAngle.angle = 3.14 * radian;
myYouBotManipulator->getArmJoint(5).setData(desiredJointAngle);
desiredJointAngle.angle = 1.9526 * radian;
myYouBotManipulator->getArmJoint(2).setData(desiredJointAngle);
desiredJointAngle.angle = -2.0192 * radian;
myYouBotManipulator->getArmJoint(3).setData(desiredJointAngle);
/*desiredJointAngle.angle = 1.73184 * radian;
myYouBotManipulator->getArmJoint(4).setData(desiredJointAngle);
*/LOG(info) << "unfold arm";
SLEEP_MILLISEC(4000);

int n =0, m=0;
double grid_spacing_n =0, grid_spacing_m =0, x_cordinate=0,y_cordinate=0,z_cordinate =0;

std::cout << " ----- Co-ordiantes of Point Location ----- : " << endl;
if(res.myPCB=0){
/* co-ordinates taken empirically for end positions where we are going to put our PCB */
x_cordinate=-0.1225;
y_cordinate=0.1124;
z_cordinate=0.0239;

std::cout << "PCB in bad conditions! " << endl;

std::cout << "Co-ordinates of end position are : " << endl;
std::cout << " X: " << x_cordinate << endl;
std::cout << " Y: " << y_cordinate << endl;
std::cout << " Z: " << z_cordinate << endl;

}
```

```

if(res.myPCB=1){

x_cordinate=-0.2225;
y_cordinate=0.1124;
z_cordinate=0.0239;

std::cout << "PCB in good conditions! " << endl;

std::cout << "Co-ordinates of end position are : " << endl;
std::cout << " X: " << x_cordinate << endl;
std::cout << " Y: " << y_cordinate << endl;
std::cout << " Z: " << z_cordinate << endl;

}

// -----

std::cout << " ----- GRID GENERATON ----- : " << endl;
std::cout << "Enter the Number of points in X-Direction : ";
cin>> n;
std::cout << endl;

std::cout << "Enter the Number of points in Y-Direction : ";
cin>> m;
std::cout << endl;

std::cout << "Enter the Grid Spacing in X-Direction : ";
cin>> grid_spacing_n;
std::cout << endl;

std::cout << "Enter the Grid Spacing in Y-Direction : ";
cin>> grid_spacing_m;
std::cout << endl;

```

```
//-----
```

```
for (double x=0; x<n; x++){
  for (double y=0; y<m; y++){
    double px_p = grid_spacing_n*x;
    double py_p = grid_spacing_m*y;
```

```
double px = x_coordinate + px_p;
double py = -y_coordinate + py_p;
double pz = z_coordinate; //0.0432;
```

```
////////// THETAS FOR ARM-LINK 1 2 3 //////////
```

```
double l1 = 0.302 - 0.147;
double l2 = 0.437 - 0.302;
double l3 = 0.655 - 0.437;
```

```
double xc = sqrt(px*px + py*py);
double zc = pz;
double phi_c = 0;
```

```
double d = sqrt (xc*xc + zc*zc);
```

```
std::cout << " d = " << d << std::endl;
```

```
if (d >= 0.500){
  double theta_link_1 = atan2(zc,xc) ;
  //break;
}
```

```
else if (d == 0.508) {
  double theta_link_1 = atan2(zc,xc) ;
  //break;
```



```

}

else if (d > 0.508) {
    std::cout << " Co-ordinate Points are out of the work space. \n" ;
    std::cout << "Enter New Co-ordinates Points. \n" ;
    //break;
}

else {

double xw = xc - l3*cos(phi_c);
double zw = zc - l3*sin(phi_c);

double alpha = atan2 (zw,xw);

double cos_beta = (l1*l1 + l2*l2 -xw*xw -zw*zw)/(2*l1*l2);
double sin_beta = sqrt (abs(1 - (cos_beta*cos_beta)));
double theta_link_2 = 3.1416 - atan2 (sin_beta , cos_beta) ;

double cos_gama = (xw*xw + zw*zw + l1*l1 - l2*l2)/(2*l1*sqrt(xw*xw + zw*zw));
double sin_gama = sqrt (abs (1 - (cos_gama * cos_gama)));

double theta_link_1 = alpha - atan2(sin_gama, cos_gama);

double theta_link_1p = theta_link_1 + 2*atan2 (sin_gama , cos_gama);
double theta_link_2p = - theta_link_2;

double theta_2 = theta_link_1p;
double theta_3 = theta_link_2p;
double theta_4 = (theta_2 + theta_3);

```

```
std::cout << " Base Angle " << theta_base << std::endl;
std::cout << " Link-1 Angle " << theta_2 << std::endl;
std::cout << " Link-2 Angle " << theta_3 << std::endl;
std::cout << " Link-3 Angle " << theta_4 << std::endl;
```

```
desiredJointAngle.angle = theta_base * radian;
myYouBotManipulator->getArmJoint(1).setData(desiredJointAngle);
//SLEEP_MILLISEC(1000);
```

```
desiredJointAngle.angle = (2.5988 - theta_2) * radian;
myYouBotManipulator->getArmJoint(2).setData(desiredJointAngle);
//SLEEP_MILLISEC(1000);
```

```
desiredJointAngle.angle = (-2.4352 - theta_3) * radian;
myYouBotManipulator->getArmJoint(3).setData(desiredJointAngle);
//SLEEP_MILLISEC(1000);
```

```
desiredJointAngle.angle = (1.7318 + theta_4) * radian;
myYouBotManipulator->getArmJoint(4).setData(desiredJointAngle);
//SLEEP_MILLISEC(1000);
```

```
//desiredJointAngle.angle = -3.3760 * radian;
//myYouBotManipulator->getArmJoint(3).setData(desiredJointAngle);
SLEEP_MILLISEC(5000);
```

```
desiredJointAngle.angle = (1.7318-0.4 + theta_4) * radian;
myYouBotManipulator->getArmJoint(4).setData(desiredJointAngle);
SLEEP_MILLISEC(1000)
```

```
}
```

```
}
```

```

}

//desiredJointAngle.angle = 0.0 * radian;
//myYouBotManipulator->getArmJoint(1).setData(desiredJointAngle);

/*/* fold arm (approx. home position) using empirically determined values for the positions */
desiredJointAngle.angle = 0.1 * radian;
myYouBotManipulator->getArmJoint(1).setData(desiredJointAngle);

desiredJointAngle.angle = 0.011 * radian;
myYouBotManipulator->getArmJoint(2).setData(desiredJointAngle);

desiredJointAngle.angle = -0.1 * radian;
myYouBotManipulator->getArmJoint(3).setData(desiredJointAngle);
desiredJointAngle.angle = 0.1 * radian;
myYouBotManipulator->getArmJoint(4).setData(desiredJointAngle);
LOG(info) << "fold arm";
SLEEP_MILLISEC(4000);

}

} catch (std::exception& e) {
std::cout << e.what() << std::endl;
std::cout << "unhandled exception" << std::endl;
}

/* clean up all variables*/
if (myYouBotBase) {
delete myYouBotBase;
myYouBotBase = 0;
}
if (myYouBotManipulator) {
delete myYouBotManipulator;
myYouBotManipulator = 0;
}

LOG(info) << "Done.";

```

```
return 0;
```

## 2. keyboard.cpp

```
1 #include <iostream>
2 #include <vector>
3 #include <signal.h>
4 #include <ncurses.h>
5
6 #include "youbot/YouBotBase.hpp"
7
8 using namespace std;
9 using namespace youbot;
10
11 bool running = true;
12
13 void sigintHandler(int signal) {
14     running = false;
15     printf("End!\n\r");
16 }
17
18 int main() {
19
20     signal(SIGINT, sigintHandler);
21
22     try {
23
24         rude::Config configfile;
25
26         if (!configfile.load("../config/applications.cfg"))
27             throw ExceptionOODL("../config/applications.cfg file no found");
28
29         int ch = 0;
30         double linearVel = 0.05; //meter_per_second
31         double angularVel = 0.2; //radian_per_second
32         configfile.setSection("KeyboardRemoteContol");
33         linearVel = configfile.getDoubleValue("TanslationalVelocity_[meter_per_second]");
34         angularVel = configfile.getDoubleValue("RotationalVelocity_[radian_per_second]");
```

```

35
36 YouBotBase myYouBotBase("youbot-base");
37
38 JointVelocitySetpoint setVel;
39 quantity<si::velocity> longitudinalVelocity = 0 * meter_per_second;
40 quantity<si::velocity> transversalVelocity = 0 * meter_per_second;
41 quantity<si::angular_velocity> angularVelocity = 0 * radian_per_second;
(void) initscr(); /* initialize the curses library */
44 keypad(stdscr, TRUE); /* enable keyboard mapping */
45 (void) nonl(); /* tell curses not to do NL->CR/NL on output */
46 (void) cbreak(); /* take input chars one at a time, no wait for \n */
47 // (void) echo(); /* echo input - in color */
48
49 def_prog_mode();
50
51 refresh();
52 printf("up = drive forward\n\r"
53 "down = drive backward\n\r"
54 "left = drive left\n\r"
55 "right = drive right\n\r"
56 "y = turn right\n\r"
57 "x = turn left\n\r"
58 "any other key = stop\n\r\n\r");
59 refresh();
60
61 while (running) {
62
63 ch = getch();
64
65 switch (ch) {
66 case KEY_DOWN:
67 longitudinalVelocity = -linearVel * meter_per_second;
68 transversalVelocity = 0 * meter_per_second;
69 angularVelocity = 0 * radian_per_second;
70 LOG(info) << "drive backward";
71 printf("\r");
72 break;
73 case KEY_UP:

```

```
74 longitudinalVelocity = linearVel * meter_per_second;
75 transversalVelocity = 0 * meter_per_second;
76 angularVelocity = 0 * radian_per_second;
77 LOG(info) << "drive forward";
78 printf("\r");
79 break;
80 case KEY_LEFT:
81 transversalVelocity = linearVel * meter_per_second;
82 longitudinalVelocity = 0 * meter_per_second;
83 angularVelocity = 0 * radian_per_second;
84 LOG(info) << "drive left";
85 printf("\r");
86 break;
87 case KEY_RIGHT:
88 transversalVelocity = -linearVel * meter_per_second;
89 longitudinalVelocity = 0 * meter_per_second;
90 angularVelocity = 0 * radian_per_second;
91 LOG(info) << "drive right";
92 printf("\r");
93 break;
94 case 'y':
95 angularVelocity = angularVel * radian_per_second;
96 transversalVelocity = 0 * meter_per_second;
97 longitudinalVelocity = 0 * meter_per_second;
98 LOG(info) << "turn right";
99 printf("\r");
100 break;
101 case 'x':
102 angularVelocity = -angularVel * radian_per_second;
103 transversalVelocity = 0 * meter_per_second;
104 longitudinalVelocity = 0 * meter_per_second;
105 LOG(info) << "turn left";
106 printf("\r");
107 break;
108
109 default:
110 longitudinalVelocity = 0 * meter_per_second;
111 transversalVelocity = 0 * meter_per_second;
112 angularVelocity = 0 * radian_per_second;
113 LOG(info) << "stop";
```

```

114 printf("\r");
115 break;
116 }
117
118 myYouBotBase.setBaseVelocity(longitudinalVelocity, transversalVelocity, angularVelocity);
119
120 refresh();
121 SLEEP_MILLISEC(100);
122 }
123
124 setVel.angularVelocity = 0 * radian_per_second;
125 myYouBotBase.getBaseJoint(1).setData(setVel);
126 myYouBotBase.getBaseJoint(2).setData(setVel);
127 myYouBotBase.getBaseJoint(3).setData(setVel);
128 myYouBotBase.getBaseJoint(4).setData(setVel);
129
130 endwin();
131 SLEEP_MILLISEC(500);
132
133 } catch (std::exception& e) {
134 std::cout << e.what() << std::endl;
135 } catch (...) {
136 std::cout << "unhandled exception" << std::endl;
137 }
138
139 return 0;
140 }

```

## 7.2. OCTAVE COMPUTER VISION CODE

### Morphology function

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                               Morphology function                               %%
%% Author: Alberto Justo Lobato                                             %%
%%                                                                           %%

```



```
%% Bachelor's Graduation Thesis: %%
%% E-waste material classifier with KUKA youBot and %%
%% computer vision %%
%% %%
%% Vilnius Gediminas Technical University , Lithuania %%
%% %%
%% Explanation: function responsible of applying %%
%% morphological operations, in this case dilation, on %%
%% two images to obtain differences between each other. %%
%% Moreover, we asked users previously for structuring %%
%% element size and also number of times to repeat %%
%% binarization until they are satisfied with results. %%
%% Finally, we get number of elements of image obtained %%
%% from this transformation. %%
%% %%
%% MATLAB/OCTAVE functions used: %%
%% -rgb2gray %%
%% -graythresh %%
%% -imbinarize %%
%% -imfilter %%
%% -bwconncomp %%
%% -imerode %%
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [T]=myhomography(I,A)

if(islogical(I))
disp('Image is already binarized, so it will not be modified')
T=I;
return
end
repeat=1;
```



```

while(repeat==1)

umbral=graythresh(I);
BW=im2bw(I,umbral);
BW2=im2bw(A,umbral);
BW2inv=~BW2;
BWinv=~BW;
radius=input('Introduce radius of structuring element between 1-5: ');
ee=strel('disk',radius);
% We apply dilation for splitting elements and detecting them

B=imerode(BWinv,ee);
B2=imerode(BW2inv,ee);
L=~B;
L2=~B2;
figure
imshow(I),title('original image');
figure
imshow(A),title('testing image');
figure
subplot(1,4,1),imshow(BWinv), title('binarized original image')
subplot(1,4,2),imshow(L), title('original image with dilation')
subplot(1,4,3),imshow(BW2inv), title('binarized test image')
subplot(1,4,4),imshow(L2), title('test image with dilation')

% We see split objects after transformations
cc=bwconncomp(B); % creating two data structures for knowing n°
elements
a=cc.NumObjects
ccs=bwconncomp(B2);
as=cc.NumObjects

if (a~=as)
    disp('Number of elements are really different. Second image has
mistakes');
    myPCB=0; %% we create a variable called myPCB which value is 0
when it has mistakes and 1 when it is in good conditions

```



else

```
    disp('Number of elements are similar in both images. So, testing  
image is correct');
```

```
    myPCB=1; %% this variable will be shared with ROS
```

end

```
    fprintf('\n Is obtained result the one you wanted?');
```

```
    fprintf('\n t1. Yes (press 0)');
```

```
    fprintf('\n t2. No (press 1)');
```

```
    fprintf('\n');
```

```
    repeat=input(' '); %Asking user about results
```

end

## Filtering function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%           Filtering function           %%  
%% Author: Alberto Justo Lobato           %%  
%%                                         %%  
%% Bachelor's Graduation Thesis:         %%  
%% E-waste material classifier with KUKA youBot and %%  
%% computer vision                       %%  
%%                                         %%  
%% Vilnius Gediminas Technical University, Lithuania %%  
%%                                         %%  
%% Explanation: we apply two gauss filters at the same %%  
%% time to one image to obtain its contour, just %%  
%% detecting both filtered pictures       %%  
%%                                         %%  
%% MATLAB/OCTAVE functions used:         %%  
%% -rgb2gray                             %%  
%% -mat2gray                             %%  
%% -imfilter                             %%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

function filtering(I)
%Obtain image edge from subtract of two gauss filters
if ndims(I)==3
    Ig=rgb2gray(I);
end
[rows,columns]=size(Ig);
Ig=double(Ig); %passing to double format
I_contour=zeros(rows,columns);
Sx=[-1 0 1;-2 0 2;-1 0 1];
Sy=[-1 -2 -1;0 0 0; 1 2 1];

Ix=imfilter(Ig,Sx);
Iy=imfilter(Ig,Sy);
I_contour=abs(Ix)+abs(Iy);
S=mat2gray(I_contour);
figure
imshow(I), title('original image')
figure
imshow(S), title('contour image with mat2gray')

```

## Matching function

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                               Matching function                               %%
%% Author: Alberto Justo Lobato                                           %%
%%                                                                           %%
%% Bachelor's Graduation Thesis:                                         %%
%% E-waste material classifier with KUKA youBot and                       %%
%% computer vision                                                         %%

```



```
%%
%% Vilnius Gediminas Technical University, Lithuania
%%
%% Explanation: function responsible of searching for
%% similar elements on an image, based on correlation
%% finding their centroid.
%% After this search, it plots an asterisk in each
%% element that matches.
%% To sum up, it is a good way for searching elements in
%% common between images, but it is quite inconsistent,
%% as this means we should use other methods previously.
%%
%% MATLAB/OCTAVE functions used:
%% -rgb2gray
%% -imcrop
%% -normxcorr2
%% -bwconncomp
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function matching (I,A)

if ndims(I) == 3
disp( 'Image in color')
N=rgb2gray(I);
L=rgb2gray(A);

else
disp( 'Image in B&W')
N=I;
L=A;
end

jig=imcrop(N);%select jig
[x,y]=size(jig);
C=normxcorr2(jig,L);
B=C>0.9; %Only select points that have correlation bigger than 0.9
cc=bwconncomp(B);
s=regionprops(cc,'Centroid');
num_el=length(s) %number of elements in common
figure
```

```

imshow(N);
figure
imshow(L);
hold on
for k=1: num_el
plot((s(k).Centroid(1))-(x)/2, (s(k).Centroid(2))-(y)/2, '*r'); %draw
asterisks in elements in common between images
end
hold off
end

```

## Main script

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                               Main                               %%
%% Author: Alberto Justo Lobato                                   %%
%%                               %%                               %%
%% Bachelor's Graduation Thesis:                               %%
%% E-waste material classifier with KUKA youBot and             %%
%% computer vision                                             %%
%%                               %%                               %%
%% Vilnius Gediminas Technical University , Lithuania         %%
%%                               %%                               %%
%% Explanation: compile all functions mentioned in             %%
%% previous parts.                                             %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

I=imread('imag1.jpg');
J=imread('imag2.jpg');
S=imread('imag3.jpg');
W=imread('imag4.jpg');
R=imread('imag5.jpg');
T=imread('imag6.jpg');
U=imread('imag7.jpg');

```



```
% filtering all images introduced
filtering(I);
filtering(J);
filtering(S);
filtering(W);
filtering(R);
filtering(T);
filtering(U);

% comparing elements between standard and taken image of each PCB
mymorphology(I,J);
mymorphology(S,W);
mymorphology(R,T);
mymorphology(T,U);

% applying template matching

matching(I,J);
matching(S,W);
matching(R,T);
matching(T,U);
```

### 7.3. MATLAB FORWARD AND INVERSE KINEMATICS CODE

#### 1. Forward.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                               Forward kinematics function           %%
%% Author: Alberto Justo Lobato                                       %%
%%                                                                           %%
%% Bachelor's Graduation Thesis:                                       %%
%% E-waste material classifier with KUKA youBot and                   %%
%% computer vision                                                     %%
%%                                                                           %%
%% Vilnius Gediminas Technical University, Lithuania                 %%
%%                                                                           %%
%% Explanation: function responsible of applying the                   %%
```

```

%% generic forward equations of KUKA youBot and then      %%
%% obtaining its parameters, which will be used for      %%
%% inverse kinematics problem afterwards                  %%
%% MATLAB/Robotics Toolbox functions used:               %%
%% -eye                                                    %%
%% -subs                                                  %%
%% -simplify                                              %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

clear all; close all;
syms x y z alpha beta q1 q2 q3 q4 q5
syms l1 l2 l3 l4 l5 l6 PM

```

```

L=[l1 l2 l3 l4 l5 l6]; %%Matrix relative to each arm length
T00=eye(4,4); %%Build a matrix 4x4

```

```

%Now we need to find homogeneous transformation of KUKA youBot
T01=T00*ScrewZ(q1,L(2))*ScrewX(-PM,L(1)); %each generic equation for
point O
T02=T01*ScrewZ(q2,0)*ScrewX(0,L(3));
T03=T02*ScrewZ(q3,0)*ScrewX(0,L(4));
T04=T03*ScrewZ(q4,0)*ScrewX(PM,0);
T05=T04*ScrewZ(q5,L(5)+L(6))*ScrewX(0,0);

```

```

%PM now has the value of PI/2

```

```

T01=subs(T01,PM,pi/2);
T02=subs(T02,PM,pi/2);
T03=subs(T03,PM,pi/2);
T04=subs(T04,PM,pi/2);
T05=subs(T05,PM,pi/2);

```



```
%Next step is simplifying all equations
```

```
T01=simplify(T01);  
T02=simplify(T02);  
T03=simplify(T03);  
T04=simplify(T04);  
T05=simplify(T05);
```

```
%Obtaining components of our main point
```

```
r13=T05(1,3);  
r23=T05(2,3);  
r33=T05(3,3);
```

## 2. Inverseorientation.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%           Forward kinematics function           %%  
% Author: Alberto Justo Lobato                    %%  
%                                                    %%  
% Bachelor's Graduation Thesis:                  %%  
% E-waste material classifier with KUKA youBot and %%  
% computer vision                                %%  
%                                                    %%  
% Vilnius Gediminas Technical University, Lithuania %%  
%                                                    %%  
% Explanation: thanks to this function, we obtain %%  
% KUKA youBot parameters for decoupling from 5 DOF to %%  
% 3                                               %%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```



```

function [myxc,myyc,myzc]=inverseorientation(x,y,z,l5,l6,Q1,Q2,Q3,Q4)

%Applying symbolic forward equations

r13=cos(Q1)*sin(Q2+Q3+Q4);
r23=sin(Q1)*sin(Q2+Q3+Q4);
r33=cos(Q2+Q3+Q4);

%Position of the point Oc

myxc=x-(l5+l6)*r13;
myyc=y-(l5+l6)*r23;
myzc=z-(l5+l6)*r33;

end

```

### 3. Inverseposition.m

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%      Inverse kinematics position function          %%
%% Author: Alberto Justo Lobato                      %%
%%                                                    %%
%% Bachelor's Graduation Thesis:                    %%
%% E-waste material classifier with KUKA youBot and  %%
%% computer vision                                  %%
%%                                                    %%
%% Vilnius Gediminas Technical University, Lithuania %%
%%                                                    %%
%% Explanation: thanks to this function, we obtain  %%
%% KUKA youBot angles, putting equations obtained in %%
%% theoretical part                                  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```



```
function [Q2,Q3,D2]= inverseposition(XC,YC,ZC,L1,L2,L3,L4)

D=((XC.^2+YC.^2- 2*sqrt(XC.^2+YC.^2)*L1 +L1.^2)+(ZC-L2).^2-13.^2-
14.^2)/(2*13*14);
D2=1-D.^2;
    if D2<0
        Q3=0;
        Q2=0;
    else
        Q3=atan2(sqrt(1-D.^2),D);
        Q2=atan2((sqrt(xc.^2+yc.^2)-L1),ZC-12)-
atan2(14*sin(Q3),L3+L4*cos(Q3));
    end
end
```

## 8. APPENDIXES

### 8.1. KUKA YOUTBOT ARM SPECIFICATIONS



**KUKA**

OPEN INTERFACES [+]

[+] 5 DOF ARM

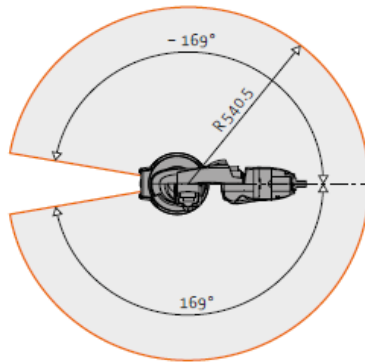
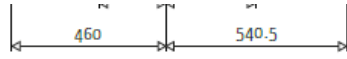
FREELY PROGRAMMABLE [+]

[+] REAL-TIME ETHERCAT COMMUNICATION

OMNIDIRECTIONAL MOBILE PLATFORM [+]

**KUKA**

**KUKA youBot**  
Research & Application Development  
in Mobile Robotics



Axis 3 (A3)	+ 146°/- 151°	90°/s
Axis 4 (A4)	+/- 102°	90°/s
Axis 5 (A5)	+/- 167°	90°/s

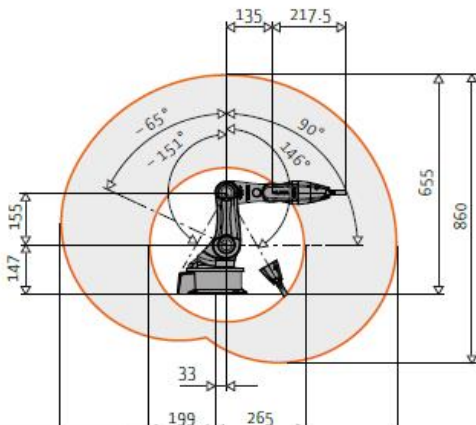
<b>Gripper</b>	<b>Detachable, 2 fingers</b>
Gripper stroke	20 mm
Gripper range	70 mm
Motors	2 independent stepper motors


Details provided about the properties and usability of the products are purely for information purposes and do not constitute a guarantee of these characteristics. The extent of goods delivered and services performed is determined by the subject matter of the specific contract. No liability accepted for errors or omissions.

## youBot Arm

### Experience Manipulation

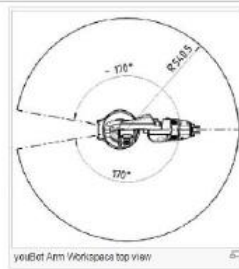
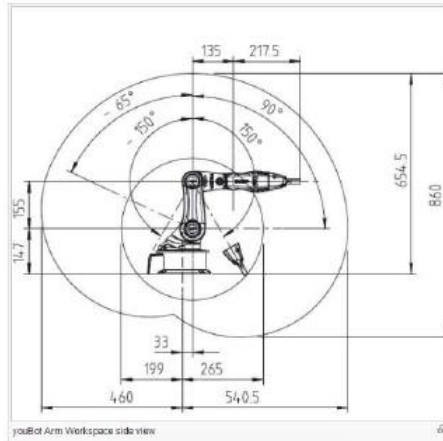
**ENTER INTO THE AREA OF ROBOTIC MANIPULATORS.** With the external, controllable 5-axis kinematics you can practically demonstrate and analyze the basics of robotics kinematics and dynamics. It is possible to perform basic grasping applications and, if extended with sensors, you can realize applications with grasp planning.



	<b>youBot arm</b>
Serial kinematics	5 axes
Height	655 mm
Work envelope	0.513 m <sup>3</sup>
Weight	5.8 kg
Payload	0.5 kg
Structure	Magnesium Cast
Positioning repeatability	0.1 mm
Communication	EtherCAT: 1 ms cycle
Voltage connection	24 V DC
Drive train power limitable to	80 W

Axis data	Range	Speed
Axis 1 (A1)	+/- 169°	90°/s
Axis 2 (A2)	+ 90°/- 65°	90°/s

Workspace



## Arm Actuator

Axis data Range Velocity

```
axis 1 +/- 169° 90 °/s
axis 2 + 90°/- 65° 90 °/s
axis 3 + 146°/ - 151° 90 °/s
axis 4 +/- 102,5° 90 °/s
axis 5 +/- 167,5° 90 °/s
```

## Actuator Data

Actuator Data	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Wheel
<b>Motor</b>						
Nominal Voltage (V)	24	24	24	24	24	24
Nominal Current (A)	2.32	2.32	2.32	1.07	0.49	2.32
Nominal Torque (mNm)	82.7	82.7	82.7	58.8		82.7
Terminal Inductance (mH)	0.573	0.573	0.573	2.24	7.73	0.573
Terminal Resistance (Ω)	0.978	0.978	0.978	4.48	13.7	0.978
Moment of Inertia (kg*mm <sup>2</sup> )	13.5	13.5	13.5	9.25	3.5	13.5
Rated speed (RPM)	5250	5250	5250	2850	2800	5250
Weight (g)	110	110	110	75	46	110
<b>Gearbox</b>						
Reduction Ratio	156	156	100	71	71	26
Moment of Inertia (kg*mm <sup>2</sup> )	0.409	0.409	0.071	0.07	0.07	0.14
Weight (g)						224
<b>Encoder</b>						
Counts per Revolution	4000	4000	4000	4000	4000	4000

## Battery

Caution: The battery is in general quite sensitive and should never be left connected when the youBot doesn't work because it slowly discharges. It may eventually results in battery failure.

- Type : Maintenance-free lead acid
- Rechargeable : Yes
- Voltage : 24 V
- Capacity : 5 Ah
- Approximate Runtime of youBot : 90 min

## Internal PC

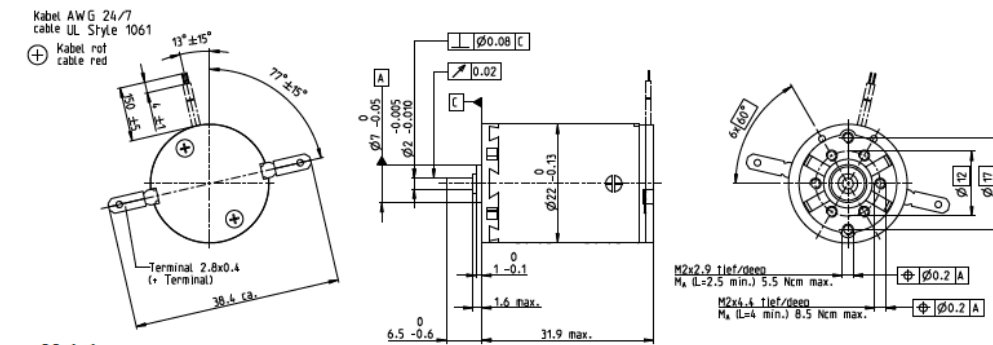
- Type : Mini-ITX
- CPU : Intel Atom D610 Dual Core 1.66 GHz
- RAM : 2 GB; SO-DDR2-667 200PIN
- Hard Drive : 32 GB SSD
- Ports : 6 x USB 2.0, 1 x VGA, 2 x LAN (1 available)
- DC Input : 12 V

## Motor controller

All used motor controller boards came from TCMC series by Titanix.

## 8.2. MAXON JOINT MOTORS

### A-max 22 Ø22 mm, Precious Metal Brushes CLL, 5 Watt



- Stock program
- Standard program
- Special program (on request)

#### Part Numbers

	110117	110119	110120	110121	110122	110123	110124	110125	110126	110127	110128	110129
with terminals												
with cables	139838	218799	238798	202413	258367	137255	134267	134666	267423	137476	310003	342390

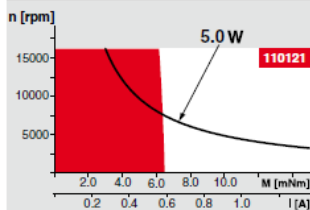
#### Motor Data

Values at nominal voltage		6	9	9	12	12	15	18	24	30	36	48	48
1 Nominal voltage	V	6	9	9	12	12	15	18	24	30	36	48	48
2 No load speed	rpm	9630	9970	8760	10400	9400	10300	9970	10700	10800	9800	9280	8370
3 No load current	mA	29.5	20.8	16.8	16.8	14.2	13.1	10.4	8.81	7.18	5.06	3.47	2.93
4 Nominal speed	rpm	7390	7300	6100	7770	6700	7530	7220	7970	8070	7000	6420	5520
5 Nominal torque (max. continuous torque)	mNm	4.81	6.22	6.3	6.24	6.18	6.1	6.05	6.02	5.98	5.94	5.83	5.9
6 Nominal current (max. continuous current)	A	0.84	0.745	0.661	0.586	0.523	0.451	0.362	0.291	0.234	0.175	0.122	0.111
7 Stall torque	mNm	20.1	22.9	20.5	24.3	21.4	22.9	22	23.5	23.5	20.8	19	17.4
8 Stall current	A	3.42	2.68	2.11	2.23	1.77	1.65	1.28	1.11	0.894	0.599	0.387	0.32
9 Max. efficiency	%	83	84	83	84	83	83	83	83	83	83	82	82
<b>Characteristics</b>													
10 Terminal resistance	Ω	1.76	3.36	4.27	5.39	6.78	9.07	14	21.6	33.5	60.1	124	150
11 Terminal inductance	mH	0.106	0.222	0.288	0.362	0.445	0.584	0.89	1.37	2.1	3.68	7.29	8.95
12 Torque constant	mNm/A	5.9	8.55	9.73	10.9	12.1	13.9	17.1	21.2	26.2	34.8	48.9	54.3
13 Speed constant	rpm/V	1620	1120	981	875	790	689	558	450	364	274	195	176
14 Speed / torque gradient	rpm/mNm	482	438	430	432	443	451	458	459	465	474	494	486
15 Mechanical time constant	ms	20.5	19.8	19.7	19.7	19.8	20.2	20.1	20.2	20.3	20.3	20.5	20.4
16 Rotor inertia	gcm <sup>2</sup>	4.07	4.32	4.37	4.36	4.26	4.27	4.2	4.2	4.16	4.09	3.97	4.01

#### Specifications

- Thermal data**
- 17 Thermal resistance housing-ambient 20 K/W
  - 18 Thermal resistance winding-housing 6.0 K/W
  - 19 Thermal time constant winding 10.2 s
  - 20 Thermal time constant motor 313 s
  - 21 Ambient temperature C -30...+65°C
  - 22 Max. winding temperature +85°C
- Mechanical data (sleeve bearings)**
- 23 Max. speed 16000 rpm
  - 24 Axial play 0.05 - 0.15 mm
  - 25 Radial play 0.012 mm
  - 26 Max. axial load (dynamic) 1 N
  - 27 Max. force for press fits (static) 80 N
  - 28 Max. radial load, 5 mm from flange 2.8 N
- Mechanical data (ball bearings)**
- 23 Max. speed 16000 rpm
  - 24 Axial play 0.05 - 0.15 mm
  - 25 Radial play 0.025 mm
  - 26 Max. axial load (dynamic) 3.3 N
  - 27 Max. force for press fits (static) 45 N
  - 28 Max. radial load, 5 mm from flange 12.3 N
- Other specifications**
- 29 Number of pole pairs 1
  - 30 Number of commutator segments 9
  - 31 Weight of motor 54 g
- CLL = Capacitor Long Life
- Values listed in the table are nominal.  
Explanation of the figures on page 79.
- Option**
- Ball bearings in place of sleeve bearings
  - Without CLL

#### Operating Range



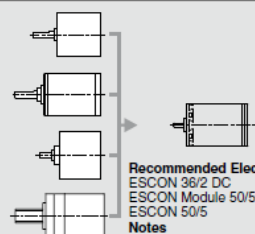
#### Comments

- Continuous operation**  
In observation of above listed thermal resistance (lines 17 and 18) the maximum permissible winding temperature will be reached during continuous operation at 25°C ambient.  
= Thermal limit.
- Short term operation**  
The motor may be briefly overloaded (recurring).
- Assigned power rating**

#### maxon Modular System

Overview on page 20-25

- Planetary Gearhead**  
Ø22 mm  
0.1 - 0.6 Nm  
Page 260/261
- Planetary Gearhead**  
Ø22 mm  
0.5 - 2.0 Nm  
Page 262/264
- Spur Gearhead**  
Ø24 mm  
0.1 Nm  
Page 269
- Spindle Drive**  
Ø22 mm  
Page 299/300



- Recommended Electronics:**
- ESCON 36/2 DC Page 842
  - ESCON Module 50/5 843
  - ESCON 50/5 844
- Notes 22

## 8.3. SIEMENS MV440 CAMERA SPECIFICATIONS

### 13.1 General technical specifications

#### Mechanical environmental conditions for SIMATIC MV420 and SIMATIC MV440

---

##### Mechanical environmental conditions for operation

---

SIMATIC MV420 and SIMATIC MV440 are designed for fixed installation in an environment protected from the weather and meet the conditions for use complying with DIN IEC 60721-3-3:

- Class 3M3 (mechanical requirements);
- Class 3K3 (climatic environmental conditions).

---

##### Mechanical environmental conditions, sine-shaped oscillations

---

Frequency range in Hz	Test values
$10 \leq f < 58$	0.075 mm amplitude
$58 \leq f < 500$	1 g constant acceleration

---

##### Test for mechanical environmental conditions

---

Test for / test standard	Comments
Vibrations Oscillation test complying with IEC 60068-2-6 (sine)	<ul style="list-style-type: none"> <li>• Vibration type: Frequency cycles with a rate of change of 1 octave/minute.               <ul style="list-style-type: none"> <li>– <math>10 \text{ Hz} \leq f &lt; 58 \text{ Hz}</math>, constant amplitude 0.075 mm</li> <li>– <math>58 \text{ Hz} \leq f &lt; 500 \text{ Hz}</math>, constant acceleration 1 g</li> <li>– <math>10 \text{ Hz} \leq f \leq 55 \text{ Hz}</math>, amplitude 1 mm (only sensor head and lighting unit)</li> </ul> </li> <li>• Vibration duration: 10 frequency cycles per axis in each of the 3 mutually perpendicular axes.</li> </ul>

Test for / test standard	Comments
Shock Shock test complying with IEC 60068-2-29	<ul style="list-style-type: none"> <li>• Type of shock: Half-sine</li> <li>• Strength of the shock for the reader:               <ul style="list-style-type: none"> <li>– 10 g peak value</li> <li>– 16 ms duration</li> </ul> </li> <li>• Direction of shock: 100 shocks in each of the 3 mutually perpendicular axes</li> </ul>

---



## Climatic environmental conditions for SIMATIC MV420 and SIMATIC MV440

Table 13- 1

<b>Ambient climatic conditions for operation</b>		
<b>Ambient conditions</b>	<b>Permitted range</b>	<b>Comments</b>
Temperature	0 ... +50 °C	
Temperature change	Max. 10 °C/h	
Relative humidity	max. 95 % at +25 °C	No condensation, corresponds to relative humidity degree 2 to IEC 61131-2.

## Transportation and storage of SIMATIC MV420 and SIMATIC MV440

### **Transportation and storage of modules**

SIMATIC MV440 exceeds the requirements of IEC 61131-2 for transportation and storage conditions. The following information applies to modules transported or stored in their original packaging.

The climatic conditions correspond to IEC 60721-3-3, Class 3K7 for storage and IEC 60721-3-2, Class 2K4 for transportation.

The mechanical conditions correspond to IEC 60721-3-2, Class 2M2.

<b>Conditions</b>	<b>Permitted range</b>
Free fall	≤ 1 m (up to 10 kg)
Temperature	-30 to +70° C
Atmospheric pressure	660 ... 1080 hPa, corresponds to a height of 0 ... 3500 m
Relative humidity (at +25 °C)	5 to 95%, no condensation
Sine-shaped oscillations complying with IEC 60068-2-6	5 - 9 Hz: 3.5 mm 9 - 500 Hz: 9.8 m/s <sup>2</sup>
Shock complying with IEC 60068-2-29	250 m/s <sup>2</sup> , 6 ms, 1000 shocks

## Power supply for SIMATIC MV420 and SIMATIC MV440

<b>Power supply</b>	
Supply voltage (UN)	24 ; (19.2 V DC ...28.8 V DC, safety extra low voltage, SELV).
Fuse	Max. 4 A
Safety requirements complying with	IEC 61131-2 corresponds to DIN EN 61131-2

## Electromagnetic compatibility for SIMATIC MV420 and SIMATIC MV440

<b>Electromagnetic compatibility</b>		
<b>Pulse-shaped interference</b>		
<b>Interference</b>	<b>Test voltage</b>	<b>Corr. to severity</b>
Electrostatic discharge according to IEC 61000-4-2	<ul style="list-style-type: none"> <li>Air discharge: <math>\pm 8</math> kV</li> <li>Contact discharge: <math>\pm 6</math> kV</li> </ul>	3
Burst pulse (fast transients) complying with IEC 61000-4-4	<ul style="list-style-type: none"> <li>2 kV (power supply cable)</li> <li>2 kV (signal line)</li> </ul>	3
<b>Surge complying with IEC 61000-4-5</b>		
<b>Coupling</b>	<b>Test voltage</b>	<b>Corr. to severity</b>
Asymmetrical	2 kV (power supply cable) direct voltage with protective elements	3
Symmetrical	1 kV (power supply cable) direct voltage with protective elements	3
<b>Sine-shaped interference</b>		
<b>RF interference (electromagnetic fields)</b>	<b>Test values</b>	<b>Corr. to severity</b>
Conforming to IEC 61000-4-3	10 V/m at 80 % amplitude modulation of 1 kHz in the range from 80 to 1000 MHz	3
Conforming to IEC 61000-4-3	10 V/m at 50% pulse modulation at 900 MHz	3
<b>RF interference on cable/cable shields</b>	<b>Test values</b>	<b>Corr. to severity</b>
Conforming to IEC 61000-4-6	Test voltage 10 V at 80% amplitude modulation of 1 kHz in the range from 9 kHz to 80 MHz	3
<b>Emission</b>		
Limit class	<ul style="list-style-type: none"> <li>Emitted interference of electromagnetic fields in accordance with EN 55011: Limit class A, group 1;</li> <li>Emitted interference via the AC input power supply in accordance with EN 55011: Limit class A, group 1.</li> </ul>	