Sheffield Hallam University

Assignment Submission Cover Sheet

Please complete all sections and attach to your assignment. You can find these details on your Assessment Statement

Student Name:	Rafael Gómez Hortelano
Student Number:	29044206

If this is a group submission the details of the student responsible for submitting the work is to be entered above and the name and student number of other group members below.

Course Title:	MSc. Automation Control & Robotics
Module Title:	PROJECT AND DISSERTATION
Module Code:	(CONT2 KF-2018/9) 55-704710-KF-20189
Tutor:	Dr Lyuba Alboul
Assignment Title:	Design and implementation of an artificial intelligence module in a Fetch. Application: Pick&place operation using colour detection
Deadline:	7th May 2020

Declaration

I confirm that this assessment is my own work and that I have duly acknowledged and correctly referenced the work of others. I am aware of and understand that any breaches to the Code of Academic Conduct will be investigated and sanctioned in accordance with the Academic Conduct Regulation, found on shuspace| Rules and Regulations| Conduct and Discipline

Signature:	GOMEZ HORTELANO RAFAEL - 23326705J Firmado digitalmente por GOMEZ HORTELANO RAFAEL - 23326705J Fecha: 2020.04.25 15:53:11 +02'00'
Date Submitted:	25th June 2020

Learning contracts

If you have a learning contract recommendation for adjusted marking you will need to use blue stickers to alert the tutor to this.

*You **must** attach a blue sticker on all your assignments and exam scripts. If you submit work electronically you must type the wording of the sticker in blue on the front of your assignment where tutors can easily see it.*

Acknowledgements

To my family, which helped and supported me in this little Erasmus adventure that has concluded with this project.

To Lyuba Alboul, that guided me through my stay at the Sheffield Hallam University and always trusted me.

Contents

1. Introduction 7
1.1. Preface 7
1.2. Abstract 8
1.3. Motivation 10
2. Objectives 11
3. Literature Review 12
3.1. Robots vs. Humans 12
3.2. Related researches 13
3.3. Other Solutions 14
4. Tools Review 16
4.1. Programming and Software 16
4.1.1. Robot Operating System (ROS) 16
4.1.2. MATLAB 19
4.1.3. Virtual Box and Linux 19
4.2. Human Vision 20
4.3. Machine Vision 20
4.3.1 Introduction 20
4.3.2. Vision System Components 21
4.4. Colour Detection 23
4.5. Hardware: Fetch Robot and Robot Movement 26
5. Methodology 28
5.1. Programming 28
5.1.1. ROS Topics 28
5.1.2. MATLAB environment with ROS 29
5.1.3. MATLAB scripts 30
5.2. Simulation 31
5.2.1. Goal of the simulation 31
5.2.2. Setup of the simulation 31
5.3. Real test 34
6. Results 36
6.1. Simulation results 36
6.2. Simulation full programs 38
6.2. Real tests results 39
7. Conclusions 40
8. Future work 41

References 42

Appendix 44

Process Initialisation 44

Process Script 47

Process Shutdown 51

List of figures

Figure 1. IRB 140 from ABB [9] 13
Figure 2. Pick&place of croissants [12] 14
Figure 3. Example of a tracking system [13] 15
Figure 4. ROS Master Node [15] 16
Figure 5. Publisher and Subscriber Graph [15] 17
Figure 6. Services [15] 18
Figure 7. Actions [15] 18
Figure 8. Human Eye. [20] 20
Figure 9. Converging and Diverging Lenses (Google Images) 21
Figure 10. Lighting techniques [23] 22
Figure 11. Fetch's head [25] 23
Figure 12. RGB (Google Images) 23
Figure 13. HSV colour space [24] 24
Figure 14. Flux diagram of colour detection 25
Figure 15. Fetch Robot [25] 26
Figure 16. Fetch Robot Joints [26] 27
Figure 17. Rosinit instruction 29
Figure 18. Successful connection 29
Figure 19. Part of the topics we can access 29
Figure 20. Initial Position 30
Figure 21. Flux Diagram of the Process 31
Figure 22. Virtual Box Main Page 32
Figure 23. Net Configuration 32
Figure 24. Configuration in VM (Ubuntu 18.04) 32
Figure 25. Configuration in Host Operating System (Windows 10) 33
Figure 26. Gazebo simulation.launch 33
Figure 27. Gazebo playground.launch 33
Figure 28. Acknowledging the safe place for the test 34
Figure 29. Prismatic piece vs Two pounds coin. 35
Figure 30. Fetch Robot with the prism picked up 35
Figure 31. Power On (left) – Pre-pick position (right) 36
Figure 32. Camera position (head moved) 37
Figure 33. Place position (right of the robot) 37
Figure 34. First program. Single cube 38
Figure 35. Aerial schematic view. Single cube. 38
Figure 36. Second program. Four cubes 39

List of Tables

Table 1. Joint Limits [26] 27

1. Introduction

1.1. Preface

The title of the dissertation is *"Design and implementation of an artificial intelligence module in a Fetch Robot. Application: Pick&place operation using colour detection"*. The document presented here aims to describe the work developed in the Department of Engineering and Mathematics and Centre for Automation and Robotics Research at Sheffield Hallam University during the second semester of the academic year 2019/2020 as part of my Master Dissertation.

El título del Proyecto es "Diseño e Implementación de un módulo de inteligencia artificial en un robot Fetch. Aplicación: Operación de pick&place utilizando detección de color". *El presente documento pretende describir el trabajo realizado en el Departamento de Ingeniería y Matemáticas y Centro de Automatización e Investigación Robótica de la Universidad de Sheffield Hallam durante el segundo semestre del año académico 2019/2020 como parte de mi Trabajo de Final de Máster.*

1.2. Abstract

Sometimes, and every day more, we need flexibility in industrial environments. Humans can adapt to certain changes in these environments and can take actions quickly and effectively. But humans can get tired and may not have the same accuracy as a robot could have. Because of that, making the robots capable of acting in some of these changing environments would result in an increase of productivity and a great improvement in industry.

The goal is to make the robot be capable of handling a group of different objects that may have not been yet pre-processed (not being in a database) so the robot must decide what to do with every object depending on their colour. For this, we are going to make a pick&place application that uses some information from the outside to help the robot make decisions.

A Fetch robot from Fetch Robotics [1] is used for simulation and verification of this application, which is useful because it was built as an industrial robot. It has a base that can move and a manipulator arm with 7 degrees of freedom that can perform various tasks. It has many sensors like a laser scan at the base which can be useful for obstacle detection and a camera at the head which can get images of what is seeing the robot. These images will be processed and with the right coding the robot will achieve the goal of the project.

The simulation environment has been chosen to be Gazebo and the programming has been done in ROS using MATLAB as the coding and communication environment.

Keywords: robot, pick&place, colour detection, decision making, artificial intelligence, ROS, MATLAB.

De vez en cuando, y cada vez más, se necesita flexibilidad en entornos industriales. Los humanos se pueden adaptar a ciertos cambios en este entorno y pueden responder rápida y eficazmente, pero nos cansamos y no tenemos la misma precisión que un robot. Por ello, conseguir que los robots puedan responder a algunos de estos escenarios resultaría en un aumento de la productividad y una gran mejora en la industria.

El objetivo es conseguir que el robot sea capaz de manejar un grupo de objetos diferentes que no tienen por qué estar en una base de datos preprogramada y este debe decidir qué hacer con cada objeto en función del color que tengan. Para ello, se realizará una aplicación de pick&place que utilizará información del exterior para ayudar al robot a tomar decisiones.

El robot Fetch de la empresa Fetch Robotics será el utilizado para similar y verificar la aplicación, lo cual es ideal pues es un robot de índole industrial. Este robot tiene una base móvil y un brazo manipulador con 7 grados de libertad. Además, cuenta con sensores láser para la detección de obstáculos y una cámara situada en la cabeza con la que se pueden obtener imágenes de lo que ve el robot. Las imágenes obtenidas de esta manera serán las que, mediante un algoritmo, se utilizarán para conseguir el objetivo del proyecto.

El entorno de simulación será Gazebo y la programación se hará siguiendo la arquitectura ROS utilizando MATLAB como entorno para dicha programación y para la comunicación.

Palabras clave: robot, pick&place, detección de color, toma de decisiones, inteligencia artificial, ROS, MATLAB.

1.3. Motivation

Robotics is a huge and interesting world in which we have learned a lot and still have a lot more to learn. Robots are very useful in many parts of our lives, carrying tasks that we do not want, or just cannot perform, making our life easier and granting also economic benefits. [1-4]

I have worked before with robots but in a "safer" environment with less programming complexity and I really liked it. But I knew that there are more things that can be done with robots. So, with this project, I decided to tackle this world more ambitiously and start learning more about hardware, software, and procedures to interact with robots and make them useful in every way possible.

The Centre for Automation and Robotics Research (CARR) at the University of Sheffield Hallam is a great place to start learning. I have the opportunity to work with a mobile robot manipulator and deepen my knowledge of ROS (Robotic Operating System). Some steps may require some great effort as it may be the first time using some software/hardware, but it will sure be worth it.

2. Objectives

Before starting, the objectives, goals, and features that our final pick&place application must have or accomplish must be set.

First, since this application will be used in an industrial environment, some problems may be encountered there and must be considered. Although the robot will not be working near other operators, it must have some security measures and its paths must be smooth enough so it cannot hurt other tools in the working area or even itself.

Now, with the security issues solved, the application should be as efficient as it could using the resources available. Unnecessary moves should be avoided and lose the less time possible in every step of the work.

Moreover, as the robot is complex with many different tools installed, the application should be developed taking into consideration that it could be applied in different industrial environments and could be useful in lots of them.

Finally, this project represents the Master Dissertation done by the researcher at the Sheffield Hallam University as part of an Erasmus Exchange from the *Universidad Politécnica de Valencia*, finishing with it my last year in the *Master en Ingeniería Industrial.*

3. Literature Review

First, there must be done a literature review of the current situation at the industry and learn from other researches what have been done in industrial robotics.

3.1. Robots vs. Humans

At the beginning of the history of the industry, humans were the ones that did most of the repetitive and less specialised work exposing themselves not only to injures because of the (sometimes heavy) machinery that was used, but also because humans got tired and their accuracy was reduced after working for a long period of time. [5]

Many years later, after some industrial revolutions, this fact triggered the beginning of the creation and programming of other machines (robots) that could take on those jobs. These lead to a massive increase in efficiency and become a revolution in industry as many authors have described in their researches in this area. [6]

Nevertheless, even nowadays, some processes are still be done by humans entirely. This is inefficient and sometimes potentially harmful. These processes are the perfect candidates for an upgrade.

Depending on the nature of the process there are many ways to improve it. If the process is simple enough, the human operator could be removed and instead a robot can take place to perform the task in a programmed way, and/or let the human controls when this robot executes its program.

However, there are processes that still need the presence of a human. This is mostly because automatisation is not powerful enough or because the nature of the process involves a task that only can be performed by humans. In this case, the solution may go by using collaborative robots that interact with the human in a safe way and help them get the job done. [7-8]

One example of a process that can be improved, and is related to the project here, is a cheese industry in which one of these pick&place operations was performed. Here, the human operator, picked pieces of three kinds of cheese and had to decide where to put them differentiating the cheese by its kind. This process involves two key concepts: (1) identify things and (2) place them based on a feature they have.

This example is not isolated and there are a lot of other places where a solution to it could be applied with minor changes. So that is why solving this problem could lead to an improvement in industry.

3.2. Related researches

Pick&place applications are not new in robotics and there are a lot of robots (even more simple that the one proposed in the introduction) that can perform the task given the order. However, these robots have their program fixed in terms that they cannot change the path or the destination. They have no intelligence and cannot decide for themselves.

There is a lot of work in this area as it is the most common starting point for every student that begins learning robotics. A robotic arm as the one showed below is the most common tool to perform these operations.

Figure 1. IRB 140 from ABB [9]

On the other hand, there is machine vision. Machine vision is a more sophisticated field in which there is still some work to do and it covers a wide range of applications: from printing images, to helping autonomous cars, performing colour detection in the in most of applications.

In relation to the project and to highlight the importance of machine vision, there must be mentioned some other research works done by students at the Centre for Automation and Robotics Research (CARR) at the University of Sheffield Hallam.

In the project carried out by Sohail Saeed, machine vision was used to allow a collaborative robot to work with a human operator detecting where the operator was and avoiding hitting them [10].

This is a clear example that the pick&place operation can be improved in great ways with information received from the outside.

There is also the project of Armando Arturo Sánchez Alcázar in which a transport system based on a mobile manipulator was designed to help the nurses work at a hospital. [11]

The work proposed a machine vision system to help the robot guide itself through the hospital and reach required destinations.

With that being said, it can be concluded that the combination of manipulation and machine vision has got many possibilities and applications. With this idea in mind and focusing just on the industry, our goal will be to design a more generic application that will use information from the outside to perform the desired actions. Also, we will aim to a generic solution that could be applied in many situations with a few changes in the parameters to adapt to the objects and the environment.

3.3. Other Solutions

Making use again of the example of the cheese industry, the goal of automatizing the cell of the process, which makes the selection of the cheese, could be reached by some different paths and they are going to be discussed here.

In this project, the use of a Fetch Robot is proposed that combines the robotic arm to perform the pick&place operation and the machine vision system through a camera to identify the objects. But there are other possibilities.

First, both modules of operation (pick&place module and machine vision module) could be separated; having a machine vision system that tracks the objects and a robotic arm that picks them at the position ordered by the camera. This is a solution that has been used in a sweet factory in Spain. [12]

Figure 2. Pick&place of croissants [12]

This solution has the advantage and disadvantage of having two systems. They must interact with each other which adds another problem to solve. But on the other hand, as they are separate, one camera could interact with more than one robotic arm (lifting the budget, of course), giving flexibility that could patch the connectivity problems.

Another solution that could be valid for certain production distributions would be to introduce a mechanism that drives the object into the desired box while it is still going on the conveyor. This solution would work if the objects are being put into a box but without order, just tossing them there, or just accumulating them in a platform for being processed later by another cell. In this case, we would not need an arm to properly pick up them, just a way to divert the flux of objects.

Again, a tracking system using a camera would identify the object and giving the order to deliver the object into one path or another [13]. The figure below illustrates an example of this tracking system:

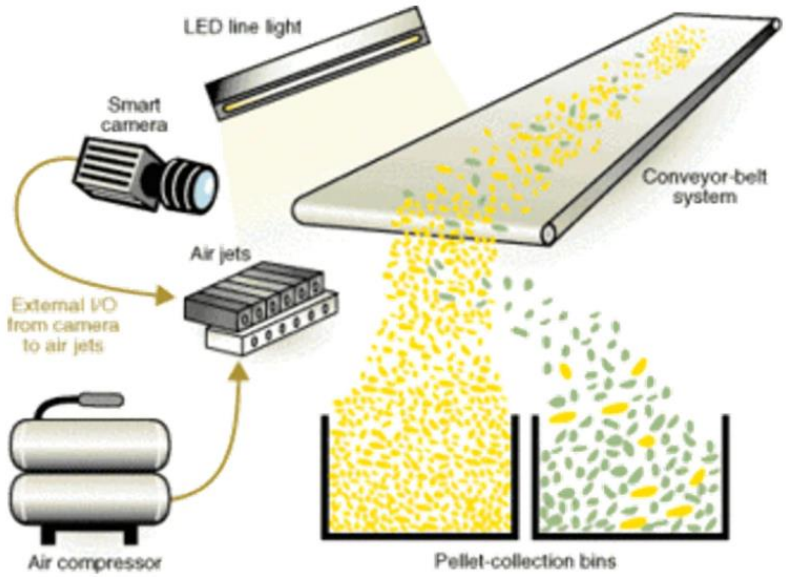


Figure 3. Example of a tracking system [13]

As it is been said, this system, while it is cheaper in some scenarios, can only be applied in those scenarios, so it is not generic enough for the scope of the project.

4. Tools Review

4.1. Programming and Software

This section will describe all the software needed in this project.

4.1.1. Robot Operating System (ROS)

Robot Operating System (from now on, ROS) is a flexible open source software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms.

It does not replace the operating system but instead, it works alongside it and provides services, such as low-level device control, hardware abstraction layer, message-passing between processes, implementation of commonly used functionality, and package management. [14-17]

This architecture works in distributed computation. It is a great advantage as robotic systems tend to rely on multiple software processes that are run in multiple computers with different operation systems. Also, this software allows to have a user interface in a mobile device, laptop or computer that communicates with the robot and it is an extension of the robot's software.

ROS can be used for a variety of styles: mobile robots, swarm robots, robotic arm manipulators and manipulator combined with a mobile base.

For this project, the distribution used is ROS Melodic. To be installed, the tutorials in the official website have been followed. Below is a brief overview of the different parts of this software.

ROS Master Node

The ROS Master Node is responsible for providing registration and naming services to the rest of the nodes in the robot operating system. It keeps track of subscribers and publishers to topics as well as services and actions, which is described later. The main job of the ROS-Master Node is to enable each node to locate one another. Once they locate each other, then the communication is set peer-to-peer as shown in figure no 1.

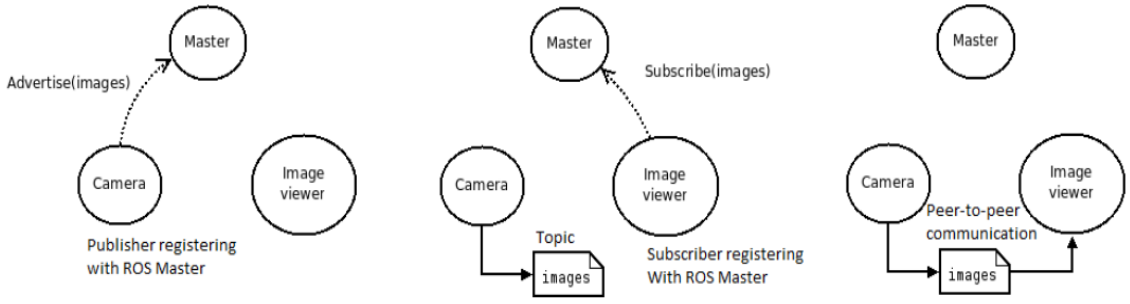


Figure 4. ROS Master Node [15]

ROS Node

A node is a single process that performs computation. All nodes are combined into a graph and they communicate with each other via topics, services, and action services.

As a node typically controls a single part of a robot system, robot systems normally have got many nodes that may communicate between them.

As the full system is divided into many nodes, it gives fault tolerance to the system if one node crashes, as the whole system will keep on working or even know the error and stop before hurting anyone. This helps maintain a high level of security in robot manipulators for example.

ROS Topics, Publishers and Subscribers

Topics are the channels of communication where nodes can exchange information with each other. Generally, the nodes are unaware of 'whom' they are talking to. Publishers generate data and publish them to relevant topic and subscribers subscribe to the relevant topic to receive those data, but they do not necessarily know what is sending or receiving the information. To a single topic, multiple publishers can publish data and several subscribers can access that data.

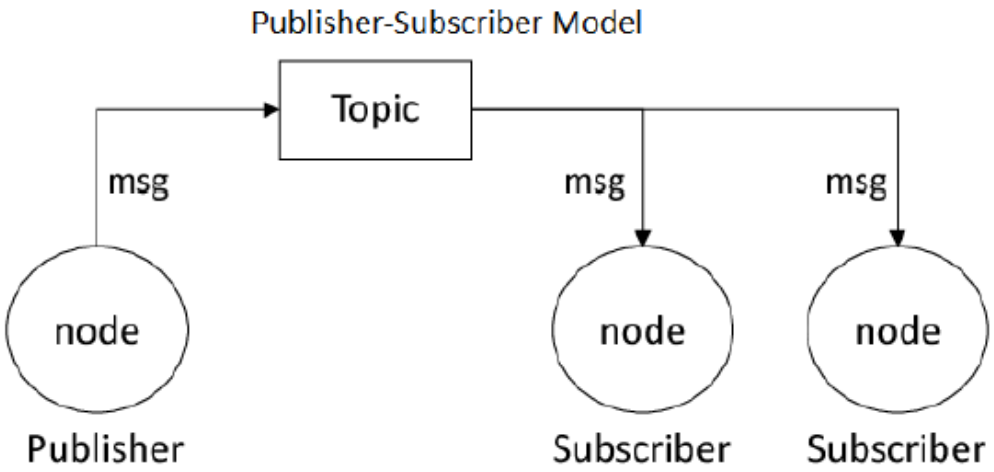


Figure 5. Publisher and Subscriber Graph [15]

ROS Service, Service Server and Client

Publisher-subscriber model is a flexible and robust communication model; it is one-way many-to-many communication. But it is not appropriate for a remote procedure call, which is often required in a distributed system. Request and replying to this request are done via a "*Service"*, the message that is sent is divided into two parts: one for the request and one for the reply. Service Server is a node offering a service under a string name and can be called by a client node by sending the request message and waiting for the reply.

To summarize, *Services* can be taken as *Functions* like in Python, C++, or any other coding language. These are called by a node with a set of parameters and return their result to another node in the net.

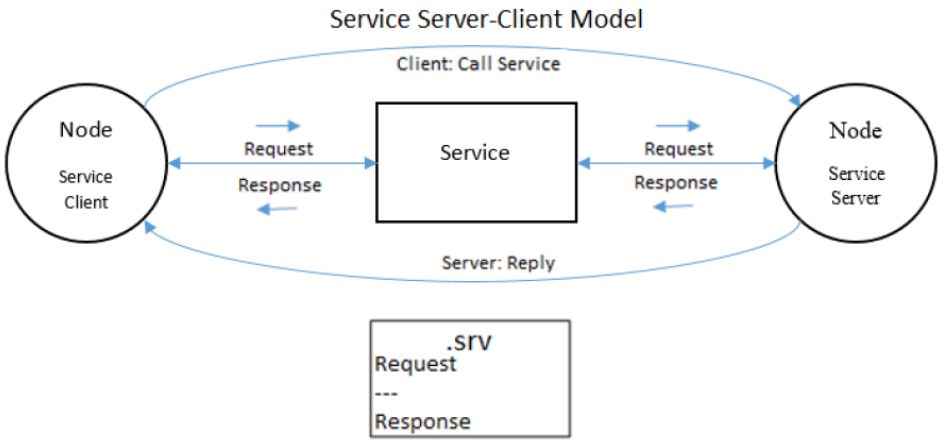


Figure 6. Services [15]

ROS Action, Action Server, and Client

Actions are like services in terms of functionality except actions are asynchronous while services are synchronous. The difference comes when talking about waiting for a response. When a node calls an action server, the caller does not necessarily have to wait for the action to complete, while, at services, the caller must wait until the service finishes.

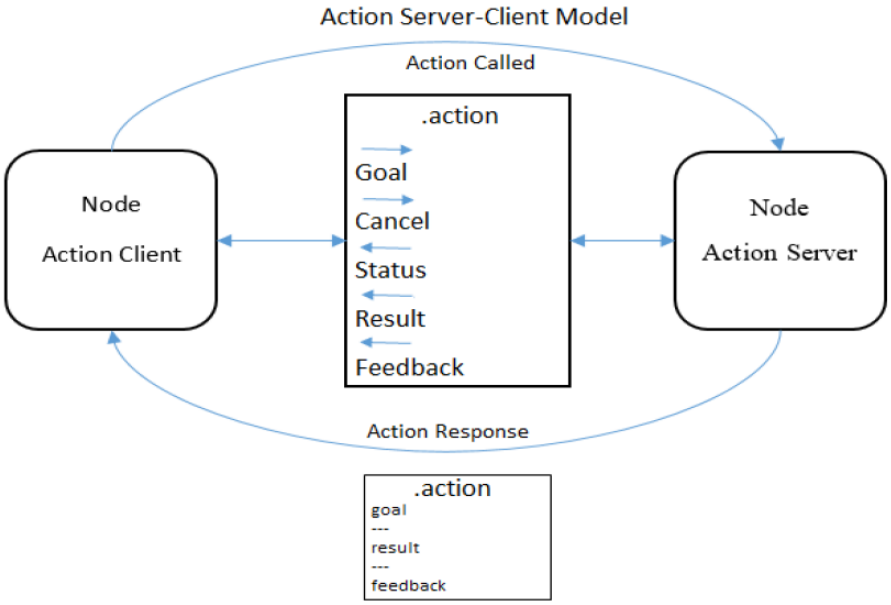


Figure 7. Actions [15]

Gazebo Package

Gazebo is a set of ROS packages that simulate an environment in which you can see the interaction of the robot with this environment. [18]

It is useful because it is a safe place in which the application can be developed and the robot movements be tested without the possibility of harming people or the robot itself. Moreover, in terms of topics, subscribers and publishers, the simulation uses the same names as it was the real robot, so any program wrote to be executed using Gazebo can also be executed with the real robot without needing to change anything.

4.1.2. MATLAB

MATLAB is an abbreviation of Matrix Laboratory and it is a high-level programming language and an interactive environment which is designed specifically for quick and easy scientific computations, visualisation, and programming of large amount of data. [19]

Matrix manipulations, data plotting, creation of user interfaces, algorithms implementation, interfacing with programs written in other languages (for example, Java, C/C++, Python), analysing data, algorithms development, creating models and applications are some examples of things that can be done in MATLAB. It has thousands of built-in functions for computations in a wide variety of scientific fields and lots of toolboxes developed and designed for specific research disciplines, including control system, neural networks, robotics, image processing, computer vision …

As for this project, MATLAB 2019 is used and there are a couple of toolboxes that are also needed to make the application itself and must be installed before commencing work.

- **Image Processing Toolbox:** allows to work with the images received from the robot's camera and processing them.
- **ROS Toolbox:** allows to make a node in MATLAB which can connect to the robot's master node in the same net and perform publisher and subscriber orders as well as action/server enquiries.

4.1.3. Virtual Box and Linux

Despite the advantages of ROS and Gazebo, it is only available to Ubuntu operating systems and it is still in development on Windows 10. Moreover, as MATLAB is better supported in Windows 10, there is the need of some way to connect both applications.

This can be solved using either two different PCs or using a virtual machine in only one PC to install a copy of Ubuntu 18.04 and ROS Melodic. The last option has been chosen as it is less dependent on different hardware, and a strong enough PC is available to support MATLAB executing and the simulation in a Virtual Machine.

Virtual Box is a free software that aids to create a virtual machine choosing the virtual hard disk space, the CPUs and RAM that the virtual operating system can use from the real PC. With this, it will be setup a nice working environment.

4.2. Human Vision

On this topic it will be exposed how human vision is, in order to better understand how we will approach machine vision in the next topics.

Human eyes are complex organs of the human body and allow to have light perception, colour vision and depth perception. These features make the human vision to be the best vision in the animal kingdom and it will be looked to have these features in our machine vision module but translated to a "machine kingdom".

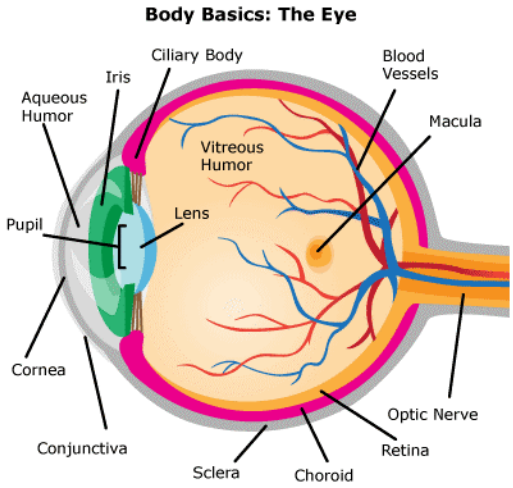


Figure 8. Human Eye. [20]

With human eyesight, cone cells are responsible for colour vision. Using the cone cells in the retina, humans perceive images in colour. Each type of cone specifically sees in regions of red, green, or blue, (RGB), in the colour spectrum of red, orange, yellow, green, blue, indigo, violet. The colours in between these absolutes are different linear combinations of RGB. [21]

RGB is the colour space used in the eyes and the most used to in common life (TVs, computer screens, and other electronics). However, later, its disadvantages are discussed later as well as how the colour space feature is employed in programming.

4.3. Machine Vision

This topic talks about machine vision and its implication in the project.

4.3.1 Introduction

Robots can do a variety of tasks much better than a human does, but, normally, this concerns repetitive tasks without any thinking or decision making involved. With machine vision, the goal here is to let the robot make the decision of where to put the object it has picked using the information it has seen through its camera.

As it has been said, humans have the best vision in the animal kingdom. They can interpret complex and unstructured scenes but with a limited speed, accuracy and, also, repeatability. These last features are no problem for a robot with the right technology (optics and camera resolution). These advantages add to the fact that robots could work in hazardous areas, so we are opening more possibilities.

4.3.2. Vision System Components

The term "right technology" is used in the later paragraphs. It will be now described and, with it, the different options to handle different environments and problems.

Lenses

Lenses are the main part of the system since it converges the reflected light onto the image sensor properly. They determine the resolution and the quality of the image captured. There are two main types of lenses: *converging* and *diverging*. There are also subtypes of lenses, but the important feature is how they treat the light through them which all share depending on the main type. Lenses are mounted and this mount can either be fixed or interchangeable.

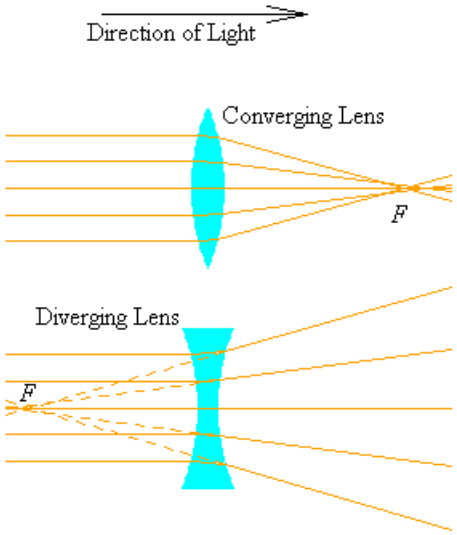


Figure 9. Converging and Diverging Lenses (Google Images)

Lighting

Lenses can direct light through them, but it is also important how are we throwing this light in the first instance. Ibn Al-Haytham corrected the old way of thinking and concluded that the human vision system produces an image by the light reflected from the object [22]. Machine vision system works in a similar way.

The two parameters that must be taken in account in lighting are the source of light and the relative placement of it, the object, and the camera. The lighting techniques used can also depend on the colour of light. Some lighting techniques are:

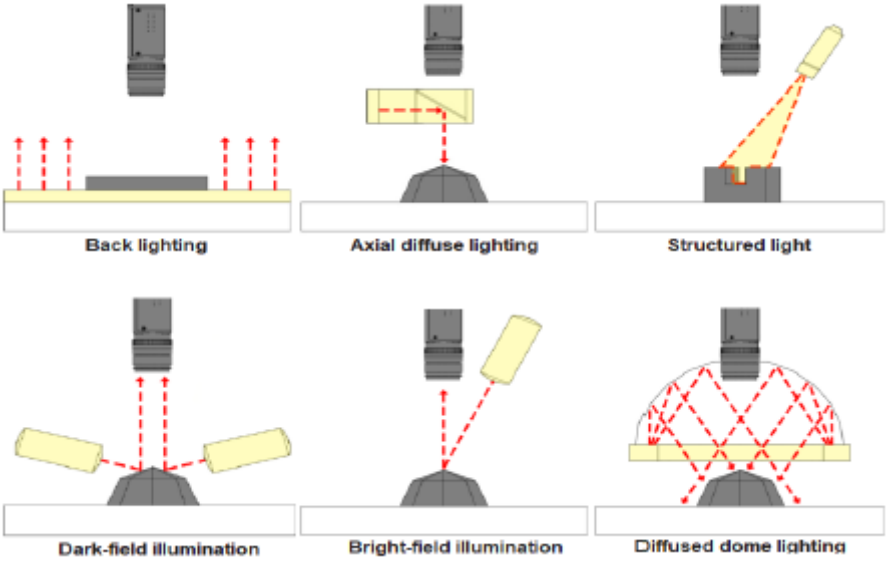
- Backlighting
- Axial-diffuse lighting
- Structured light
- Dark-field illumination
- Bright-field illumination
- Diffused dome lighting
- 

Figure 10. Lighting techniques [23]

Image Sensor

The image sensor is the part of the system that converts the photons to the electrical signal. Most used types of image sensor are complementary metal-oxide semiconductors known as CMOS or charge-coupled devices known as CCD. The key role of the image sensor is to convert the reflected light into a digital image.

An image is a collection of pixels and each pixel is composed of red, green and blue intensities in an RGB image. It is important to choose the right camera resolution since the higher the resolution, the more detailed the picture will get, but also more complex to process.

In this case, the resolution will be 480x640 pixels (3 layers for red, green and blue) as it is the resolution of the robot's camera. It is sufficient in terms of details.

Types of Machine Vision System

Regarding the dimension of the result, machine vision systems can be divided in 1D, 2D and 3D.

- **One-Dimensional System (1D):** scans a line of the full object, then they can be combined to make the full object image. Most inspection systems use 1D vision.
- **Two-Dimensional System (2D):** this is the most common type of system. It is just a simple camera like the one that our robot has got. It is the one that we are going to use.

Figure 11. Fetch's head [25]

- **Three-Dimensional System (3D):** this system consists in using more than one camera to get the RGB-D image of an object.

4.4. Colour Detection

In RGB images, a single image is composed by three layers of pixels representing each of the three primary colours (red, green, and blue). But this colour space is not the only one and it is not the one that it is going to be used in this project.

RGB colour space originated in colour television when Cathode Ray Tubes were used. It is a relative colour standard dependant on the display device. RGB colour representations assign a value from 0 to 255 to each of R, G, and B. [0, 0, 255] is pure blue, [0, 255, 0] is pure green, [0, 0, 0] is black. These colours are additive to form white, i.e. [255,255,255] is white. [24]

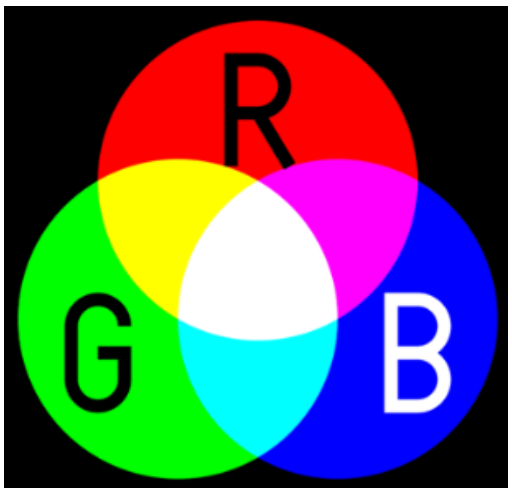


Figure 12. RGB (Google Images)

These spaces have got the advantage that they are linear and use a cartesian coordinate system, but it is difficult to name all perceivable colours and extract a conclusion from the pixels of an image.

On the other hand, the HSV colour space, unlike its former colleague, it is not linear. Each possible colour has three parameters:

- Hue, which is the dominant wavelength of the colour. It ranges from 0 to 360 degrees.
- Saturation, which reflects the "purity" of the colour. It ranges from 0% to 100%.
- Value, (or luminance) which is the brightness of the colour. It ranges from 0% to 100%.

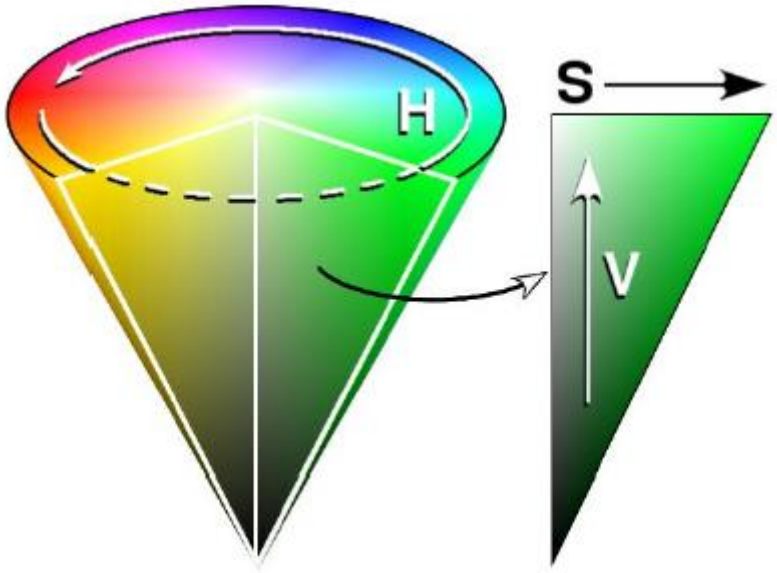


Figure 13. HSV colour space [24]

With this colour space the evaluation of which colour (in general terms) is the robot seeing it is sufficient to check just the "hue" parameter. Adding the "saturation" and "value" parameters allows to completely define the colour in every pixel.

RGB images can be converted to HSV using MATLAB. This is possible thanks to the functions from the Image Processing Toolbox exposed earlier.

For the purpose of this project, some colours are detected using only the "hue" parameter, i.e. red, orange, yellow, green, cyan, blue, purple, magenta, and pink. But also, black and white are detected comparing to the lower values of the other two parameters.

Here there is a flux diagram of the process to detect the colour of an image:

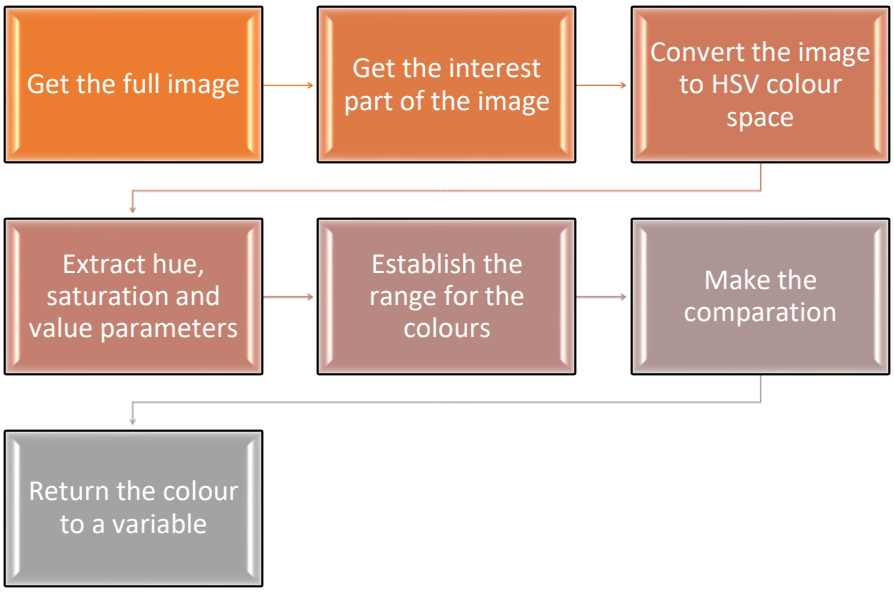


Figure 14. Flux diagram of colour detection

4.5. Hardware: Fetch Robot and Robot Movement

To accomplish the goal of this project, a Fetch Robot from Fetch Robotics is used. Here is an image of it.

Figure 15. Fetch Robot [25]

It is a mobile manipulator robot with several features. Extracted from the documentation of the robot [26], from bottom to the top of the robot, it has:

- A mobile base and a laser scan to make map planning and help avoiding obstacles.
- An arm with 7 DOF (degrees of freedom) and a gripper to pick up objects (6 kg of payload).
- A head, with a camera installed, that can move around the vertical axis and can also look up and down in a range of -90º to 45º.

However, it has got some peculiarities that must be considered when making use of it. As it is a robot that must be programmed, it does not know that its own arm may be sometimes blocking the view from the camera. Also, seeing the design, the control of the arm joints may be problematic because it can hurt itself during movement (especially at the start of it), so further care to avoid this is required when using the robot.

To solve these two problems, first, there is going to be a start routine that moves the arm in a way that does not harm the robot and gets it to an initial position. And second, also a movement planning that does not mess with the camera view during operation.

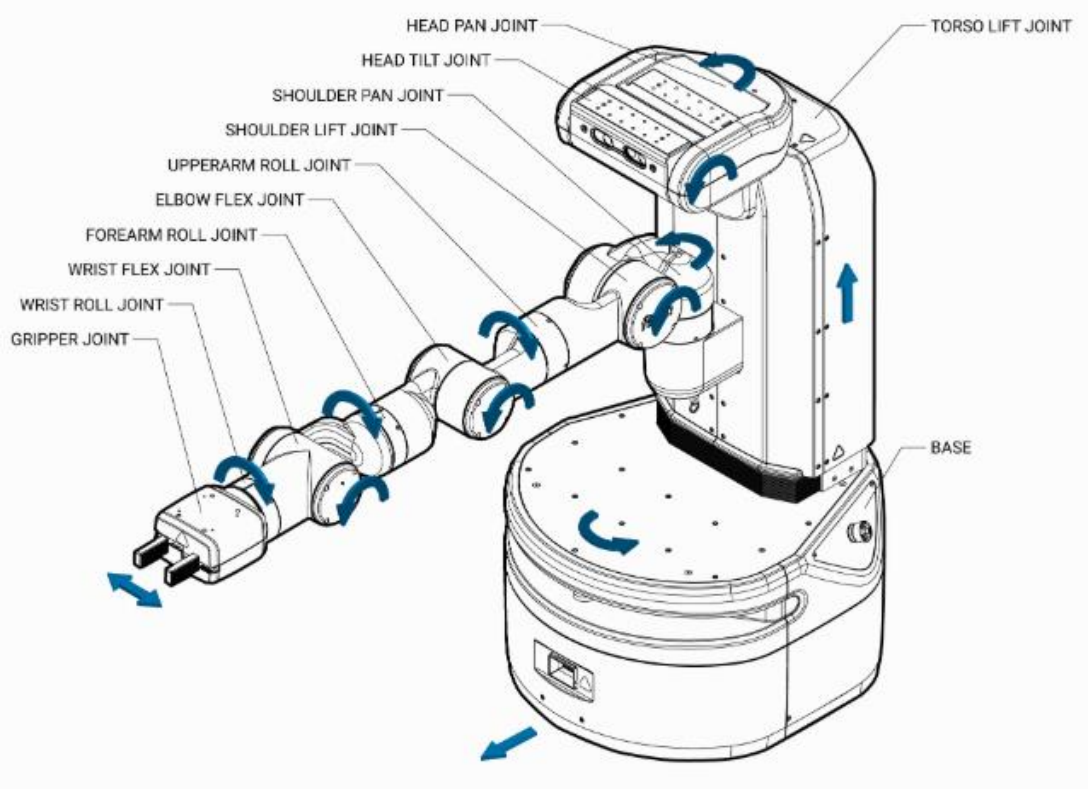


Figure 16. Fetch Robot Joints [26]

Joint	Type	Limit (+)	Limit (-)
"*"_wheel_joint	continuous	–	–
torso_lift_joint	prismatic	400mm	0mm
head_pan_joint	revolute	90°	90°
head_tilt_joint	revolute	90° (down)	45° (up)
shoulder_pan_joint	revolute	92°	92°
shoulder_lift_joint	revolute	87°	70°
upperarm_roll_joint	continuous	–	–
elbow_flex_joint	revolute	129°	129°
forearm_roll_joint	continuous	–	–
wrist_flex_joint	revolute	125°	125°
wrist_roll_joint	continuous	–	–
"*"_gripper_finger_joint	prismatic	50mm	0mm

Table 1. Joint Limits [26]

5. Methodology

In this section, the methodology followed to make the application is presented. The description covers all the steps involved starting from the setup of the simulation until how the final test with the robot is done and the programming involved in it.

5.1. Programming

In terms of programming, the subscriber/publisher architecture of ROS is used u. Some publishers are set in MATLAB in order to send the commands to the active parts of the robot (head, torso, arm and gripper) and some subscribers to retrieve the information from the camera and joint topics.

An advantage that can be found here is that topics (and their messages needed) do not change when addressing the simulation or the real robot, so anything built to work with the simulation will work fine on the robot. This helped at the beginning where further learning of ROS and its architecture was needed, so with the simulation began the tries of which instructions would work and which would not.

The full script can be checked at the end of the document in the appendix section.

5.1.1. ROS Topics

Head topic

The head is the less dangerous part of the robot as it cannot do much harm nor to people or objects around the robot or the robot itself. It is going to be moved down to achieve a better view of the object. New values are going to be sent for the angles of the two joints of the head.

When programming, in order to change the values of head joints, it has appeared the need of creating an auxiliary variable to initialise part of the message that will be sent to the robot.

Torso topic

The torso itself is not dangerous either, but as the arm could be extended, there must be care when moving this joint to avoid hitting anything with the arm. An auxiliary variable is also needed to set up the full message. Here, just set the parameter of the torso joint to be positive or negative results in the robot going up or down, respectively.

Moreover, the parameter "effort" is set to a relatively high value, so the torso moves gently and does not make the arm move too much in consequence.

Arm topic

The arm is, obviously, the most dangerous part of the robot if no safety measures are taken within the path planning.

The position of every degree of freedom (DOF) must be set. And, in order to avoid undesirable paths, the arm is going to be moved step by step, moving only a joint at the same time (with some harmless exceptions that have been largely tested). Thanks to the simulation, it has been possible to find the movements that accomplish the goal safely.

Camera topic

There are lots of different topics from where information about what the robot is seeing can be extracted. The differences affect mostly the format of the image extracted and how or by which programs can it be used.

As there is not a direct HSV image, the raw RGB image topic is chosen since it can be processed in order to get it in the HSV colour space.

Joint State topic

With this topic it is possible to get the state of every joint and notice whether its working as intended or not. This topic is not used in the final program as the robot should work automatically without too much supervision, but it can be accessed at any time for debugging purpose.

5.1.2. MATLAB environment with ROS

Starting Gazebo with fetch robot package automatically starts a master node as if the real robot were initialised so it is possible to connect to that node from MATLAB. Here, every instruction related to ROS is available thanks to the ROS Toolbox provided in MATLAB Apps.

Using instruction "rosinit" followed by the IP address allows MATLAB to create a node that will connect to the master node which is running in that address.

```
%% Connect to VM via Host-link (which is running a master node in Gazebo)
rosinit('http://192.168.56.101:11311')
```

Figure 17. Rosinit instruction

```
>> rosinit('http://192.168.56.101:11311')
Initializing global node /matlab_global_node_70367 with NodeURI http://192.168.56.1:49428/
```

Figure 18. Successful connection

Now, if browsing for topics, it appears which topics can be published in or subscribed to.

```
>> rostopic list
/arm_controller/follow_joint_trajectory/cancel
/arm_controller/follow_joint_trajectory/feedback
/arm_controller/follow_joint_trajectory/goal
/arm_controller/follow_joint_trajectory/result
/arm_controller/follow_joint_trajectory/status
/arm_with_torso_controller/follow_joint_trajectory/cancel
/arm_with_torso_controller/follow_joint_trajectory/feedback
/arm_with_torso_controller/follow_joint_trajectory/goal
/arm_with_torso_controller/follow_joint_trajectory/result
/arm_with_torso_controller/follow_joint_trajectory/status
/base_controller/command
/base_scan
```

Figure 19. Part of the topics we can access

5.1.3. MATLAB scripts

The application has been divided in three phases: initialisation, routine and shutdown. As for MATLAB, there have been developed three scripts that relate to that three phases.

The initialisation script is devoted to make the robot get to an initial position safely while setting up the publishers/subscribers needed for the architecture. The initial position is discussed later when talking about the simulation results.

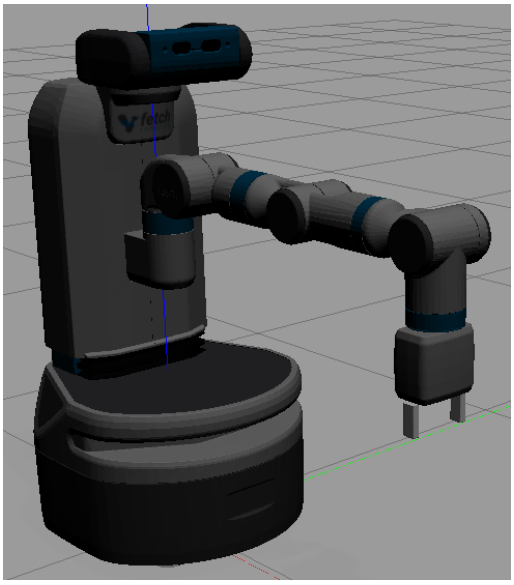


Figure 20. Initial Position

The routine script is the "loop" part of the application. This script has been created in order to be executed as many times as we need to pick objects from the pick position. The program starts from the initial position and returns to it at the end while placing the object in the correct place depending on its colour.

Finally, the shutdown script is meant to return the arm to the retracted position and stop communications with the robot cleaning also the workspace. When ending the work, this is the script that should be called.

5.2. Simulation

As told previously, the simulation is done in Gazebo using MATLAB for programming.

5.2.1. Goal of the simulation

Basically, the goal of the simulation is to verify that the robot is going to do what has been stated, and it is not going to hurt itself by doing it. The robot will start in a pre-pick position and, when ordered, it picks up the object, checks it out with the camera, then places it where the algorithm says and return to the initial position.

Within this simulation lots of path planning, movements and joint position have been tried and checked to avoid damage to the robot and its environment.

Below there is the flux diagram that the application should follow in and out of the simulation.

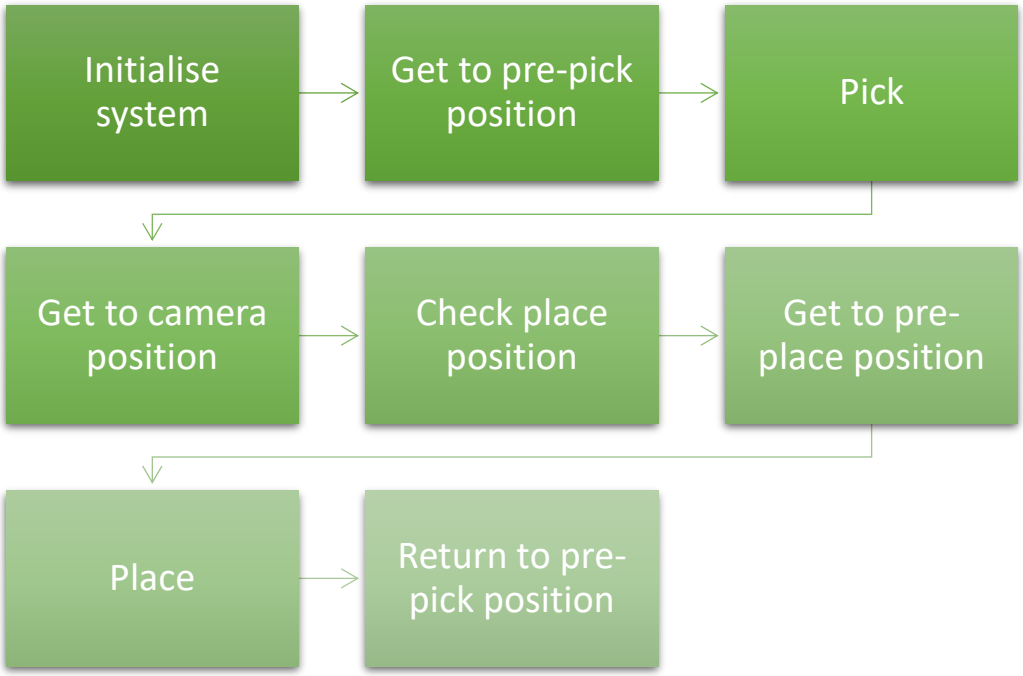


Figure 21. Flux Diagram of the Process

5.2.2. Setup of the simulation

The first step is to create a virtual machine and install both Ubuntu 18.04 and ROS Melodic, including the Gazebo packages which work with Fetch Robot.

The settings for the virtual machine used are the following:

- 40 GB of virtual hard disk
- 4 cores of the CPU (1,60 GHz per core)
- 4096 MB RAM (half of the total 8 GB of the PC)

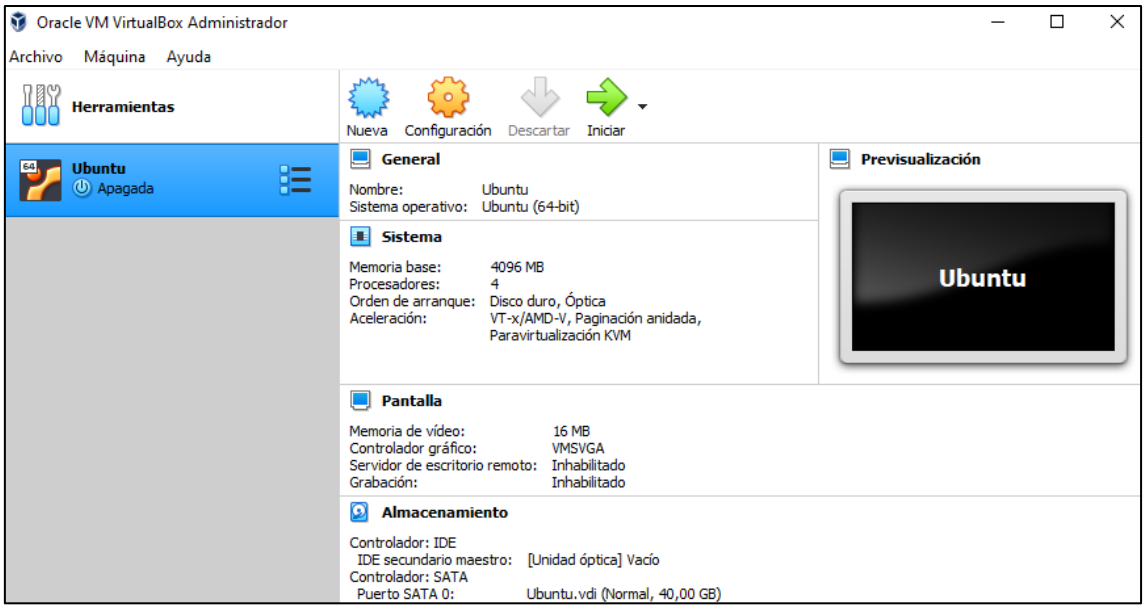


Figure 22. Virtual Box Main Page

Now, for MATLAB to communicate with ROS being executed inside the virtual machine there are additional configurations to make.

In net configurations, the communication to "Host-only adapter" must be set. This option makes the virtual machine to be in a subnet inside the PC, so it can connect now to other applications running outside the virtual machine, but it will not be able to connect to anything outside the computer. With this, the connection via IP address from MATLAB can be done.

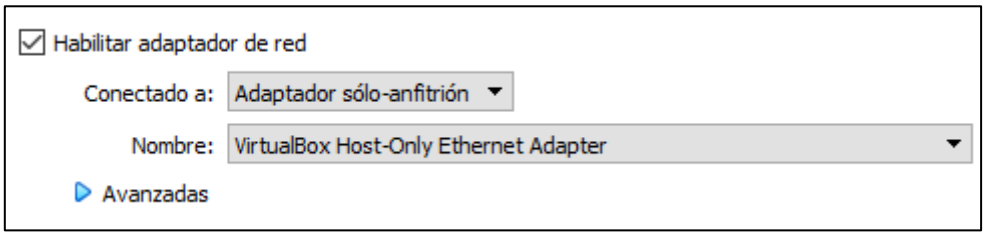


Figure 23. Net Configuration

After this, connection parameters can be double-checked using "ifconfig" in a Linux terminal and "ipconfig" in Windows command terminal.

Figure 24. Configuration in VM (Ubuntu 18.04)

```
Adaptador de Ethernet VirtualBox Host-Only Network:

   Sufijo DNS específico para la conexión. . :
   Vínculo: dirección IPv6 local. . . : fe80::a0de:6609:b43:b3d5%19
   Dirección IPv4. . . . . . . . . . . . : 192.168.56.1
   Máscara de subred . . . . . . . . . . : 255.255.255.0
   Puerta de enlace predeterminada . . . . . :
```

Figure 25. Configuration in Host Operating System (Windows 10)

The Gazebo environment has been a fundamental part of the simulation. Two "launch" files have been used to make the testing of the movement: "*simulation.launch*" and "*playground.launch*". In the former there is an empty environment with just the Fetch Robot while in the last one there are also other objects the robot may interact with.

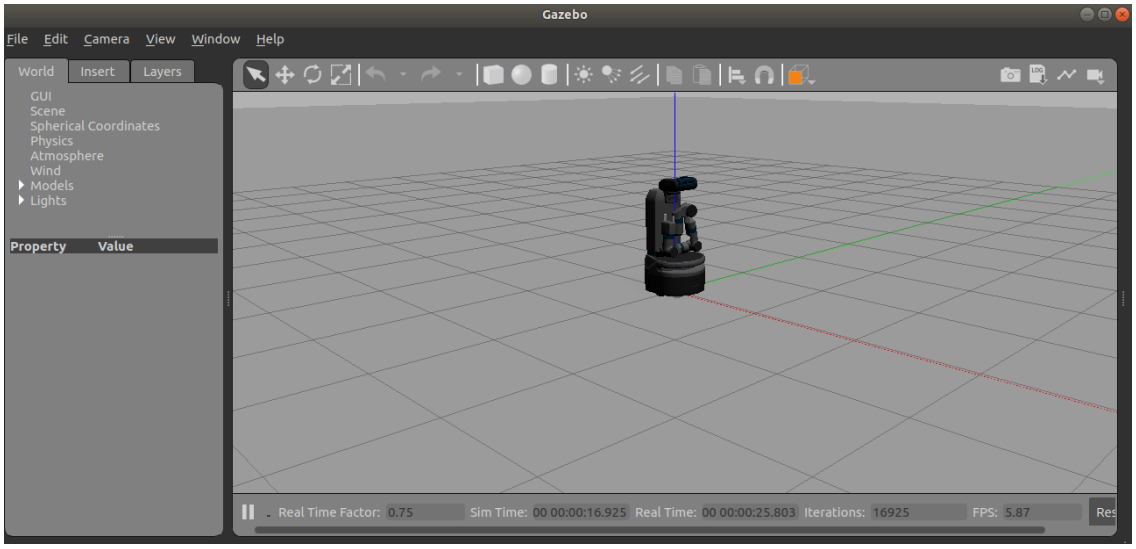


Figure 26. Gazebo simulation.launch

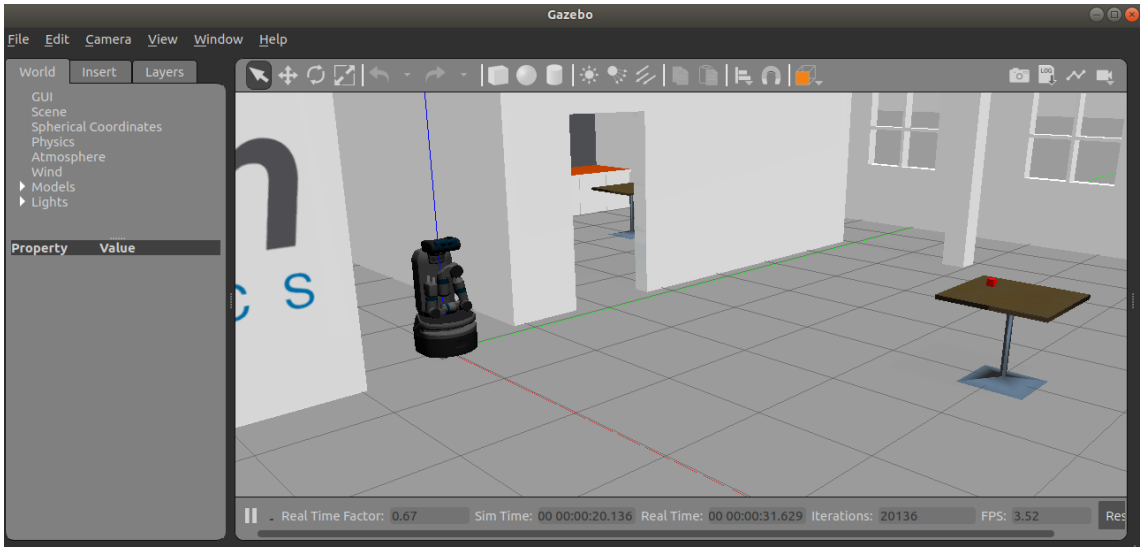


Figure 27. Gazebo playground.launch

Now everything is set to start MATLAB and begin working with topics in ROS.

5.3. Real test

To carry out the real test, first, there must be set a safe place in which the robot can start working and will not harm anything.

Robot's arm measures 1 m fully extended from its torso and the pre-pick position is 70cm from the torso. Taking this into account, as the arm may move near the full extension, enough space must be disposed for the robot to move freely even if the pick position is nearer than the full extension.

In the end, the safe place was set in the Robotics Lab since the robot was already there and there was enough space to proceed with the test without harmful collisions.

Figure 28. Acknowledging the safe place for the test

The test objects were some prismatic polyhedron made from plastic. Some of their faces were covered with different colours to test the colour detection feature of the robot. These objects have been chosen taking in account their size for the robot to be able to pick them easily with its gripper.

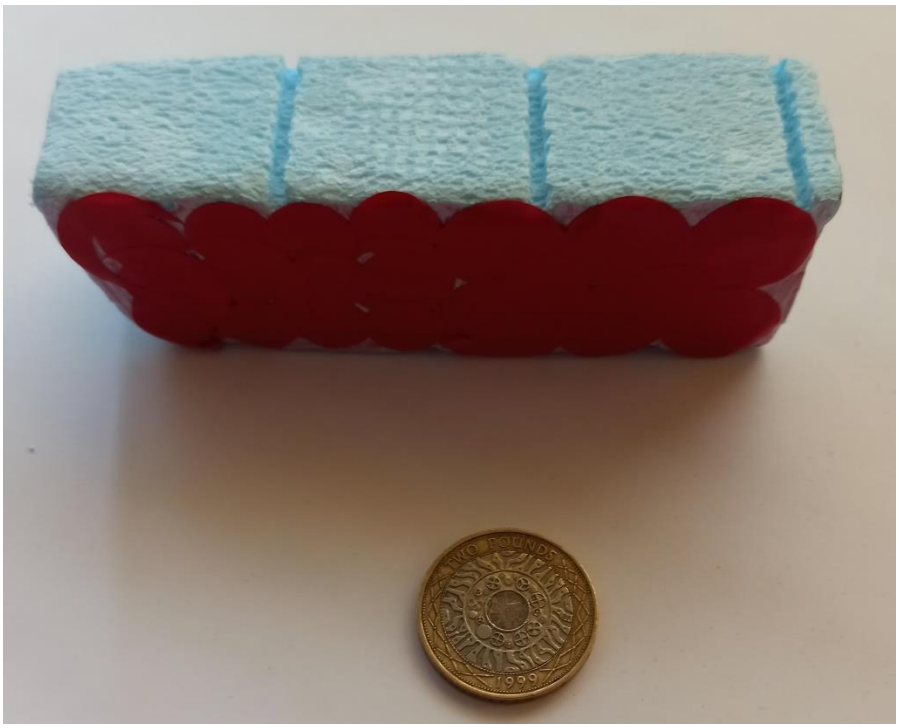


Figure 29. Prismatic piece vs Two pounds coin.

In the above figure there is an example of a test object. Its faces are blue but one that was covered in red with stickers.

Figure 30. Fetch Robot with the prism picked up

6. Results

6.1. Simulation results

Thanks to the simulation, better understanding of the robot's movement has been achieved. Also, it has also led to the discovery of which paths are better in order to avoid self-harming and fulfil the robot's duty at the same time.

At the beginning, the simulated robot started moving one joint at a time following the orders allowing to learn which paths were better to develop later. Then, it came up with the idea to combine joint movements to increase efficiency but keeping safety.

The starting position of the robot that comes by design is problematic. It is necessary to understand that joints in that position are not in the "zero radians" position, but they have another value.

Through a couple of tests, the joint states were established and there was managed to get the arm to a better initial position. In this position, the robot has the arm almost fully extended except for the last two joints that are prepared to pick an object from the table.

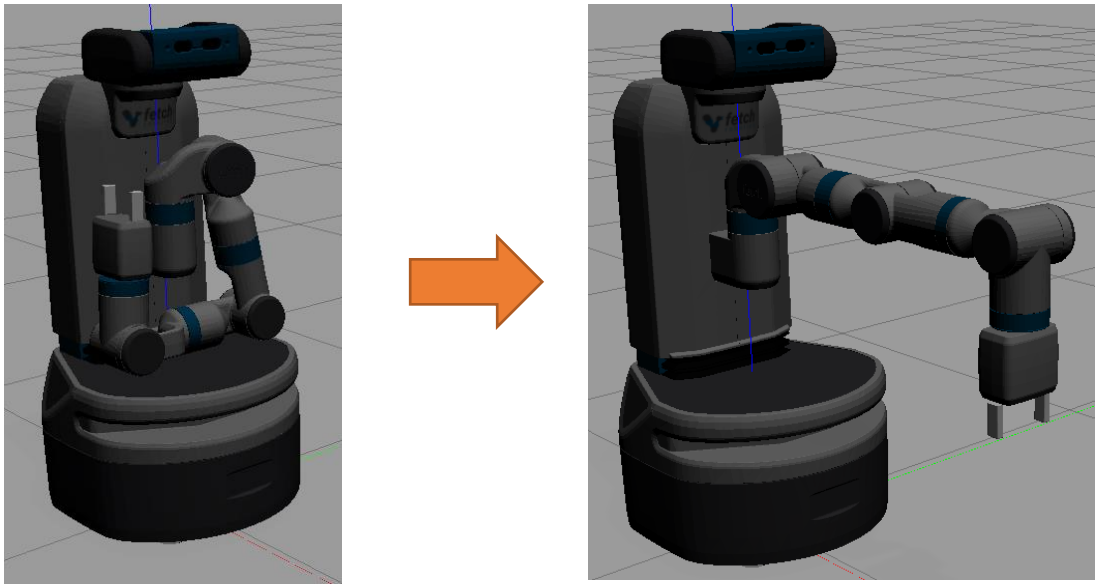


Figure 31. Power On (left) – Pre-pick position (right)

The torso joint is used to let the robot reach advantageous pre-pick positions. The height can be adjusted to make the robot pick anything up to 40 cm from its base (measured from the top part of the base).

After this, testing was continued to know how the object could be held by the robot. This is important so the object could be seen by the camera without interruptions in the line of sight. Head joints were also used to help get a reasonable position that does not make the arm to move too much. Finally, the object reaches around 10mm from the camera which gives enough image quality in the postprocessing step.

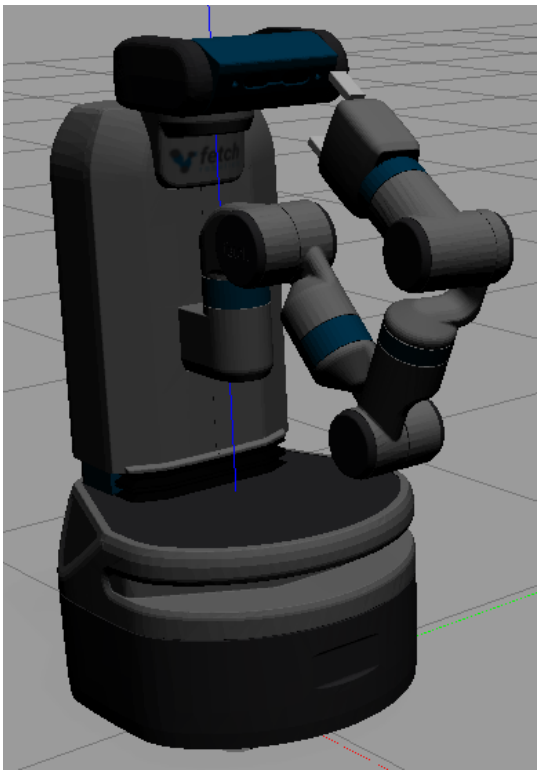


Figure 32. Camera position (head moved)

Finally, place positions are arbitrary in some way since the application can be used in many fields with different spatial conditions. For this project, two different place positions at both sides of the robot were chosen; with the arm almost fully extended. The reason of this is to test the limits of the arm and the robot itself without using its mobile base.

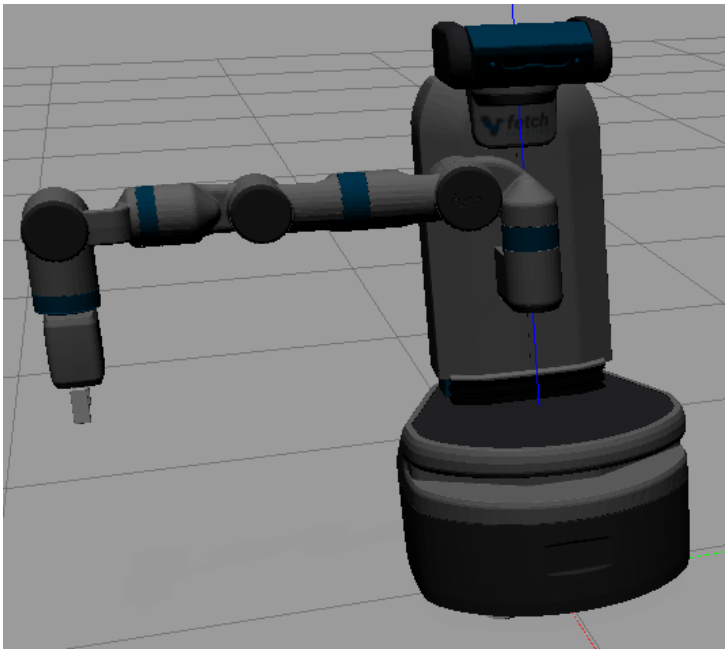


Figure 33. Place position (right of the robot)

6.2. Simulation full programs

Two full programmed tests have been made.

In the first one, the robot will pick a single cube and deposit it in one of the boxes (right or left) depending on its colour. This is the simplest use of the application and needs no change.

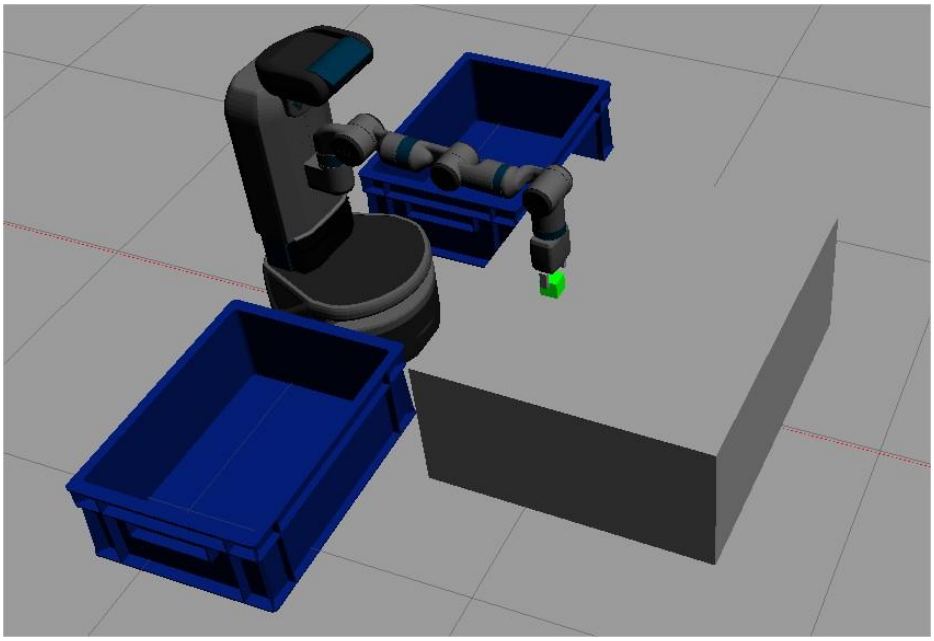


Figure 34. First program. Single cube

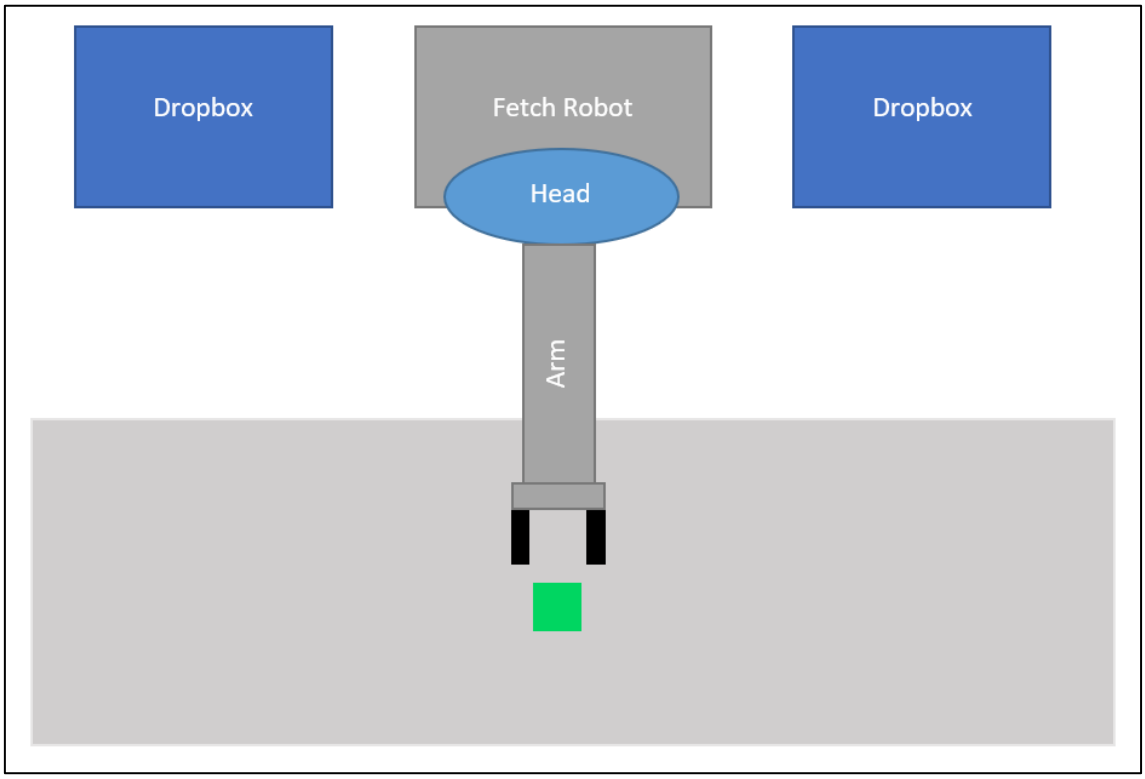


Figure 35. Aerial schematic view. Single cube.

In the second one, a different scenario was created. In this scenario the robot must perform the pick&place task four times to place four cubes that are on the table. The robot must know the cubes' position and it will pick and place one by one, repeating the "routine script" four times, and then shutdown.

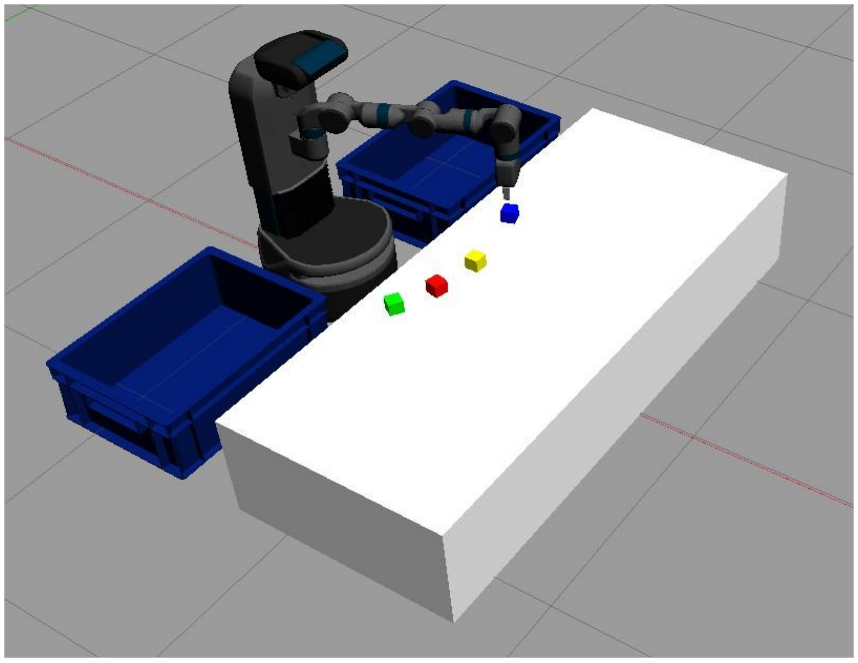


Figure 36. Second program. Four cubes

6.2. Real tests results

There has been a lot learned after doing some tests in the Gazebo simulation. However, at this point, there was still the need to see tests in real applications and take a closer look of how well it is working.

The first thing that came up during the tests was that some movements were too fast and too strong that the robot was having a hard time using its compensators to perform these movements. This gave the idea to change some parameters in the code.

The "effort" parameter in the arm message is responsible for this. If it is increased to high levels, the robot moves more gently and avoids harming the object.

Also, regarding the object, the gripper suffered from this as well. If the effort parameter was left to a low value, the gripper put too much pressure on the object. So, it was needed to increase its value again.

This test is mandatory for working with fragile objects since they could need different effort values to be picked firmly but also gently, avoiding breaking them.

Real tests also provided a simple and efficient way to test working heights the robot could reach.

7. Conclusions

At the beginning of this document, it was indicated that the aim was to create a pick&place application which would let the robot decide where to place the objects. At this point that goal has been fulfilled. And through this project, there have been learnt lots of useful tools that could be used in future projects working with robots.

From the first steps of the simulation to the final real test, it has been aimed to be as efficient and safe as it could. In order to reach a good result, it has been key to balance the different goals but also to keep in mind the possible industry in which this application could have its place.

The application has been designed using both common tools in robotic programming: MATLAB and ROS architecture. It has been designed to be flexible to the situation, the objects, and the hardware used. The application can be easily adapted if using the same architecture.

Thanks to the simulations and the real testing it has been able to produce a high-quality application despite the limitations in software or hardware.

8. Future work

In this project, a versatile application that can work in many different industries has been made. But, with more time, there are some other features that can be added to this application to make it even better and self-sufficient.

For now, the application works under instruction. A person or another machine must send the order to pick the object. But the application could be made to recognise when it should start working, so it would not need external aid to work properly. The robot's camera would be used to detect the object to be picked and work standalone.

Apart from that, with further research, the artificial intelligence of the robot could be improved to consider more characteristics from the object when classifying it, such as shape or even texture using deep learning.

Following the beginning of this project, robotics is a huge world in which humans have still a lot to research, and doing so, lives would be easier.

References

[1] Article: Multiclass object classification for retail products - https://towardsdatascience.com/multi-class-object-classification-for-retail-products-aa4ecaaaa096

[2] Article: Benefits of Robots - https://www.robots.com/articles/benefits-of-robots

[3] Article: Advantages and Disadvantages of Industrial Robots - https://www.plastikmedia.co.uk/advantages-disadvantages-of-industrial-robots/

[4] Article: Benefits of Automation - https://www.productivity.com/benefits-of-automation/

[5] Rosen, William (2012). The Most Powerful Idea in the World: A Story of Steam, Industry and Invention. University of Chicago Press. p. 149. ISBN 978-0-226-72634-2.

[6] Article: On Humans, Robots and the Future of Work - https://www.industryweek.com/technology-and-iiot/article/22027828/on-humans-robots-and-the-future-of-work

[7] ABB *(Company)* - https://new.abb.com/es

[8] Human–robot collaborative assembly in cyber-physical production: Classification framework and implementation , Xi Vincent Wang, Zsolt Kemény, József Váncza, Lihui Wang

[9] CLASSIFICATION OF COBOTIC SYSTEMS FOR INDUSTRIAL APPLICATIONS, Théo Moulières-Seban, Jean-Marc Salotti, Bernard Claverie, David Bitonneau

[10] Confined Configuration Space for Collaborative Robots, S. Saeed 2020

[11] Design of the implementation of a collaborative robot for optimising the internal flow of equipment and materials in the healthcare industry, A. A. Sánchez Alcázar, 2019

[12] Article: Los robots industriales irrumpen en el sector de la alimentación y bebida, November 2010 - https://www.youtube.com/watch?v=8G59zTXVHHU

[13] Article: Custom cameras classify plastic pellets precisely - https://www.vision-systems.com/cameras-accessories/article/16738359/custom-cameras-classify-plastic-pellets-precisely

[14] General information about ROS: URL: ros.org/about-ros/

[15] Material to learn ROS: Udemy course: "Programming Robots with ROS"

[16] Material to learn ROS: Course from ETH Zürich - https://rsl.ethz.ch/education-students/lectures/ros.html

[17] ROS by Example. A Do-It-Yourself Guide to Robot Operating System, R.Patrick Goebel

[18] Gazebo tutorials: http://gazebosim.org/tutorials?tut=ros_overview

[19] MATLAB https://es.mathworks.com/products/matlab.html

[20] Human Eye image and documentation https://kidshealth.org/en/kids/eyes.html

[21] Human Eye learning https://courses.lumenlearning.com/boundless-physics/chapter/the-human-eye/

[22] Ibn al-Haytham. (2019, 10 1). Retrieved from Encyclopaedia Britannica: https://www.britannica.com/biography/Ibn-al-Haytham

[23] Cognex *(Company)*, 2016 - https://www.cognex.com/es-es/what-is/machine-vision/components/lighting

[24] Machine Vision Notes. Image Processing, L. Alboul 2020

[25] Fetch Robot from Fetch Robotics: https://fetchrobotics.com/robotics-platforms/fetch-mobile-manipulator/

[26] Documentation of the Fetch Robot - https://docs.fetchrobotics.com/robot_hardware.html

Appendix

Here are presented the three algorithms developed that make the pick&place application as has been explained in the corresponding section of this document:

Process Initialisation

```
%% Initial configuration %%

%% Connect to ROS master in fetch robot from Matlab %%
rosinit('http://192.168.0.3:11311') % 192.168.0.3 fetch54

%% Or connect to ROS master in simulation %%
rosinit('http://192.168.56.102:11311') % 192.168.56.102

%% Init Subscribers %% (head camera and joint states)
camera_sub = rossubscriber('/head_camera/rgb/image_raw');
joint_states = rossubscriber('/joint_states'); % In case we need to
know the states during the process

%% Init Publishers %% (head, arm, torso and gripper controllers)
move_head_publisher =
rospublisher('/head_controller/follow_joint_trajectory/goal','control_
msgs/FollowJointTrajectoryActionGoal');
move_arm_publisher =
rospublisher('/arm_controller/follow_joint_trajectory/goal','control_m
sgs/FollowJointTrajectoryActionGoal');
move_torso_publisher =
rospublisher('/torso_controller/follow_joint_trajectory/goal','control
_msgs/FollowJointTrajectoryActionGoal');
move_gripper_publisher =
rospublisher('/gripper_controller/gripper_action/goal','control_msgs/G
ripperCommandActionGoal');

%% Move head so camera can see objects %%
head_move = rosmessage(move_head_publisher);

joint_send = rosmessage('trajectory_msgs/JointTrajectoryPoint'); % Aux
variable
joint_send.Positions = zeros(2,1);
joint_send.Velocities = zeros(2,1);
joint_send.Accelerations = zeros(2,1);
joint_send.Effort = zeros(2,1);
time = rosduration(1,0);
head_move.Goal.Trajectory.Points = joint_send;

head_move.Goal.Trajectory.JointNames = ["head_pan_joint";
"head_tilt_joint"];
head_move.Goal.Trajectory.Points.Positions = [0.0; 0.48];
head_move.Goal.Trajectory.Points.TimeFromStart = time;
send(move_head_publisher,head_move);

%% Setting torso initial position %%
torso_move = rosmessage(move_torso_publisher);

joint3_send = rosmessage('trajectory_msgs/JointTrajectoryPoint'); %
Aux variable
joint3_send.Positions = zeros(1,1);
```

```
joint3_send.Velocities = ones(1,1)*0.1;
joint3_send.Accelerations = zeros(1,1);
joint3_send.Effort = 500*ones(1,1);
time = rosduration(1,0);
torso_move.Goal.Trajectory.Points = joint3_send;
torso_joint_names = "torso_lift_joint";
torso_move.Goal.Trajectory.JointNames = torso_joint_names;
torso_positions = 0.2; % Positive = up, Zero/Negative = down (max 0.5)
torso_move.Goal.Trajectory.Points.Positions = torso_positions;
torso_move.Goal.Trajectory.Points.TimeFromStart = time;
send(move_torso_publisher,torso_move);

%% Setting arm message conditions and pickup position from initial
fetch position %%
arm_move = rosmessage(move_arm_publisher);

joint2_send = rosmessage('trajectory_msgs/JointTrajectoryPoint'); %
Aux variable
joint2_send.Positions = zeros(7,1);
joint2_send.Velocities = 0.01*ones(7,1);
joint2_send.Accelerations = zeros(7,1);
joint2_send.Effort = 50000*ones(7,1);
time = rosduration(1,0);
arm_move.Goal.Trajectory.Points = joint2_send;
arm_joint_names = ["shoulder_pan_joint", "shoulder_lift_joint",
"upperarm_roll_joint", "elbow_flex_joint", "forearm_roll_joint",
"wrist_flex_joint", "wrist_roll_joint"];
arm_move.Goal.Trajectory.JointNames = arm_joint_names;
arm_move.Goal.Trajectory.Points.TimeFromStart = time;

% Step by step arm movement
% arm_positions = [deg2rad(90); deg2rad(90); deg2rad(0); deg2rad(90);
deg2rad(0); deg2rad(90); deg2rad(0)];
% arm_move.Goal.Trajectory.Points.Positions = arm_positions;
% send(move_arm_publisher,arm_move); % 1 % Initial Position
% pause(4)
% arm_positions = [deg2rad(90); deg2rad(90); deg2rad(-90);
deg2rad(90); deg2rad(0); deg2rad(90); deg2rad(0)];
% arm_move.Goal.Trajectory.Points.Positions = arm_positions;
% send(move_arm_publisher,arm_move); % 2 % Can be skipped
% pause(4)
% arm_positions = [deg2rad(0); deg2rad(90); deg2rad(-90); deg2rad(90);
deg2rad(0); deg2rad(90); deg2rad(0)];
% arm_move.Goal.Trajectory.Points.Positions = arm_positions;
% send(move_arm_publisher,arm_move); % 3 % Can be skipped
% pause(4)
arm_positions = [deg2rad(0); deg2rad(90); deg2rad(180); deg2rad(90);
deg2rad(0); deg2rad(90); deg2rad(0)];
arm_move.Goal.Trajectory.Points.Positions = arm_positions;
send(move_arm_publisher,arm_move); % 4 %
pause(4)

%%%%%%%%%%%%
% arm_positions = [deg2rad(0); deg2rad(90); deg2rad(-180);
deg2rad(90); deg2rad(0); deg2rad(0); deg2rad(0)];
% arm_move.Goal.Trajectory.Points.Positions = arm_positions;
% send(move_arm_publisher,arm_move); % 5.1 % Extend 6° DOF % Skippable
% pause(4)
arm_positions = [deg2rad(0); deg2rad(0); deg2rad(-180); deg2rad(90);
deg2rad(0); deg2rad(0); deg2rad(0)];
arm_move.Goal.Trajectory.Points.Positions = arm_positions;
```

```
send(move_arm_publisher,arm_move); % 5.2 % Extend 2° DOF
pause(4)
arm_positions = [deg2rad(0); deg2rad(0); deg2rad(-180); deg2rad(0);
deg2rad(0); deg2rad(0); deg2rad(0)];
arm_move.Goal.Trajectory.Points.Positions = arm_positions;
send(move_arm_publisher,arm_move); % 5.3 % Extend 4° DOF
pause(4)
%%%%%%%%%%%%%

arm_positions = [deg2rad(0); deg2rad(0); deg2rad(-180); deg2rad(0);
deg2rad(0); deg2rad(-90); deg2rad(0)];
arm_move.Goal.Trajectory.Points.Positions = arm_positions;
send(move_arm_publisher,arm_move); % 6 %
pause(4)

%% Opening gripper in case it's closed %%
gripper_move = rosmessage(move_gripper_publisher);

gripper_move.Goal.Command.Position = 1; % Positive opens,
Zero/Negative closes
gripper_move_Goal.Command.Effort = 500;

send(move_gripper_publisher,gripper_move);

%% RAFAEL GOMEZ HORTELANO %% RGH %%
```

Process Script

```
%% Procces script %%

%% Wait input -- then pick %%

% Moving torso down to pick
torso_positions = 0.06; % Positive = up, Zero/Negative = down
torso_move.Goal.Trajectory.Points.Positions = torso_positions;
send(move_torso_publisher,torso_move); % Picks at 53 mm from ground
pause(6)
% Closing gripper
gripper_move.Goal.Command.Position = 0; % Positive opens,
Zero/Negative closes
send(move_gripper_publisher,gripper_move);
% Tested with 12 mm object [---4mm-gripper-7mm---]
pause(4)

% Moving torso up
torso_positions = 0.2; % Positive = up, Zero/Negative = down
torso_move.Goal.Trajectory.Points.Positions = torso_positions;
send(move_torso_publisher,torso_move); % Up to 14 mm torso extended
pause(4)

% Moving arm to camera position
% arm_positions = [deg2rad(0); deg2rad(0); deg2rad(-180); deg2rad(0);
deg2rad(0); deg2rad(90); deg2rad(0)];
% arm_move.Goal.Trajectory.Points.Positions = arm_positions;
% send(move_arm_publisher,arm_move); % 7 % Skippable
% pause(4)

%%%%%%%%
% arm_positions = [deg2rad(0); deg2rad(0); deg2rad(-180); deg2rad(45);
deg2rad(0); deg2rad(90); deg2rad(0)];
% arm_move.Goal.Trajectory.Points.Positions = arm_positions;
% send(move_arm_publisher,arm_move); % 8.1 % Skippable
% pause(4)
arm_positions = [deg2rad(0); deg2rad(45); deg2rad(-180); deg2rad(45);
deg2rad(0); deg2rad(90); deg2rad(0)];
arm_move.Goal.Trajectory.Points.Positions = arm_positions;
send(move_arm_publisher,arm_move); % 8.2 %
pause(4)
%%%%%%%%

arm_positions = [deg2rad(0); deg2rad(45); deg2rad(-180); deg2rad(110);
deg2rad(-5); deg2rad(85); deg2rad(90)];
arm_move.Goal.Trajectory.Points.Positions = arm_positions;
send(move_arm_publisher,arm_move); % 9 %
pause(4)  % Object at 10 mm from camera

%% Get info from camera and interprete it %%

% image_msg =  receive(camera_sub, 8);
img = receive(camera_sub);
image_work = readImage(img);
figure(1), imagesc(image_work)

part_object = image_work(250:370, 60:500, :);
```

```
figure(2), imshow(part_object)

% Test color HSV method
hsv_object = rgb2hsv(part_object);
hue = mean(hsv_object(:,:,1));
hue = mean(hue);
saturation = mean(hsv_object(:,:,2));
saturation = mean(saturation);
value = mean(hsv_object(:,:,3));
value = mean(value);

color(1) = (0+0.0833)/2;
color(2) = (0.0833+0.1665)/2;
color(3) = (0.1665+0.333)/2;
color(4) = (0.333+0.5)/2;
color(5) = (0.5+0.667)/2;
color(6) = (0.667+0.75)/2;
color(7) = (0.75+0.8335)/2;
color(8) = (0.8335+0.9167)/2;
color(9) = (0.9167+1)/2;

if saturation<=0.1
    color_str = 'white';
elseif value<=0.1
    color_str = 'black';
else
    if (hue>color(9) || hue<color(1))
        color_str = 'red';
    elseif (hue>color(1) && hue<color(2))
        color_str = 'orange';
    elseif (hue>color(2) && hue<color(3))
        color_str = 'yellow';
    elseif (hue>color(3) && hue<color(4))
        color_str = 'green';
    elseif (hue>color(4) && hue<color(5))
        color_str = 'cyan';
    elseif (hue>color(5) && hue<color(6))
        color_str = 'blue';
    elseif (hue>color(6) && hue<color(7))
        color_str = 'purple';
    elseif (hue>color(7) && hue<color(8))
        color_str = 'magenta';
    elseif (hue>color(8) && hue<color(9))
        color_str = 'pink';
    end
end

 %% Choose where to place %%
if (strcmp(color_str,'red') == true) || (strcmp(color_str,'white') ==
true || strcmp(color_str,'black') == true ||
strcmp(color_str,'yellow') == true || strcmp(color_str,'blue') ==
true)
   place = 1;
elseif (strcmp(color_str,'orange') == true || strcmp(color_str,'cyan')
== true || strcmp(color_str,'green') == true ||
strcmp(color_str,'purple') == true || strcmp(color_str,'magenta') ==
true || strcmp(color_str,'pink') == true)
   place = 2;
end
```

```
%% Place %%
% Moving arm to place
switch place
    case 1 % Right
        arm_positions = [deg2rad(-90); deg2rad(45); deg2rad(-180);
deg2rad(110); deg2rad(-5); deg2rad(85); deg2rad(90)];
        arm_move.Goal.Trajectory.Points.Positions = arm_positions;
        send(move_arm_publisher,arm_move); % Rotate to left/right
        pause(4)
%         arm_positions = [deg2rad(-90); deg2rad(45); deg2rad(-180);
deg2rad(45); deg2rad(0); deg2rad(90); deg2rad(0)];
%         arm_move.Goal.Trajectory.Points.Positions = arm_positions;
%         send(move_arm_publisher,arm_move); % 8.2 % Skippable
%         pause(4)
%         arm_positions = [deg2rad(-90); deg2rad(0); deg2rad(-180);
deg2rad(45); deg2rad(0); deg2rad(90); deg2rad(0)];
%         arm_move.Goal.Trajectory.Points.Positions = arm_positions;
%         send(move_arm_publisher,arm_move); % 8.1 % Skippable
%         pause(4)
%         arm_positions = [deg2rad(-90); deg2rad(0); deg2rad(-180);
deg2rad(0); deg2rad(0); deg2rad(90); deg2rad(0)];
%         arm_move.Goal.Trajectory.Points.Positions = arm_positions;
%         send(move_arm_publisher,arm_move); % 7 % Skippable
%         pause(4)
        arm_positions = [deg2rad(-90); deg2rad(0); deg2rad(-180);
deg2rad(0); deg2rad(0); deg2rad(-90); deg2rad(0)];
        arm_move.Goal.Trajectory.Points.Positions = arm_positions;
        send(move_arm_publisher,arm_move); % 6 %
        pause(4)
        % Moving torso down
        torso_positions = 0.06; % Positive = up, Zero/Negative = down
        torso_move.Goal.Trajectory.Points.Positions = torso_positions;
        send(move_torso_publisher,torso_move);
     case 2 % Left
        arm_positions = [deg2rad(90); deg2rad(45); deg2rad(-180);
deg2rad(110); deg2rad(-5); deg2rad(85); deg2rad(90)];
        arm_move.Goal.Trajectory.Points.Positions = arm_positions;
        send(move_arm_publisher,arm_move); % Rotate to left/right
        pause(4)
%         arm_positions = [deg2rad(90); deg2rad(45); deg2rad(-180);
deg2rad(45); deg2rad(0); deg2rad(90); deg2rad(0)];
%         arm_move.Goal.Trajectory.Points.Positions = arm_positions;
%         send(move_arm_publisher,arm_move); % 8.2 %
%         pause(4)
%         arm_positions = [deg2rad(90); deg2rad(0); deg2rad(-180);
deg2rad(45); deg2rad(0); deg2rad(90); deg2rad(0)];
%         arm_move.Goal.Trajectory.Points.Positions = arm_positions;
%         send(move_arm_publisher,arm_move); % 8.1 %
%         pause(4)
%         arm_positions = [deg2rad(90); deg2rad(0); deg2rad(-180);
deg2rad(0); deg2rad(0); deg2rad(90); deg2rad(0)];
%         arm_move.Goal.Trajectory.Points.Positions = arm_positions;
%         send(move_arm_publisher,arm_move); % 7 %
%         pause(4)
        arm_positions = [deg2rad(90); deg2rad(0); deg2rad(-180);
deg2rad(0); deg2rad(0); deg2rad(-90); deg2rad(0)];
        arm_move.Goal.Trajectory.Points.Positions = arm_positions;
        send(move_arm_publisher,arm_move); % 6 %
        pause(4)
```

```
        % Moving torso down
        torso_positions = 0.03; % Positive = up, Zero/Negative = down
        torso_move.Goal.Trajectory.Points.Positions = torso_positions;
        send(move_torso_publisher,torso_move);
        pause(4)
    otherwise

end

% Opening gripper
gripper_move.Goal.Command.Position = 1; % Positive opens,
Zero/Negative closes
send(move_gripper_publisher,gripper_move);
pause(4)
% Moving torso up
torso_positions = 0.1; % Positive = up, Zero/Negative = down
torso_move.Goal.Trajectory.Points.Positions = torso_positions;
send(move_torso_publisher,torso_move); % Up to 14 mm torso extended
pause(4)

%% Returning to pick position %%
switch place
  case 1 % from right
        arm_positions = [deg2rad(-90); deg2rad(0); deg2rad(-180);
deg2rad(90); deg2rad(0); deg2rad(-90); deg2rad(0)];
        arm_move.Goal.Trajectory.Points.Positions = arm_positions;
        send(move_arm_publisher,arm_move); % Ret 1 %
        pause(4)
   case 2 % from left
        arm_positions = [deg2rad(90); deg2rad(0); deg2rad(-180);
deg2rad(90); deg2rad(0); deg2rad(-90); deg2rad(0)];
        arm_move.Goal.Trajectory.Points.Positions = arm_positions;
        send(move_arm_publisher,arm_move); % Ret 1 %
        pause(4)

    otherwise

end

% arm_positions = [deg2rad(0); deg2rad(0); deg2rad(-180); deg2rad(90);
deg2rad(0); deg2rad(45); deg2rad(0)];
% arm_move.Goal.Trajectory.Points.Positions = arm_positions;
% send(move_arm_publisher,arm_move); % Ret 3 % Skippable
% pause(4)
% arm_positions = [deg2rad(0); deg2rad(0); deg2rad(-180); deg2rad(90);
deg2rad(0); deg2rad(-90); deg2rad(0)];
% arm_move.Goal.Trajectory.Points.Positions = arm_positions;
% send(move_arm_publisher,arm_move); % Ret 4 % Skippable
% pause(4)
arm_positions = [deg2rad(0); deg2rad(0); deg2rad(-180); deg2rad(0);
deg2rad(0); deg2rad(-90); deg2rad(0)];
arm_move.Goal.Trajectory.Points.Positions = arm_positions;
send(move_arm_publisher,arm_move); % Ret 5 % Ready to pick again
pause(4)

torso_positions = 0.1; % Positive = up, Zero/Negative = down (max 0.5,
torso_move.Goal.Trajectory.Points.Positions = torso_positions;
torso_move.Goal.Trajectory.Points.TimeFromStart = time;
send(move_torso_publisher,torso_move);
pause(4)
```

Process Shutdown

```
% Move head
head_move.Goal.Trajectory.Points.Positions = [0.0; 0.0];
send(move_head_publisher,head_move);

% Move arm
arm_positions = [deg2rad(0); deg2rad(0); deg2rad(-180); deg2rad(0);
deg2rad(0); deg2rad(0); deg2rad(0)];
arm_move.Goal.Trajectory.Points.Positions = arm_positions;
send(move_arm_publisher,arm_move); % Shutdown 1 %
pause(4)
arm_positions = [deg2rad(0); deg2rad(0); deg2rad(-180); deg2rad(90);
deg2rad(0); deg2rad(0); deg2rad(0)];
arm_move.Goal.Trajectory.Points.Positions = arm_positions;
send(move_arm_publisher,arm_move); % Shutdown 2 %
pause(4)
arm_positions = [deg2rad(0); deg2rad(90); deg2rad(-180); deg2rad(90);
deg2rad(0); deg2rad(0); deg2rad(0)];
arm_move.Goal.Trajectory.Points.Positions = arm_positions;
send(move_arm_publisher,arm_move); % Shutdown 3 %
pause(4)
arm_positions = [deg2rad(0); deg2rad(90); deg2rad(-180); deg2rad(90);
deg2rad(0); deg2rad(90); deg2rad(0)];
arm_move.Goal.Trajectory.Points.Positions = arm_positions;
send(move_arm_publisher,arm_move); % Shutdown 4 %
pause(4)
arm_positions = [deg2rad(90); deg2rad(90); deg2rad(-180); deg2rad(90);
deg2rad(0); deg2rad(90); deg2rad(0)];
arm_move.Goal.Trajectory.Points.Positions = arm_positions;
send(move_arm_publisher,arm_move); % Shutdown 5 %
pause(4)
arm_positions = [deg2rad(90); deg2rad(90); deg2rad(-90); deg2rad(90);
deg2rad(0); deg2rad(90); deg2rad(0)];
arm_move.Goal.Trajectory.Points.Positions = arm_positions;
send(move_arm_publisher,arm_move); % Shutdown 6 %
pause(4)
arm_positions = [deg2rad(90); deg2rad(90); deg2rad(0); deg2rad(90);
deg2rad(0); deg2rad(90); deg2rad(0)];
arm_move.Goal.Trajectory.Points.Positions = arm_positions;
send(move_arm_publisher,arm_move); % Shutdown 7 %
pause(4)

% Move torso down without hitting itself
torso_positions = 0.03; % Positive = up, Zero/Negative = down
torso_move.Goal.Trajectory.Points.Positions = torso_positions;
send(move_torso_publisher,torso_move);
pause(4)

% Node shutdown
rosshutdown
clear all

%% RAFAEL GOMEZ HORTELANO %% RGH %%
```