Tampere University

Álvaro Pajares Barroso

APPLICATION OF PERCEPTION TECHNOLOGIES FOR ROBOTIC MANIPULATION

Faculty of Engineering and Natural Sciences
Master's thesis
May 2020

Abstract

Álvaro Pajares Barroso: APPLICATION OF PERCEPTION TECHNOLOGIES FOR ROBOTIC MANIPULATION
Master's thesis
Tampere University
Automation Engineering, Factory Automation and Robotics
May 2020

Robots are highly present in manufacturing processes, being relevant assets for manufacturing companies that develop tasks that increase the value of the product. Those tasks are normally considered repetitive and in some cases hazardous for humans, as could cause boredom and lead to injuries (for example, painting a car) or directly because the tasks that are developed are hazardous (such as welding or the manipulation of toxic elements). The use of robots in manufacturing processes improve both the safety and the efficiency of processes in a company.

Nowadays, the range of tasks that are developed by robots is getting wider, and are able to develop tasks that require higher dexterity and precision. To improve the performance of robots, the use of visual perception techniques and the use of Collaborative Robots are two of the main research fields in Robotics, with the purpose of giving robots the possibility to interact with humans without compromising worker´s safety and developing the tools to overcome changes in the environment that could compromise the development of the task.

The aim of this thesis is to demonstrate that Cobots can be an adequate solution to work in manufacturing environments developing dexterous and precise tasks that nowadays are developed by humans, by putting together the two main challenges that manufacturing industries are facing nowadays: Cobots and Visual Perception. This demonstration will be done by using a Cobot to assemble a connection panel by using visual perception techniques to detect cables. In addition to the detection of cables, the manipulation of deformable linear objects such as cables is faced in this project.

To fulfill the objectives of the thesis, the first part of the document presents a review of the literature that is related to those research fields, highlighting different Visual Perception techniques and the grasp taxonomies. After this literature review, an approach taking into account the recent advances in these fields is exposed to solve the problem.

After the project development, it has been proved that by using simple image processing operations high-accuracy (97,14% testing 25 different situations) can be achieved to solve an object-detection problem. Indeed, the time consumed to ma-

nipulate the cables is also reduced compared to humans.

Keywords: Cobots, Visual Perception, dexterous tasks, Automotive industry, Image Processing, Collaborative Robots.

The originality of this thesis has been checked using the Turnitin Originality Check service.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Robots are part of most of the manufacturing processes that occur in industry. Those robots are used to develop repetitive tasks that could cause boredom and could lead to injuries to the worker [1], improving the quality of manufacturing processes with high precision. The market of robotics will continue growing in the incoming years, where the improvement of quality and productivity will be even more because of the decrease of prices and the new capabilities industrial robots will have. The trends in industrial robotics, according to [2], are: collaborative robots, Industrial Internet of Things (IIoT), Industry 4.0. and moving into smaller markets. These trends tend to create a safer and more connected manufacturing environment, where workers will work alongside with collaborative robots and where data acquired from the devices integrated in the manufacturing processes will increase efficiency with machine visibility, data analytics or a better predictive maintenance. The automotive manufacturing industry is one of the most automated supply chains in the world, where robots are an important asset, since Ford started them in 1970s doing tasks such as welding (the most important task for robots in automotive manufacturing) or painting. Even if robots have improved production lines efficiency, some of the assembly tasks are still done by humans, as robots are supposed to be too robust or not good enough to adapt their requirements to a changing environment (tasks as the assembly of cables or the assembly of small components). Collaborative robots are also being incorporated in automotive supply chains [3], even at first some people were skeptical about the integration of robots with workers altogether. This integration has enhanced the human productivity, and some studies revealed that this productivity could have been increased around 85%. Nowadays, the main challenge for Cobots is the fine dexterity and rapid decisions in changing environments that would ensure a high production efficiency [4].

To address the challenges that Cobots (and robots in general) industry are facing, the main approaches that are being proposed go in the same direction, the use of Perception Technologies. During the last decade, computer vision has being a field where research and new applications are being developed. Many of these applications are related to identifying objects or persons, but also many other applications have been developed, as solving healthcare problems from images. The main problems of making computer vision solutions that could be applied are related with the amount of data that has to be processed in an image to find a solution, and how to obtain the patterns to find proper information. However, deep learning and artificial neural

networks have simplified this problem by searching for patterns in an automated way, reducing the degree of complexity of computer vision and making possible the application of these technologies in many applications.

1.2 Problem definition

In this work, the assembly of a connection panel will be done by using a Cobot, being considered a task which is not automated in manufacturing processes where many interconnected variables have a role in the process that have to be connected and where the gripper is crucial as the grasp and connection of cables is a task which require high dexterity. This topic is considered a research topic where visual perception, robot controller and grippers have to be connected adequately to develop the task.

The purpose of this project is to combine Cobots and Vision Perception and demonstrate how the combination of them could improve the dexterity and the adaptation of robots to changing environments.

1.3 Objectives and scope

The main objective of this work is to demonstrate that Cobots could be an adequate solution to manipulate cables in manufacturing environments with dexterity and being faster than humans that currently develop this task, avoiding time consumed by humans to be focused in just one grasp of the cable (with Cobots it would be possible to grasp both grasps of a cable at the same time) and increasing the velocity to connect the cables, as robots make faster movements compared to humans. In future works, the scope could be wider, using robots to manipulate the cables that are assembled in some parts of a car in real manufacturing environments.

1.4 Outline

The structure followed in this project is: Chapter 1 is a brief introduction of the project, where objectives and scope of the thesis are introduced; Chapter 2 reviews the literature related to this work, taking into account the advances in different technologies and some applications that could be useful for this project; Chapter 3 explains the proposal done to approach the objective and how it has been implemented; Chapter 4 presents different situations that have been tested and the results obtained, to evaluate how effective the proposed solution is; Finally, in Chapter 5 the main conclusions are defined and the future works that the author thinks could be done are determined.

2 Literature review

With the 4th Industrial Revolution, manufacturing processes evolve constantly, improving to satisfy the demands of the client while optimizing processes and using an approach where modifications can be applied easily, being possible to satisfy new requirements. In this section, technologies that are most related to this thesis are reviewed and the main advances that have been achieved are exposed.

2.1 Industrial robots manipulators and cobots

The main advance in manufacturing processes in the last years have been the automation of processes. By definition, automation is "the use of machines and computers that can operate without needing human control". Even if the definition is not the most appropriate in my opinion (as human control is what makes possible automated processes), it can be understood that automation is the "replacement of man by machine for the performance of tasks, and it can provide movement, data gathering, and decision making", as defined in [5], as human control is still needed, but the physical human labour to fulfil a task is not needed when a process is automated. This automation has been achieved during the last years by the presence of robots in manufacturing processes, but robots have been there long time ago, since 1961, when General Motors included a robot in a manufacturing process. In the beginning, this field was hesitant to include robots in their manufacturing lines, not only because of the fear of people to lose their jobs because of robots (even if it has been demonstrated that robots enhance manufacturing processes by helping humans, not by substituting them), but also because in some cases they thought that robots wouldn´t be useful in their processes. Little by little, this perception has changed along with a reduction in costs and complexity in terms of use and an increase in terms of capabilities, and nowadays robots are present in most of the processes involved in manufacturing, in Small and Medium Sized Manufacturers and also in Big Sized Manufacturers.

In this section, a brief summary of the history of robots will be presented along with the most related tasks, followed by a categorization based on the topologies of industrial robots. Collaborative robots will be presented at the end as a new trend in manufacturing processes.

Brief historical review of Industrial Robots

As commented above, in 1961, the first industrial robot was incorporated in a manufacturing process. This robot was developed by the company Unimation and was

called Unimate 1, being moved by an hydraulic driven motor and stacking die cast parts as first task, having as memory a magnetic drum that stored the instructions needed for the task. After this achievement, General Motors expanded the use of robots and in 1969 a major installation was installed by developing also welding tasks. In 1973, KUKA developed the first robots with 6 electromechanical driven axes, and Hitachi incorporated the first vision sensors to track moving objects.

The following ten years were incredibly active in the field of robotics, and many achievements were fulfilled: the first fully electric microprocessor-controlled robot was launched (IRB 6, 1974), assembly operations were incorporated (SIGMA robot, 1978), the first Selective Compliance Assembly Robot Arm was developed (SCARA, 1978) and others. This achievements increased the use of industrial robots, and, in 1983, 66.000 robots were in operation (10 years earlier, just 3.000 were in use). One year later, in 1984, Adept simplified the design of robots by including the motors directly to the arms, what was considered as a great achievement as from then robots could have a smaller size and accuracy and reliability improved. By 2003, 800.000 robots were in operation. After that, the main improvements have been achieved in robot controllers, being the main research field during the last years, among with the inclusion of collaborative robots in working environments where humans and robots work together.

Since the inclusion of robots in manufacturing processes, many sectors have incorporated them to optimize their systems, being always the automotive sector the one that has been more related to them. Figure 2.1 (taken from the IFR World Robotics(2013)) shows the estimated supply of industrial robots by industries, where the automotive sector is the one that needs a higher supply of industrial robots. The IFR World Robotics also made a study showing the number of robots used in different industries per number of employees (Figure 2.2), where it is noted how important robots are for the automotive industry. Even with these results, new robot applications can be applied to this industry, particularly in trim and final assembly operations, such as the assembly of cables in cars.

Industrial robots can be categorised into hard or soft automation, depending on their role in the automated system. Hard automation involves a specific task that is optimised and without flexibility, while soft automation allows different tasks by using similar equipment, but less optimization is achieved. From the beginning, industrial robots have been used to develop intuitive and repetitive tasks such as welding or painting with predefined paths, but nowadays more complex tasks that require other senses such as visual perception or the hearing sense to apply more accurate tasks.

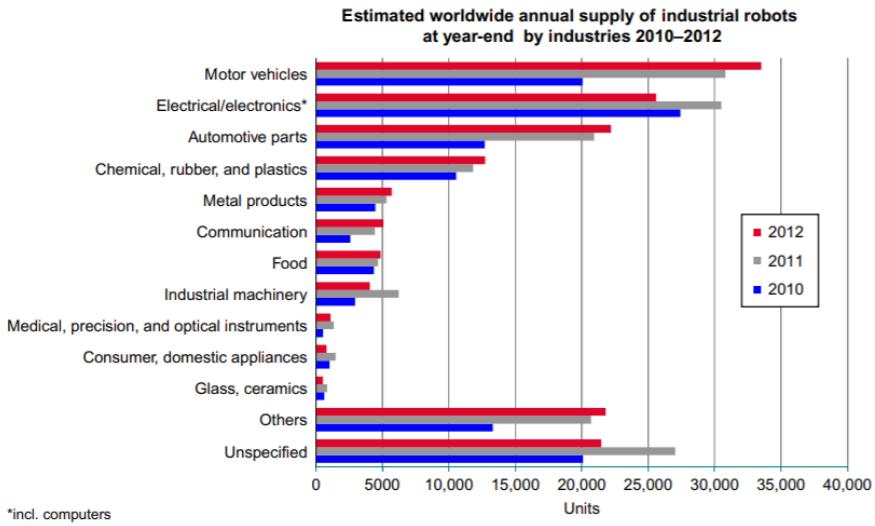


Figure 2.1 *Estimation of industrial robots supplied worldwide [5].*

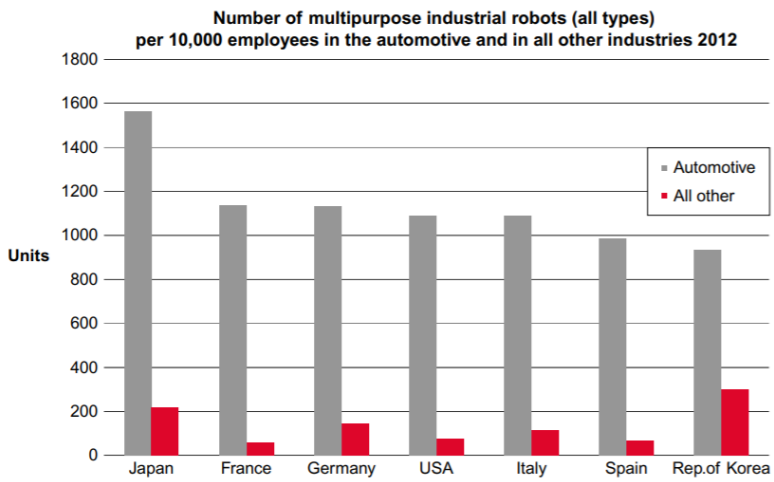


Figure 2.2 *Number of robots per 10.000 employees [5].*

Categorization based on topologies

Robots in industry have a wide range of operation and, depending on that, will include certain features to develop properly the task required. The most-common categorization of industrial robots based on topologies is the following [5]:

- Articulated: robots that are composed by rotary joints of more than two joints, having as many joints as needed (each joint represent an axis and a degree of freedom). The main advantage of this type of robots is the flexibility that the design provides, being one of the most used in manufacturing lines.

- Cartesian: those robots make linear movements in the three axes (X, Y, Z), and the gripper could attach a wrist to allow rotational movement. The main advantage

of this type of robots are the simplicity, the range and heavy capacities, being cheaper than other solutions and making them a good choice for simple operations such as picking up objects or palletising.

- Cylindrical: the main movement is cylindrical and two linear movements are allowed to go up and down and to allow movements in the around the cylindrical pole. Those robots are used in some manufacturing lines to do tasks because are faster to go from one point to another if predefined.

- Spherical: a twisting joint is the main component, in addition to a linear movement allowed. Even if those robots where used years ago, nowadays the footprint that they have in industrial processes is small.

- SCARA: Selective Compliance Assembly Robot Arm. The joints allow both vertical and horizontal movements. Two joints allow the movement along the horizontal plane and one linear movement is allowed along the vertical axis (have just 4 DoF). Those robots have small capacities and the range is about 1m, but is normally faster than articulated robots.

- Parallel: those robots are built from a single base that connects many parallelograms. The main advantage of these robots is the high-speed repetition, such as separating certain types of objects (for example, are used to separate rubbish to recycle by using visual sensors).

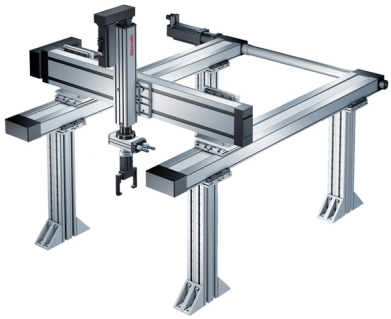
Those robots can be used to interact with humans developing tasks by collaborating with them. The following subsection focuses on these Collaborative Robots or Cobots.

Collaborative robots

Collaborative robots intend to share with humans a workspace and work together were introduced in 1999 at Northwestern University, in Evaston (Illinois), defined as "an apparatus and method for direct physical integration between a person and a general-purpose manipulator controlled by a computer"[4]. However, it wasn´t until 2004 that the first collaborative robot was built by FANUC. At that time, the main problem of those robots was the safety that workers had to work with and how restrictions could be applied to robots to avoid risk situations. Four years later, Universal Robots released the UR5, that was able to operate safely with humans, settling Cobots as a flexible, safe and cost-efficient robots. Since then, many Cobots have been presented, existing currently 19 different Cobots on sale (some examples are ABB YuMi, Yaskawa Motoman or UR10) with different features (the main differences are the Degrees of Freedom and the number of arms). The main reasons that make Cobots unique and a good solution are [4]:

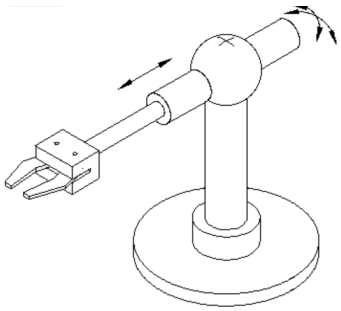
- Affordable: the average cost for a cobot is 24.000$. Depending the size of the company, this price could be low or high, but, even for Small and Medium Sized

(a) Articulated robot.



(b) Cartesian robot.

(c) Cylindrical robot.



(d) Spherical robot.

(e) SCARA robot.

(f) Delta Robot

Figure 2.3 *Types of industrial robots [5].*

Manufacturers, the payback times are quite fast, being normally about 6 months.

- Flexible: Cobots are able to work in versatile tasks.

- Fast set-up: it is extremely easy to unpack, mount and program a Cobot, taking in some cases less than an hour.

- Simple to Use: it will depend on the robot´s company, but in some cases even operators without programming experience will be able to program a Cobot. Cobots can be programmed also by teaching them manually. It is possible to manipulate their arms with our hands and out them in the desired position.

- Collaborative and safe: the main advantage of Cobots. Safety is ensured when these robots work alongside employees.

At the beginning, as happened with industrial robots, people were skeptic about the possibility of using them in industry processes. Nowadays, however it is a market that is growing at a 50%/year pace and could have around 3 billion dollars in revenue by 2020 [4]. The current trend that this industry is experiencing focuses on the fine dexterity and making rapid decisions to avoid changes in the environment to keep on with the production. The approach that the Cobot industry is taking to address these challenging trends is the use of faster processors and integrated vision systems, to be able not only to detect problems and changes in the environment and differentiate them from an operator (where safety should be the main objective), but also to react fast to find a new path to avoid an obstacle without stopping.

It looks that Cobots are useful to work alongside people and develop tasks in changing environments, but: What are the industries that need these solutions to improve their processes? According to [6], as for industrial robots as a whole, the automotive industry is again the leading industry to use Cobots in their assembly lines, developing more accurate tasks such as the assembly of the inside components of a door. Another industry that feels Cobots are an appropriate solution to develop some of the tasks done in their production lines is the food processing industry, where Cobots are used to package the more delicate items to prepare them for transportation. The third field where Cobots are more used is electronics, where Cobots adapt themselves to fulfil different applications, such as assembly or inspection. This industry´s forecast points that there will be more high mix, low volume production (on-demand or micro-manufacturing), what makes Cobots even more fitful for this industry as are easy to reprogram and with vision sensors these changes could be detected even directly. This list keeps on and it is noted that Cobots are being used in a big list of industries, comprising food processing, manufacturing, construction, healthcare, steel and chemical industries (where awkward workpieces are handled by Cobots), textile industry and others.

2.2 Grasp and manipulation taxonomies

Manipulating objects is not an easy task to be applied in robots. Even if for humans it is instinctive or natural to grasp objects and manipulate them, this task is defined by the geometry of the object, kinematics, dynamics and constitutive relations (such as joints, fingertip deformations, friction conditions or others). To determine the way a robot should manipulate an object, the grasp taxonomy of humans has to be studied, not only studying the object that has to be grasped, but also the task that has to be done with that object (the taxonomy to grasp a cable is different if it has to be moved than if it has to be inserted in a pin), as shown in Figure 2.4. In

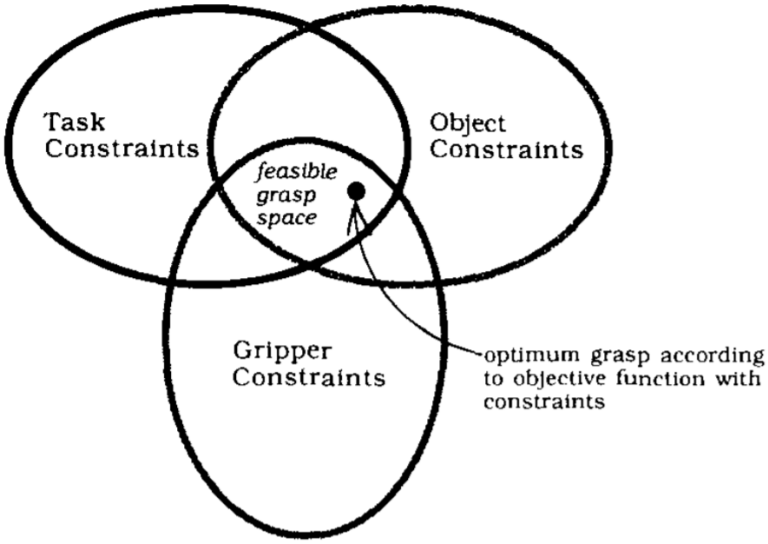


Figure 2.4 *Optimum grasp for a certain operation [7].*

this section, two main articles will be studied and many others will be mentioned, explaining different authors approaches. The main articles that are going to be studied are [7] and [8], being considered the ones that are most used and most complete, respectively.

The approach done in [7] studied small-batch manufacturing single-handed operations in order to design and control robotic hands in manufacturing operations. In this work, Cutkosky reviews previous works that study the grasp modelling with many assumptions (such as [9] and [10]), the grasp choice ([11] and [12] among others) and the grasp selection process itself ([13], defining six possible human grasps: cylindrical, fingertip, hook, palmar, spherical and lateral). After the evaluation, a partial taxonomy of manufacturing grasps is defined, as shown in Figure 2.5. This partial taxonomy differentiate at first level precision grasps and power grasps, where precision grasps will use fingertips and the thumb to do accurate tasks and power grasps will use larger areas of contact to do a task that requires more stability and security. This first split was suggested by Napier in [14], and the rest of the taxonomy is suggested by Cutkosky, where both the object and the task have the same importance while evaluating the grasp. For example, to write with a pencil, probably the grasp chosen would be between 6, 7, 8 and 9, as is a task that requires precision and where the object is long, while to grasp a tennis ball, grasps 10 or 11 would be chosen, as stability and clamping are required for the task and the object is compact. It is worth noting that many tasks could require different grasps, having situations that require precision grasps and others that require power grasps.

After this partial taxonomy, many limitations were found while observing the

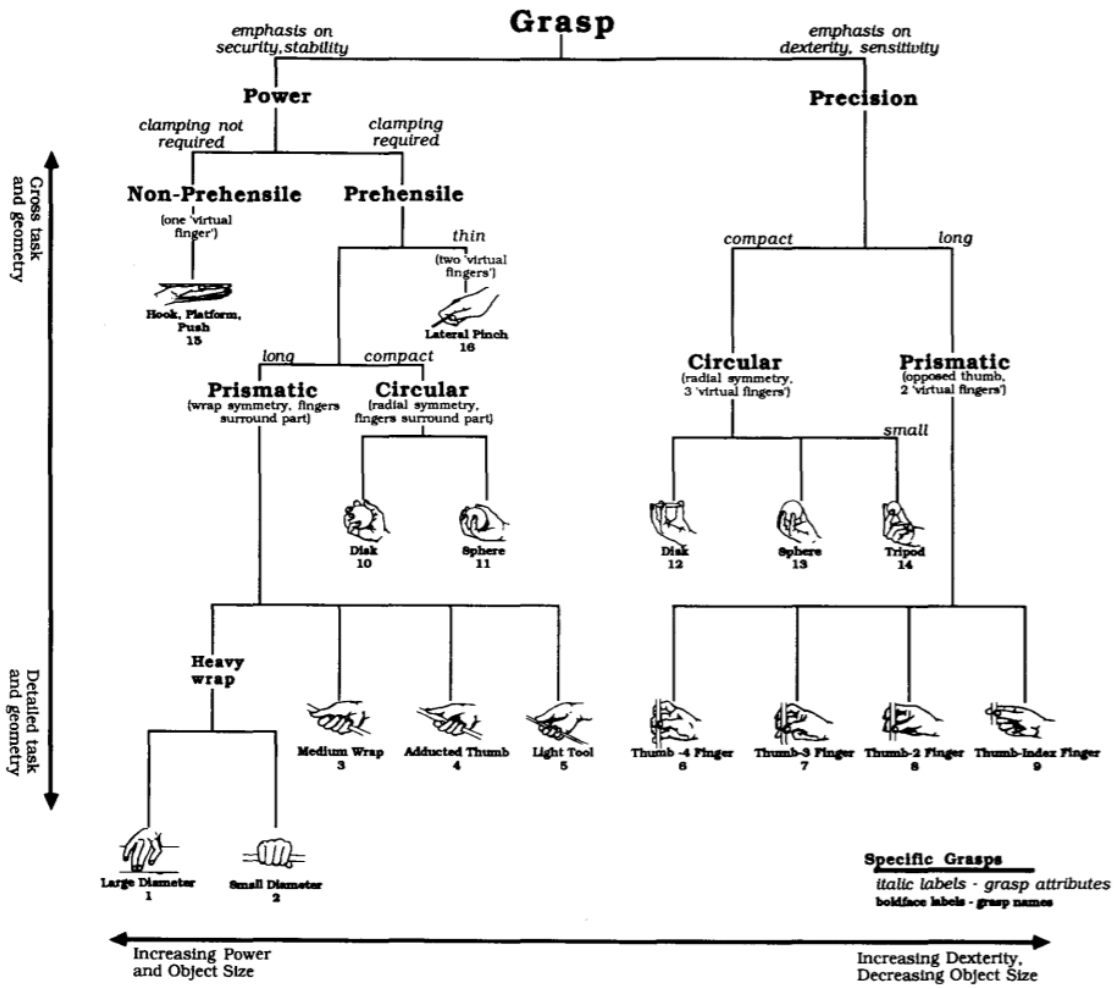


Figure 2.5 *Partial taxonomy of manufacturing grasps [7].*

way manufacturers where working, where many grasps were not included and where machinists also used to grasp things in similar ways as the ones suggested, but with small differences. To fix that, the author proposed Grasp-Exp, a system for manufacturing grasps, where many constrains (for example, if only 3 fingers fit to grasp a particular tool) and requirements (for example, what if I only had three fingers) are added. This system is composed of a framework that get information from the user by answering questions related to the object and the task to be done. The considerations that were done in this work to evaluate the taxonomy of manufacturing grasps were used in many other articles related to this topic.

One of this articles is the one presented in [8], where a more current view of the grasp taxonomy of humans was suggested, including many considerations that were done in 22 previous works and still working with static situations and one-handed situations. Before mentioning this article, other publications about this topic are going to be mentioned.

In [15], a grasp taxonomy is proposed based on the contact points between the hand and the object. In this proposal, grasps are divided into volar (if the palm surface is involved in the grasp) and non-volar (if not, and are further classified as fingertip grasps and composite non-volar grasps), which is similar to precision and power grasps, excepting the lateral pinch, that is considered non-volar and power grasp.

In 1999, C.M. Light [16] did a review of natural and prosthetic hands, with an approach that is more related to daily activities, while the approaches related to robots is more related to manufacturing activities. Later on, in 2002 [17], a statistical approach was presented, where volunteers elaborated some tasks (12 abstract tasks and 14 daily activities) and the grasp was evaluated, establishing only 6 classified grasps: lateral, power, tripod, tip, extension and spherical. Those studies note that the study of grasp taxonomies was not only done to be applied in the field of robotics, but also in others such as medicine.

Not a long time ago, [18] in 2014 and [19] in 2015, were more concerned on defining a grasp taxonomy easier to apply on robots. [18] evaluated the activities two subjects did during a typical day, trying to classify them into an already defined grasp. After the evaluation, it was not possible to classify many grasping actions (out of 179 grasping actions, 40 were not classified), and to extend the grasp possibilities, including motion, force and stiffness were proposed. So, the features that were highlighted in order to classify daily actions in this article were: hand shape, force type, direction and flow, focusing on the task that is going to be done, so that this classification is related not only to grasp taxonomies, but also to manipulation taxonomies (See Figure 2.6). This classification goes deeper and in the end every high-level and the tree is long-enough to define the daily activities that were recorded. This approach is helpful for robots that have been built to help at home with daily activities. In [19], five subjects were required to grasp objects that were randomly situated in a tablet, being recorded by three cameras and doing 100 trials per participant, grasping different objects from a tablet, simulating a table. This study distinguished a total of six primitive grasps (reach, close, slide, edge-grasp, flip and fail), that represent the main hand movement, where a grasp strategy consists of many grasp primitives that ends with the object lifted up. These primitive grasps have also modifiers, where each one can be done top/side, constrained/unconstrained and with rotation/no-rotation. In this article, the main primitive grasps are considered as suitable to be transferred to a robot, as having being tested previously, and an example is shown to grasp an object.

Back to [8], after the evaluation of these related-works and the combination of all of them, excluding the ones that were not included in the definition of the problem ("A grasp is every static hand posture with which an object can be held securely

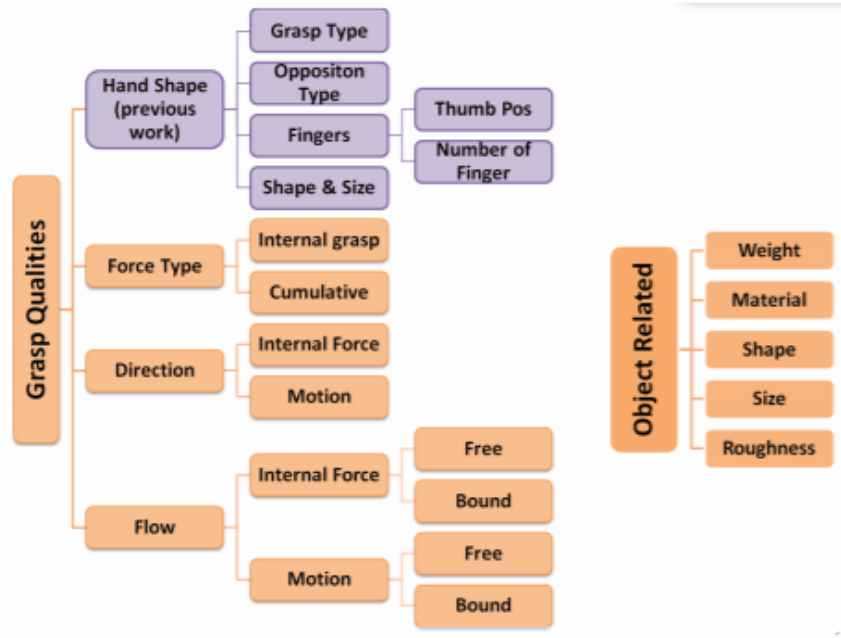


Figure 2.6 *Simple classification after evaluating everyday tasks [18].*

with one hand, irrespective of the hand orientation.”), the number of possible grasp types was 33. This work classify grasps according to four aspects (it is worth noting that the object size is not considered):

- The opposition type: it refers to the direction of the force that the hand applies over the object. It can be pad opposition (direction parallel to the palm, for example when an object is grasped between two fingers), palm opposition (direction perpendicular to the palm, for example when an object is grasped between palm and many fingers) and side opposition (between hand surfaces along a direction generally transverse to the palm, for example when a key is grasped). Figure 2.7 represents the types of opposition and the number of virtual fingers according to the situations shown.

- The virtual finger assignments: a virtual finger is defined as fingers that apply forces in the same direction. For example, when grasping a hammer, there would be two virtual fingers: one referred to the thumb and another referred to the rest of the fingers. Virtual fingers opposed their forces in order to do a proper grasp, and the only way there could be a third virtual finger is if a task-related force or torque is applied.

- Type in terms of precision, power or intermediate grasp: as defined in [7] and many other articles.

- The position of the thumb: thumb abducted (where it oppose the fingertips) and thumb adducted (where there could be forces on the side or just leave the thumb

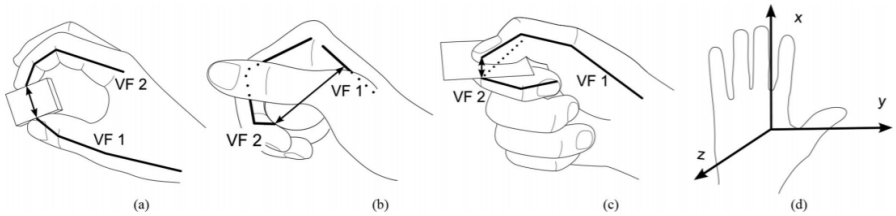


Figure 2.7 *Types of opposition: (a) Pad opposition, (b) Palm opposition and (c) Side opposition. Virtual Fingers (VF) are also represented. [8]*

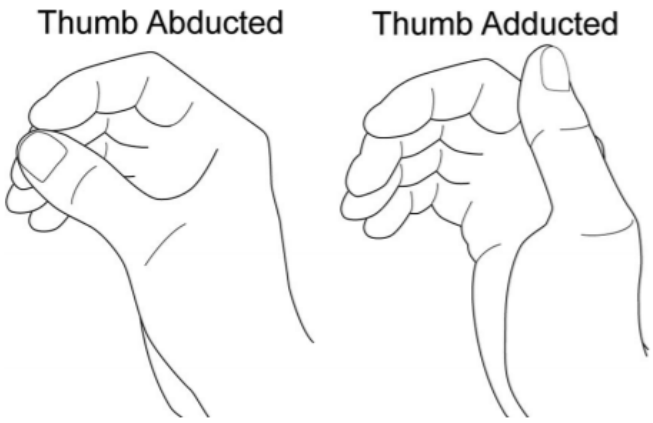


Figure 2.8 *Positions of the thumb used in GRASP taxonomy [8].*

without helping to grasp). These situations are presented in Figure 2.8.

According to these aspects, Figure 2.9 shows the 33 possible configurations in GRASP taxonomy. Can be seen that there are many cells with many possible grasps. The main difference between them is the object to be grasped (property that was included by Cutkosky [7]). This publication was completed in other articles [20] [21],to include in this list of grasps the object that is normally grasped (size, weight and rigidity) and the task that was completed (comparing if the grasp force was dominated by the weight of the object or by the interaction).

Once that the grasp taxonomy for humans has been boarded, the state-of-the-art of manipulation taxonomies will be approached. The manipulation of objects have many components that together compose the task. Some of the main are approaching the object, grasping the object and the way those objects are moved later. Those aspects are equally important and, as well as the different types of grasps have been studied, the way a task has to be planned is also presented.

In 1987, [22] proposed a grasp planner for human prehensile movements. In this approach, the worker has to send a command telling the object that is going to be grasped with its most important features and the task that has to be performed.

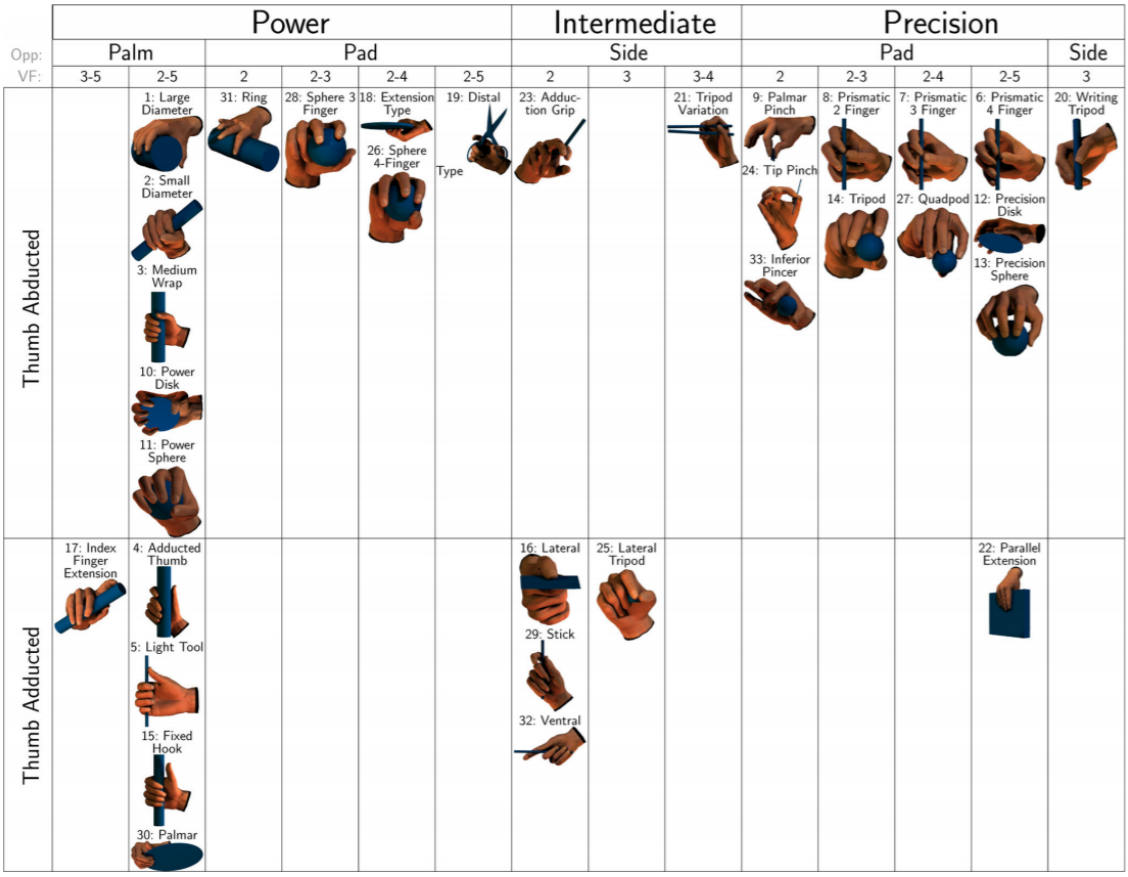


Figure 2.9 *GRASP taxonomy [8].*

The task requirements are evaluated on the first phase of the planner, evaluating functional (more related to the task) and physical constraints (more related to the object, evaluating also the forces, torques,... that will appear, being important features such as the center of mass of the object). On the second and last phase of the planner, these internal requirements are evaluated and the hand variables are described, according to them. This author define the hand postures as combination of three basic oppositions (pad, palm and side). These positions are constrained by anatomical constraints (wrist angle, length of the fingers,...) and object constraints (such as length, width and height.

More recently, grasp planning systems are normally done with the help of vision systems. One approach that use this technology is [23], where a digitizer and a pair of movable stereo cameras are used to evaluate the environment and the objects that it has and, after that, a grasp plan is presented. Those objects to be grasped were novel for the system, and were influenced by obstacles and must-touch regions in the objects. This approach represent the workspace (highly detailed 3D models of many objects were detailed and textured), evaluate the optimum grasp with the help of the software "GraspIt!" from [24] and a continuous collision detection is integrated

to avoid possible detections between objects. In this article, to present a good grasp planning, stable plans and semantic knowledge are required. The stable plans of the object to be grasped are obtained from the 3D models, and are thought as a collision for the grasp plan. If a grasp is not good enough at a first sight, regrasp operations could be done to find a new better grasp. Finally, semantic knowledge for grasping is required. As this approach is mostly done for service robots, that work in domestic and daily activities, the task to be done is crucial. For example, if a glass is going to be grasped to put milk in, it has to be grasped from the side parts. This information is proportioned by a human teacher, that will help by indicating the places where an object can´t be grasped. This part of the grasp planning was not in the scope of this paper, and was indicated as a further job to do.

One of the articles whose approach is more related to the manipulation taxonomies that are being studied at this point is [25]. In this article, the taxonomy proposed is referred to what is the hand doing while doing a task (hand-centric), instead of how an object is being contacted. Figure 2.10 shows the different configurations for human manipulation. The features that define one configuration or other are:

- Contact/No contact: there is contact between the hand and an object or not.

- Motion/No motion: the hand is moving with respect to a body coordinate frame or not.

- Prehensile/ Non-prehensile: a prehensile task is considered when the contact cannot be represented with a single contact point.

- Within hand/ Non-within hand: motion within the hand is considered if parts from the hand (fingers) are moving with respect to a fixed frame from the base of the hand.

- Motion at contact/ No motion at contact: there is motion at contact if there is significant translation or rotation with respect to the contact point.

According to this classification, inserting a key, for example, would be classified as Contact/Prehensile/Motion/Within Hand/No Motion at Contact, as there is contact between object and hand, the object has to be grasped with more than a single contact, the hand is moving, there is rotation between hand and base and there is no significant relative motion at the contact point

In this article, [25] also propose a taxonomy related to dexterous manipulation. Dexterous manipulation is referred to prehensile manipulations within hand manipulation. Figure 2.11(a) shows the taxonomy, where each movement is subcategorized into three rotational and transational movement with respect to the fixed frame in the back of the hand. Back to the example of inserting a key, it would be classified as rotation among the y-axe.

I.M. Bullock made a more-detailed article in 2013 [26], where the same classifica-

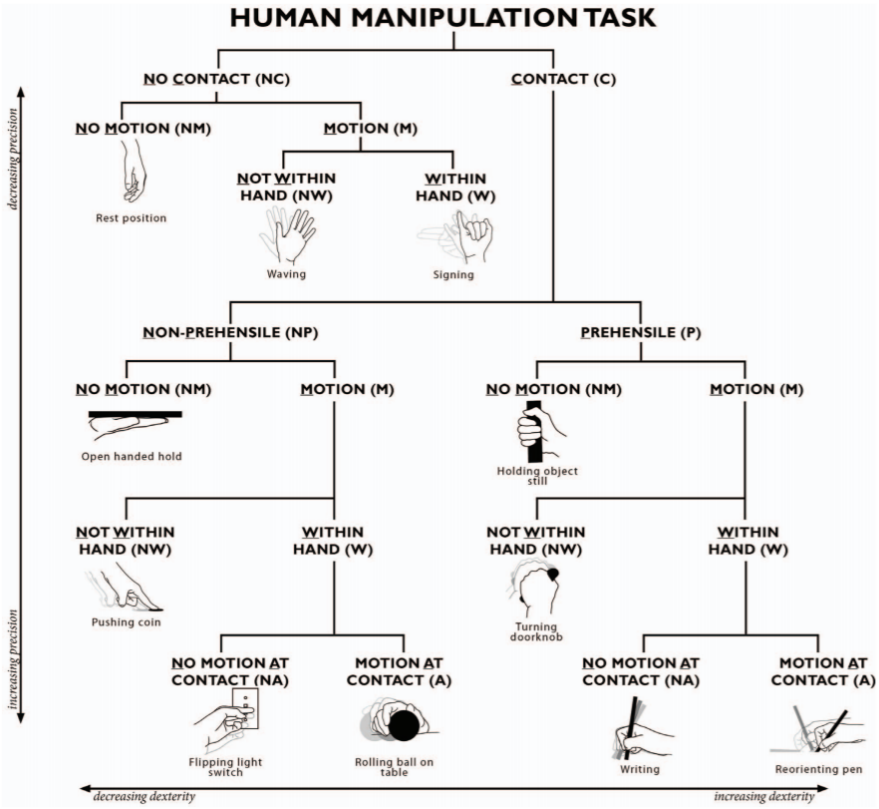
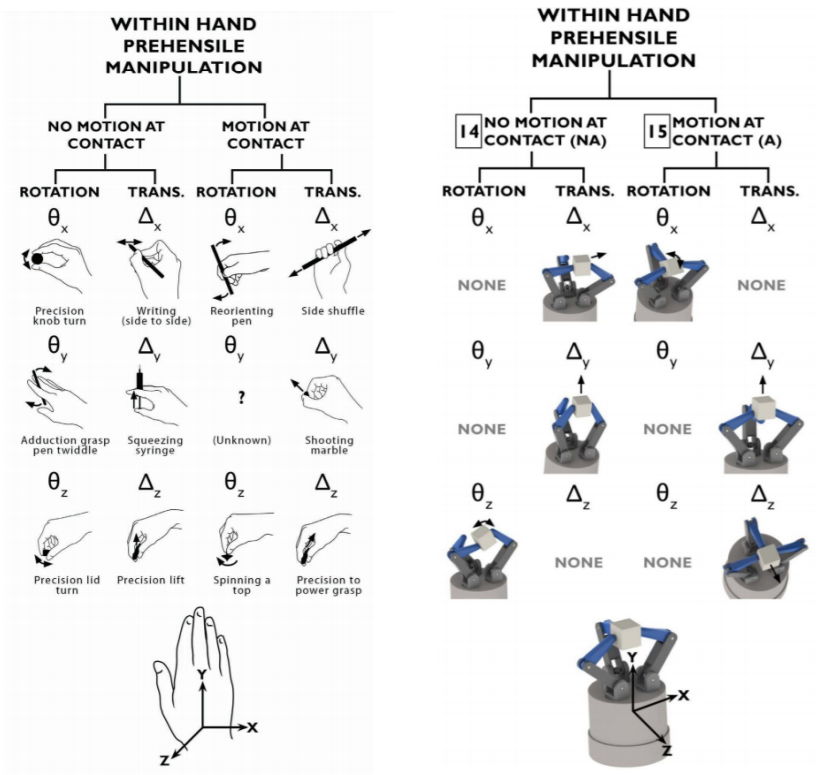


Figure 2.10 *Human manipulation taxonomy [25].*

tion is defined and the differences between hand and arm dexterity are exposed. At this point, the author concludes that with a simple grasper (gripper) and a dexterous arm are enough to accomplish most of the tasks needed (such as, for example, putting an object onto a bin or insert and turn a key). Moreover, this taxonomy was applied to many daily activities, analyzing also the hand design (or gripper) and how appropriate it could be for the task involved. To know what kind of tasks that a gripper could accomplish, the available movements from Figure 2.11(a), as done by the author with the three-fingers gripper (Figure 2.11(b)), help to evaluate the manipulation task. The activities that were analyzed were: taking a drink, opening a door (including the use of keys) and putting on socks. A large amount of statistics is given by [26], highlighting the importance of using both hands and the main differences between the tasks: taking a drink did not needed too much within manipulation, opening the door involves a dexterous task with one hand (turning the key) and non-prehensile capacity with the other (pushing the door), and putting on socks involves contact and motion during the whole task. This approach suggest the evaluation of human hand and gripper capabilities for the task that has to be done, to analyze if the gripper fits with the task.

More recent approaches have been done during this years. [27] made an approach



(a) General configurations for dexterous manipulation taxonomy

(b) Configurations for dexterous manipulation taxonomy in a three-fingers gripper

Figure 2.11 *Dexterous manipulation taxonomy [25].*

about grasps, including the gripper constrains that affect the grasp as a factor that influence the grasp choice. In addition, the final grasp choice according to this article will be dictated as a combination of five factors: object constraints, task constraints, gripper constraints, habits of the grasper (experience and social convention) and chance (environmental constraints and the initial position of the object). In this work, two experiments were done to different persons, evaluating the way humans interact to grasp an object and perform two different tasks (put the object onto a box and an object-specific task) by themselves (non-interactive sessions) and how do they hand objects over a partner, that had to do the same tasks that did before (handover sessions). The results showed that during handover sessions, the grasps done by the first person where precision grasps (considered as grasps where only distal and intermediate phalanges were used), by using two or three fingers and trying to have more space for the second person that had to perform the task. The approach done by D. Paulius et al [28] was related with cooking activities, trying to transfer learned manipulations into unlearned manipulations, focusing on the possibility of transferring the movements to robots and not only on evaluating the manipulation

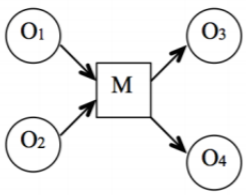
Table 2.1 *Manipulation taxonomy according to [29].*

Manipulation Attributions	*Description of Attributes*
Contact Type	• 0 : **Non-contact** – there is little to no contact between active and passive items. • 1 : **Contact** – there is contact between active and passive items.
Engagement Type	• 0 : **Rigid engagement** – tool (active) and object(s) (passive) do not change in state or structure. Rigid engagement sub-classes: – 00 : **Stationary** – there is no movement of the passive item from the action. – 11 : **Moving** – passive object is moved as a result of the manipulation. • 1 : **Soft engagement** – the manipulation causes change in the state of tools (active) or objects (passive). Soft engagement sub-classes: – 00 : **Admitting/Penetrative** – contact or action allows for penetration, or the manipulated object is permeable in some way for the tool to enter. – 1 : **Deforming** – the manipulation causes deforming, either to the: ∗ 0 : **Manipulator (active deforming)** – deformation of active tool ∗ 1 : **Manipulatee (passive deforming)** – deformation of passive object(s)
Trajectory Type	• (0 – False, 1 – True) **Prismatic** – the movement trajectory is on a line, plane or surface. • (0 – False, 1 – True) **Revolute** – the movement is about axes of rotation; the trajectory moves in its orientation domains.
Contact Duration (between tool and objects)	• 0 : **Discontinuous** – active tool or object makes inconsistent contact with the passive object(s). • 1 : **Continuous** – active tool or object makes constant contact with the passive object(s).
Manual Operation	• 0 : **Unimanual** – the action uses a single hand mainly. • 1 : **Bimanual** – the action uses two hands to manipulate the tool.

activities. The way it is done is by studying the mechanics of human manipulations (mostly, trajectory and contact, instead of other approaches that prioritize finger kinematics) from a large amount of data obtained from cooking tasks. This data is presented in terms of a graph knowledge representation called Functional Object-Oriented Network (FOON), defined in [29]. This FOON representation is built by nodes (motion nodes contain information about the task involved and object nodes about the state of the object, for example: ”dirty” and ”clean” fora knife), edges (connect nodes) and functional units (represent a task, and need at least as input one object node and one motion node and one output). The idea of this representation is to define by hand simple functional units and build an activity (as could be cook a dish), known as the whole network, algorithmically from a video. Figure 2.12 shows an example of a functional unit and a whole network. Once the information about the activity is represented as FOON, the proposed manipulation taxonomy is shown in Table 2.1, where a binary code is obtained evaluating an activity. Obviously, different activities could have the same code, as the requirements in terms of motion could be the same.

In the following section, visual perception is analyzed, as this technology is having an incredible development in the last decades in order to enhance machine learning and deep learning models, areas that are being crucial to apply the visual sense in robots.

(a) FOON network



(b) Simple functional unit

Figure 2.12 *Functional Object-Oriented Network [29].*

2.3 Visual Perception

As humans, we have the possibility of understanding the world around us, differentiating effortlessly different objects and the way we can interact with those objects (for example, we know how to grasp a ball and how will this ball be deformed).

Computer Vision tends to recover the three-dimensional structure of the world

from images by acquiring, processing, analysing and understanding digital images, giving important information from them. Computer vision is a rapidly grown area, that has claimed the interest of important companies to solve problems related to autonomous vehicles, face recognition, object detection and others. As explained in [30], the first attempts to develop Computer Vision algorithms were done around 1966, when trying to improve digital image processing techniques to obtain useful information by attaching a digital camera to a computer during a summer project at Massachusetts Institute of Technology. It was supposed to be an easy task, but obviously it was not. The first attempts to build a system that could obtain information from a picture was to study the vision system from humans, and to try to apply it to a computer. After noticing it was not possible to understand good enough the vision system because of the lack of information and it was not an easy task to apply such a complex system to a computer neither, later approaches decided to look for new solutions, developing mathematical techniques that could obtain enough information to build the 3D model of an environment. From these mathematical models, it has been possible to build models to remove the background, to track a person or to recognize the number of the people that are in a picture,for example. The problem of computer vision is that, even if many fields have been approached and many solutions have been reached, most of the applications are specialized and their use is restricted to narrow domains. Normally, computer vision problems need many approaches to obtain an accurate solution: engineering approach (it is important to know the problem definition and the main constraints and specifications), scientific approach (more related to physics, where the scene is evaluated: lights, sensors, noise and more) and statistical approach (this approach helps to reduce the noise that an image could have and to choose the best solution and how accurate the solution could be). All of these have to be implemented in algorithms that not only work among ideal conditions, but also that are robust enough to admit some noise (to achieve this, Bayesian techniques are used) and deviations and that are efficient in terms of time and space (if possible, linear systems could be used to ensure better efficiency). The first step of computer vision is to understand how images are built. Once this has been achieved, image processing can be done and, based on these techniques, many applications and algorithms can be build to obtain feature detection, segmentation, motion estimation and other applications.

In this work, image formation will be exposed briefly, while image processing will be presented deeper, to have a better knowledge on what kind of information can be obtained from an image and how is this information obtained.

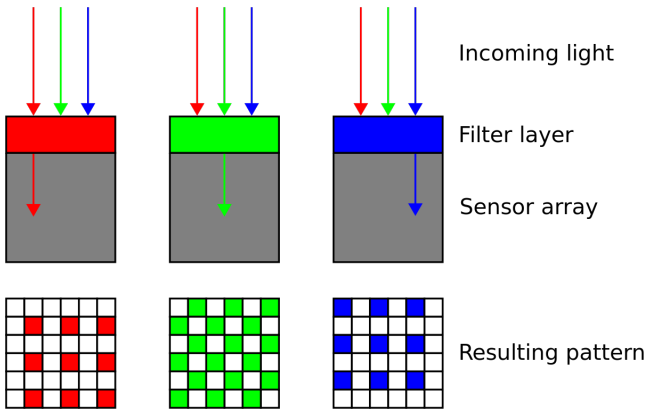


Figure 2.13 *Bayer Filter [30].*

2.3.1 Image formation

While going from the 3D real world to the 2D image, different rays are passing through a pinhole (that can be modified in order to let less/more light pass trough it, depending on light conditions) that is situated in the origin of the camera and then arrive to the image plane, situated at a fixed distance from the origin (focal length). This rays have information about the objects that are in the surroundings as they reflect the light that is present with a wavelength that depends on the color the object, person or animal that reflects the light. Digital cameras capture this information collected in the image plane and process it with a sensor the camera have inside. In the moment somebody takes a photo, light is passed through the camera and the light from the image plane arrives to the sensor. Once the light arrives to the sensor, a Bayer filter is applied. The Bayer filter collects different information depending on the photodiode (pixel) of the sensor. Each pixel will collect information of just one color: Red, Green or Blue, and the other two values will be estimated from algorithms depending on the values that neighbors have. Figure 2.13 shows how this filter works, where can be seen that the number of green pixels collected is double. This is because human eyes are more sensible to this color. After passing through this filter, the light that has been collected at each pixel generates a current that depends on the wavelength of the photon, being able to save this information. After an algorithm is applied, each pixel will have information about Red, Green and Blue, generating colors from this values (for example, white is (255,255,255) and black is (0,0,0)).

Once the image is obtained, information is lost, as we are going from 3D to 2D. For example, straight lines will stay straight, but angles and lengths won´t be preserved. To make simpler the operations (rotation, translation,...) involved when going from 3D to 2D, homogeneous coordinates are essential, as Euclidean transfor-

mation can be complex if many are involved. To go from Cartesian coordinates to homogeneous, a third coordinate for the 2D system and a fourth coordinate for the 3D system is added, 1, representing the same point proportional homogeneous coordinates ($[1, 2, 3]$ is the same as $[3, 6, 9)$. When going from homogeneous to Cartesian coordinates, the "normal" coordinates (x,y,z) are divided by the "extra" coordinate, w.

$$(x,\ y) => [x\ y\ 1]\ \ ;\ \ (x,\ y,\ z) => [x\ y\ z\ 1] \tag{2.1}$$

$$[x\ y\ w] => (x/w,\ y/w)\ \ ;\ \ [x\ y\ z\ w] => (x/w,\ y/w,\ z/w) \tag{2.2}$$

Once the points (or pixels) are in homogeneous coordinates, some basic operations are presented: the line $ax + by + z = 0$ is expressed as $line_i = [a_i b_i c_i]$, and the cross product of two points gives the homogeneous coordinates of a line. Indeed, the cross product of two lines gives the coordinates of the intersection point. In 3D coordinates, the transformations can be concatenated with matrix multiplications, what makes easier to apply rotations and translations from the Euclidean transforms. An example of a transformation that includes rotations and translations in a 3D system is presented as follows (it is worth noting that in 2D occurs the same, but being even more simple by dropping the z coordinate).

$$R_z = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \quad E = \begin{bmatrix} R & t \\ 0^T 1 & \end{bmatrix} \tag{2.3}$$

$$\begin{bmatrix} X_2 \\ 1 \end{bmatrix} = E_{21} E_{10} \begin{bmatrix} X_0 \\ 1 \end{bmatrix} \tag{2.4}$$

This simple addition of the third coordinate in an image enables transformations such as projective (gives a 3D impression from the image) or similarity (rotate, translate and scale the image). Furthermore, if the 3D Cartesian coordinates of a point from an image and the focal length of the camera are known, it is possible to know the position of that point in the image (some additional information about the camera position and it´s calibration could be required too if conditions are not ideal). The following equation define the transformation from 3D coordinates (world point) to 2D coordinates (image point).

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} X^W \\ 1 \end{bmatrix} \ where\ \ K = \begin{bmatrix} f & sf & u_0 \\ 0 & \gamma f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

K represents the camera´s intrinsic calibration (consider the hardware properties

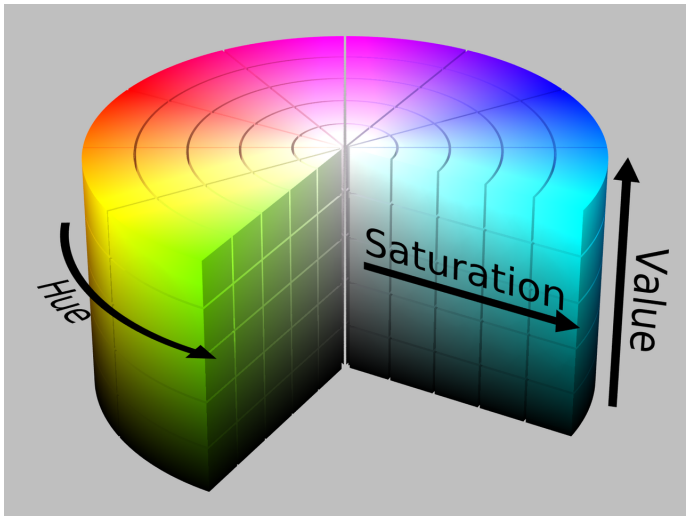


Figure 2.14 *HSV model [30].*

of a camera, taking into account that it is not ideal) , [I 0] the projection matrix (to change from 3D to 2D) and [R t] the camera´s extrinsic calibration (to align the camera with world´s coordinates).

The last thing to consider in this subsection are the models that are most used to represent colors:

- RGB: already explained, a color is formed from three channels: Red, Green and Blue (additive primary colors), where combinations of those can form every color.

- CMYK: in this case, four channels are used representing the subtractive primary colors (Cian, Magenta, Yellow and Black).

- HSV: it is based on the Hue, Saturation and Value of a color. While the previous models where based on Cartesian coordinates, in this case variables are in cylindrical coordinates. The Hue is the main variable, representing a color (Red is 0 degrees, Green is 120 degrees and Blue is 240 degrees, for example). Figure 2.14 shows the palette of colors.

2.3.2 Image processing

Once the image is formed, the next stage to consider when looking for a solution is to process the image, making the needed operations to have a suitable image for further analysis (operations like reducing noise, color balancing and others). This operations can be done pixel by pixel depending only on the pixel value without caring about neighbor´s values (what is known as point operators, which also could take into account general information of the image) or taking into account those values (neighborhood operators, which can be followed by tools that could speed up the process).

Point operators

Point operators are the simplest transforms that can be applied to an image to process it. The main point operators are described as follow:

- Pixel transforms: simple operations are applied to the pixels of the image. This could be used to modify the contrast (multiplying each pixel from the picture or from a region of the pixel by a gain value) or the brightness (adding a bias value to pixel´s values from the picture or from a region of the picture) of an image.

- Color transforms: to balance colors, each channel could be multiplied separately or the whole image transformed to the XYZ color space could be processed by more complex methods to obtain the desired visual effects

- Compositing and matting: matting is defined as the process to extract an object from an image by cutting the background, while compositing is the process of inserting it into another image. To make this possible, an intermediate stage is needed to obtain good results. An alpha channel is added to the RGB image describing the opacity (or fractional coverage) at each pixel, where pixels from the object are opaque ($\alpha = 1$), pixels from outside the object are transparent ($\alpha = 1$) and pixels that are around the boundary are between those values. With these values, the composite image is built as follows: $C = (1 - \alpha)B + (\alpha)F$, where F is the foreground with the background in black color and B is the new background.

- Histogram equalization: an histogram is the display that plots all three channels and luminance values (form 0 to 255) in the x-axis and with the number of pixels that have that value in the y-axis. This display could show relevant information about the image and simple operations could be applied to the image depending on the values obtained. One of these simple operations is histogram equalization, where the final histogram should be flat. This is done by using the cumulative distribution function, where y-axis from this distribution is re-scaled to [0,255] and where the final value of each pixel will depend on the previous value of the same pixel and the value that previous value has in the new y-axis. This operations could also be applied partially, compensating only histogram unevenness.

Linear filtering

Linear filtering is the most commonly used neighborhood operator, where the output value of a pixel is determined by a weighted sum of a collection of pixel values in the vicinity. This is determined by the weight kernel or mask, that imposes how is the input matrix weighted. Equation 2.6 represents this operator:

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l) \tag{2.6}$$

It is also known as convolutional filter, when expressed as defined in Equation 2.7. This second definition adds commutative and associative properties to the equation.

$$g(i,j) = \sum_{k,l} f(k,l)h(i-k,j-l) \tag{2.7}$$

Some properties that are common to linear and convolutional filters are linear shift-invariance and shift invariance. To obtain an output image with the shape unmodified, there are many ways of considering the values that are out of borders. This is called padding or extension modes, and the most used are: zero (values outside the image are set to 0), constant (values outside the image are set to a specified value), clamp (values outside the image are set to edge´s values), wrap (values outside the image are set in a toroidal configuration) and mirror (values outside the image are set as the reflected values from the edge).

To speed up the process of convolutional filtering or to improve this operator, many operations can be applied to the image.

- Separable filtering: this speeds up the convolutional filter operation. Instead of doing the required K^2 operations for each pixel, $2K$ operations are done, by separating the 2D kernel into one 1D horizontal convolution followed by one 1D vertical convolution. Not all kernels can be separated like this, and the ones that can are called separable. This procedure is very useful for computer vision, and, to know if a kernel is separable or not, the Singular Value Decomposition of the matrix is needed, where if the first value is non-zero, it is separable.

- Bartlett filter: the Kernel is built as a linear tent function([1,2,1];[2,4,2];[1,2,1], for example). This is used to smooth the image. It is called the bilinear Kernel.

- Gaussian filter: it is the result of convolving the linear tent with itself (cubic approximating spline).

- Sobel operator: it is used to obtain horizontal edges from pictures. The Kernel in this case is built by a horizontal central difference and a vertical tent filter.

- Simple corner detector: a simple Kernel to detect corners is done by using second derivatives horizontally and vertically.

Figure 2.15 shows those operations that can be applied to convolutional filtering.

It is worth mentioning that Kernel convolutions can be also understood as a filter that modifies the values and the phases in the frequencies that an image has (being frequency understood as the change of pixel values in the image), and the Fast Fourier Transforms can be applied, obtaining faster results and where frequencies of images and filters can be studied to understand better the process. This Fourier transforms are used, for example, to resize or sample an image.

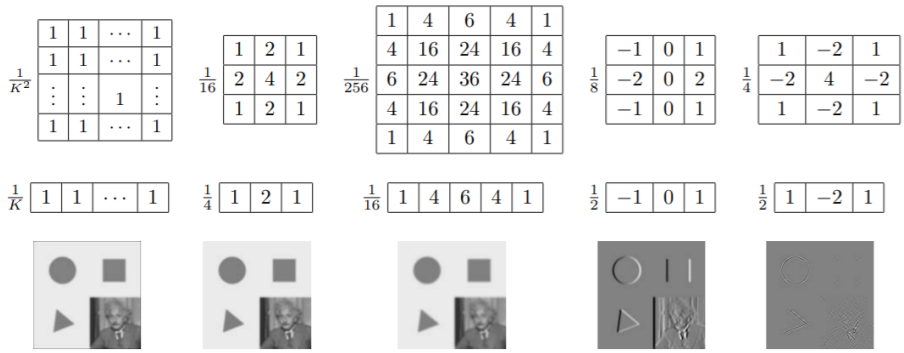


Figure 2.15 *Linear Filtering [30]. (a) Separable filtering. (b) Bilinear kernel. (c) Gaussian kernel. (d) Sobel operator. (e) Simple corner detector.*

More neighborhood operators.

Even with linear filters good results can be achieved and relevant information can be obtained, there are other neighborhood operators that could perform even better. Those operators are: non-linear filtering, morphology, distance transforms and connected components.

- Non-linear filtering: linear filters were composed by weighted summations of some inputs (pixels values). Now, with non-linear filtering, more complex methods are applied to obtain a better performance. Two examples of non-linear filtering are median filtering and bilateral filtering. Median filtering selects the median value from each pixel´s neighborhood and works fine to remove shot noise, where Gaussian filters do not work properly (some variants of this filter work even better, as using the weighted median). Bilateral filtering uses a weighted filter kernel but rejecting pixel values that differ too much from the central pixel value.

- Morphology: morphological operations are applied to binary images (where values are white or black that often come from a thresholding operation in greyscale images) in order to change the shape of the objects from the input image. Morphological operations use structuring matrices to modify output values (some with common structures and others with more complex structures), and the most used are: dilation (makes objects thicker), erosion (shrinks objects, the opposite of dilation), majority (the output will be the value that is more present in the matrix), opening (tend to remove small objects) and closing (tend to close small holes that an image could have). Figure 2.16 represents this operations.

- Distance transforms: in many applications, the distance between pixels is important and has to be measured. The aim of this operation is to compute the distance from every pixel to the nearest black pixel neighbor from a binary image. It is done in two steps: first, the image is swept from top to bottom and left to right

Figure 2.16 *Morphological operations [30].(a) Original image. (b) Dilation. (c) Erosion. (d) Majority. (e) Opening. (f) Closing.*

and the number of pixels in-between the pixel to evaluate and the nearer black pixel is included, and in the second stage the opposite route is followed. By this procedure, the Manhattan distance is obtained (i.e. the minimum vertical or horizontal distance), but there are also some variants to obtain the Euclidean distance.

- Connected components: it is also important for some applications to know if determined pixels are connected in an image, to understand, for example, if two pixels are from the same letter in a picture. The idea is to first swept the image horizontally and find the connected values in images (considering both left and top previous values), so that later an overview of the components is done and the ones that still can be connected are connected. There are many libraries in computer vision whose aim is to obtain the connected components with relevant information (area, perimeter and centroid, for example).

Having this information about image processing as a background and with more operations such as pyramids and wavelets to make smoother a blending and obtain proper results for example when trying to match a template, geometric transformation to modify the geometry of the image and others, images can be processed and information or a better output can be obtained from them. With the image processed, feature recognition and matching can be implemented, obtaining information that can be useful for an application. Edge and corner detection are two common features to be detected. To obtain corner and edge information, normally partial derivatives are used for both x and y directions, using the gradient direction by combining both directions to obtain the edges (rapid changes in the image intensity function along the edge direction) and the corners (significant changes in all directions). To obtain the best possible results, smooth filters are used previously.

Many applications could be satisfied with the information that could be obtained as edge, corners and other features by applying those operators and by processing the image, but in many other some applications (for face recognition or object detection in different environments, for example) more complex methods such as machine learning or deep learning may be needed.

An approach related to image processing was done in [31]. In this article, a machine-learning algorithm is proposed to detect objects (by using graph segmen-

tation algorithm) and to decide the best-grasping option, by using a depth camera to obtain RGBD images (Kinect V2). This approach has many advantages, one of which is that large amount of data is not required, neither training and testing a model (as is required for deep learning models). The steps taken are: 1) Image processing with graph segmentation and morphological image processing, 2) Data is processed and a random forest classifier is trained and 3) Robot control using the robot inverse kinematics.

To process the image, firstly, image segmentation is applied. The background is removed by evaluating the intensity if the pixels and the depth information and comparing the differences between intensities with a threshold. To reduce the noise, convolution filters (this will smooth the image) and area opening are used (that will reduce delete small amount of pixels that are isolated). With this information and by doing blob detection (connected pixels that share common features), the number of objects is identified. After that, morphological image processing is applied.

2.3.3 Applications of Deep Learning

Deep learning is a subset of machine learning, that is at the same time a subset of artificial intelligence, as shown in Figure 2.17. With the arrival of computers, artificial intelligence was intended to solve problems that were difficult for human beings but that were easily described by mathematical rules. After a while, the point of view changed, and the aim of artificial intelligence is now to solve problems that are easy and intuitive for human beings, but that are not followed by easy mathematical rules such as recognizing objects or faces, understanding a speech or reading words. These problems are not easy to solve, as computers have to acquire a lot of knowledge about the world that is subjective and intuitive. Some early projects have tried to apply this knowledge to computers by hard-coding in formal languages with formal rules, what did not succeed at all, as it is too complex to describe the world in an accurate way with formal rules.

As hard-coding was not an option to make computers understand the world, in the 1990s, machine learning techniques were proposed to let computers acquire their own knowledge from raw data, developing patterns that could describe that data meaning. As explained in [32], to solve this problems it is important to have a good representation of the data, that is composed by features of the problem to solve. For example, while trying to detect if a tumor is bad or not, the size of the tumor is crucial, and would be a feature to include. So, for every machine learning problem, the key is to set the right features that represent accurately and properly the problem. But, what if these features are not easy to find? Representation learning is the approach that not only solve the problem, but also discover the representation of the problem to be solved. With this approach, it is easier to make

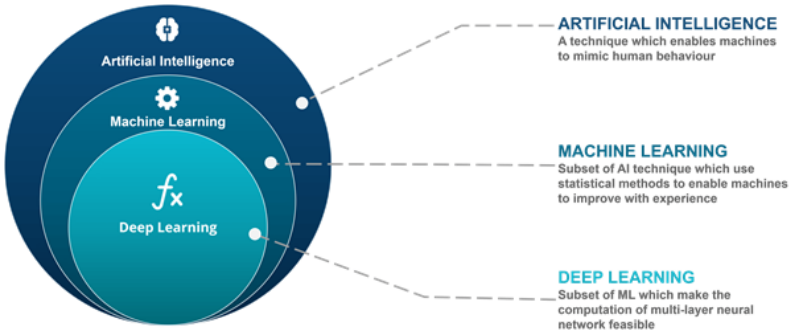


Figure 2.17 *Deep Learning as part of Artificial Intelligence [32].*

a machine learning algorithm complete a new task that differs from the one for what the algorithm was first created.

Representation learning struggles when the problem is not just the problem to be solved, but also to obtain the representation. The factors of variation of the features that are to be evaluated could make this representation difficult to be built. For example, if we want to find a color, it will be different to find it depending of the source of light. At this point is where deep learning is useful. Deep learning introduce simpler representations so that the computer could define complex concepts out of the simpler ones.

Deep learning uses a series of layers where each one is built to identify simple features. The first layer will map some easy feature (as could be edges in an image), and the following ones will use the previous identified features to identify new ones (for example, to determine corners once edges have been determined). The fact of having many layers is also a pro in terms of computer programming, as each layer can be executed in parallel, making possible the flow of information from previous layers. To sum up with the differences between the different models of AI, Figure 2.18 taken from [32] shows the different steps that each model take.

Now, focusing on deep learning models, the last wave, the one that is now being used to train computers, started around 2006, with the idea of copying the way our brains work, with computational models of biological learning (that is why those networks are sometimes called Artificial Neural Networks). This way of applying deep learning has been interesting, not only because computers would be able to think somehow as a person, but also because it would be possible to understand better how the brain works. The problem is this last point, as not enough information about the brain has been acquired and it is too difficult to copy the way it works. Nowadays, the term ”deep learning” makes sense in terms of multiple levels of composition and the field of neuroscience is not anymore the predominant guide of deep learning (even if many ideas for deep learning models can be inspired by neuroscience). The fields that are most linked to deep learning are linear algebra,

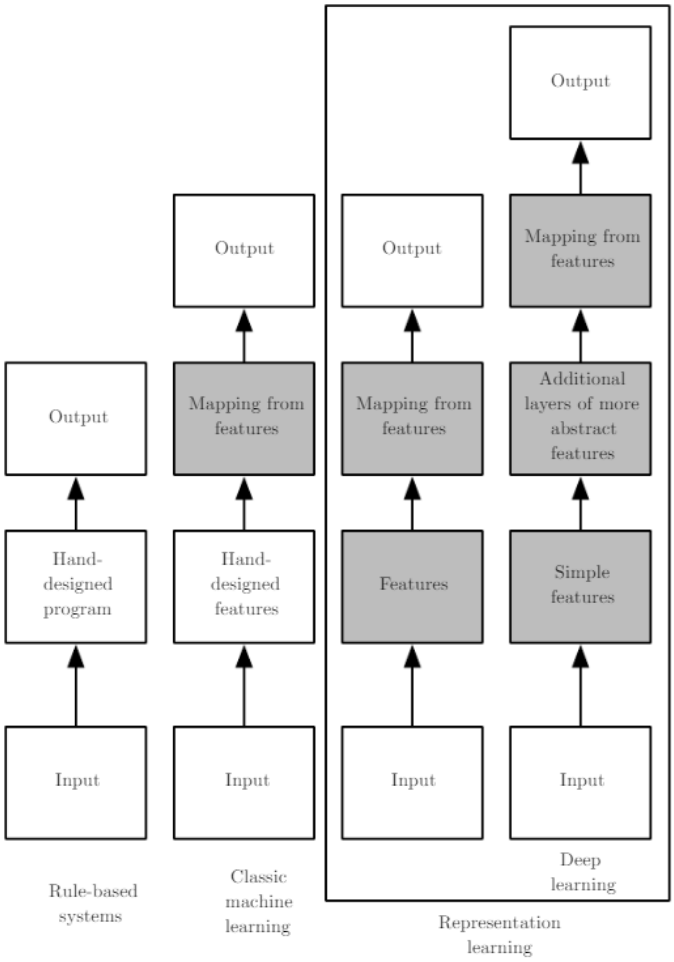


Figure 2.18 *Artificial Intelligence systems [32].*

probability, information theory and numerical optimization, according to [32], while the study of how brain works is related to computational neuroscience.

It is worth noting that the neural networks that during this last wave of deep learning that still continues had a change on the goal of research: firstly, around 2006, the goal was to solve unsupervised learning algorithms from small datasets, while nowadays, with the increasing amount of data that can be collected (for example, ImageNet dataset collects around 14 million of labeled images) and because of computer infrastructure has been improving during the last decade, the goal is to solve supervised learning algorithms with large labeled datasets.

As explained before, deep learning algorithms have multiple layers that evaluate from an input some simple features in the first layers and more complex information is obtained while going from one layer to another. Many neural networks are used to interconnect these layers, but the most common in image applications are Convolutional Neural Networks (CNNs), that are defined as neural networks that

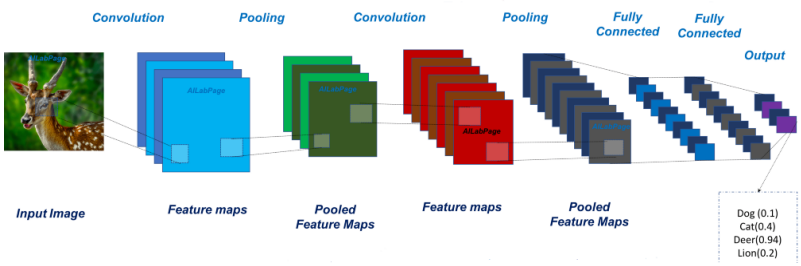


Figure 2.19 *Convolutional Neural Network [32].*

use convolution in at least one of their layers (filter that has been explained in the previous subsection). In addition to the convolution layers, there are many other hidden layers and parameters to build the CNN:

- Pooling layers: replace the output of a layer with an output that collect information about the neighborhood. For example, max pooling takes the maximum value within a specified neighborhood.

- Fully connected layers: as its name indicates, two layers are fully connected (i.e., every neuron from one layer is connected to every neuron of the following layer).

- Weights: values that are applied to the output of each neuron. Those outputs could be biased too. Normally, iterative processes are followed to achieve good results by varying the weights of the neurons that compose a neural network.

-ReLU: rectified linear unit. It is a unit that uses an activation function that activates the positive part of the function.

Figure 2.19 shows an example of a simple Convolutional Neural Network.

Once that a brief review of what is deep learning, firstly the most important image classification algorithms will be explained briefly and later some applications already applied to real-life situations will be exposed.

Currently, the most important image classification algorithms are (all of them have used the ImageNet dataset):

- AlexNet: composed by 5 consecutive convolutional filters (11x11 filters), max pooling layers and 3 fully-connected layers.The error-rate of this algorithm was 15.3% (ImageNet 2012).

- VGG16 model: composed by 16 convolutional layers, many max-pool layers and 3 fully-connected layers. This algorithm also used ReLU activation funcitons to connect multiple convolutional layers, and introduced smaller filters to the convolutional layers (3x3 filters). The error-rate of this algorithm was 7.3% (ImageNet 2014).

- Inception V3: after the development of the concept ”inception modules” in [33], what is known as training many convolutional layers simultaneously and link them with a multi-layer perceptron to achieve non-linear transformations, the first

version of this algorithm used this concept with modules composed by 1x1, 3x3, 5x5 convolutional layers with a 3x3 max-pool layer (6.7% error-rate, ImageNet 2014). The second version made some modifications (removed the 5x5 filter and added two 3x3 filters, a 3x3 convolution and a 3x1 fully-connected layer) to achieve a 5.6% as error-rate (ImageNet 2012). Finally, Inception V3 modified the first two layers to analyze higher-resolution images, achieving an error-rate of 3.58% (ImageNet 2012).

- ResNet: this algorithm incorporated the idea of residual learning, where output layers where connected to their inputs to understand the outputs and modify slightly some parameters to improve the accuracy. This algorithm was made of 152 convolutional layers with 3x3 filters and residual learning is included by block of two layers. The error-rate has been 3.57% (ImageNet 2015). After defining residual learning, Inception V4 mixed residual learning with inception modules to achieve an error-rate of 3.08% (ImageNet 2012) (ImageNet 2012).

- SE-ResNet: [34] linked the previous explained concepts (fully-connected layers, inception modules and residual blocks) and used a reduced number of parameters, obtaining an error-rate of 2.25% (ImageNet 2017)

It is worth mentioning that this algorithms must be trained before being tested, modifying parameters until the optimum network is reached.

In the industrial field, many deep learning applications have been used to include visual perception in robots, not only by using this most-used networks directly, but also using the main structure of these networks and modifying some parameters or training a new dataset more accurate to the application. In many cases, new deep learning models are proposed to develop the application.

In 2015, [35] presented a deep learning model with just 4 layers (input layer, self-organized layer, reservoir layer and output layer) to learn the inverse dynamics models for robotics. The inputs that the network receives are the position, velocity and acceleration of each joint, the self-organized layer received this inputs with the weights related and apply the GHL rule (an unsupervised learning rule). After this layer, the reservoir layer is composed by many fully interconnected layers that exchange information with other layers and with the output layers.

Another application of deep learning in robotics was done in 2016 by [36]. In this case, the goal was to obtain more tactile information of an object for robots, by applying a deep learning model to physical and visual interaction with the surfaces of an object. The approach done in this project was composed by one LSTM model for haptic inputs (composed by 10 recurrent units with a fully-connected layer and a ReLU) and a CNN model for visual inputs (uses the weights used for material recognition and the Inception V1 architecture), followed by a fusion layer that links both.

Finally, an article where grasp detection is approached by using deep learning

is presented. [37] used a Convolutional Neural Network where an 400x400 pixels RGBD image is down-sampled and evaluated with convolutional 3x3 filters and then up-sampled to obtain the grasps position. In this project, a high-accuracy for grasp detection and a high-speed is obtained, being a proper solution to detect the best grasping position for the robot.

2.4 Summary

After reviewing the literature related to the technologies that can be applied to solve the problem defined in this project, in this section a summary emphasizing the most important characteristics, advantages and disadvantages is presented. It can be used later in Chapter 3 to decide how the solution is going to be approached and the main decisions the author has proposed.

Firstly, the different characteristics of each category that have been introduced in Section 2.1 are summarized in Table 2.2, being the main topic of this Section and making easier the robot´s choice depending on the task.

Table 2.2 *Categorization based on topologies of Industrial Robots*

Industrial Robot	Characteristics
Articulated	Robots composed by more than two rotary joints that have high flexibility to perform tasks in a wide range of the workspace.
Cartesian	Robots that make linear movements along the three axes (X, Y, Z). A wrist in the gripper can add the rotational movement. The range of work is wide and is able to manage heavy capacities.
Cylindrical	The main movement is cylindrical and two more linear movements are included to enable 3D movements. Are fast to develop tasks with predefined points.
Spherical	A twisting joint is the main component, in addition to a linear movement allowed. Even if those robots where used years ago, nowadays the footprint that they have in industrial processes is small.
SCARA	The joints allow both vertical and horizontal movements. Two joints allow the movement along the horizontal plane and one linear movement is allowed along the vertical axis. Allow small capacities, but are fast compared to other types of robots.
Parallel	Robots that are built from a single base that connects many parallelograms. The main advantage of these robots is the high-speed repetition, such as separating certain types of objects.

In addition, in Section 2.1 the features that make Cobots an interesting advance in robotics are presented:

- Affordability: Cobots are a cheap solution to incorporate to manufacturing processes.

- Flexibility: Cobots are able to work in versatile tasks, being an adequate solution for dexterous tasks that require precision.

- Simplicity to use: in many cases Cobots can be programmed even by operators without programming experience, and also by teaching them manually manipulating their arms with our hands.

- Safety: the main advance that Cobots incorporate, making possible the interaction between humans and robots in the same workspace, developing together tasks and ensuring employee´s safety.

In Section 2.2 different categorizations of grasp and manipulation taxonomies are presented. The grasp taxonomies that are more accepted and complete are the exposed in [7] and [8], where different grasps done by humans are evaluated depending on different variables that are considered by the authors essential to describe the grasp (Figure 2.5 and Figure 2.9 show the different grasps that humans use to grasp different objects). Manipulation taxonomies are more concerned on how humans develop tasks that require the manipulation of objects, considering the entire process and not only the grasp. The most complete article that reviews this topic is the one presented by [26], where not only the different possibilities for human manipulation task are presented (Figure 2.10), but also a comparison to a three-fingers gripper is done, indicating the ones that couldn´t proceed for that gripper (Figure 2.11).

Finally, in Section 2.3 different approaches that have been done to study visual perception and apply it into computers are presented, and in Table 2.3 a summary of the advantages and disadvantages is presented.

Table 2.3 *Advantages and disadvantages of Visual Perception techniques.*

	Advantages	Disadvantages
Image processing operations	- Simplicity of the models. - High accuracy. - No need of large datasets to deploy a model. - Facility to include new requirements to the model from features that the developer could notice.	- Time consumed. - Undesired modifications in the image can lead to inaccurate solutions.
Deep Learning	- Time consumed to process the image. - High accuracy. - Adaptability to undesired changes in images.	- Time required to deploy the model. - Need of large datasets with labeled images. - Complexity of the models. - Complexity to add new requirements to the model.

3 Proposal and Implementation

In this chapter, the path to reach the solution is exposed. Firstly, the main components to be chosen for the thesis are defined, explaining the reasons to select them. Then, the implementation is presented explaining the procedure to fulfill the requirements.

3.1 Proposal

As previously defined, the main objective of this thesis is to demonstrate that perception technologies can be applied to robots to develop dexterous and precise tasks. To define the problem, the main components of the project have to be chosen: task to be developed, robot chosen for the task, technique to apply perception technologies and how the user could interact with the process.

First of all, the task to be fulfilled has to be exposed. In industrial processes, different tasks can be developed by robots in a proper way, where most of them are repetitive tasks where no visual perception is needed, but there are some others that require visual perception, where the workplace could be a changing environment. This task will be chosen in order to solve real industrial situations that are present nowadays in industry.

Once the task to be done is known, the robot used for manipulation has to be chosen. As explained in Section 2.1, there are different types of robots to work in industrial processes. The robot could be dual-arm, a cobot, 6 Degrees-of-freedom robot, etc., depending on the task to be developed.

The technique that is going to be used depends on the task and on the workspace. As explained previously, the visual perception could be achieved by processing the image, by using deep learning and others.

The user interface that will be applied has to interconnect all of the components needed to complete the process, facilitating the change of the main parameters of the process by the user directly and intuitively. The idea is to have a process where the user just have to define some parameters and the robot would complete the task according to the parameters introduced.

Task selection

The task selection will define the project, being a decision that affects following choices. There are many tasks that could be done by robots by using visual perception, but three were selected to decide which one could be applied to the project:

- Pick and place: this task would require 9 cubes to pick with a drawing and a box with nine spots where the same drawing is drawn. The objects would be positioned randomly in a table, and the robot would first detect the position and the drawing of the cubes and then detect the drawings from the box where the cubes should be placed. In this task, the user could modify the drawings to be detected, but the same drawings should be in the box.

- Assemble a connection panel: this task is similar to the task that was required around the 1920s, when phone companies needed personal to connect cables depending on the phone call that was received. In this case, cables would be positioned randomly in a table and a panel would be situated in front of the robot. The user would decide the number of cables needed, the color and the places to connect the cables in the panel. Once the cables would be connected, a circuit built besides the panel would be closed and some output would be set (for example, a led would be switched on). This kind of tasks that involve the manipulation of cables are normally developed by humans because of the complexity that shapes of cables have, but the possibility of using robots to do such tasks would improve many manufacturing processes, as could be the process that involves manufacturing a car, where a big amount of cables are manipulated by humans.

-Separate two types of objects in a conveyor belt: in this task, two different types of objects would be needed. Objects would approach the robot in randomly positions and the robot would detect what type of object is, separating the objects in two different boxes. The objects could be, for example, cubes and cylinders. This is a task that is being used in industries such as recycling, manufacturing,... where different products are separated by using visual perception in robots.

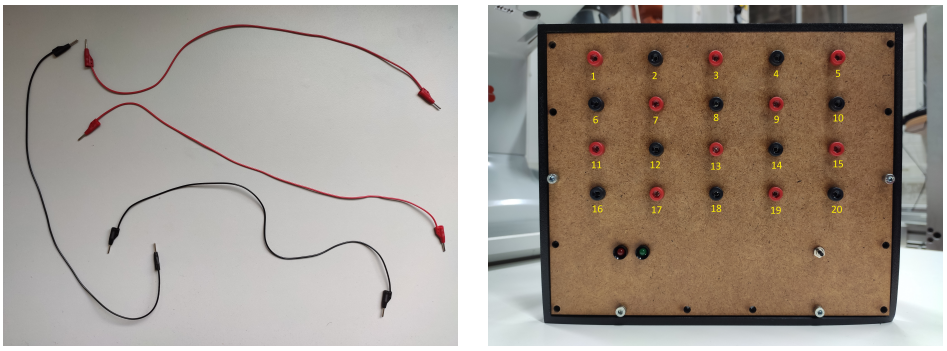
The task selected for this project is the assembly of a connection panel, where cables will be manipulated by a robot by using visual perception technologies. This task has been chosen as is a research topic that is being studied nowadays to satisfy the new requirements that the automotive industry is having to keep automating its processes. In this project, the task that involves the manipulation of cables is simpler than the one automotive industry is facing, but is a simulation of how this could be achieved by using perception technologies.

The workplace is shown in Figure 3.1, where the components used are:

- Banana cables: red and black 2mm banana cables 500mm length will be used.

- Board with a circuit: a board with a circuit built backwards is required. 20 pins have been situated in the board with a circuit composed by two leds (one red and one green), a switch and a battery. The user specify which pins should be connected to close the circuit.

As it is a researching field, there are a few works where robots have been used to detect and manipulate cables. One of these works is the one presented by J.Zhu et



(a) Table in the workspace (b) Board

Figure 3.1 *Workspace for the task.*

al [38], that defines a framework to manipulate a cable so that it reaches a desired shape. The components that are used in the final solution are a cable gripper (where the cable is already grasped, so the robot doesn´t have to grasp it), two different cables (one of 5,88mm and another of 7,66mm), a dual-arm robot and an image with resolution 1936x1456. The final solution uses image processing to obtain the current image of the cable and after that an algorithm (that uses Fourier series and evaluates the relation between the motion of the robot and the cable deformation shape) is applied to determine the movements that the robot must do to obtain the desired shape.

As explained in Section 2.1, two of the main challenges of Industrial Robots nowadays is to improve the dexterity of robots and to adapt robots to changing environments. To develop this task, both challenges have to be satisfied.

The main challenges that are going to be faced by developing this task are:

- Detection of cables whichever the initial shape is.

- Manipulation of deformable objects where the shape that the object will take is difficult to predict.

- Grasping an object that is slight where dexterity is needed to achieve good performance.

Robot selection

Once the task has been selected, the following decision is the robot that will develop the task.

In this case, the robot should be a dual-arm robot to grasp the cables in a proper way, and it is not needed a robot to manage heavy objects, as cables are light objects. Taking into account this two main considerations, and also because of the fact that this robot has integrated vision in the right hand, the robot selected is the ABB IRB 14000 (YuMi, shown in Figure 3.2), a collaborative robot that fits

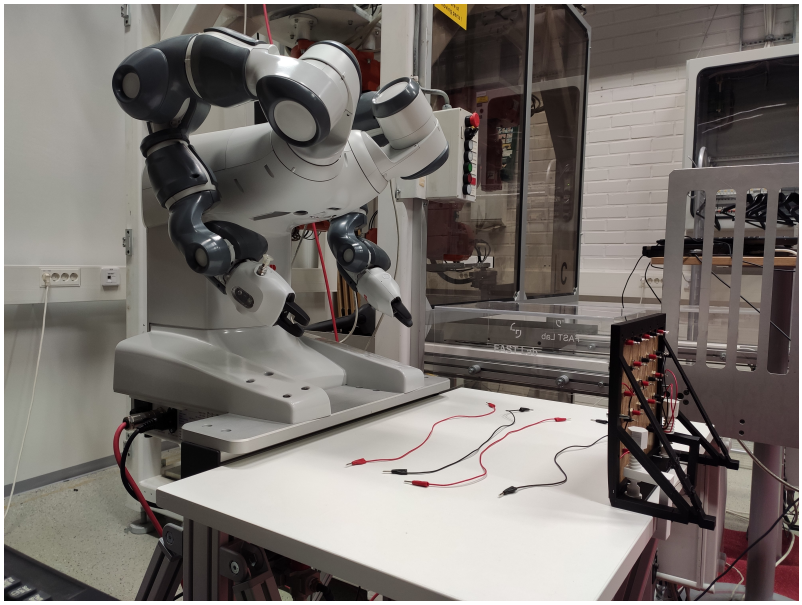


Figure 3.2 *Robot environment.*

with the task selected and whose range is enough to grasp cables of 500mm length. The characteristics of this robot are:

- Payload: 0,5kg per arm.
- Reach: 559mm.
- Accuracy: 0,02mm.
- Footprint: 339mm * 497mm.
- Grippers: SmartGrippers from ABB, where the gripper is able to recognize when an object is being picked. A gripper with two fingers and parallel motion will be used.

The Cobot is mounted on a table of 700mm*500mm, where cables are positioned. Additionally, a camera will be mounted on top of the Cobot to have a top vision of the table to detect the cables.

The selection of this Cobot allows the use of the RobotStudio software. It is a virtual environment where robots can be tested by using the same controller that is installed in the real robot. This enables the possibility of testing the robot with the RAPID code virtually and, once the task works, represent it in the real environment. Figure 3.3 shows a representation of the real environment in RobotStudio, where the cables are represented by cylinders, simulating the rigid parts of the cables.

To define the movements that the robot should do to grasp and insert the cables, the grasp and manipulation taxonomies presented in Section 2.2 must be evaluated. To evaluate the process, the task has been done by a person by using just two fingers (simulating a robot). During the evaluation, the person grasped the cable with a precision grasp (type 9 from Figure 2.5 or type 9 from Figure 2.9), and was accurate

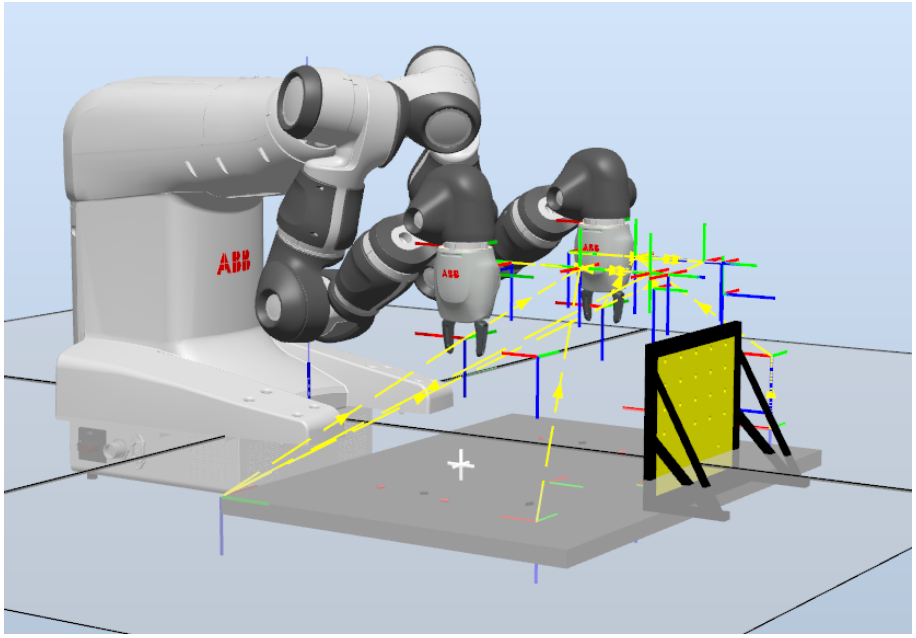


Figure 3.3 *Virtual environment.*

and strong enough to insert the cable, overcoming the resistance done by the female connections. However, when evaluating this grasp movement with the robot, it was noticed that the grasp was not strong enough to overcome the resistance from female connectors, so an intermediate step is introduced to grasp the cable with a precision grasp, but insert the cable with a power grasp (type 16 from Figure 2.5 or type 15 from Figure 2.9).

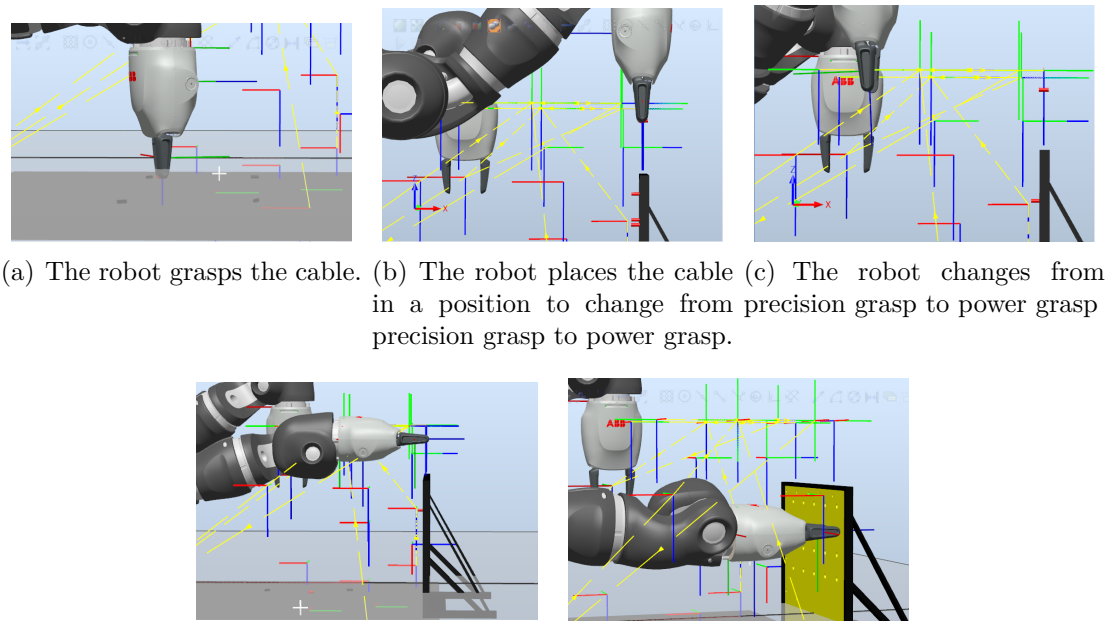
As representing cables in the virtual environment is a tough task where physics considerations must be taken into account, to simulate a simple and intuitive process, the process has been simulated with cylinders in RobotStudio, where the steps followed by the robot are shown in Figure 3.4.

Visual perception technique selection

The selection of the visual perception technique is also crucial for the project. It will determine the amount of data required, the amount of time consumed for the detection of cables and how accurate the grasp detection is.

The main techniques to determine grasp´s positions are:

- Deep Learning: it is the most common technique to develop object detection. Many libraries have been developed to facilitate the training and testing for models, and also many models have been already trained and are ready to be used. For example, Amazon Web Services have many models that can be used, as object detection, face recognition, cat or dog recognition or hot dog-not hot dog. Indeed, this models can be deployed easily by using the AWS console to a compatible device



(a) The robot grasps the cable. (b) The robot places the cable in a position to change from precision grasp to power grasp. (c) The robot changes from precision grasp to power grasp.

(d) The robot grasps the cable with a power grasp (e) The robot inserts the cable in the pin to close the circuit.

Figure 3.4 *Steps to develop the task.*

(for example, the DeepLens camera). When building a deep learning model, real-time detection could be achieved but, even if different neural networks already built can be adapted to new solutions, a big amount of labeled data is required to obtain proper results and normally high amount of time is consumed to train and deploy a model.

- Image processing: the second alternative to detect the cables and the grasping positions is by simply processing the image, understanding the environment. Basically this alternative takes into account the kind of image that will be processed, and according to the characteristics of the environment and applying different operations to the image, the grasping coordinates and angles would be calculated. Operations that could be applied to the image could be, for example, color filtering to filter the red cables and the convolutional filter, to reduce noise in the picture. To apply this technique, an image should be taken at the beginning of the process and once it is processed, coordinates would be sent to the robot.

In this project, image processing is used to analyze the image and to obtain the grasping coordinates, with the aim of being more accurate, without the need of gathering a big amount of data and reducing the complexity of the algorithm. In Section 2.3 some projects that detect objects are presented, where most of them use deep learning models. Sometimes, those models increase the complexity of the solutions to approach a solution that could be implemented by simpler solutions.

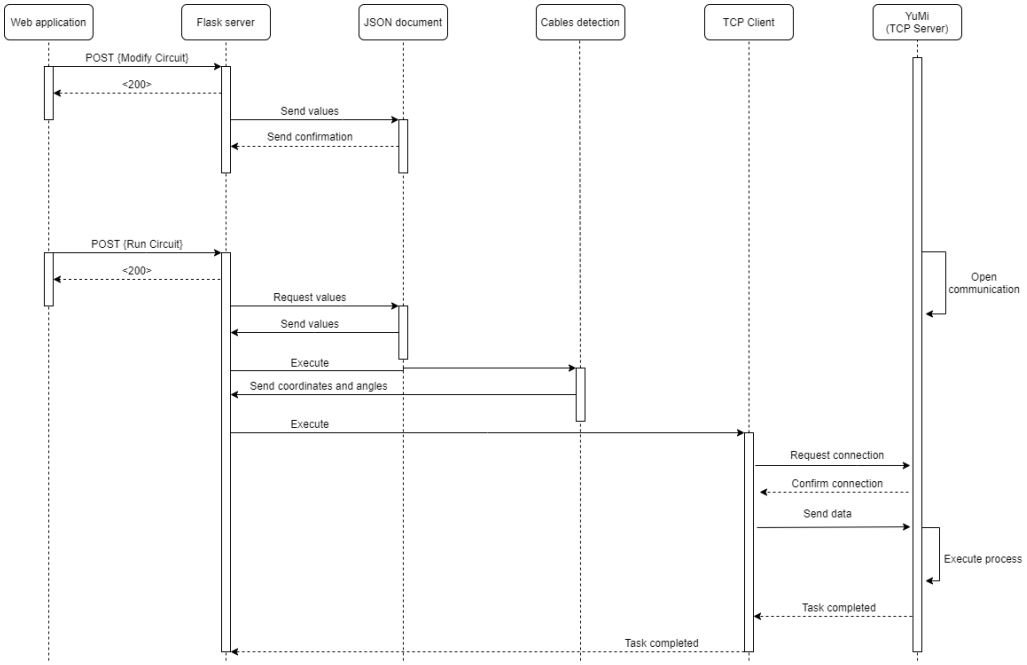


Figure 3.5 *Sequence diagram.*

For example, if the environment that is approached changes but in a range that is known and that can be determined by image processing solutions, there is no need to implement more complex deep learning solutions, as it would increase the complexity and the time and data required to develop the algorithm.

Guide User Interface

The last decision to make is the interface where the user could interact with the task. The final decision have been to use an interface where the user would be able to modify the pins where the cables should be inserted and the amount of cables and its color. Also the possibility of saving many circuits is included, being the user the one that selects the circuit to run.

This GUI must include the connections that are needed in this project: support and connection between front-end and back-end and TCP/IP connections between robot and laptop via sockets.

Figure 3.5 shows a sequence diagram that sums up the general idea on how the interface would interrelate the different programs and scripts that will be present in the solution. In addition to this diagram, a Use Case Diagram is also added to facilitate the understanding of the process (Figure 3.6).

This GUI will be implemented using a Service-Oriented Architecture that will allow interoperability between the different machines that are used, by means of Web Services. To accomplish this, a web framework will be needed to support the

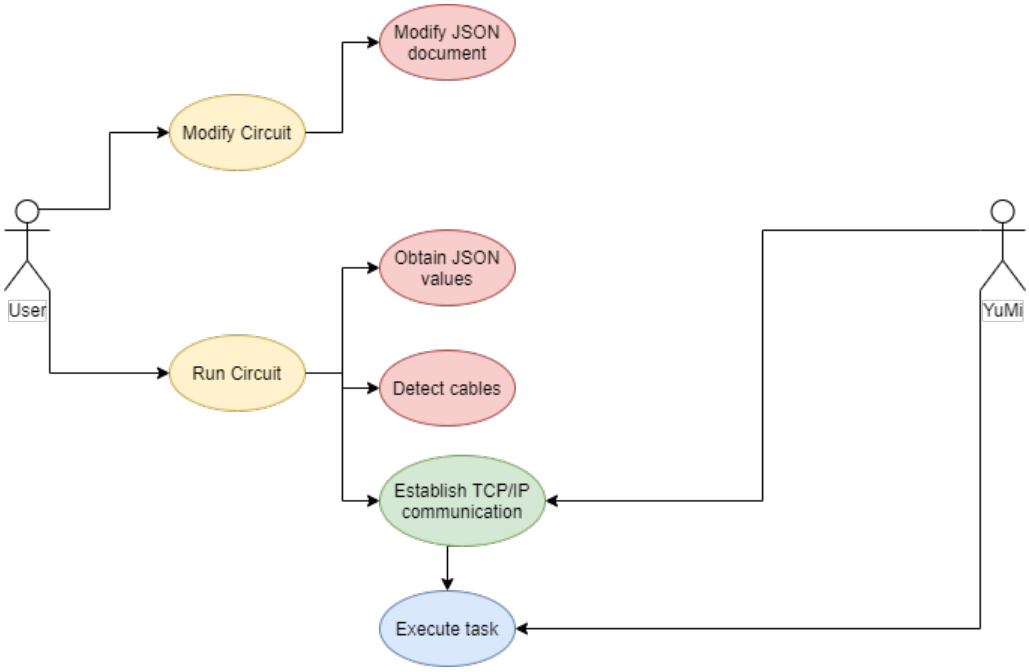


Figure 3.6 *Use Case diagram.*

development, being the Flask micro web framework a good solution for simple applications that do not require high security. RESTful Web Services (REpresentational Stateless Transfer) will be used, in order to achieve scalable, lightweight and modifiable Web Services, and using the method HTTP, whose requests can be used to create, read, update or delete data: POST, GET, PUT or DELETE. By using these methods, the client can make those operations to the server, and a response will be obtained meaning if the request was successful or not: 2xx means successful request, 4xx means problem with the request sent by the client and 5xx means server failed to process the request.

Data format

To interconnect different programs, it is important to define which data format is going to be used, as it will determine how data is saved and how has to be accessed. REST can work with different types of data: XML, JSON, HTML,... The three main formats that are used as data format are:

- XML: eXtensible Markup Language. It is a markup language that is easy to understand for humans and where data is typeless (all data is a string and it is needed to be parsed). The advantages of these format is the security that it has, the display options and the possibility of using various data types such as images, charts, numbers, text, ... To encode, it supports various encoding.

- JSON: JavaScript Object Notation. It is an open standard data interchange

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<root>
        <Circuit 1>
                <id>01</id>
                <Color>Red</Color>
                <Pin 1>3</Pin 1>
                <Pin 2>7</Pin 2>
        </Circuit 1>
        <Circuit 1>
                <id>02</id>
                <Color>Black</Color>
                <Pin 1>10</Pin 1>
                <Pin 2>15</Pin 2>
        </Circuit 1>
</root>
```

(a) XML format.

```json
{
        "Circuit 1":[
                {
                        "Color":"Red",
                        "Pin 1":"3",
                        "Pin 2":"7"
                },
                {
                        "Color":"Black",
                        "Pin 1":"10",
                        "Pin 2":"15"
                }
                ]
}
```

(b) JSON format.

```csv
Circuit 1,Red,3,7,Black,10,15
```

(c) CSV format.

Figure 3.7 *Most used data formats.*

format that is easy to read and write by humans and to generate by machines. In addition, it is lightweight, but only supports text and number data type (JSON types are string, number, array and Boolean). The only encoding that supports is UTF-8.

- CSV: Comma Separated Values. This is the simpler data format, separating values just with commas. Its simplicity is also a disadvantage, as it is difficult to modify data, to understand the data and to mix different data types.

Figure 3.7 shows an example of how the same information could be saved in these three formats. It could be used a data format to store some data and another data format to store other data. In this project, the JSON format is used, as is considered the most intuitive and easy to use.

3.2 Implementation

To implement the requirements above indicated as the proposal, three main steps have been followed: image processing, connection between PC and robot and development of a Guide User Interface to simplify the interaction and the modification of parameters. It is worth mentioning that the programming language that has been chosen to implement the code has been Python, being considered an intuitive language where a big amount of open-source libraries are available for a big range of applications. In this project, the libraries that has been used are:

- OpenCV: this library enables the application of computer vision. In this

project, the main use of this library will be the image processing, where operations can be completed at real-time, simplifying and reducing the time consumed by the final application.

- Flask: this library facilitates the creation of a micro web framework where an application can be created. In this project, it will be used to create the GUI where the user will be able to interact with the robot and with the process.

- Socket: this library will be used to do the communication between the laptop and the robot via TCP/IP.

- Numpy: this library is used to operate with arrays and to make algebra operations. In this project, as images can be considered as arrays, this library will be used to do operations in images.

- Json: this library facilitates the use of the JSON format in Python. With this library it is possible to convert a dictionary or a list into a JSON file and vice-versa.

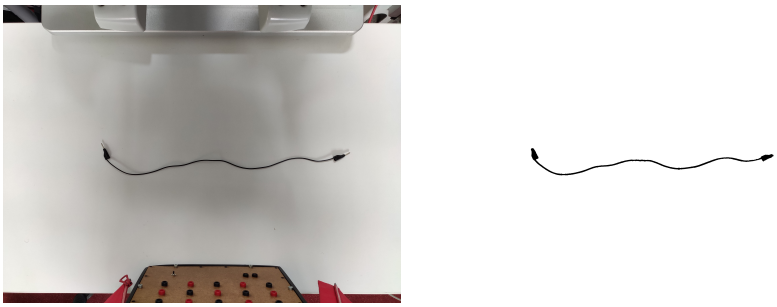
3.2.1 Image processing

As commented above, image will be processed to obtain the coordinates of the grasps. Image processing has three main goals: separate cables by colors, detect the grip zone of the cable and detect which grasps belong to the same cable. It is worth mentioning that the first operation to do is to remove the background from the image (robot components that are on the table and the board that is also in the image). Due to the COVID-19 pandemic and the lockdown that had to be implemented in the university, the images that are going to be used just have the Region of Interest (ROI) of the environment, i.e., the cables with different orientations and positions (no robot or board are in the picture), as just one image was taken with a cable and the real environment was taken. The operation that should be applied in real operation is the one that is shown in Figure 3.8, where the background is removed and just the cables remain (this picture is the only one that was taken with the whole environment).

The first stage when processing the image is to do some operations that will separate the image by colours, smooth the image and then separate the image into two (per colour): an image with the cables and an image with just the grasps. To obtain these filtered images, the following steps have been followed (Figure 3.9):

- Convert the RGB input image (2661x1637 pixels) to HSV and apply masks to separate colors (Red and Black). From this input image, two images are obtained: one with black cables and the other one with red cables, being both output images with information in RGB even if the input image was in HSV format.

- Apply 2D convolutional filter, with a kernel matrix 10x10. After applying this filter, a smoother image will be obtained, deleting random holes and the noise the image could have.



(a) Cable with its full environment. (b) Black cable detected.

Figure 3.8 *Removal of the background.*

- Convert both images (black and red) to black and white images, with the background in black and the cables in white.

- Apply the opening morphological operation with a Kernel matrix 20x20. By doing this, the output image will just have the grasps of the cables, as are the regions where a higher concentration of white pixels is observed.

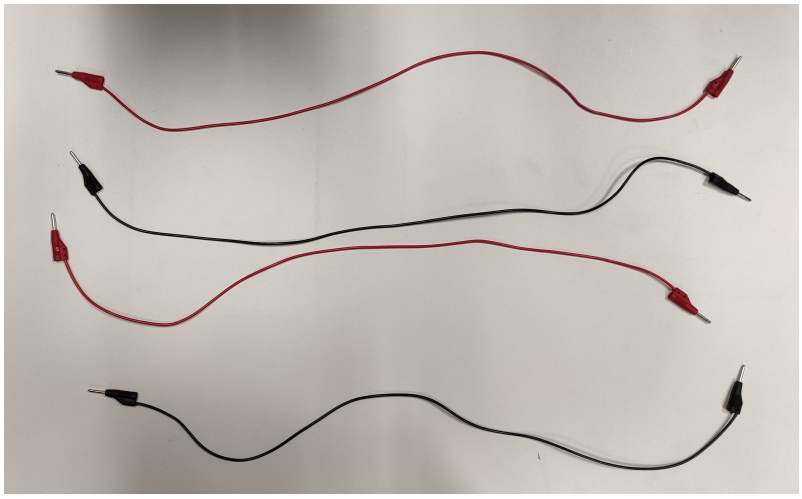
The second stage is to determine from that images what is a grasp and what is a cable, by sweeping the image and establishing relations. The idea is to have gathered the pixels that belong to the same grasp or cable. It is worth mentioning that before applying this stage, the cables are modified to black color and background to white, and the size is changed to an appropriate one that reduce the time consumed, relates pixels with real information and obtains proper results (explained in Subsection 3.2.1). To gather this information, the following steps have been followed:

- Sweep the image looking for black pixels, gathering the ones that have a distance of less than 5 ”pixels” as pixels from the same group (the distance is more than 1 pixel because still some noise could have remained in the image). This is done with both the image that has the cables and the one that has the grasps. The image must be swept in both directions to avoid problems clustering black pixels. (Figure 3.10 for the cables and Figure 3.11 for the grasps.)

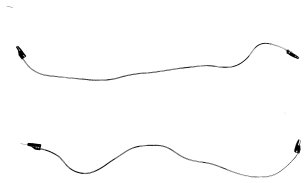
- Once the black pixels have been clustered, the false positives must be removed. The groups that cluster less than 10 pixels are removed. This has been considered evaluating the number of pixels that a cable and a grasp has. Those false positives can occur in the image that contains the cables, as the one with the grasps has been opened wider.

The final stage is to process this information and obtain useful data, understanding which grasps belong to the same cable and obtaining the center and the angle of each grasp. To obtain this final data, the following steps have been followed:

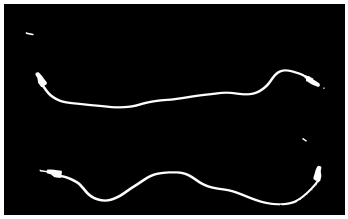
- For each group of pixels that determine the grasp, obtain the average value



(a) Input image.



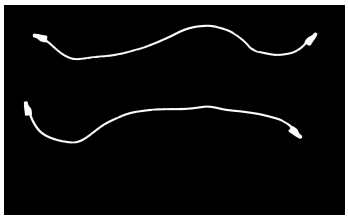
(b) Black mask.



(c) Black Convolutional filter.

(d) Black Opening filter.

(e) Red mask.



(f) Red Convolutional filter.

(g) Red Opening filter.

Figure 3.9 *Image preprocessing.*

that would represent the center of the grasp.

- Once that the center of each grasp has been calculated, the pixel´s group that represents the grasp is swept, and the pixel whose distance is bigger is saved to determine the orientation of the grasp.

- To calculate the angle that defines the orientation of the grasp, must be considered that this information will be sent to the robot. The sense that represents the orientation of the grasps is clockwise (Figure 3.12).

With this last stage, the information required to grasp the cables has been collected: position and orientation of the grasps. The information regarding the position of the grasps saves the pixels where the grasps are. Before sending that information to the robot, the real coordinates from the table must be calculated.

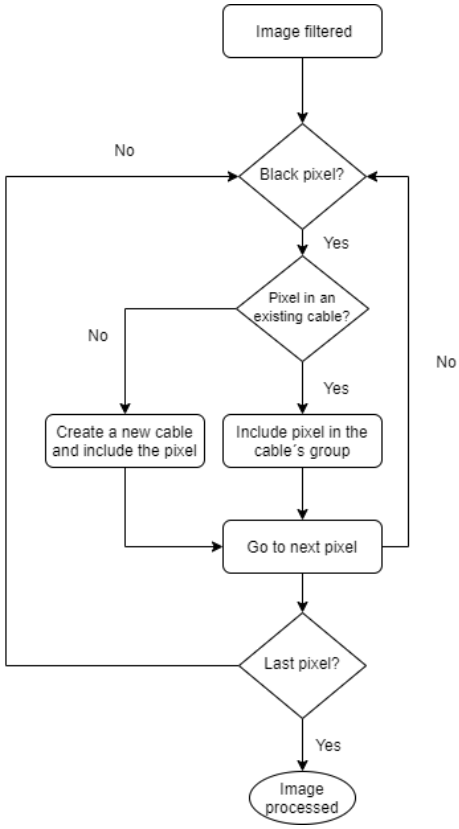


Figure 3.10 *Filtered image processing flow chart.*

Transformation from pixels to real coordinates

As explained in Subsection 2.3.1, to obtain real information about the image, it has to be transformed from 2D to 3D. As the table where cables will be placed is fixed at a predefined height and the camera is fixed too, the coordinate Z is already known (the camera is positioned at a height to fit the sides of the table in the image). To relate the real world with the image, the relation between some points from the image and the real world must be calculated.

To obtain this relation, the corners of the table are evaluated, as are known points from the real world and are easy to detect from the image. Once these points are related, as both the camera and the table are on parallel planes, this relation between corner points in the image and in the real world will remain for the rest of the points.

Figure 3.8(b) represents the situation of the robot in its workplace. After applying some operations to that picture, it was determined that by resizing the image to 700x550 (width and height), the distance of one pixel would represent 1mm in the real environment. It is worth mentioning that image is resized to 350x275 before doing the Stage 2 of Image Processing, to preprocess the image without losing information and to make the detection of the grasps and calculation of the angles faster.

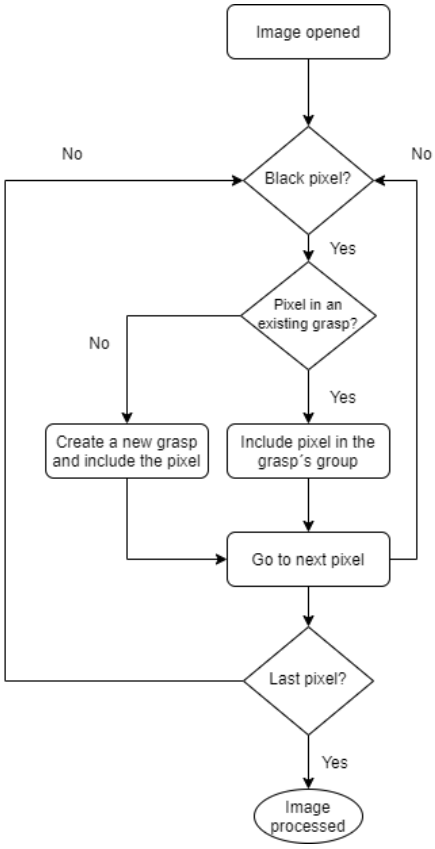


Figure 3.11 *Opened image processing flow chart.*

Moreover, in the simulation done during this project, due to the lack of images will the full environment, it has been obtained after evaluating the images that the ROI covered around 500mm in width and 322mm in height (having the same relation to ensure the ratio width/height), so before the Stage 2 during the simulation, the image will be resized to 250x161 pixels as a first step to evaluate the image. In the following sections, this resolution will be evaluated to determine the most adequate to ensure adequate results and reduced time consumed.

Once this relation has been applied and the information regarding the detection of cables is stored, it is time to create the communication between robot and laptop to send it.

3.2.2 Networking communication

At this point, and assuming that the pins where the cables should be inserted have been collected too (in the GUI, the user is able to modify them, what will be explained in Subsection 3.2.3), the connection between the robot and the laptop must be done.

To communicate many devices by using the Internet, there are different internet

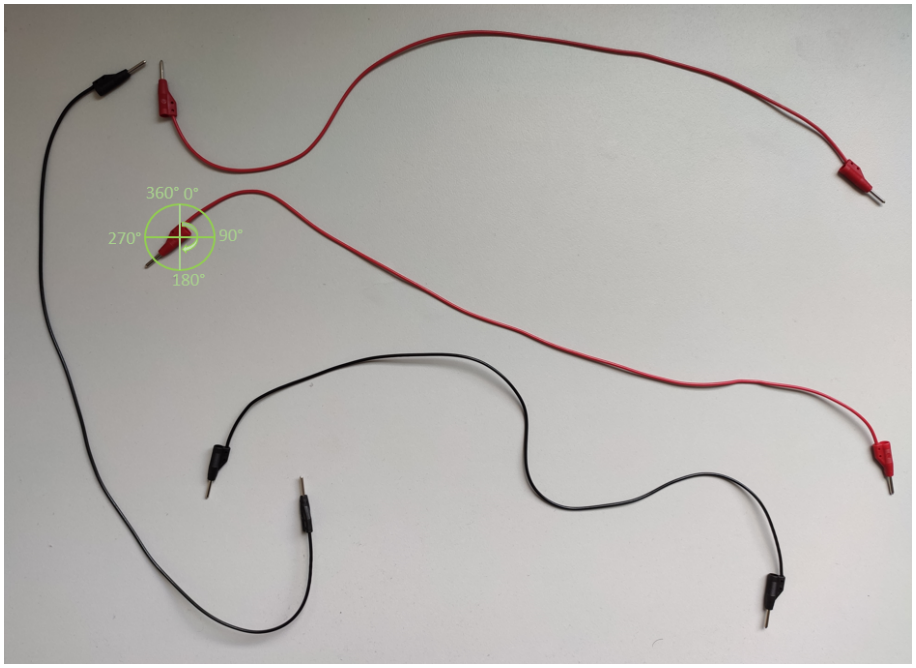


Figure 3.12 *Orientation of the grasps.*

protocol suites based on different layers to provide this connection. The most extense protocol suite that is used currently is the OSI model [39], which is defined by 7 layers that build this communication between two networked systems, performing functions to the layers that are above a specific layer and offering services to the layers from below. Those 7 layers are: (1) Physical; (2) Data link; (3) Network; (4) Transport; (5) Session; (6) Presentation; and (7) Application. Depending on the protocols that are used in each layer to develop a task, some of this layers can be joined. For example, the most known protocol suite used is the TCP/IP model. This model joins layers (1) and (2) as the network access layer, layer (3) is the network layer, layer (4) is the transport layer and layers (5), (6) and (7) are the process layer. Figure 3.13 shows the relation between TCP/IP model and the OSI layers.

In this project, the networkwing communication has been done using the TCP/IP model. The data link protocol that is going to be used is Ethernet, but WiFi could also be used, being both devices connected to the same network (FASTory), being the physical communication between the devices. The network protocol that will be used is the Internet Protocol version 6 (IPv6), which is responsible for addressing host interfaces, encapsulating the data to send into datagrams and routing those datagrams from a source host interface to a destination host interface, defining the format of packets and providing an addressing system. The transport protocol that will be used is the Transmission Control Protocol (TCP) as TCP is considered more reliable than the User Datagram Protocol (UDP), which is considered faster

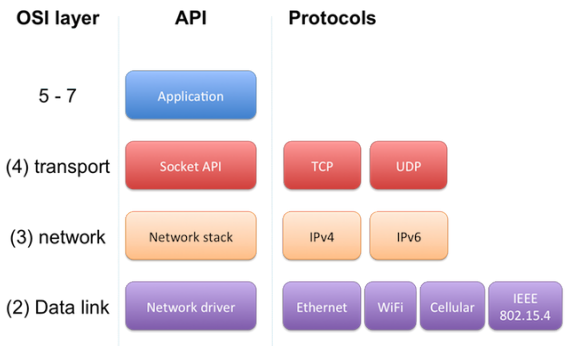


Figure 3.13 *TCP/IP model [39].*

to transfer data. It is worth mentioning the role of Socket APIs to enable this networking communication between devices.

Network sockets are internal endpoints that connect two networking devices in order to send and receive data within a node on a computer network. RAPID and Python have both libraries to simplify the connection between two networking devices. In this case, the server host is the robot, that will wait until the laptop sends a requirement to establish a connection by using a port and the IP of the robot. Once the connection is done, all the information that the robot needs is sent and the robot is ready to develop the task.

3.2.3 Guide User Interface

The Guide User Interface, as explained in Section 3.1, is developed using the Web Services, by using the localhost. The main functionalities of this GUI are:

- Show the streaming of the camera that is being used to detect the cables, showing the detection of cables at the same time and the cable that is being manipulated by the robot. As in this case it is not possible to make real simulations, with the purpose of testing the project, images taken from the real environment are used to show what is supposed to be the workspace.

- Modify the circuit that has to be built in the board. This modifications will be done by defining the number of cables that are needed and the pins where cables should be connected. Also will be showed the information regarding the connections that are saved at the time the user is using this interface. Moreover, three predefined circuits are included that the user can select.

- Run the process that makes the detection of the cables and where the robot insert the cables in the position required. It is worth mentioning that before this button is clicked, the code from the robot has to be run to enable the server and to let the communication start.

As explained in Section 3.1, the flask framework is used to built this web application, and the HTTP method is used to GET or POST the parameters to build the circuit, being this information saved in a JSON document.

4 Tests and results

Due to the COVID-19 pandemic, the tests that have been done are in the virtual environment, and with conditions that differ from those from the real environment (light conditions, pictures taken from a nearer position to the table, ...). With this situation, what has been done is simulate the process in the virtual environment, processing different images with different conditions (cables in horizontal position, one cable, many cables, different colors, ..) and making a simulation where cylinders are inserted instead of cables. It is worth mentioning that in this project crossed cables won´t be approached, as while doing the detection of cables that have the same color and are crossed, to understand which cable is above and which one is below, a side-image would be needed and processed by using the camera that the robot has in its right hand (it is left as a future work that could be approached in the future).

So, 10 pictures of cables positioned in horizontal and 10 pictures of cables positioned in random positions will be tested to prove the detection of grasps and cables. Moreover, 5 pictures of crossed cables with different colors will be tested too.

As while doing the simulation the coordinates of the cylinders have to be set by the user (as it is the virtual environment, if it was done in the real environment, it would not be needed), the manipulation of the cables will be evaluated just once, as for the rest of the tested images it will behave similarly.

The evaluation will be divided in three: detection of grasps, detection the cable which grasps belong to and orientation of the grasps.

The time consumed to evaluate and save 25 images have been 64.625 seconds.

Figure 4.1 shows the images that has been evaluated, with the grasps that have been detected and the orientation they have. The results of this evaluation are presented in Table 4.1.

Table 4.1 *Results of the tests.*

Grasps detected	Cable-Grasps relation detected	Orientation detected
136 of 140	68 of 70	123 of 140
97.14%	97.14%	87.86%

The results show how accurate is the proposed solution, detecting 136 grasps out of the 140 grasps the different images had, detecting the relation between those grasps and the cables for all the grasps detected and obtaining the proper orientation of the grasps for 123 out of 140 grasps, meaning a 87.87% of accuracy in this last

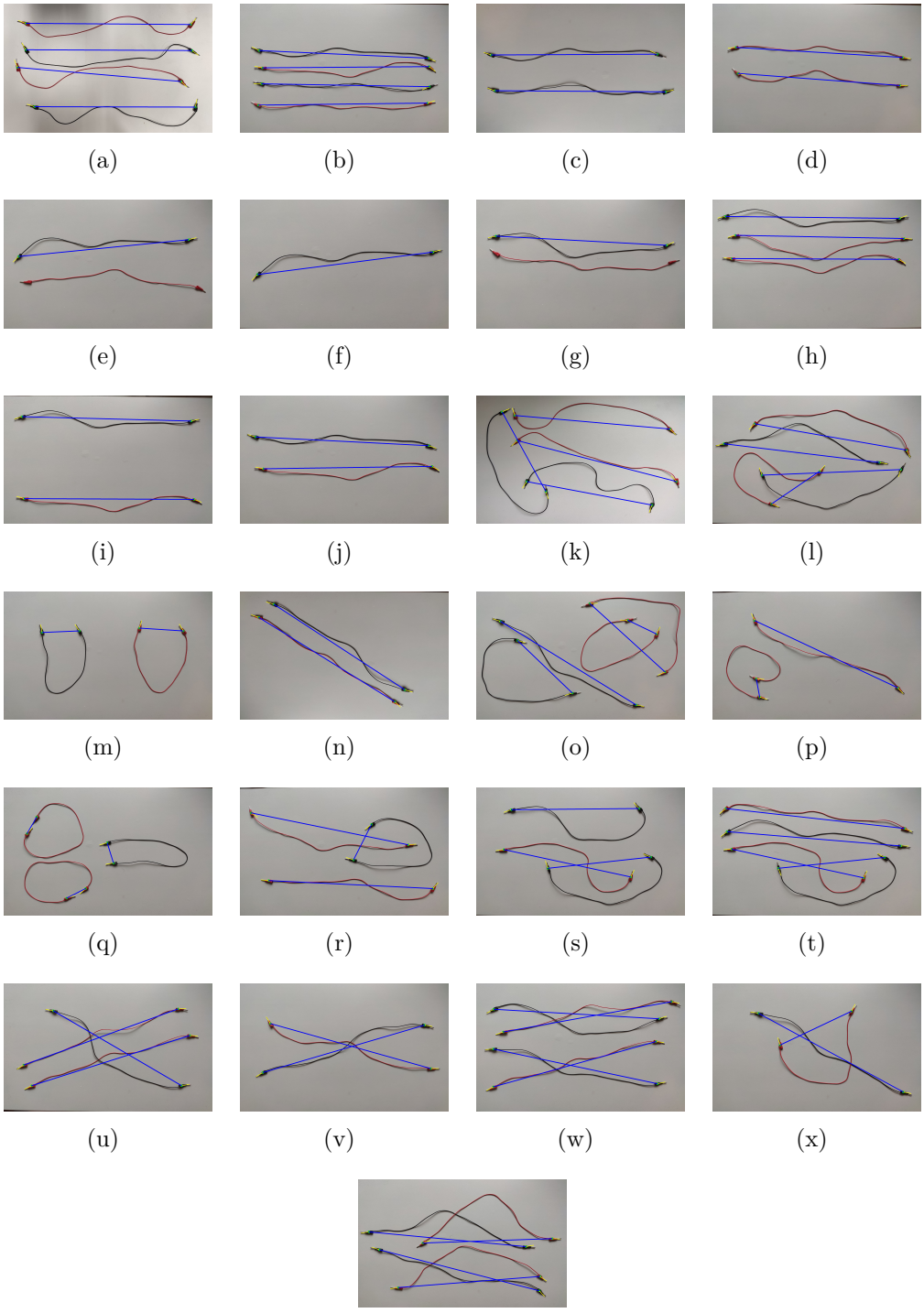


Figure 4.1 *Images tested.*

feature.

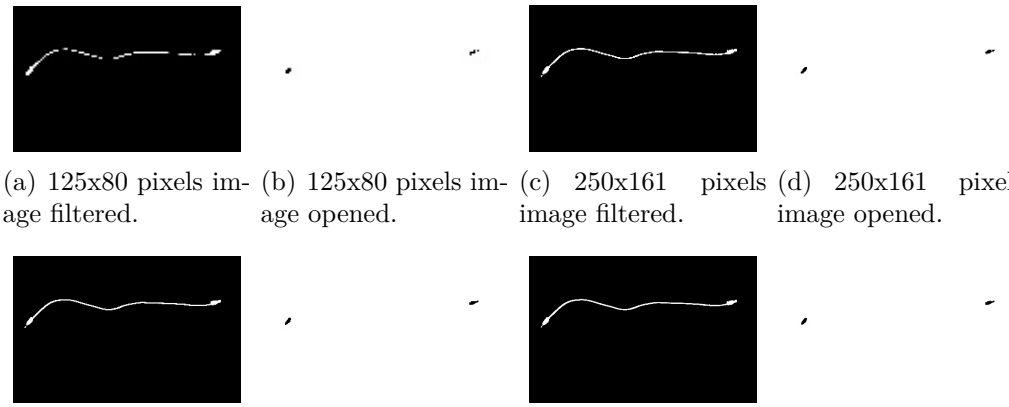
Before evaluating the whole process, it is also interesting to evaluate how does the size of the image to evaluate affects to the results, evaluating the time consumed to process the images and the accuracy obtained. It has to be mentioned that by modifying the size of the image to evaluate (second stage of Subsection 3.2.1), the values that defined how far two pixels can be to be considered from the same cable (because of the noise an image could have) and how many pixels are needed at least to define a grasp must be changed according to the new size to evaluate, for example, if, instead of evaluating 250x161 pixels, 125x80 pixels are evaluated, the values that should be defined are 2.5 pixels to consider two pixels being from the same cable and at least 5 pixels to consider a cluster of pixels being a grasp. So, the same 25 images will be tested by using different sizes: 62x40, 125x80, 250x161 (already evaluated) and 500x322. Obviously, the pixels that define the grasp have to be adapted to be used in the real environment depending on the size used to evaluate the image. The results obtained depending on the size of the image evaluated are shown in Table 4.2.

Table 4.2 *Results depending on the resolution of the image to evaluate.*

Image size	Time consumed	Grasps detected	Cable-Grasps relation detected	Orientation detected
125x80	22.36s	5.71%	5.71%	5%
187x120	34.34s	64.29%	64.29%	60.71%
250x161	67.87s	97.14%	97.14%	87.86%
375x241	217.25s	97.14%	97.14%	88.57%
500x322	602.31s	95.71%	95.71%	88.57%

This evaluation shows the lost of information that an image can undergo if the size chosen is not adequate, and how a process can be accelerated by reducing it. The balance between both parameters define the best solution. In this case, it is easy to find the best approach, as with 250x161pixels high accuracy is achieved and by evaluating more pixels too much time is consumed and the increase in accuracy is not enough (by using 500x322pixels, accuracy is worse).The detection of cables and grasps is good enough, but when detecting the orientation, better results could be obtained (this can be achieved ensuring that the point that is furthest from the center point of the grasp represents the ending point of the grasp, evaluating the surroundings of this pixel from the image that has the cables in the direction of the line that joins this point with the center, to ensure that those are white pixels, representing the end of the cable).

Figure 4.2 shows the modifications images have when modifying the resolution.



(a) 125x80 pixels image filtered. (b) 125x80 pixels image opened. (c) 250x161 pixels image filtered. (d) 250x161 pixels image opened.

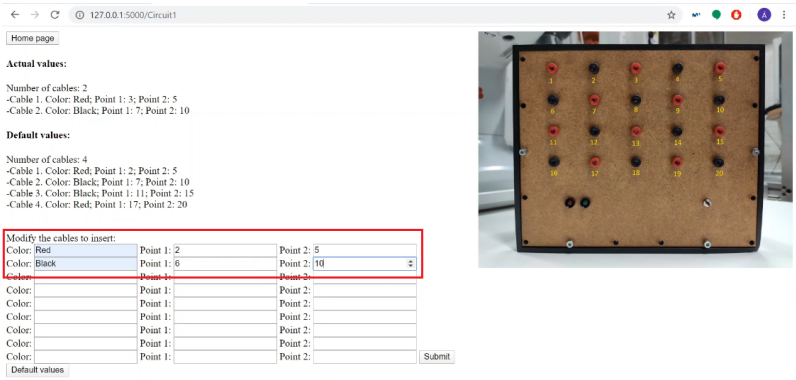
(e) 375x241 pixels image filtered. (f) 375x241 pixels image opened. (g) 500x322 pixels image opened (h) 500x322 pixels image filtered.

Figure 4.2 *Different resolutions to process the image.*

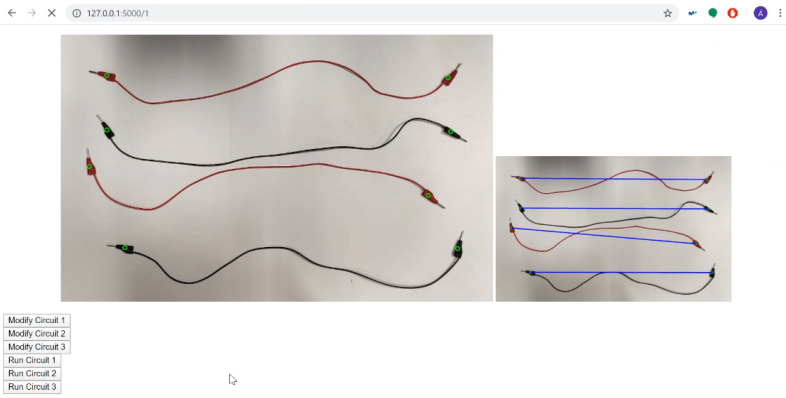
It is possible to see how the cable loses information if the resolution is not high enough, in some cases the grasps are isolated in a proper way but cables seem to be separated.

Once the detection of grasps have been evaluated, the whole procedure is evaluated. In this case, as mentioned above, the simulation will be done just with one image in the virtual environment as cylinders are positioned according to the information obtained from the image (a future work would be to test the application in real environment to see if the procedure works properly with different cables configurations).

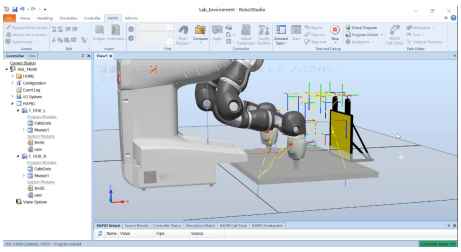
The results that have been obtained after executing the whole process are presented in Figure 4.3, where a red cable is selected to be connected between pins 2 and 5 and a black cable is selected to be connected between pins 6 and 10. The accuracy of the insertion is highlighted, as it is possible to see that the center of the cylinders are exactly in the position of the holes. In this figure it is possible to observe also how does the GUI looks like and its functionalities.



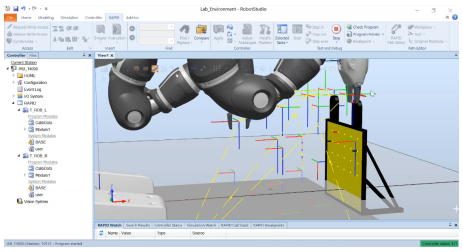
(a) Modification of parameters.



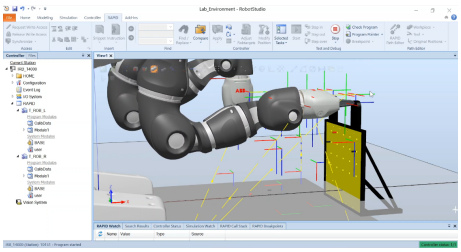
(b) Detection of cables.



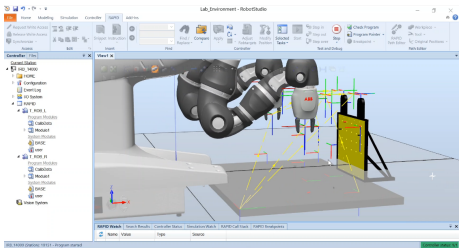
(c) Robot grasps a cable.



(d) Robot approach the intermediate step with a precision grasp.



(e) Robot approach the intermediate step with a power grasp.



(f) Process finished.

Figure 4.3 *Evaluation of the whole process.*

5 Conclusions

Robots are already a big asset for manufacturing companies from different sizes. We can find robots welding, picking and placing objects, painting, etc. making processes more efficient and profitable for companies. Nowadays, new technologies are being developed and used to improve the dexterity and performance of robots. As explained in Chapter 1, perception technologies are being introduced to robots to have a better understanding of the environment and make the robot self-supported to detect variations in the environment and act depending on it.

In this project, it has been proved that the use of perception technologies by simply processing images can achieve proper results, reducing the time consumed to deploy the model if it is compared to the use of Deep Learning models and obtaining high accuracy in the detection (97,14% of accuracy). During the evaluation of the project, it has been proved that by using robots the time consumed to identify the endpoints of the connections has been reduced, as robots do not need any time to detect them. It has been also noticed the importance of the resolution of the image to obtain proper results and its relation with the time consumed to detect the cables.

In addition, it has been also proved that robots can handle deformable linear objects in simple tasks, evaluating which are the most adequate parts of the cable for the manipulation.

Due to the COVID-19, it was not possible to do some tests to the model, what could be included as future works to do. Indeed, there are also some other works that could be done to improve the algorithm and compare different approaches to the same problem. Those future works that the author considers would complement this work are the following ones:

- Test the project with the real robot in the real environment. Make adjustments to the relation between the coordinates from the table and pixels from the image if needed.

- Compare time consumed developing the task a human and the robot.

- Improve the script to detect which cable is above if there are cables with the same color. The camera that the robot has in the right hand could be used to obtain images from different sides.

- Extend the capabilities of the detection by using different types of cables with different dimensions.

- Develop a Deep Learning model to detect the grasps and orientations and compare both models.

- Evaluate different grippers that could assemble the connection panel to determine the best solution for the real environment.

- Make deeper research about the compression of images without losing useful information for the detection. This would lead to a reduction in the time consumed to process the image, but the accuracy of the model should be the same or higher.

Bibliography

[1] Calderone, L. (2016, February 8). Robots in Manufacturing Applications. Retrieved from https://www.manufacturingtomorrow.com/article/2016/07/robots-in-manufacturing-applications/8333

[2] Keene, M. (2020, January 28). Trends in Industrial Robotics to Watch in 2020. Retrieved from https://www.roboticstomorrow.com/story/2020/01/trends-in-industrial-robotics-to-watch-in-2020/14711/

[3] Francis, S. (2020, January 29). Collaborative robots contribute towards changing perceptions of automotive robotics. Retrieved from https://roboticsandautomationnews.com/2020/01/29/collaborative-robots-contribute-towards-changing-perceptions-of-automotive-robotics/29260/

[4] Hand, S. (2020, January 8). Collaborative Robotics: History and Recent Trends. Retrieved from https://www.newequipment.com/industry-trends/article/21120041/collaborative-robotics-history-and-recent-trends

[5] Wilson, M. (2014). Implementation of robot systems: an introduction to robotics, automation, and successful systems integration in manufacturing. Butterworth-Heinemann.

[6] Owen-Hill, A. (2019, December 19). The Top 10 Collaborative Robot Industries in 2020. Retrieved from https://blog.robotiq.com/the-top-10-collaborative-robot-industries-in-2020

[7] Cutkosky, M. R. (1989). On grasp choice, grasp models, and the design of hands for manufacturing tasks. IEEE Transactions on robotics and automation, 5(3), 269-279.

[8] Feix, T., Romero, J., Schmiedmayer, H. B., Dollar, A. M., Kragic, D. (2015). The grasp taxonomy of human grasp types. IEEE Transactions on Human-Machine Systems, 46(1), 66-77.

[9] Hanafusa, H., Asada, H. (1978). A robot hand with elastic fingers and its application to assembly process. In Information-Control Problems in Manufacturing Technology (pp. 127-138). Pergamon.

[10] Ji, Z. (1987). Dexterous hands: Optimizing grasp by design and planning(Ph. D. Thesis).

[11] Kerr, J., Roth, B. (1986). Analysis of multifingered hands. The International Journal of Robotics Research, 4(4), 3-17.

[12] Nakamura, Y., Nagai, K., Yoshikawa, T. (1989). Dynamics and stability in coordination of multiple robotic mechanisms. The International Journal of Robotics Research, 8(2), 44-61.

[13] Schwarz, R. J., Taylor, C. L. (1955). The anatomy and mechanics of the human hand. Artificial limbs, 2(2), 22-35.

[14] Napier, J. R. (1956). The prehensile movements of the human hand. The Journal of bone and joint surgery. British volume, 38(4), 902-913.

[15] Kang, S. B., Ikeuchi, K. (1992, July). Grasp Recognition Using The Contact Web. In IROS (Vol. 92, pp. 194-201).

[16] Light, C. M., Chappell, P. H., Kyberd, P. J., Ellis, B. S. (1999). A critical review of functionality assessment in natural and prosthetic hands. British Journal of Occupational Therapy, 62(1), 7-12.

[17] Light, C. M., Chappell, P. H., Kyberd, P. J. (2002). Establishing a standardized clinical assessment tool of pathologic and prosthetic hand function: normative data, reliability, and validity. Archives of physical medicine and rehabilitation, 83(6), 776-783.

[18] Liu, J., Feng, F., Nakamura, Y. C., Pollard, N. S. (2014, November). A taxonomy of everyday grasps in action. In 2014 IEEE-RAS International Conference on Humanoid Robots (pp. 573-580). IEEE.

[19] Heinemann, F., Puhlmann, S., Eppner, C., Élvarez-Ruiz, J., Maertens, M., Brock, O. (2015, May). A taxonomy of human grasping behavior suitable for transfer to robotic hands. In 2015 IEEE International Conference on Robotics and Automation (ICRA) (pp. 4286-4291). IEEE.

[20] Feix, T., Bullock, I. M., Dollar, A. M. (2014). Analysis of human grasping behavior: Object characteristics and grasp type. IEEE transactions on haptics, 7(3), 311-323.

[21] Feix, T., Bullock, I. M., Dollar, A. M. (2014). Analysis of human grasping behavior: Correlating tasks, objects and grasps. IEEE transactions on haptics, 7(4), 430-441.

[22] Iberall, T. (1987, August). Grasp Planning from Human Prehension. In IJCAI (Vol. 87, No. 1987, pp. 1153-1157).

[23] Xue, Z., Kasper, A., Zoellner, J. M., Dillmann, R. (2009, June). An automatic grasp planning system for service robots. In 2009 International Conference on Advanced Robotics (pp. 1-6). IEEE.

[24] Miller, A. T., Allen, P. K. (2004). Graspit! a versatile simulator for robotic grasping. IEEE Robotics Automation Magazine, 11(4), 110-122.

[25] Bullock, I. M., Dollar, A. M. (2011, June). Classifying human manipulation behavior. In 2011 IEEE International Conference on Rehabilitation Robotics (pp. 1-6). IEEE.

[26] Bullock, I. M., Ma, R. R., Dollar, A. M. (2012). A hand-centric classification of human and robot dexterous manipulation. IEEE transactions on Haptics, 6(2), 129-144.

[27] Cini, F., Ortenzi, V., Corke, P., Controzzi, M. (2019). On the choice of grasp type and location when handing over an object. Science Robotics, 4(27), eaau9757.

[28] Paulius, D., Huang, Y., Meloncon, J., Sun, Y. (2019). Manipulation Motion Taxonomy and Coding for Robots. arXiv preprint arXiv:1910.00532.

[29] Paulius, D., Huang, Y., Milton, R., Buchanan, W. D., Sam, J., Sun, Y. (2016, October). Functional object-oriented network for manipulation learning. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 2655-2662). IEEE.

[30] Szeliski, R. (2010). Computer vision: algorithms and applications. Springer Science Business Media.

[31] Zhang, J., Li, M., Feng, Y., Yang, C. (2020). Robotic grasp detection based on image processing and random forest. Multimedia Tools and Applications, 79(3), 2427-2446.

[32] Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep learning. MIT press.

[33] Lin, M., Chen, Q., Yan, S. (2013). Network in network. arXiv preprint arXiv:1312.4400.

[34] Hu, J., Shen, L., Sun, G. (2018). Squeeze-and-excitation networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7132-7141).

[35] Polydoros, A. S., Nalpantidis, L., Krüger, V. (2015, September). Real-time deep learning of robotic manipulator inverse dynamics. In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 3442-3448). IEEE.

[36] Gao, Y., Hendricks, L. A., Kuchenbecker, K. J., Darrell, T. (2016, May). Deep learning for tactile understanding from visual and haptic data. In 2016 IEEE International Conference on Robotics and Automation (ICRA) (pp. 536-543). IEEE.

[37] Wang, S., Jiang, X., Zhao, J., Wang, X., Zhou, W., Liu, Y. (2019). Efficient Fully Convolution Neural Network for Generating Pixel Wise Robotic Grasps With High Resolution Images. arXiv preprint arXiv:1902.08950.

[38] Zhu, J., Navarro, B., Fraisse, P., Crosnier, A., Cherubini, A. (2018, October). Dual-arm robotic manipulation of flexible cables. In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 479-484). IEEE.

[39] Simoneau, P. (2006). The OSI Model: understanding the seven layers of computer networks. Expert Reference Series of White Papers, Global Knowledge.

APPENDIX A. RAPID code for the left hand.

```
MODULE Module1
    ! Socket variables
    VAR socketdev server_L;
VAR socketdev client_L;
    VAR num length;
    VAR num number_cables_int;
    VAR num number_cables_int_firstimage;
    VAR rawbytes data_mes;
    VAR bool cables;
    VAR num cables_remaining;
     ! Message variables
    VAR string tras;
    VAR string rot;
    VAR string config;
    VAR string exax;
    VAR string tras_pin;
    VAR string rot_pin;
    VAR string config_pin;
    VAR string exax_pin;
    VAR string number_cables;
    VAR string data;
    VAR robtarget pTarget;
    VAR robtarget pinTarget;
    VAR robtarget Approach_pinTarget;
    VAR robtarget Approaching_point;

    ! Para la sincronización de los robots:
    PERS tasks multi_task{2} := [["T_ROB_R"],["T_ROB_L"]];
    VAR syncident sync0;
    VAR syncident sync1;
    VAR syncident sync2;
    VAR syncident sync3;
    VAR syncident sync4;
    VAR syncident sync5;
    VAR syncident sync6;
    VAR syncident sync7;
```

```
CONST robtarget HomePosition_L:=[[525, 250, -200],[0.707106781, 0, 0,
-0.707106781],[-1,2,1,4],[106.804443962,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_50:=[[575,225,100],[0.707106781,0,0.707106781,0],
[1,-2,2,0],[-101.964430857,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_40:=[[450,400,-100],[0,0,0.707106781,0.707106781],
[-1,-1,-1,4],[106.804433997,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_10:=[[500,200,-20],[0,0,1,0],[-1,2,0,4],
[106.804443962,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_70:=[[618,460,-38],[0.707106781,0,0,-0.707106781],
[-1,2,0,6],[106.804434689,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_30:=[[500,200,80],[0,0,1,0],[-1,2,0,4],
[106.804443962,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_80:=[[575,125,50],[0.707106781,0,0.707106781,0],
[-1,2,0,5],[106.804443962,9E+09,9E+09,9E+09,9E+09,9E+09]];

CONST robtarget Approach_Precision_L:=[[475,72.5,247],[0,0.707106781,
0.707106781,0],[-1,2,1,4],[106.804443962,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Step_Precision_L:=[[589.5,72.5,247],[0,0.707106781,
0.707106781,0],[-1,2,1,4],[106.804443962,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Step_Precision_Power_1_L:=[[375,72.5,247],[0,0.707106781,
0.707106781,0],[-1,2,1,4],[106.804443962,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Step_Precision_Power_2_L:=[[475,72.5,247],[0.5,0.5,
0.5,0.5],[-1,2,1,4],[106.804443962,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Step_Power_L:=[[572,72.5,227.5],[0.5,0.5,0.5,0.5],
[-1,2,1,4],[106.804443962,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Approach_Insert_L:=[[574,72.5,247],[0.5,0.5,0.5,0.5],
[-1,2,1,4],[106.804443962,9E+09,9E+09,9E+09,9E+09,9E+09]];

PROC main()
    ReceiveNumber;
    WaitSyncTask sync0, multi_task;
    WaitSyncTask sync1, multi_task;
    number_cables_int_firstimage:= number_cables_int;
    cables_remaining:=number_cables_int_firstimage;
    IF number_cables_int_firstimage = 0 THEN
        EXIT;
    ENDIF
    FOR i FROM 1 TO number_cables_int_firstimage DO
    ReceivePoint;
```

```
    WaitSyncTask sync4, multi_task;
    WaitSyncTask sync5, multi_task;
    cables_remaining:=cables_remaining-1;
    ReceiveNumber;
    WaitSyncTask sync6, multi_task;
    WaitSyncTask sync7, multi_task;
    IF cables_remaining <> number_cables_int THEN
        EXIT;
    ELSEIF cables_remaining = 0 THEN
        EXIT;
    ENDIF
    ENDFOR
ENDPROC
PROC ReceiveNumber()
    SocketCreate server_L;
    SocketBind server_L, "127.0.0.1", 55000;
    SocketListen server_L;

    SocketAccept server_L, client_L \Time:=300;
    SocketReceive client_L, \RawData:=data_mes;
    length := RawBytesLen(data_mes);
    UnpackRawBytes data_mes,1,number_cables,\ASCII:=length;
    cables:= StrToVal(number_cables, number_cables_int);
    SocketSend client_L, \Str:=number_cables;
    SocketClose client_L;
    SocketClose server_L;
ENDPROC
PROC ReceivePoint()

    SocketCreate server_L;
    SocketBind server_L, "127.0.0.1", 55000;
    SocketListen server_L;

    ! For the translation data
    SocketAccept server_L, client_L \Time:=300;
    SocketReceive client_L, \RawData:=data_mes;
    length := RawBytesLen(data_mes);
    UnpackRawBytes data_mes,1,tras,\ASCII:=length;
    SocketSend client_L, \Str:=tras;
```

```
SocketClose client_L;
SocketClose server_L;

! For the rotation data
SocketCreate server_L;
SocketBind server_L, "127.0.0.1", 55000;
SocketListen server_L;
SocketAccept server_L, client_L \Time:=300;
SocketReceive client_L, \RawData:=data_mes;
length := RawBytesLen(data_mes);
UnpackRawBytes data_mes,1,rot,\ASCII:=length;
SocketSend client_L, \Str:=rot;
SocketClose client_L;
SocketClose server_L;

! For the config data
SocketCreate server_L;
SocketBind server_L, "127.0.0.1", 55000;
SocketListen server_L;
SocketAccept server_L, client_L \Time:=300;
SocketReceive client_L, \RawData:=data_mes;
length := RawBytesLen(data_mes);
UnpackRawBytes data_mes,1,config,\ASCII:=length;
SocketSend client_L, \Str:=config;
SocketClose client_L;
SocketClose server_L;

! For the external axis data
SocketCreate server_L;
SocketBind server_L, "127.0.0.1", 55000;
SocketListen server_L;
SocketAccept server_L, client_L \Time:=300;
SocketReceive client_L, \RawData:=data_mes;
length := RawBytesLen(data_mes);
UnpackRawBytes data_mes,1,exax,\ASCII:=length;
SocketSend client_L, \Str:=exax;
SocketClose client_L;
SocketClose server_L;
```

```
    ! For the translation data of the pin target
    SocketCreate server_L;
    SocketBind server_L, "127.0.0.1", 55000;
    SocketListen server_L;
    SocketAccept server_L, client_L \Time:=300;
    SocketReceive client_L, \RawData:=data_mes;
    length := RawBytesLen(data_mes);
    UnpackRawBytes data_mes,1,tras_pin,\ASCII:=length;

    rot_pin := "0.5 0.5 0.5 0.5";
    config_pin := "-1 2 1 4";
    exax_pin := "106.804443962 9E+09 9E+09 9E+09 9E+09 9E+09";
    pinTarget :=StringToTarget(tras_pin,rot_pin,config_pin,exax_pin);
    Approach_pinTarget:= Offs(pinTarget,-100,0,0) ;

    ! When the point is already created, I move to the point
    pTarget:=StringToTarget(tras,rot,config,exax);
    Approaching_point:= Offs(pTarget,0,0,-100) ;
    Grasping_cable;
    SocketSend client_L, \Str:="Information for Left Hand received";
    SocketClose client_L;
    SocketClose server_L;
    WaitSyncTask sync2, multi_task;
    WaitSyncTask sync3, multi_task;
    WaitTime(1);
    Inserting_cable;


ENDPROC
FUNC robtarget StringToTarget(string tras,string rot,string config,
string exax)
    VAR robtarget tmpTarget;
    VAR bool bResult;

    VAR num posX;
    VAR num posY;
    VAR num posZ;
    VAR num posQ1;
    VAR num posQ2;
```

```
VAR num posQ3;
VAR num posQ4;
VAR num poscf1;
VAR num poscf4;
VAR num poscf6;
VAR num poscfx;
VAR num poseaxa;
VAR num poseaxb;
VAR num poseaxc;
VAR num poseaxd;
VAR num poseaxe;
VAR num poseaxf;
! Find split positions in string
posX:=StrFind(tras,1,STR_WHITE);
posY:=StrFind(tras,posX+1,STR_WHITE);
posZ:=StrFind(tras,posY+1,STR_WHITE);
posQ1:=StrFind(rot,1,STR_WHITE);
posQ2:=StrFind(rot,posQ1+1,STR_WHITE);
posQ3:=StrFind(rot,posQ2+1,STR_WHITE);
posQ4:=StrFind(rot,posQ3+1,STR_WHITE);
poscf1:=StrFind(config,1,STR_WHITE);
poscf4:=StrFind(config,poscf1+1,STR_WHITE);
poscf6:=StrFind(config,poscf4+1,STR_WHITE);
poscfx:=StrFind(config,poscf6+1,STR_WHITE);
poseaxa:=StrFind(exax,1,STR_WHITE);
poseaxb:=StrFind(exax,poseaxa+1,STR_WHITE);
poseaxc:=StrFind(exax,poseaxb+1,STR_WHITE);
poseaxd:=StrFind(exax,poseaxc+1,STR_WHITE);
poseaxe:=StrFind(exax,poseaxd+1,STR_WHITE);
poseaxf:=StrFind(exax,poseaxe+1,STR_WHITE);
! read all strings
! pos data
bResult:=StrToVal(StrPart(tras,1,posX-1),tmpTarget.trans.x);
bResult:=StrToVal(StrPart(tras,posX+1,posY-posX-1),tmpTarget.trans.y);
bResult:=StrToVal(StrPart(tras,posY+1,posZ-posY-1),tmpTarget.trans.z);
! orient data
bResult:=StrToVal(StrPart(rot,1,posQ1-1),
tmpTarget.rot.q1);
bResult:=StrToVal(StrPart(rot,posQ1+1,posQ2-posQ1-1),
```

```
    tmpTarget.rot.q2);
    bResult:=StrToVal(StrPart(rot,posQ2+1,posQ3-posQ2-1),
    tmpTarget.rot.q3);
    bResult:=StrToVal(StrPart(rot,posQ3+1,posQ4-posQ3-1),
    tmpTarget.rot.q4);
    ! conf data
    bResult:=StrToVal(StrPart(config,1,poscf1-1),tmpTarget.robconf.cf1);
    bResult:=StrToVal(StrPart(config,poscf1+1,poscf4-poscf1-1),
    tmpTarget.robconf.cf4);
    bResult:=StrToVal(StrPart(config,poscf4+1,poscf6-poscf4-1),
    tmpTarget.robconf.cf6);
    bResult:=StrToVal(StrPart(config,poscf6+1,poscfx-poscf6-1),
    tmpTarget.robconf.cfx);
    ! external axis conf
    bResult:=StrToVal(StrPart(exax,1,poseaxa-1),
    tmpTarget.extax.eax_a);
    bResult:=StrToVal(StrPart(exax,poseaxa+1,poseaxb-poseaxa-1),
    tmpTarget.extax.eax_b);
    bResult:=StrToVal(StrPart(exax,poseaxb+1,poseaxc-poseaxb-1),
    tmpTarget.extax.eax_c);
    bResult:=StrToVal(StrPart(exax,poseaxc+1,poseaxd-poseaxc-1),
    tmpTarget.extax.eax_d);
    bResult:=StrToVal(StrPart(exax,poseaxd+1,poseaxe-poseaxd-1),
    tmpTarget.extax.eax_e);
    bResult:=StrToVal(StrPart(exax,poseaxe+1,poseaxf-poseaxe-1),
    tmpTarget.extax.eax_f);


    RETURN tmpTarget;

ENDFUNC
PROC Gripping()
    g_Init\maxSpd:=10 \holdForce:=10 \Calibrate;
    g_SetForce 10;
    g_SetMaxSpd 10;
    g_GripOut;
    g_GripIn \holdForce:=20;
ENDPROC
PROC Inserting_cable()
```

```
        ConfJ\On;
        MoveL Approaching_point,v1000,fine,Servo\WObj:=Workobject_2;
        MoveJ HomePosition_L,v1000,fine,Servo\WObj:=Workobject_2;
        MoveJ Approach_Precision_L,v1000,fine,Servo\WObj:=wobj0;
        MoveL Step_Precision_L,v1000,fine,Servo\WObj:=wobj0;
        Reset Gripper_L;
        WaitTime 1;
        MoveJ Step_Precision_Power_1_L,v1000,fine,Servo\WObj:=wobj0;
        MoveJ Step_Precision_Power_2_L,v1000,fine,Servo\WObj:=wobj0;
        MoveL Step_Power_L,v1000,fine,Servo\WObj:=wobj0;
        Set Gripper_L;
        WaitTime 1;
        MoveL Step_Precision_Power_2_L,v1000,fine,Servo\WObj:=wobj0;
        MoveJ Approach_pinTarget,v1000,fine,Servo\WObj:=wobj0;
        MoveL pinTarget,v1000,fine,Servo\WObj:=wobj0;
        Reset Gripper_L;
        WaitTime 1;
        MoveL Approach_pinTarget,v1000,fine,Servo\WObj:=wobj0;
        MoveJ HomePosition_L,v1000,fine,Servo\WObj:=Workobject_2;
    ENDPROC
    PROC Grasping_cable()
        ConfJ\On;
        MoveJ Approaching_point,v400,fine,Servo\WObj:=Workobject_2;
        MoveL pTarget,v400,fine,Servo\WObj:=Workobject_2;
        Set Gripper_L;
        WaitTime 1;
    ENDPROC
    PROC Path_20()


    ENDPROC
ENDMODULE
```

APPENDIX B. RAPID code for right hand

```
MODULE Module1
    ! Socket variables
    VAR socketdev server_R;
VAR socketdev client_R;
    VAR num length;
    VAR num number_cables_int_R;
    VAR num number_cables_int_firstimage_R;
    VAR rawbytes data_mes;
    VAR bool cables_R;
    VAR num cables_remaining_R;
    CONST num socket_port_R:= 4500;
     ! Message variables
    VAR string tras;
    VAR string rot;
    VAR string config;
    VAR string exax;
    VAR string tras_pin;
    VAR string rot_pin;
    VAR string config_pin;
    VAR string exax_pin;
    VAR string number_cables;
    VAR string data;
    VAR robtarget pTarget;
    VAR robtarget pinTarget;
    VAR robtarget Approach_pinTarget;
    VAR robtarget Approaching_point;

    ! Para la sincronización de los robots:
    PERS tasks multi_task{2} := [["T_ROB_R"],["T_ROB_L"]];
    VAR syncident sync0;
    VAR syncident sync1;
    VAR syncident sync2;
    VAR syncident sync3;
    VAR syncident sync4;
    VAR syncident sync5;
    VAR syncident sync6;
```

```
VAR syncident sync7;
CONST robtarget HomePosition_R:=[[175, 250, -200],[0.707106781, 0, 0,
-0.707106781],[2,2,-1,4],[-86.964432191,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_70:=[[575,-125,100],[0.707106781,0,0.707106781,0],
[1, 2, 0, 4],[-101.964434909,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Approach_Precision_R:=[[475,-72.5,247],[0,-0.707106781,
0.707106781,0],[2,2,-1,4],[-86.964432191,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Step_Precision_R:=[[589.5,-72.5,247],[0,-0.707106781,
0.707106781,0],[2,2,-1,4],[-86.964432191,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Step_Precision_Power_1_R:=[[375,-72.5,247],[0,-0.707106781,
0.707106781,0],[2,2,-1,4],[-86.964432191,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Step_Precision_Power_2_R:=[[475,-72.5,247],[0.5,-0.5,
0.5,-0.5],[2,2,-1,4],[-86.964432191,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Step_Power_R:=[[572,-72.5,227.5],[0.5,-0.5,0.5,-0.5],
[2,2,-1,4],[-86.964432191,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Approach_Insert_R:=[[574,-72.5,247],[0.5,-0.5,0.5,-0.5],
[2,2,-1,4],[-86.964432191,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_80:=[[100,400,0],[0.707106781, 0, 0, -0.707106781],
[1,2,0,4],[-101.964431052,9E+09,9E+09,9E+09,9E+09,9E+09]];
PROC main()
    WaitSyncTask sync0, multi_task;
    ReceiveNumber;
    WaitSyncTask sync1, multi_task;
    number_cables_int_firstimage_R:= number_cables_int_R;
    cables_remaining_R:=number_cables_int_firstimage_R;
    IF number_cables_int_firstimage_R = 0 THEN
        EXIT;
    ENDIF
    FOR i FROM 1 TO number_cables_int_firstimage_R DO
    WaitSyncTask sync2, multi_task;
    ReceivePoint;
    WaitSyncTask sync5, multi_task;
    WaitSyncTask sync6, multi_task;
    cables_remaining_R:=cables_remaining_R-1;
    WaitSyncTask sync7, multi_task;
    !IF cables_remaining_R <> number_cables_int_R THEN
    !    EXIT;
    !ELSEIF cables_remaining_R = 0 THEN
    !    EXIT;
```

```
    !ENDIF
    ENDFOR
ENDPROC
PROC ReceiveNumber()
    SocketCreate server_R;
    SocketBind server_R, "127.0.0.1", socket_port_R;
    SocketListen server_R;

    SocketAccept server_R, client_R \Time:=300;
    SocketReceive client_R, \RawData:=data_mes;
    length := RawBytesLen(data_mes);
    UnpackRawBytes data_mes,1,number_cables,\ASCII:=length;
    cables_R:= StrToVal(number_cables, number_cables_int_R);
    SocketSend client_R, \Str:=number_cables;
    SocketClose client_R;
    SocketClose server_R;
ENDPROC
PROC ReceivePoint()

    SocketCreate server_R;
    SocketBind server_R, "127.0.0.1", socket_port_R;
    SocketListen server_R;

    ! For the translation data
    SocketAccept server_R, client_R \Time:=300;
    SocketReceive client_R, \RawData:=data_mes;
    length := RawBytesLen(data_mes);
    UnpackRawBytes data_mes,1,tras,\ASCII:=length;
    SocketSend client_R, \Str:=tras;
    SocketClose client_R;
    SocketClose server_R;

    ! For the rotation data
    SocketCreate server_R;
    SocketBind server_R, "127.0.0.1", socket_port_R;
    SocketListen server_R;
    SocketAccept server_R, client_R \Time:=300;
    SocketReceive client_R, \RawData:=data_mes;
    length := RawBytesLen(data_mes);
```

```
UnpackRawBytes data_mes,1,rot,\ASCII:=length;
SocketSend client_R, \Str:=rot;
SocketClose client_R;
SocketClose server_R;

! For the config data
SocketCreate server_R;
SocketBind server_R, "127.0.0.1", socket_port_R;
SocketListen server_R;
SocketAccept server_R, client_R \Time:=300;
SocketReceive client_R, \RawData:=data_mes;
length := RawBytesLen(data_mes);
UnpackRawBytes data_mes,1,config,\ASCII:=length;
SocketSend client_R, \Str:=config;
SocketClose client_R;
SocketClose server_R;

! For the external axis data
SocketCreate server_R;
SocketBind server_R, "127.0.0.1", socket_port_R;
SocketListen server_R;
SocketAccept server_R, client_R \Time:=300;
SocketReceive client_R, \RawData:=data_mes;
length := RawBytesLen(data_mes);
UnpackRawBytes data_mes,1,exax,\ASCII:=length;
! When the point is already created, I move to the point
pTarget:=StringToTarget(tras,rot,config,exax);
Approaching_point:= Offs(pTarget,0,0,-100) ;
Grasping_Cable;
WaitSyncTask sync3, multi_task;
SocketSend client_R, \Str:=exax;
SocketClose client_R;
SocketClose server_R;

! For the translation data of the pin target
SocketCreate server_R;
SocketBind server_R, "127.0.0.1", socket_port_R;
SocketListen server_R;
SocketAccept server_R, client_R \Time:=300;
```

```
    SocketReceive client_R, \RawData:=data_mes;
    length := RawBytesLen(data_mes);
    UnpackRawBytes data_mes,1,tras_pin,\ASCII:=length;
    rot_pin := "0.5 -0.5 0.5 -0.5";
    config_pin := "2 2 -1 4";
    exax_pin := "-86.964432191 9E09 9E09 9E09 9E09 9E09";
    pinTarget :=StringToTarget(tras_pin,rot_pin,config_pin,exax_pin);
    Approach_pinTarget:= Offs(pinTarget,-100,0,0);
    Inserting_Cable;
    SocketSend client_R, \Str:="Information for Right Hand received";
    SocketClose client_R;
    SocketClose server_R;
ENDPROC
FUNC robtarget StringToTarget(string tras,string rot,string config,
string exax)
    VAR robtarget tmpTarget;
    VAR bool bResult;

    VAR num posX;
    VAR num posY;
    VAR num posZ;
    VAR num posQ1;
    VAR num posQ2;
    VAR num posQ3;
    VAR num posQ4;
    VAR num poscf1;
    VAR num poscf4;
    VAR num poscf6;
    VAR num poscfx;
    VAR num poseaxa;
    VAR num poseaxb;
    VAR num poseaxc;
    VAR num poseaxd;
    VAR num poseaxe;
    VAR num poseaxf;
    ! Find split positions in string
    posX:=StrFind(tras,1,STR_WHITE);
    posY:=StrFind(tras,posX+1,STR_WHITE);
    posZ:=StrFind(tras,posY+1,STR_WHITE);
```

```
posQ1:=StrFind(rot,1,STR_WHITE);
posQ2:=StrFind(rot,posQ1+1,STR_WHITE);
posQ3:=StrFind(rot,posQ2+1,STR_WHITE);
posQ4:=StrFind(rot,posQ3+1,STR_WHITE);
poscf1:=StrFind(config,1,STR_WHITE);
poscf4:=StrFind(config,poscf1+1,STR_WHITE);
poscf6:=StrFind(config,poscf4+1,STR_WHITE);
poscfx:=StrFind(config,poscf6+1,STR_WHITE);
poseaxa:=StrFind(exax,1,STR_WHITE);
poseaxb:=StrFind(exax,poseaxa+1,STR_WHITE);
poseaxc:=StrFind(exax,poseaxb+1,STR_WHITE);
poseaxd:=StrFind(exax,poseaxc+1,STR_WHITE);
poseaxe:=StrFind(exax,poseaxd+1,STR_WHITE);
poseaxf:=StrFind(exax,poseaxe+1,STR_WHITE);
! read all strings
! pos data
bResult:=StrToVal(StrPart(tras,1,posX-1),tmpTarget.trans.x);
bResult:=StrToVal(StrPart(tras,posX+1,posY-posX-1),tmpTarget.trans.y);
bResult:=StrToVal(StrPart(tras,posY+1,posZ-posY-1),tmpTarget.trans.z);
! orient data
bResult:=StrToVal(StrPart(rot,1,posQ1-1),
tmpTarget.rot.q1);
bResult:=StrToVal(StrPart(rot,posQ1+1,posQ2-posQ1-1),
tmpTarget.rot.q2);
bResult:=StrToVal(StrPart(rot,posQ2+1,posQ3-posQ2-1),
tmpTarget.rot.q3);
bResult:=StrToVal(StrPart(rot,posQ3+1,posQ4-posQ3-1),
tmpTarget.rot.q4);
! conf data
bResult:=StrToVal(StrPart(config,1,poscf1-1),tmpTarget.robconf.cf1);
bResult:=StrToVal(StrPart(config,poscf1+1,poscf4-poscf1-1),
tmpTarget.robconf.cf4);
bResult:=StrToVal(StrPart(config,poscf4+1,poscf6-poscf4-1),
tmpTarget.robconf.cf6);
bResult:=StrToVal(StrPart(config,poscf6+1,poscfx-poscf6-1),
tmpTarget.robconf.cfx);
! external axis conf
bResult:=StrToVal(StrPart(exax,1,poseaxa-1),tmpTarget.extax.eax_a);
bResult:=StrToVal(StrPart(exax,poseaxa+1,poseaxb-poseaxa-1),
```

```
    tmpTarget.extax.eax_b);
    bResult:=StrToVal(StrPart(exax,poseaxb+1,poseaxc-poseaxb-1),
    tmpTarget.extax.eax_c);
    bResult:=StrToVal(StrPart(exax,poseaxc+1,poseaxd-poseaxc-1),
    tmpTarget.extax.eax_d);
    bResult:=StrToVal(StrPart(exax,poseaxd+1,poseaxe-poseaxd-1),
    tmpTarget.extax.eax_e);
    bResult:=StrToVal(StrPart(exax,poseaxe+1,poseaxf-poseaxe-1),
    tmpTarget.extax.eax_f);


    RETURN tmpTarget;

ENDFUNC
PROC Gripping()
    Reset Gripper;
    Set Gripper;
ENDPROC
PROC Inserting_Cable()
    ConfJ\On;
    MoveL Approaching_point,v1000,fine,Servo\WObj:=Workobject_3;
    MoveJ HomePosition_R,v1000,fine,Servo\WObj:=Workobject_3;
    WaitSyncTask sync4, multi_task;
    MoveJ Approach_Precision_R,v1000,fine,Servo\WObj:=wobj0;
    MoveL Step_Precision_R,v1000,fine,Servo\WObj:=wobj0;
    Reset Gripper;
    WaitTime 1;
    MoveJ Step_Precision_Power_1_R,v1000,fine,Servo\WObj:=wobj0;
    MoveJ Step_Precision_Power_2_R,v1000,fine,Servo\WObj:=wobj0;
    MoveL Step_Power_R,v1000,fine,Servo\WObj:=wobj0;
    Set Gripper;
    WaitTime 1;
    MoveL Step_Precision_Power_2_R,v1000,fine,Servo\WObj:=wobj0;
    MoveJ Approach_pinTarget,v1000,fine,Servo\WObj:=wobj0;
    MoveL pinTarget,v1000,fine,Servo\WObj:=wobj0;
    Reset Gripper;
    WaitTime 1;
    MoveL Approach_pinTarget,v1000,fine,Servo\WObj:=wobj0;
    MoveJ HomePosition_R,v1000,fine,Servo\WObj:=Workobject_3;
```

```
    ENDPROC
    PROC Grasping_Cable()
        ConfJ\On;
        MoveJ Approaching_point,v400,fine,Servo\WObj:=Workobject_3;
        MoveL pTarget,v400,fine,Servo\WObj:=Workobject_3;
        Set Gripper;
        WaitTime 1;
    ENDPROC
ENDMODULE
```

APPENDIX C. Python script to test the images

```
import numpy as np
import cv2
import math
import glob
import time
start = time.process_time()

image_directory = 'images/images_test'


def read_img(img_list, img):
    n = cv2.resize(cv2.imread(img), (2661,1637))
    img_list.append(n)
    return img_list


path = glob.glob("images/images_test/horizontal_4.jpg") #or jpg
list_ = []
cv_image = [read_img(list_, img) for img in path]
cv_image_black_conv = []
cv_image_red_conv = []
cv_image_black_open = []
cv_image_red_open = []
w_resize, h_resize = 250, 161
resize_shape = (w_resize, h_resize)
viewing_shape = (2*w_resize, 2*h_resize)

for img in list_:
    h_init, w_init, c = np.shape(img)
    img_res = cv2.resize(img, resize_shape)
    img_fake = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img_fake[:, :] = 0
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    lower_red = np.array([170, 160, 70])
    upper_red = np.array([180, 255, 155])
    red_mask = cv2.inRange(hsv, lower_red, upper_red)
```

```
    red = cv2.bitwise_and(img, img, mask=red_mask)
    kernel = np.ones((10, 10), np.uint8)
    red_conv = cv2.cvtColor(cv2.filter2D(red, -1, kernel), cv2.COLOR_BGR2GRAY)
    lower_black = np.array([0, 0, 0])
    upper_black = np.array([255, 255, 30])
    black_mask = cv2.inRange(hsv, lower_black, upper_black)
    black = cv2.bitwise_and(img, img, mask=black_mask)
    black_bw = cv2.cvtColor(black, cv2.COLOR_BGR2GRAY)
    difference_black = cv2.subtract(black_bw,img_fake)
    black_bw[difference_black == 0] = 0
    black_bw[difference_black != 0] = 255
    black_conv = cv2.filter2D(black_bw, -1, kernel)
    kernel = np.ones((20, 20), np.uint8)
    opening_black = cv2.morphologyEx(black_conv, cv2.MORPH_OPEN, kernel)
    opening_red = cv2.morphologyEx(red_conv, cv2.MORPH_OPEN, kernel)
    difference_black = cv2.subtract(opening_black, img_fake)
    opening_black[difference_black == 0] = 255
    opening_black[difference_black != 0] = 0
    difference_red = cv2.subtract(opening_red, img_fake)
    opening_red[difference_red == 0] = 255
    opening_red[difference_red != 0] = 0
    kernel_grad = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (10, 10))
    gradient_red = cv2.morphologyEx(opening_red, cv2.MORPH_GRADIENT,
    kernel_grad)
    gradient_black = cv2.morphologyEx(opening_black, cv2.MORPH_GRADIENT,
    kernel_grad)
    difference_red_orientation = cv2.add(gradient_red, opening_red)
    ret_red, difference_red_orientation_bw = cv2.threshold
    (difference_red_orientation, 127, 255, cv2.THRESH_BINARY)
    difference_black_orientation = cv2.add(gradient_black, opening_black)
    ret_black, difference_black_orientation_bw = cv2.threshold
    (difference_black_ orientation, 127, 255, cv2.THRESH_BINARY)
    cv_image_black_conv.append(black_conv)
    cv_image_red_conv.append(red_conv)
    cv_image_black_open.append(difference_black_orientation_bw)
    cv_image_red_open.append(difference_red_orientation_bw)

def obtaining_grasps(difference_orientation_bw, image_conv):
    zero_coord = {1: [0, 0]}
```

```
zero_coord_conv = {1: [0, 0]}
x = 0
h_or, w_or = np.shape(difference_orientation_bw)
h_conv, w_conv = np.shape(image_conv)

for max in range((h_or + w_or - 1)):
    if max <= w_or - 1:
        i = 0
        j = max
    else:
        i = max - w_or + 1
        j = w_or - 1
    flag = True
    while flag:
        if j == -1 or i == h_or:
            flag = False
        else:
            if difference_orientation_bw[i, j] == 0:
                x = x + 1
                zero_coord.update({x: [j, i]})
        i = i + 1
        j = j - 1
x = 0
for max in range((h_conv + w_conv - 1)):
    if max <= w_conv-1:
        i = 0
        j = max
    else:
        i = max - w_conv + 1
        j = w_conv - 1
    flag = True
    while flag:
        if j == -1 or i == h_conv:
            flag = False
        else:
            if image_conv[i, j] > 170:
                x = x + 1
                zero_coord_conv.update({x: [j, i]})
        i = i + 1
```

```
        j = j - 1
num_grasp = 0
num_cables = 0
previous_points = {1: [0, 0]}
previous_points_cables = {1:[0, 0]}
zero_clustered = {1: {1: []}}
zero_clustered_conv = {1: {1: []}}
i = 0
same_holder = False
for point in zero_coord:
    for prev_point in previous_points:
        if np.sqrt((int(zero_coord[point][1])-int(previous_points
        [prev_point][1]))**2+(int(zero_coord[point][0])-int
        (previous_points[prev_point][0]))**2)< 5*w_resize/250:
            if num_grasp != 0:
                for j in range(num_grasp):
                    if previous_points[prev_point] in
                    zero_clustered[j+1].values():
                        k = len(zero_clustered[j+1])
                        zero_clustered[j+1][k+1] = zero_coord[point]
            same_holder = True
            break
    if not same_holder:
        num_grasp = num_grasp + 1
        if num_grasp == 1:
            zero_clustered[num_grasp].update({1: zero_coord[point]})
        else:
            zero_clustered[num_grasp] = {1: zero_coord[point]}
    previous_points[point] = zero_coord[point]
    same_holder = False
i = 0
same_cable = False
image_res = cv2.cvtColor(image_conv, cv2.COLOR_GRAY2BGR)
for point in zero_coord_conv:
    for prev_point in previous_points_cables:
        if zero_coord_conv[point] != zero_coord_conv[prev_point]:
            if np.sqrt((int(zero_coord_conv[point][1])-int
            (previous_points_cables[prev_point][1]))**2+
            (int(zero_coord_conv[point][0]) -int
```

```
                (previous_points_cables[prev_point][0]))**2)< 5*w_resize/250:
                    if num_cables != 0:
                        for j in range(num_cables):
                            if previous_points_cables[prev_point] in
                            zero_clustered_conv[j+1].values():
                                k = len(zero_clustered_conv[j+1])
                                zero_clustered_conv[j+1][k+1] =
                                zero_coord_conv[point]
                    same_cable = True
                    break
        if not same_cable:
            cv2.circle(image_res, (zero_coord_conv[point][0],
            zero_coord_conv[point][1]), 5, (0, 0, 255), 1)
            num_cables = num_cables + 1
            if num_cables == 1:
                zero_clustered_conv[num_cables].update({1:
                zero_coord_conv[point]})
            else:
                zero_clustered_conv[num_cables] = {1:
                zero_coord_conv[point]}
        previous_points_cables[point] = zero_coord_conv[point]
        same_cable = False
    i = len(zero_clustered_conv)
    zero_clustered_conv_real = zero_clustered_conv.copy()
    while i > 1:
        flag = False
        for elements in zero_clustered_conv:
            if elements >= i:
                pass
            else:
                for element_prev in zero_clustered_conv[elements]:
                    for element in zero_clustered_conv[i]:
                        if np.sqrt((int(zero_clustered_conv[i][element]
                        [1])-int(zero_clustered_conv[elements]
                        [element_prev][1]))**2+
                        (int(zero_clustered_conv[i][element][0])-int
                        (zero_clustered_conv[elements][element_prev][0]))**2)
                        < 5*w_resize/250:
                            cv2.circle(image_res, (zero_clustered_conv[i]
```

```
                        [element][0], zero_clustered_conv[i]
                        [element][1]), 3, (0, 255, 0), 3)
                        for j in zero_clustered_conv[i]:
                            zero_clustered_conv_real[elements].update
                            ({(len(zero_clustered_conv[elements])+ j):
                            zero_clustered_conv[i][j]})
                        del zero_clustered_conv_real[i]

                        flag = True
                        break
                if flag:
                    break
            if flag:
                break
    i = i - 1
i = len(zero_clustered)
zero_clustered_real = zero_clustered.copy()
while i > 1:
    flag = False
    for elements in zero_clustered:
        if elements >= i:
            pass
        else:
            for element_prev in zero_clustered[elements]:
                for element in zero_clustered[i]:
                    if np.sqrt((int(zero_clustered[i][element][1]) - int
                    (zero_clustered[elements][element_prev][1])) ** 2 +
                    (int(zero_clustered[i][element][0]) -
                    int(zero_clustered[elements][element_prev][0])) ** 2)
                    < 5*w_resize/250:
                        for j in zero_clustered[i]:
                            zero_clustered_real[elements].update(
                                {(len(zero_clustered[elements]) + j):
                                zero_clustered[i][j]})
                        del zero_clustered_real[i]

                        flag = True
                        break
                if flag:
```

```
                    break
            if flag:
                break
    i = i - 1
key = zero_clustered_conv_real.copy().keys()
long = 0
for i in key:
    if len(zero_clustered_conv_real[i]) < 10*w_resize/250:
        del zero_clustered_conv_real[i]
key = zero_clustered_conv_real.copy().keys()
for i in key:
    if i > long:
        long = i
k = 0
for i in key:
    k = k + 1
    if i == k:
        pass
    else:
        zero_clustered_conv_real.update({k: zero_clustered_conv_real[i]})
        del zero_clustered_conv_real[i]

key = zero_clustered_real.copy().keys()
long = 0
for i in key:
    if len(zero_clustered_real[i]) < 10*w_resize/250:
        del zero_clustered_real[i]
key = zero_clustered_real.copy().keys()
for i in key:
    if i > long:
        long = i
k = 0
for i in key:
    k = k + 1
    if i == k:
        pass
    else:
        zero_clustered_real.update({k: zero_clustered_real[i]})
        del zero_clustered_real[i]
```

```
sum_x = 0
sum_y = 0
center_x = list()
center_y = list()
large_point_x = list()
large_point_y = list()
num_grasp = len(zero_clustered_real)
if num_grasp > 0:
    for i in range(num_grasp):
        center_x.insert(i, 0)
        center_y.insert(i, 0)
        large_point_x.insert(i, 0)
        large_point_y.insert(i, 0)
    for i in zero_clustered_real:
        total = 0
        for j in zero_clustered_real[i]:
            total = total + 1
            sum_x = sum_x + int(zero_clustered_real[i][j][0])
            sum_y = sum_y + int(zero_clustered_real[i][j][1])
        center_x[i-1] = sum_x/total
        center_y[i-1] = sum_y/total
        sum_x = 0
        sum_y = 0
    max_distance = 0
    for i in zero_clustered_real:
        for j in zero_clustered_real[i]:
            distance = np.sqrt((center_x[i-1]-zero_clustered_real[i][j]
            [0])**2+(center_y[i-1]-zero_clustered_real[i][j][1])**2)
            if distance >= max_distance:
                large_point_x[i-1] = zero_clustered_real[i][j][0]
                large_point_y[i-1] = zero_clustered_real[i][j][1]
                max_distance = distance
        max_distance = 0
    angle = np.zeros(num_grasp)
    for i in range(num_grasp):
        if center_x[i] > large_point_x[i] and center_y[i] >
        large_point_y[i]:
            angle[i] = - (math.pi/2 - math.atan((center_y[i]-
```

```
            large_point_y[i])/(center_x[i]-large_point_x[i])))
        elif center_x[i] < large_point_x[i] and center_y[i] >
        large_point_y[i]:
            angle[i] = - (3*math.pi/2 + math.atan((center_y[i] -
            large_point_y[i]) / (large_point_x[i] - center_x[i])))
        elif center_x[i] < large_point_x[i] and center_y[i] <
        large_point_y[i]:
            angle[i] = - (3*math.pi/2 - math.atan((large_point_y[i] -
            center_y[i]) / (large_point_x[i] - center_x[i])))
        elif center_x[i] > large_point_x[i] and center_y[i] <
        large_point_y[i]:
            angle[i] = - (math.pi/2 + math.atan((large_point_y[i] -
            center_y[i]) / (center_x[i] - large_point_x[i])))
elif num_grasp == 0:
    angle = [0]
    center_x = [0]
    center_y = [0]
cables = dict()
cables_left = dict()
cables_right = dict()
num_cables = len(zero_clustered_conv_real)
for i in range(num_grasp):
    for j in range(num_cables):
        if [int(center_x[i]), int(center_y[i])] in
        zero_clustered_conv_real[j+1].values():
            if (j+1) in cables.keys():
                k = len(cables[j + 1])
                cables[j + 1].update({k + 1: [int(center_x[i]*2),
                int(center_y[i]*2), angle[i]]})
            else:
                cables.update({j+1: {1: [int(center_x[i]*2),
                int(center_y[i]*2), angle[i]]}})
for elements in cables:
    if len(cables[elements]) == 2:
        if cables[elements][1][0] < cables[elements][2][0]:
            cables_left.update({elements: cables[elements][2]})
            cables_right.update({elements: cables[elements][1]})
        elif cables[elements][1][0] > cables[elements][2][0]:
            cables_left.update({elements: cables[elements][1]})
```

```
                cables_right.update({elements: cables[elements][2]})
    return cables, cables_left, cables_right

i = 0

for img in list_:
    cv2.imwrite('image_test_' + str(i+1) + '.jpg', img)
    h_init, w_init, c = np.shape(img)
    cables_red, cables_red_left, cables_red_right = obtaining_grasps
    (cv2.resize(cv_image_red_open[i], resize_shape),
    cv2.resize(cv_image_red_conv[i], resize_shape))
    cables_black, cables_black_left, cables_black_right = obtaining_grasps
    (cv2.resize(cv_image_black_open[i], resize_shape),
    cv2.resize(cv_image_black_conv[i], resize_shape))
    for elements in cables_red:
        if len(cables_red[elements]) == 2:
            cv2.circle(img, (int(cables_red[elements][1][0] / 2 * w_init /
            resize_shape[0]),int(cables_red[elements][1][1] / 2 * h_init /
            resize_shape[1])), 12,(0, 255, 0), 5)
            cv2.circle(img, (int(cables_red[elements][2][0] / 2 * w_init /
            resize_shape[0]),       int(cables_red[elements][2][1] / 2 *
            h_init / resize_shape[1])), 12, (0, 255, 0), 5)
            cv2.line(img, (int(cables_red[elements][1][0] / 2 * w_init /
            resize_shape[0]), int(cables_red[elements][1][1] / 2 * h_init /
            resize_shape[1])),((int(cables_red[elements][1][0] / 2 * w_init /
            resize_shape[0])) + int(100*math.sin(cables_red[elements][1][2]))
            ,(int(cables_red[elements][1][1] / 2 * h_init /resize_shape[1]))
            -int(100*math.cos(cables_red[elements][1][2]))),(0, 255, 255), 5)
            cv2.line(img, (int(cables_red[elements][2][0] / 2 * w_init /
            resize_shape[0]),int(cables_red[elements][2][1] / 2 * h_init /
            resize_shape[1])),((int(cables_red[elements][2][0] / 2 * w_init /
            resize_shape[0])) + int(100 * math.sin(cables_red[elements][2][2]
            )),(int(cables_red[elements][2][1] / 2 * h_init /
            resize_shape[1])) - int(100 * math.cos(cables_red[elements]
            [2][2]))), (0, 255, 255), 5)
            cv2.line(img, (int(cables_red[elements][1][0] / 2 * w_init /
            resize_shape[0]),int(cables_red[elements][1][1] / 2 * h_init /
            resize_shape[1])),(int(cables_red[elements][2][0] / 2 * w_init /
            resize_shape[0]),int(cables_red[elements][2][1] / 2 * h_init /
```

```
            resize_shape[1])), (255, 0, 0), 10)
    for elements in cables_black:
        if len(cables_black[elements]) == 2:
            cv2.circle(img, (int(cables_black[elements][1][0] / 2 * w_init /
            resize_shape[0]),int(cables_black[elements][1][1] / 2 * h_init /
            resize_shape[1])), 12,(0, 255, 0), 5)
            cv2.circle(img, (int(cables_black[elements][2][0] / 2 * w_init /
            resize_shape[0]),int(cables_black[elements][2][1] / 2 * h_init /
            resize_shape[1])), 12,(0, 255, 0), 5)
            cv2.line(img, (int(cables_black[elements][1][0] / 2 * w_init /
            resize_shape[0]),int(cables_black[elements][1][1] / 2 * h_init /
            resize_shape[1])),((int(cables_black[elements][1][0] / 2 * w_init /
            resize_shape[0])) + int(100 * math.sin(cables_black[elements]
            [1][2])),(int(cables_black[elements][1][1] / 2 * h_init /
            resize_shape[1])) - int(100 * math.cos(cables_black[elements]
            [1][2]))), (0, 255, 255), 5)
            cv2.line(img, (int(cables_black[elements][2][0] / 2 * w_init /
            resize_shape[0]),int(cables_black[elements][2][1] / 2 * h_init /
            resize_shape[1])),((int(cables_black[elements][2][0] / 2 * w_init /
            resize_shape[0])) + int(100 * math.sin(cables_black[elements]
            [2][2])),(int(cables_black[elements][2][1] / 2 * h_init /
            resize_shape[1])) - int(100 * math.cos(cables_black[elements]
            [2][2]))), (0, 255, 255), 5)
            cv2.line(img, (int(cables_black[elements][1][0] / 2 * w_init /
            resize_shape[0]),int(cables_black[elements][1][1] / 2 * h_init /
            resize_shape[1])),(int(cables_black[elements][2][0] / 2 * w_init /
            resize_shape[0]),int(cables_black[elements][2][1] / 2 * h_init /
            resize_shape[1])), (255, 0, 0), 10)
    i = i + 1
    cv2.imwrite('image_tested_' + str(i) +'.jpg', img)

print(time)
print(time.process_time()-start)
```