Project Report

Eden Robotics

Members:
Antoine Alessandrini, Maxime Bacquet, Anna Berger, Héloïse Boyer-Vidal, Simon Dahy, Anna Ducros, Victor Guinebertière, Thomas Jaouen, David Kirov, Simon Kurney, Noé Luxembourger, Aya Skhoun

Client:
Mr. Patrick Kedziora

Coaches:
Mr. Denis LePicart, Mr. Roland Marcoin

Consultants:
Mr. Yannick Dusch, Mr. Alexandre Kruszewski, Mr. Denis LePicart

Date:
September 2020 - February 2022

Written by:
Antoine Alessandrini

Lille

Contents

List of Figures

List of Tables

Introduction

Project

This project was realized as part of our studies in connection with the company "ALL one Robotics". The initial objective was to build a robot to pick apples. Our client had noticed a great shortage of manpower in French and American farms. In France alone, 1 million seasonal workers are needed for the harvest. This labor force remains difficult to find and the crisis of the cider has amplified this phenomenon. Thus, in the United States, only 80%[Fra] of the apples would be picked, resulting in significant economic and food losses. This problem has given the idea to our customer to remotely operate the harvest, giving birth to the project "All One Robotics".

It is a year-and-a-half project of 12 students of the Ecole Centrale de Lille whose objective is to build a robot capable of picking apples controlled by a user remotely. With the means available and the difficulties encountered, the project turned to the harvest of tomatoes. We had to build a prototype that will facilitate and accelerate the work.

Our Client, Patrick Kedziora is a French-American entrepreneur and founder of the start-up "All One Robotics". We could also count on coaches from Centrale Lille who helped us throughout the project, especially for the management of such a project: Mr. Denis le Picart and Mr. Roland Marcoin. Finally, we were able to count on the investment of Mr. Kruszewski, a professor at the Ecole Centrale, and his numerous advice throughout the project.

Team members

In this project we were 12 students in the first year of the Ecole Centrale de Lille: Anna Berger, Anna Ducros, Antoine Alessandrini, Aya Skhoun, David Kirov, Héloïse Boyer-Vidal, Maxime Baquet, Noé Luxembourger, Simon Dahy, Simon Kurney, Thomas Jaouën et Victor Guinebertière. 11 of us came from preparatory classes (from all section) and Simon K. was in double degree with his university in Germany.

Figure 0.1: The team with our client

1 Project Definition

1.1 Market study

At the request of Mr. Kedziora, we began by conducting a market study[1]. Even if he had already done it, it allowed us to better understand the expectations of our customers, and the possible difficulties. It was also a way to have the point of future users of our product.

Mr. Kedziora wanted to sell the robot in the United States. We focused on this region even if we also studied the situation in France. For that we contacted farmers by asking them questions about :

- The size of their farms
- The method of collection and the duration
- The points of attention to picking apples
- The type of soil
- The difficulties encountered in finding personnel
- The interest for the use of robots: motivation and fear.
- The possible price.

It turned out that only the big farms ($>$ 50 acres) were interested. Indeed, out of the twenty or so farms that we contacted, more than 80% were having difficulty recruiting, this phenomenon increased with the Covid crisis. This is not the case for small farms because they work mainly with local people and need fewer people. Thus, all the large farms were in favor of using robots. However, they warned us about the current existence of autonomous robots and the need to distinguish themselves. According to them, the big disadvantages of robotization today are the price of access and maintenance. A remote-controlled robot can answer their request by reducing the cost. However, our robot must keep the advantages of autonomous robots: the possibility to be used 24h and reducing the number of employees needed. The continuous work is a primordial point because the robots are generally slower than the humans but can have a better output on a complete day. Finally, technical characteristics were also put forward: Not too much pressure on the fruit, the ability to drive on potentially muddy dirt roads, and dimensions to respect.

We also went directly on the spot to visit orchards in order to have a better vision of all these constraints and to be able to better discuss with the farmers. It was thus of great help to write the specifications.

1.2 State of the art

At the same time as we were doing the market research, we were also interested in what was currently being done. There are several robot solutions for picking up fruit. All of them, however, use artificial intelligence and are autonomous. Partly because of this, their price is generally very high. Many of these are also relatively young and have not yet gained the full confidence of users. A good example is the *FFRobotics* robot[FFR] that was in late development in February 2019 but the cost of a machine is between $300,000 and $350,000. They finally launched their robot in July 2022, 6 months after our project. Similarly, the company *Abundant Robotics*[Roba] also created a robot capable of picking up apples by sucking them. However, this one had to close in July 2021 because of a lack of customers.

As the farmers we spoke with told us, the price is the main reason why they do not invest in these solutions. Our objective is to propose a solution using relatively easy and inexpensive manufacturing techniques. The desire of our customer to use a remote operator is also a way to reduce the design time and therefore the cost of the project. These robots are also usually very bulky and we want to try to reduce their size to move them more easily.

[1] Analyze, understand and measure the real functioning of the forces at work in a market.

Figure 1.1: FFRobotics robot

Finally, there are also studies that we could rely on. For example, the article by Joseph Davidson [Dav+16] gives the ways to pick apples with a robot and indicates the maximum pressure that can be exerted.

1.3 Definition

After conducting a market study and a state of the art, we were able to redefine the project. Initially, our client wanted us to design and manufacture a complete robot: a base and an arm. His main constraint was the flexibility of the robot to be able to adapt to other tasks and thus be used throughout the year. As we have seen when we did the state-of-the-art, many bases already exist and are relatively cheap. Moreover, Mr. Kedziora also wanted us to start from a blank page in order not to be influenced by what already exist. It would have taken too much time to focus on everything. With his agreement, we decided to focus on the arm, the hand, and the control system.

Indeed, the main point of this project was to recover the fruits without damaging them. The hand was therefore the central element to distinguish us. We also kept the creation of the arm because the robot had to have a low cost and we wanted to imagine the least expensive design. For the same reason, we also kept the implementation of the control system.

Finally, during the project and facing the difficulties we encountered, we also evolved the project during the year. The apple harvest became the cherry tomato harvest. This allowed us to reduce the dimensions of our robot while keeping almost all the constraints we had to respect. We were therefore able to make all the prototypes with the "traditional" tools of the school. It also showed our capacity to adapt.

1.4 Scope statement

Following this, we made a scope statement[2]. These studies as well as the evolution of the project allowed us to estimate the expectations and constraints.

[2]Defines the purpose of the project, the stages of its realization, and the elements necessary to carry it out

The general cases of use that we could list are the following :

- Picking up
- Storage of fruits
- Storage of robots
- Cleaning of the robot
- Recycling (not treated in the following)

We then made an "octopus diagram[3]" which illustrates the links between the different functions:

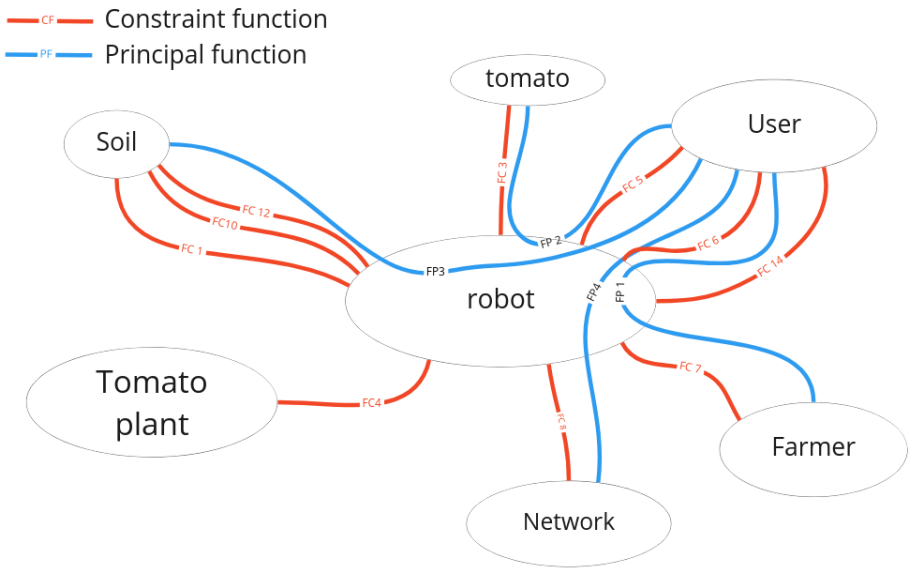


Figure 1.2: Octopus diagram

Only the main functions are presented here, the entire specification can be retained in appendix A. The main function is to be as efficient as a human over a day. Thus the arm will have to retrieve a tomato in less than 10s.

Principal Function	**Appreciation criteria**	**Level**	**flexibility**
PF 1: Be at least as efficient as a human	harvest duration	less than 10s	F2
PF 2: Picking tomatoes	The tomatoes are detached	80% of standard orchards are collect	F1
PF 3: Controlling the robot	Pick tomatoes	get 80% of tomatoes	F2
PF 4 Control the robot remotely	User can be in other place	500m distance	F2

Table 1: Principal functions extract

[3]Represents the relationship between a product and its environment

The main constraint function is to reach all the tomatoes. To measure this, we want our robot to be able to reach 80% of the tomatoes in a standard orchard.

Constraint Function	**Appreciation criteria**	**Level**	**flexibility**
CF 1: Keeping tomatoes intact	Forces exerted	less than 5N	F1
CF 2: Avoid branches and tomatoes	Precision	relative gap $<$ 1cm	F1
CF 3: Be easy to pilot	User feeling	Learn time $<$ 2h	F2

Table 2: Constraint functions extract

2 Project Management

2.1 Organization

2.1.1 Non technical aspect

To manage the project in the best possible way, we defined from the beginning the non-technical roles to be taken charge of. The objective was to avoid the "tunnel effect[4]" on these tasks which can be considered secondary. They are however essential for the success of such a project.

We appointed a project manager in charge of transmitting information to all members and especially to the different technical poles. His mission was to ensure a follow-up of the progress as well as the update of the communication tools.

In order to help him, we also put in place a person in charge of external communication. He communicated with the coaches as well as with the external consultants. Like the project manager, he was also in permanent communication with the client. In order to manage the budget as well as possible, finance provided for the client, but also the possible training, budget, and training managers were appointed.

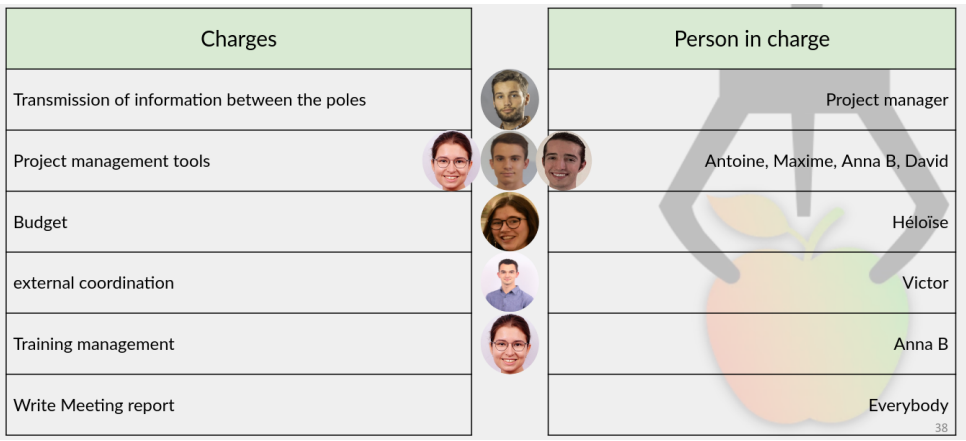


Figure 2.1: Non Technical organization

2.1.2 Technical aspect

After having defined the project we were also able to define a technical organization. Three poles were created.

First, a mechanical pole whose role was to design the different versions of the robot. Then to build it. An automatic pole was in charge of the simulation of the robot and the control system. Finally, an IT pole worked on the user interface, the video acquisition, and the remote connection.

Each pole had a person in charge to facilitate the follow-up. They regularly discussed with the project leader.

[4]case where the sponsor receives little or no information during implementation

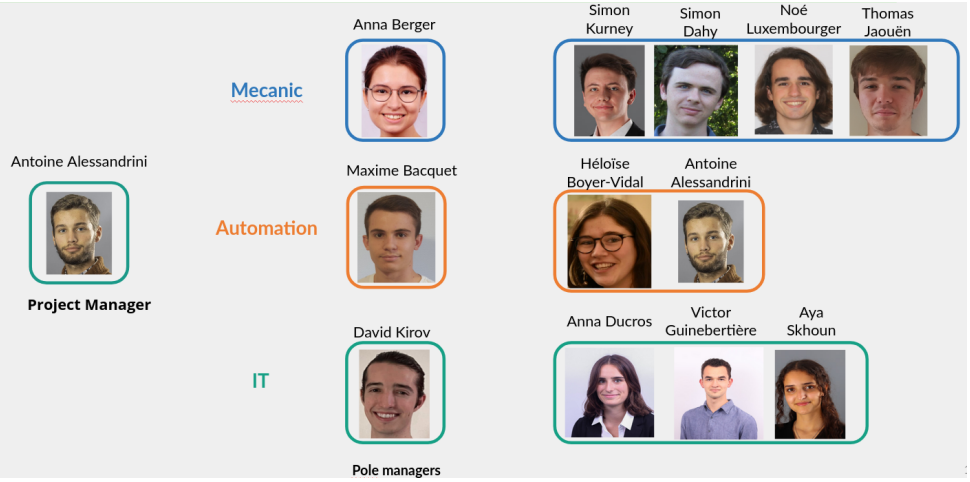


Figure 2.2: Technical organization

2.2 Planification

In order to manage the progress of the project, project management tools were put in place. They were updated at least weekly.

First of all, a WBS[5](appendix B) (Work Breakdown Structure) was created. Its purpose is to break down the project into smaller pieces. It allows us to identify more precisely all the work to be done and the distribution between the different poles.

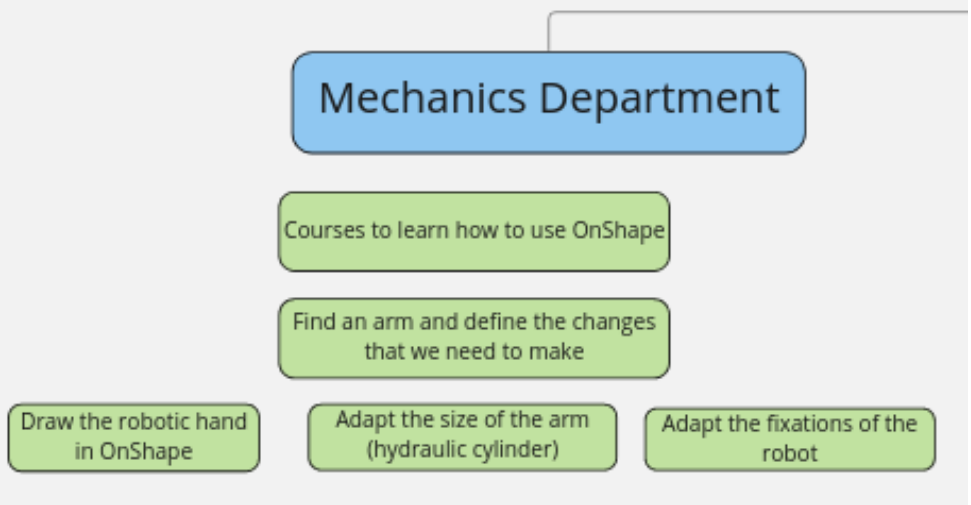


Figure 2.3: WBS example

Thanks to this document, we set up a Gantt diagram[6] (appendix C). It illustrates the project schedule by indicating the tasks to be accomplished and the time. Each pole had its Gantt chart, which was linked to a global chart. This allowed us to anticipate certain delays and adapt. Milestones[7] were also defined in order to have objectives all along the project. Their dates were set on the audits during the project.

[5]Hierarchical breakdown of the necessary work
[6]Graphical representation of a schedule
[7]Siginificant stage or event in the development of something

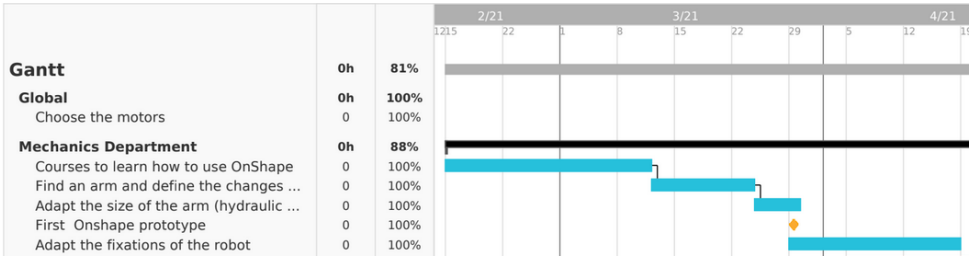


Figure 2.4: Gantt example

Finally, to manage the daily tasks and ensure both a good distribution of work between members and their achievement we used a to-do-list. It indicated the tasks to be carried out, the person in charge, the person in charge who followed the progress, the date, and the status.

Automation Pole (Responsable : Maxime) Members : Antoine, Maxime, Noé puis Héloïse	who ?	Responsable ?	when ?	State ?	Remarks
Training in the fundamentals of matlab/simulink	Antoine Maxime Noé	Maxime	18/02/21	Fini	
Theoretical kinematic study of the arm	Antoine Maxime Noé Thomas	Maxime	10/05/21	Fini	Delay because study too complex manually, but after consulting with Mr. Kruszewski Matlab could allow to do this
Dynamic study of the hand (torque equation)	Antoine Maxime Noé	Maxime	15/05/21	Fini	
Kinematics study with matlab + workspace	Antoine		10/05/21	Fini	Gets a graph representing the points reached by the arm

Figure 2.5: To do list example

2.3 Communication

Communication was a key point for success. We were fortunate to have a client who was very involved in the project. Thus, we had weekly meetings with him to review the progress. It was also an opportunity to take advantage of his expertise in the management of such a project as well as his ideas. During those meetings, we discussed both the technical and management aspects. He also allowed us to meet people such as the CTO of *BMW* who gave us precious advice. At the request of Mr. Kedziora, we did not have a person in charge of taking notes. A schedule was defined in order to have a rotation between all the members and all benefit from this experience. A template for uniformity was set up.

Each pole also met one afternoon per week to work and exchange on the technical part of the project, and a monthly meeting was held between the pole managers and the project manager to ensure the smooth running of the project.

To exchange information, we had a *Google Drive* to store and share all the documents. *GitHub*[8] folders were also created for the IT part. Finally, we also communicated with our client via *Whatsapp*. We also had a messenger group with all members.

2.4 Resource management

Two types of means can be taken into account. First, the direct financial expenses. These are all the purchases we made. As explained earlier, Héloïse was responsible for this tracking. She could therefore easily ensure that the expenses were in line with the client's wishes. These expenses also included the use and purchase of materials from central offices.

[8] Main repository ROS repository

dernière mise à jour :	Type d'achat	Lieu d'achat	Pole	Cout (€)
08/12/2020	Inscription Conférénce Fira	Site Fira	robotique	39€
13/03/2021	AZ delevery cablede remplacement	Amazon	informatique	4,99€
	Inno Maker Rasberry Pi	Amazon	informatique	13,99€
	LEICKE 72 Alimentation	Amazon	informatique	19,99

Figure 2.6: Financial budget extract

Matériel école	nom	prix	quantité	Temp d'impression
impression main et doigts	ASA PRO	16,28	50	5h50
impression main V2	ABS	18,8	110	18h
	ASA PRO	28,97	161	

Figure 2.7: Material budget extract

Many non-direct expenses were also taken into account. First, the cost of our work is summarized in the table below. But also the consulting hours. Indeed, Centrale offered the possibility to contact professors and benefit from their knowledge. So we contacted different people who redirected us to the teachers who could help us the most. To ensure full and balanced use as needed, Anna kept a document describing their use.

These hours were a great help. We were able to count on our coaches to help us manage this project. We were able to count on consultants, especially Mr. Kruszewski, who was really involved in the project. Thanks to him, we were able to benefit from an almost weekly follow-up to perpetually advance the project. It was also a question of privileged moments to deepen certain subjects with them.

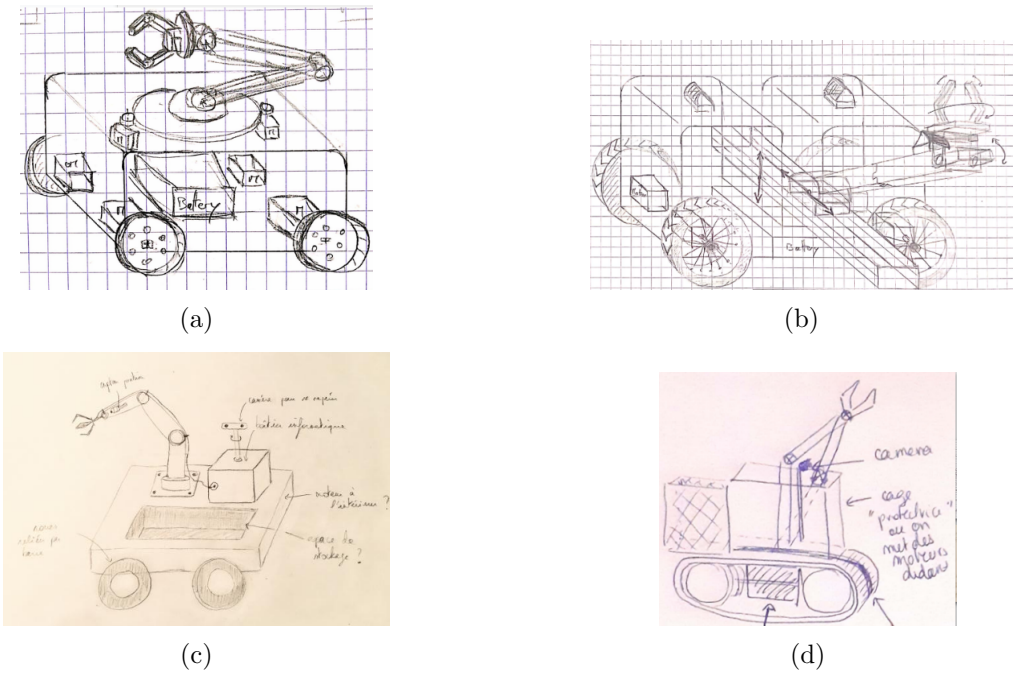
Pole	Number of days per person	Detail	Daily rate	Total
Mecanic	81	5 people available at 75% 1 day a week for 18 months	36,23 €	14 671,13 €
IT	81	4 people available at 75% 1 day a week for 18 months	36,23 €	11 736,90 €
Automation	81	3 people available at 75% 1 day a week for 18 months	36,23 €	8 802,68 €
			Total	35 211 €

Figure 2.8: Human budget extract

3 Design

3.1 Brainstorming

Right after we defined the project, we took some time to propose ideas. The goal was to abstract from the existing and imagine different configurations, shapes, or mechanisms. So we made hand sketches to illustrate our different ideas as shown in the drawings below.

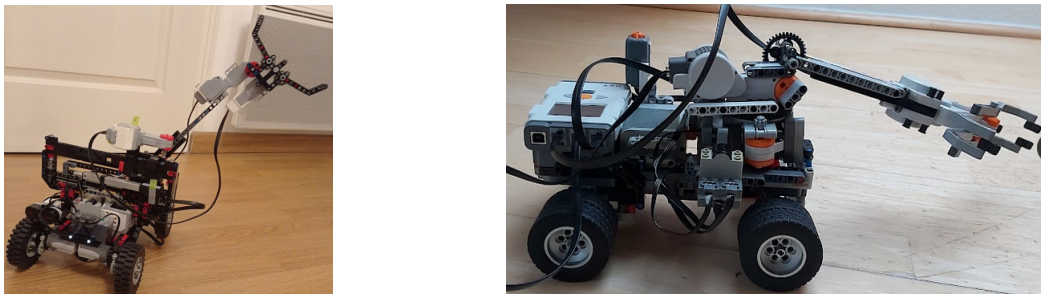


(a) (b)

(c) (d)

Figure 3.1: Robot drawings

In order to prototype and select a design quickly we also built two lego prototypes. This allowed us to define the basic design of the arm: a three-degree-of-freedom arm attached with a pivot link to the base. We hesitated with the design of the figure 3.1b with a central axis but when we built the lego prototype we found constraints too important to reach different points in space. In particular, the ability to work only in a 2D plane without moving the base.



(a) (b)

Figure 3.2: Lego robots

Like the market research and project definition, all members participated in this step in order to get as many ideas as possible but also to be sure everyone understand the future goal. We then divided ourselves into technical divisions as detailed in the previous 2.1. Once this process

was finished, the mechanical pole could start a design in CAD[9] using the software *Onshape*.

3.2 Onshape

3.2.1 Presentation

To model our robot, we used the software *Onshape* [Ind]. It is a computer-aided design software system, delivered over the Internet via a software-as-a-service model. This software offers many advantages :

- collaborative
- quick to learn
- online

The last one may limit its use because it requires an internet connection.

3.2.2 Rules

Rules to standardize our work have also been established from the beginning. As we will see in the following parts, it is necessary to respect them to be able to exploit these documents with other software.

- Name all parts with a specific name: it is the ID of the part and it has to be unique
- One subassembly by articulation: a subassembly contains all the parts of an articulation that have a fixed join between them
- One main assembly containing the motor links: the main assembly contains only the join that you want to see at the end. Usually, it is the dynamic join but it can be fixed to get a part separately.
- Assign material to each part: in *Onshape*, you precise the volumic mass, you need to pay attention in case you use PLA with a 3D printer. A part is never full at 100% and you need to adapt the volumic mass to get the correct masS.
- Motor link in the main assembly needs to be named $dof\_ < link\_name >$
- A relation between parent/child needs to be respected for every link: In *Onshape*, like all CAO software, there is a parent/child relation in every link. You need to pay attention to this, it can be a source of many errors if you want to export your model. When you create a link, fixed or motorized, the first element you click on is the child. The second one is the father. You can then check the parent and the child as shown in the following image. It is therefore necessary to start from the origin to the end of the robot, respecting these relationships at the time of assembly.

[9] the use of computers to aid in the creation, modification, analysis, or optimization of a design

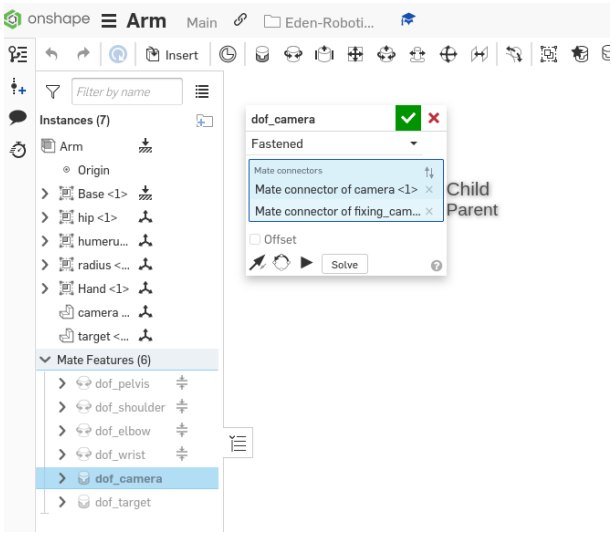


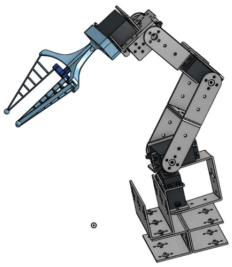
Figure 3.3: Parent child principle

3.3 Arm

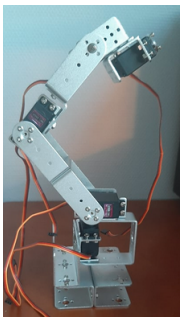
The design of the arm was done in 4 steps.

3.3.1 Step 1

For the first step, we looked for and bought an arm resembling the desired design. Thanks to this, the mechanical department was able to train on different tutorials and then to make an "easy" application that consisted only in reproducing this arm by respecting the rules stated. At the same time, the automatic pole was able to take advantage of this to make tests on a physical robot.



(a) Onshape design



(b) Physical robot

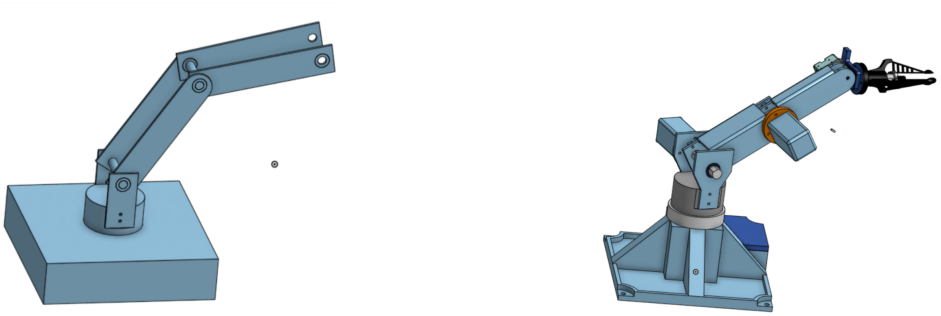
Figure 3.4: Prototype 1

The purchase of this arm allowed a first quick design on which all the members could work. We thus avoided a waiting time for the computer science and automatic poles. They were able to work on this one and carry out various tests in order to leave time for the mechanical pole to create our own design that we were going to manufacture.

3.3.2 Step 2 and 3

In steps 2 and 3, we created different prototypes entirely on the computer. Little manufacturing was done, we only tested parts of the robot in order to save time and iterate as quickly

as possible.



(a) Prototype 2 (b) Prototype 3

Figure 3.5: Prototypes 2 and 3

Prototype 2 was a rough sketch of our robot. As we said, we decided to have 3 degrees of freedom on the arm (shoulder, elbow, wrist) and a pivot link on the base to reach a maximum of points in space. We can thus have a base that will settle in one place and an arm that will recover a maximum of tomatoes without having to move it. In view of the layout of the tomatoes, we also decided to have two parts that we will call humerus and radius almost the same size.

It allowed us to make choices like the types of joints, and the lengths of each of the parts. Some of us also followed the training courses offered by Centrale on 3D printing and laser cutting. Some parts of this design, although rough, have been created to practice.

(a) 3D printed part (b) Laser cutting part

Figure 3.6: 3D printed and laser cutting examples

We then designed another robot (step 3) more complete since it integrates the wrist mechanism. The motors and the ball bearings were also represented, with realistic masses. Following the problems we noticed during the manufacturing of some parts, we also imagined a new structure for the arms to stiffen the whole. However, the geometry implied a one-piece construction of the arm rather than two laser-cut parts. It had to be 3D printed, which increased the production time. Although the structure was better bonded, these parts also lacked strength at the corners. It was therefore not retained.

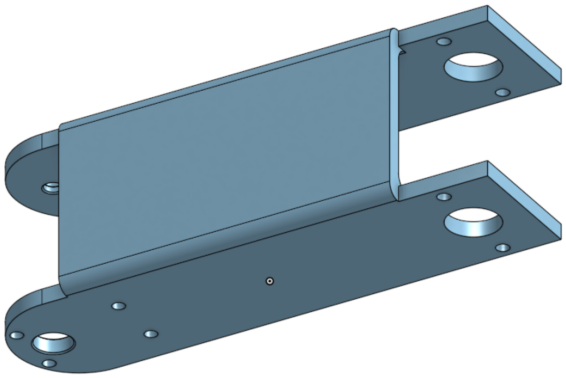
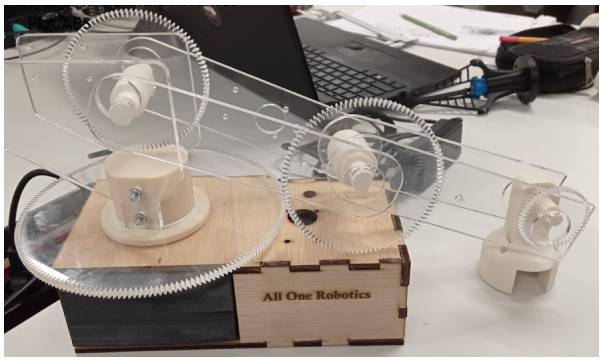


Figure 3.7: Arm modified

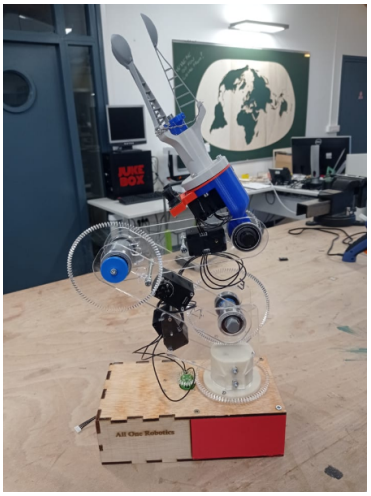
It is on this robot that the studies of part 6 were led first to select the motors for the final robot.

3.3.3 Step 4

All these tests and changes lead us to step 4: the complete design of the robot and its complete fabrication. For this, we made some parts in PVC or plywood with a laser cut, we printed in 3D some parts. Others such as the filleted rods were directly purchased. To overcome the lack of rigidity, we added rods between the two plates of the humerus and radius. A building instruction to reproduce the robot can be found in appendix D. All the STL files can also be found on the GitHub page of this project.



(a) Partial assembly



(b) Final assembly

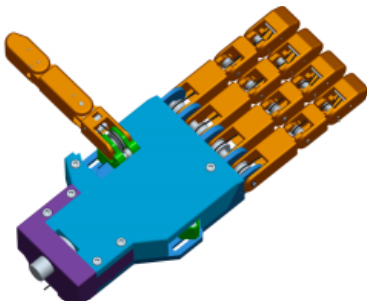
Figure 3.8: Arm step 4

This is only a prototype and improvements are still to be expected. We had some ideas that we, unfortunately, could not explore. Some PVC parts can be replaced by aluminum to increase the resistance, and a structure around the arm to manage the cables can also be added for example.

3.4 Hand

The design of the hand was a crucial step. Based on what was being done and the advantages put forward, we created our own design. The two figures below present techniques to catch objects, one based on aspiration[Roba] and the other mimicking a human hand[Bar10]. However, the first one is relatively complex and therefore expensive and long to implement. Moreover, there is a risk of damaging the fruit by sucking it too hard. The second one is not very adapted for tomatoes.

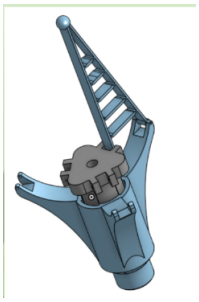
(a) Hand 1



(b) Hand 2

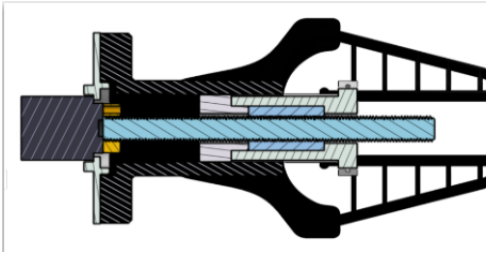
Figure 3.9: Existing hands

Our first idea was to create a hand composed of three deformable rights that would follow the shape of the tomato. These fingers are attached to a fixed part and to the same actuator to make them move at the same time. The actuator is a screw-nut system and an adapter to make the 3D-printed fingers move.



(a) Onshape design

(b) Hand



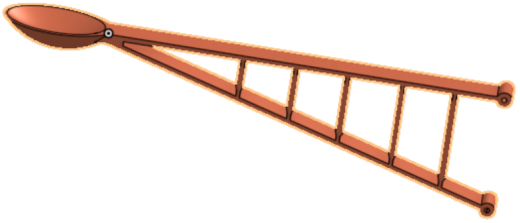
(c) Actuator system

Figure 3.10: Hand V1

Unfortunately, with our knowledge and facing the difficulty to find a material deformable enough to fit the shape of the tomato while supporting enough to pull it, we changed the shape of the fingers. While keeping the principle of three fingers, they have been adapted to have a

spoon shape and come directly into the shape of the tomato.

(a) Hand picking tomato



(b) Finger

Figure 3.11: Hand V2

This requires greater precision when picking the fruit, however. This configuration also offers less flexibility when it comes to the different shapes of tomatoes. It could be interesting to try to pursue the principle of a deformable hand.

4 Robot Characteristics

We indicate here the characteristics of the final robot but all these studies have been conducted on intermediate prototypes. They will be used in the following parts.

4.1 Definition

Definition : Given a vector $x = [x_1 x_2 x_3]^T \in \mathbb{R}^3$, we define :

$$[x] = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}$$

Definition : Given two vectors x and y in $\mathbb{R}^3$, we define : $x \times y = [x] \cdot y$

Definition : For a joint, we define the pitch $h = \frac{v}{w}$ with v: the linear speed and w the angular speed

Definition : The screw is a 6×1 vector that represents the angular velocity when $\dot{\theta} = 1$ and the linear velocity of the origin when $\dot{\theta} = 1$. $S = \begin{bmatrix} s_w \\ s_v \end{bmatrix}$ with $s_v = hw - s_w \times q$ where h is the pitch and q is a point on the axis of the screw.
Definition : For a given reference frame, a screw axis S is written as

$$S = \begin{bmatrix} s_w \\ s_v \end{bmatrix}$$

where either (i) $\|s_w\| = 1$ or (ii) $\|s_w\| = 0$ and $\|s_v\| = 1$. If the pitch is finite ($h = 0$ for a pure rotation), then $s_v = hs_w - s_w \times q$ where q is a point on the axis of the screw.

4.2 Configuration

The figure below shows the kinematic diagram[10] of the robot. The figure defines an {s} frame[11] at the bottom, an {e} frame at the end effector position, and a {c} frame at the camera position. The robot is at its home configuration[12]. The joints are represented with the rotation (positive rotation about the axes is by the right-hand rule).

The parameters can be found with Onshape and are listed below:

$L_0 = 0.069m$	$d_0 = 0m$	$h_0 = 0.06m$
$L_1 = 0.116m$	$d_1 = 0.018m$	
$L_2 = 0.16m$	$d_2 = 0.042m$	
$L_3 = 0.155m$	$d_3 = 0.01413m$	
$L_c = 0.053m$	$d_c = 0.0105m$	$h_c = 0.0815m$
$L_e = 0.2377m$	$d_e = 0.0105$	$h_e = 5.10^{-5}m$

[10] Illustates connectivity of links and joints of a mechanism
[11] An abstract coordinate system whose origin, orientation, and scale are specified by a set of reference points
[12] Corresponds to the rest position, usually all joints or links are set to zero

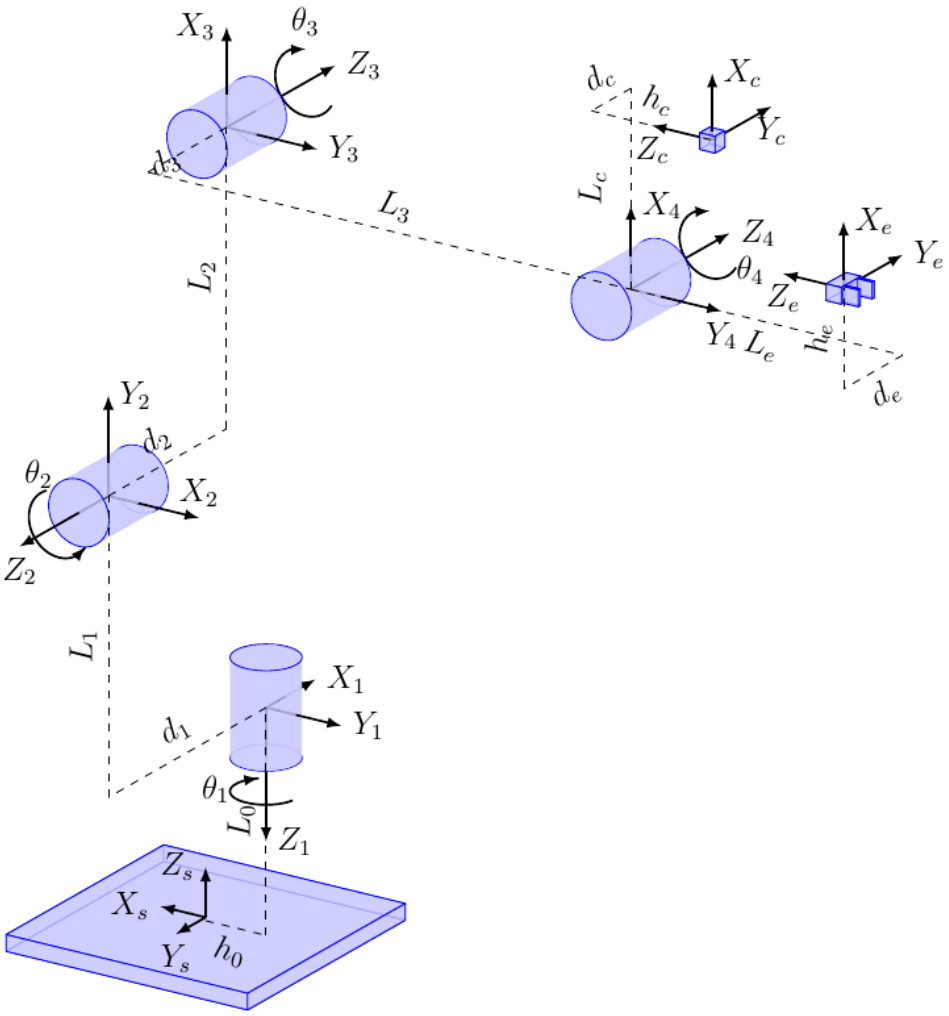


Figure 4.1: Kinematics schema

We can then define M_c and the M_e transformation matrix[13] of the camera and the end effector from the base (T_{sc} and T_{se}) when the robot is at its home configuration.

$$M_c = \begin{bmatrix} 0 & 0 & 1 & -h_0 - L_3 - h_c \\ 0 & -1 & 0 & d_1 - d_2 + d_3 + d_c \\ 1 & 0 & 0 & l_0 + l_1 + l_2 + l_c \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } M_e = \begin{bmatrix} 0 & 0 & 1 & -h_0 - L_3 - h_c \\ 0 & -1 & 0 & d_1 - d_2 + d_3 + d_c \\ 1 & 0 & 0 & l_0 + l_1 + l_2 + l_c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4.2.1 Base frame

In this subsection, we study the kinematics parameters in the base frame {s}. It will be the one used in the following sections.

The rotation S_{w_i} of each joint in {s} are :

$$S_{w_1} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}, S_{w_2} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, S_{w_3} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, S_{w_4} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix},$$

[13] Transforms one vector into another vector by the process of matrix multiplication The transformation matrix alters the cartesian system and maps the coordinates of the vector to the new coordinates

We can also write the position of each joint $q_1, q_2, q_3, q_4, q_c, q_e$ in {s}. Lining up the position as columns, we get :

$$\begin{bmatrix} -h_0 & -h_0 & -h_0 & -h_0 - L_3 & -h_0 - L_3 - h_c & -h_0 - L_3 - L_e \\ 0 & d_1 & d_1 - d_2 & d_1 - d_2 + d_3 & d_1 - d_2 + d_3 + d_c & d_1 - d_2 + d_3 + d_e \\ L_0 & L_0 + L_1 & L_0 + L_1 + L_2 & L_0 + L_1 + L_2 & L_0 + L_1 + L_2 + L_c & L_0 + L_1 + L_2 + h_e \end{bmatrix}$$

There are all pure rotation joints, using the position, the rotation axis and the formula defined above we can calculate the screw axis[14] S_1, S_2, S_3, S_4 in {s}. Lining up them as columns, we get :

$$S_{list} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & -L_0 - L_1 & L_0 + L_1 + L_2 & L_0 + L_1 + L_2 \\ -h_0 & 0 & 0 & 0 \\ 0 & -h_0 & h_0 & h_0 + L_3 \end{bmatrix}$$

4.2.2 End effector frame

In this subsection, we study the kinematics parameters in the end effector frame {e}. However, it is not the one that will be used later.

The rotation axis S_{w_i} of each joint in {e} are :

$$S_{w_1} = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}, S_{w_2} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, S_{w_3} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, S_{w_4} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix},$$

We can also write the position of each joint $q_1, q_2, q_3, q_4, q_c, q_e$ in {e}. Lining up the position as columns, we get :

$$\begin{bmatrix} -h_e - L_2 - L_1 & -h_e - L_2 & -h_e & -h_e & -h_e + L_c & 0 \\ d_e + d_3 - d_2 + d_1 & d_e + d_3 - d_2 & d_e + d_3 & d_e & d_e - d_c & 0 \\ L_e + L_3 & L_e + L_3 & L_e + L_3 & L_e & L_e - h_c & 0 \end{bmatrix}$$

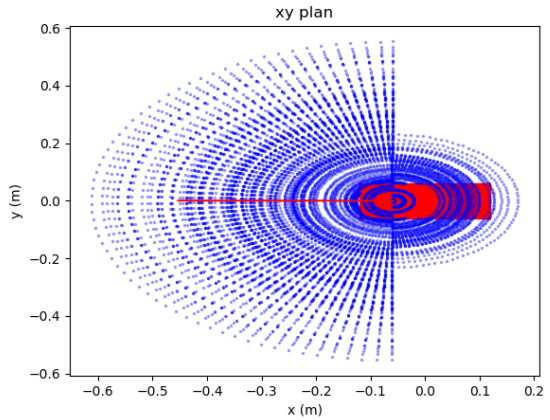
There are all pure rotation joints, using the position, the rotation axis and the formula defined above we can calculate the screw axis B_1, B_2, B_3, B_4 in {e}. Lining up them as columns, we get :

$$B_{list} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & L_e + L_3 & -L_e - L_3 & -L_e \\ -L_e - L_3 & 0 & 0 & 0 \\ d_e + d_3 - d_2 + d_1 & h_e + L_2 & -h_e & -h_e \end{bmatrix}$$

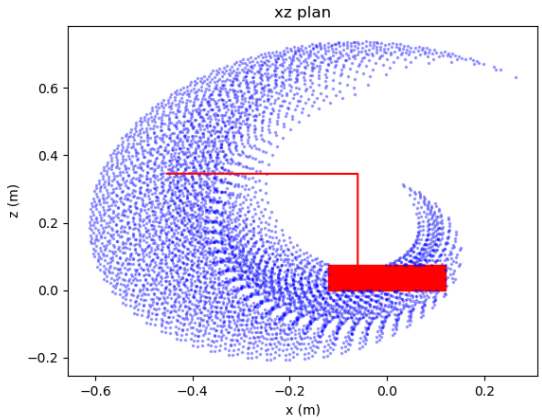
[14] Is used to describe rigid body motions and forces, be its axes of motion (such as the rotation axis of a pure rotation) or force (such as the line of action of a force)

4.3 Workspace

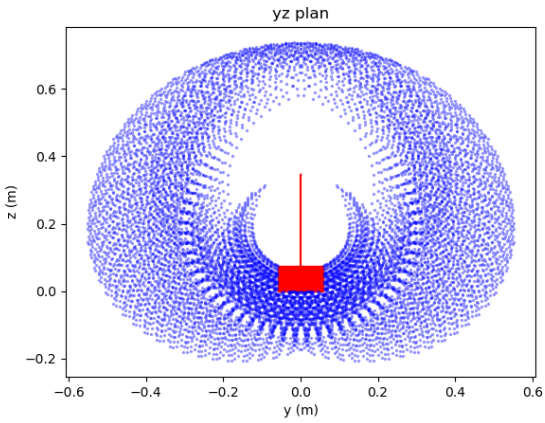
In order to define our robot, we have determined the Workspace[15]. To do this, we used the forward kinematic computed with python as explained in section 5. Knowing the limits of each link, we made loops to evaluate the reachable positions in the whole space. We then represented the working space according to the 3 planes of the robot: top, front, and side. Obviously, this does not allow us to visualize the complete workspace, but it is the clearest way to show it. However, our calculations allow us to access the full workspace, i.e. all the points reachable by the robot. We also put the python code to represent these figures. The robot is also present in the figures in red in its zero configuration to better estimate the working space. To understand how we compute the transformation matrix and the position see section 5.2.2.



(a) XY plan for z=0.07m



(b) XZ plan for y=0m



(c) YZ plan for x=-0.06m

Figure 4.2: Workspace

```
    # import kinematics parameters
    from parameters import *
    import matplotlib.pyplot as plt
    # joint values in radian
    pelvis_limits = list(np.arange(-1.57,1.57,0.2))
    shoulder_limits = list(np.arange(-1.57,0.69,0.3))
    elbow_limits = list(np.arange(-1.57,1.05,0.4))
```

[15] A specification of the reachable configuration of the end effector

```
    wrist_limits = list(np.arange(0,1.57,0.4))
    # loop on values to get all positions
    x,y,z = [],[],[]
    for pelvis in pelvis_limits:
        for shoulder in shoulder_limits:
            for elbow in elbow_limits:
                for wrist in wrist_limits:
                    thetalist = np.array([pelvis,shoulder,elbow,wrist])
                    # compute transformation matrix and point
                    t = mr.FKinSpace(m_e,screw_list,thetalist)
                    p = t.dot(np.array([0,0,0,1]))[:-1]
                    x.append(p[0])
                    y.append(p[1])
                    z.append(p[2])
    # plot xy plan
    plt.axes()
    plt.plot(x,y,".b",ms = 3,alpha=0.3) # plot position
    plt.plot([p1[0],p2[0],pe[0]], [p1[1],0,pe[1]],"-r") # plot robot arm
    rectangle = plt.Rectangle((-0.12,-0.06), 0.24, 0.12, fc='red',ec="red")
    plt.gca().add_patch(rectangle) # plot robot body
    plt.title('xy plan')
    plt.xlabel('x (m)')
    plt.ylabel('y (m)')
```

4.4 Payload

Another essential characteristic of our robot is the payload[16]. A cherry tomato weighs on average 15g, so we need a robot able to lift at least 2 times that to avoid any problems. To measure the payload, we placed the robot in the "worst" configuration: arm stretched flat. We then measured the intensity of the shoulder motor to lift a given mass. Then we held the robot in position by blocking it and we operated the same motor by measuring the current at the moment of the break. For a DC motor, the current is proportional to the torque which is proportional to the mass, it is then possible to determine the maximum mass that the robot can support.

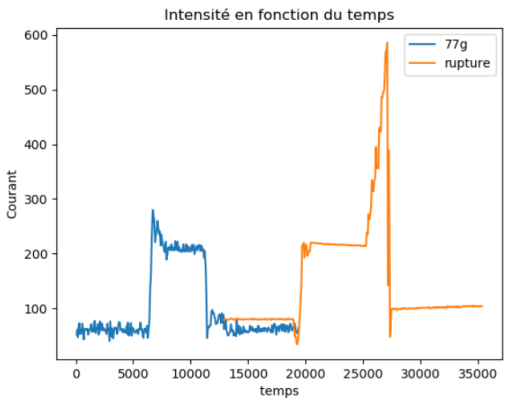


Figure 4.3: Payload mesure

[16]Maximum mass a robot can support before breaking

The graph below shows the intensity in a specific unit of the motor. We measured an intensity of 300 for a mass of 77g and an intensity of 600 at breakage. Thus, **the payload of our robot is 144g**. We can note that this is not the limit of the robot. Indeed, it is the PVC gear that broke and not the physical limits of the robot. A stronger gear would allow for increasing the maximum mass if needed.

4.5 Precision

Finally, we have determined the accuracy of our robot. For this, we subjected our robot to a movement of 10cm from left to right several times. We made sure to mark our robot's trajectory and measured the distance covered. It was 12cm on average which makes a **accuracy of 2cm**. However, as we will see later, this is sufficient with the control method we have selected because the error is naturally compensated by the user.

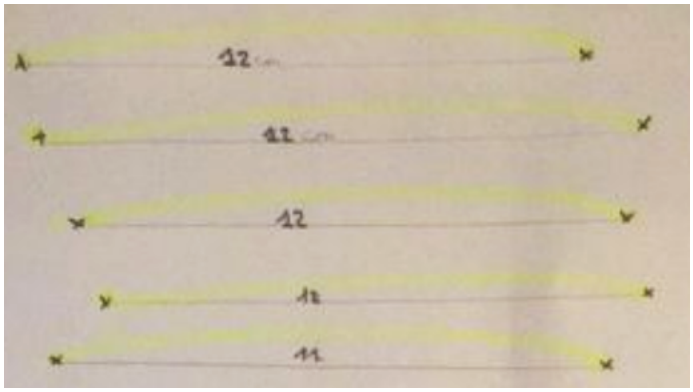


Figure 4.4: Precision

4.6 Others

We also have other characteristics that we have measured:

- height: 70cm including 50cm of arms
- mass: 2.7kg of which for 1.2kg the base

5 Kinematics

In parallel to the design, we simulated the operation of the robot using *Matlab*, *Simulink*[Mata], and python. The work explained from now on is done on the last prototype presented in the previous section. However, the principle applied has been the same throughout the project. The simultaneous work was important in order to anticipate the delays due to the manufacturing of the robot. The goal of this section is to compute the forward kinematics[17] and the inverse kinematics[18].

5.1 URDF Format

The Universal Robot Description Format (URDF) is an XML (eXtensible Markup Language) file format used by the Robot Operating System (ROS) to describe the kinematics, inertial properties, and link geometry of robots. A URDF file describes the joints and links of a robot:

- **Joints :** Joints connect two links: a parent link and a child link. A few of the possible joint types include prismatic, revolute (including joint limits), continuous (revolute without joint limits), and fixed (a virtual joint that does not permit any motion). Each joint has an origin frame that defines the position and orientation of the child link frame relative to the parent link frame when the joint variable is zero. The origin is on the joint's axis. Each joint has an axis 3-vector, a unit vector expressed in the child link's frame, in the direction of positive rotation for a revolute joint or positive translation for a prismatic joint.
- **Links :** While the joints fully describe the kinematics of a robot, the links define its mass properties. The elements of a link include its mass; an origin frame that defines the position and orientation of a frame at the link's center of mass relative to the link's joint frame described above; and an inertia matrix, relative to the link's center of mass frame, specified by the six elements on or above the diagonal. (Since the inertia matrix is symmetric, it is only necessary to define the terms on and above the diagonal.)

This format will be useful to build the *Simulink* model of the robot. Thankfully, the library **onshape-to-robot** in python can transform Onshape design into a URDF model. It is very important that you have respected the rules explained in the last section. It will download the STL file of each part and create all the joints and the links from the main assembly. The inertia matrices and the mass are also imported for each block.

As explained in the library documentation, you should create Onshape API key (see Onshape developer portal). It is recommended to store them on your bashrc or zshrc because the secret key will no longer be shown.

```
export ONSHAPE_API=https://cad.onshape.com
export ONSHAPE_ACCESS_KEY=Your_Access_Key
export ONSHAPE_SECRET_KEY=Your_Secret_Key
```

Then, you should create a folder where you want your URDF file to be constructed and write a config.json file:

[17] The use of the kinematic equations of a robot to compute the position of the end-effector from specified values for the joint parameters.

[18] The use of kinematic equations to determine the motion (appropriate joints configuration) of a robot to reach a desired position.

```
~$ mkdir -p robot\_urdf && touch robot\_urdf/config.json
```

The config file must contain at least the following fields :

```
{
    "documentId": "document-id",
    "assemblyName": "onshape assembly",
    "outputFormat": "urdf"
}
```

The documentId is the number (below XXXXXXXXX) you can find in Onshape URL: https://cad.onshape.com/documents/XXXXXXXXX/w/YYYYYYYY/e/ZZZZZZZZ.

Once this is done, if you properly installed and set up your API key, just run the following command. It will create a URDF file and put the STL file in the folder.

```
~$ onshape-to-robot robot_urdf
```

The file will contain only the joints define in the final assembly. This is why you need subassemblies with fixed parts. As we can see in the extract below, the camera has a fixed joint with the hand. Also, it downloads all STL files and describes the visual position of each. However, it has only one global parameter for each subassembly that define the inertia.

```
<link name="camera">
        <visual>
        <origin xyz="-0.0505004 -3.25261e-18 -0.083" rpy="-3.14159 8.99294e-30 -3.04281e-32" />
        <geometry>
            <mesh filename="package:///camera.stl"/>
        </geometry>
        <material name="camera_material">
            <color rgba="0.282353 0.54902 0.160784 1.0"/>
        </material>
    </visual>
    <collision>
        <origin xyz="-0.0505004 -3.25261e-18 -0.083" rpy="-3.14159 8.99294e-30 -3.04281e-32" />
        <geometry>
            <mesh filename="package:///camera.stl"/>
        </geometry>
        <material name="camera_material">
            <color rgba="0.282353 0.54902 0.160784 1.0"/>
        </material>
    </collision>
    <inertial>
        <origin xyz="-0.003 9.37179e-20 -0.0015" rpy="0 0 0"/>
        <mass value="0.000135" />
        <inertia ixx="5.0625e-10" ixy="-1.00385e-41"  ixz="7.79417e-40" iyy="5.0625e-10" iyz="6.90342e-42" izz="8.1e-10" />
    </inertial>
</link>

<joint name="camera" type="fixed">
    <origin xyz="0.0560004 0.08 -0.0105" rpy="1.5708 3.04281e-32 8.99294e-30" />
    <parent link="hand" />
    <child link="camera" />
    <axis xyz="0 0 1"/>
    <limit effort="1" velocity="20" />
    <joint_properties friction="0.0"/>
</joint>
```

Figure 5.1: URDF file extract

5.2 Forward kinematics

To calculate the forward kinematics of our arm we used two different methods. This allowed us to compare the results and validate them.

5.2.1 Matlab simulation

To begin with, *Matlab* offers the possibility to import a URDF file describing a robot to make a *Simulink* model using the simscape toolbox. This is the easiest method when you have the URDF file. It also has the advantage of visually simulating the robot. Indeed, the model is based on the STL files of each part. To create the model, simply run the following command in *Matlab* :

```
%% create robot model from urdf
smimport(<path to urdf>)
```

It will open a *Simulink* file, once it is created, the joints must be modified so that the motor torque is calculated automatically and the desired angles can be entered manually. Finally, a position sensor linking the reference frame and the target frame must be added. We can then obtain the position of the hand for a given angle vector.

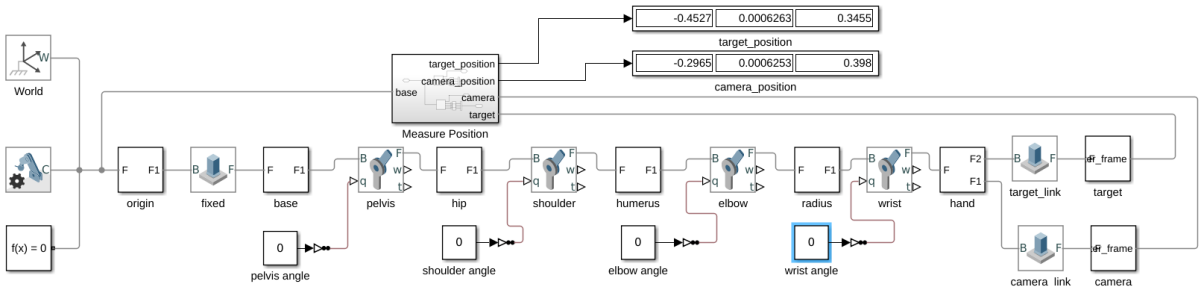


Figure 5.2: Simulink model

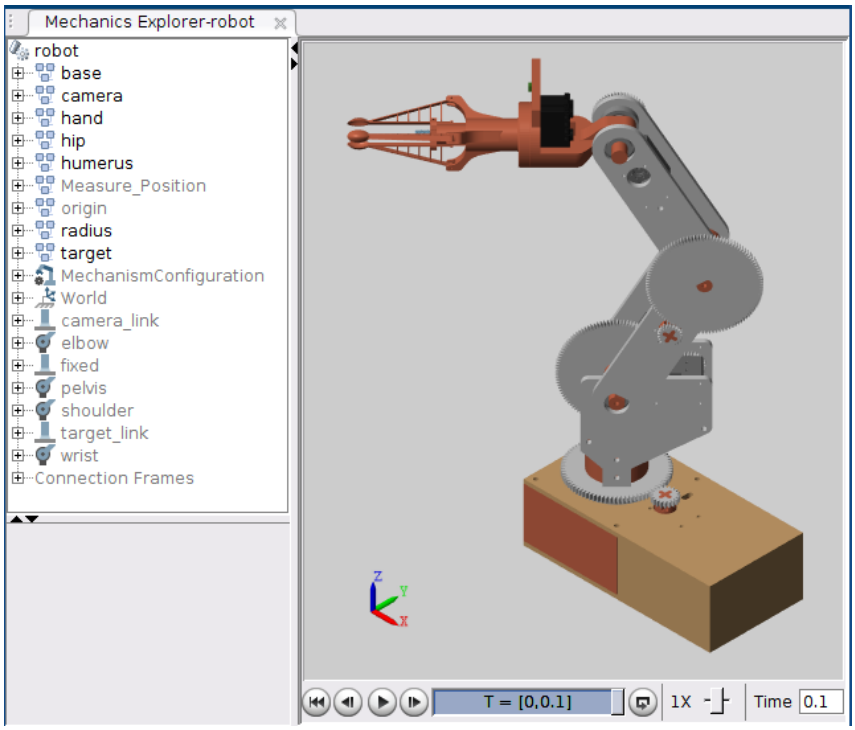


Figure 5.3: Matlab robot visualization

It should be noted that a target block positioned at the center of the fingers has been added in the Onshape model. It is in a fixed link with the hand and this link is defined in the general assembly. Thus, when creating the URDF file, this target appears separately. It allows having an easy way to locate it. The mass of this object, which must be defined, is 10^{-5} to be considered as zero. The same thing has been done for the camera position.

5.2.2 Python simulation

Definition : Let $S = (w, v)$ be a screw axis. If $\|w\| = 1$ then, for any distance $\theta \in \mathbb{R}$ traveled along the axis,

$$e^{[S]\theta} = \begin{bmatrix} e^{[w]\theta} & (I\theta + (1 - \cos\theta)[w] + (\theta - sin\theta)[w]^2)v \\ 0 & 1 \end{bmatrix}$$

If w=0 and $\|v\| = 1$ then

$$e^{[S]\theta} = \begin{bmatrix} I & v\theta \\ 0 & 1 \end{bmatrix}$$

The second approach is more theoretical and is based on the kinematic parameters seen previously. To realize the calculations we used the python library modern robotics [LP17]. It has native functions to calculate the forward kinematics. We will explain here how it computes the position of the end effector from the joints' position. Then we show how to use it in real life.

This library is based on the exponentials of matrices to calculate the position of the end effector from the coordinates of each link. It uses the following formula:

$$T(\theta) = e^{[S_1]\theta_1} e^{[S_2]\theta_2} e^{[S_3]\theta_3} e^{[S_4]\theta_4} M$$

where θ is a 4×1 vector of joint coordinates, S_i is the screw axes of the joint i and M is the transformation matrix when the robot is at its zero configuration.

So we can use the function FKinSpace which takes as arguments M,θ, and S_{list} as defined above. Depending on whether we want the position of the camera or the end effector, we just have to change the M matrix. This method was faster to perform the calculations and faster to test a large number of values. However, it does not allow visualization.

```
# import kinematics parameters
from parameters import *
import modern_robotics as mr
# define desired angles
thetalist = np.array([0,0,0,0])
# get transformation matrix and extract the position
t = mr.FKinSpace(m_e,screw_list,thetalist)
p = t.dot(np.array([0,0,0,1]))[:-1]
```

For the same set of angles, we then obtain the following results using *Matlab* and python:

As we can see, the results are identical at $10^{-4}m$. We can therefore validate the kinematic model of our robot. The position of *Matlab* is correct since it comes from an explicit model and allows visualization.

5.3 Inverse kinematics

In the same way, we used two methods to calculate the inverse kinematics. The objective is to determine the coordinates of each link from a given position.

joints (rad)	Matlab position (m)	Python Position (m)
[0 0 0 0]	[-0.4527 0.00063 0.3455]	[-0.4527 0.00063 0.3455]
[pi/4,-pi/3,0,pi/3]	[-0.1286 0.06948 0.07537]	[-0.1286 0.06949 -0.07533]
[pi/4,pi/6,-pi/6,pi/3]	[-0.2259 0.1668 0.4583]	[-0.2258 0.1667 0.4583]

Table 1: Matlab and Python result for forward kinematics

5.3.1 Matlab simulation

Once we had a *Simulink* model, we were able to create a *Matlab* variable that represents the robot. It is obtained with the script below. This variable contains information about each link (position, parent, child) but also about each body (mass, center of mass, and inertia). The bodies are numbered from 1 to 7. The number 1 corresponds to the base and 7 to the end effector. We could identify them from the information on mass and inertia.

```
%% create robot matrix
open_system('robot.slx')
S=sim('robot.slx')
[robot,importInfo] = importrobot(gcs)
robot.DataFormat = 'column';
```

The Robotic System toolbox then allows calculating the inverse kinematics. Indeed, there is a block *inverse kinematics* which takes as input the desired configuration (transformation matrix), weights which allow adjusting the importance of reaching the desired orientation and translation according to the 3 axes and returns the list of the position of each link. This block takes as a parameter the robot variable created before. You must then indicate the name of the body corresponding to the end effector.

The block also offers the possibility to choose the resolution method. We have selected the Levenber-Marquardt method leaving the original resolution parameters. This method requires providing an initial guess if possible close to the real value. As we will explain in the following parts, our robot will always start in the same position: the zero configuration. We, therefore, provided this angle vector as an origin.

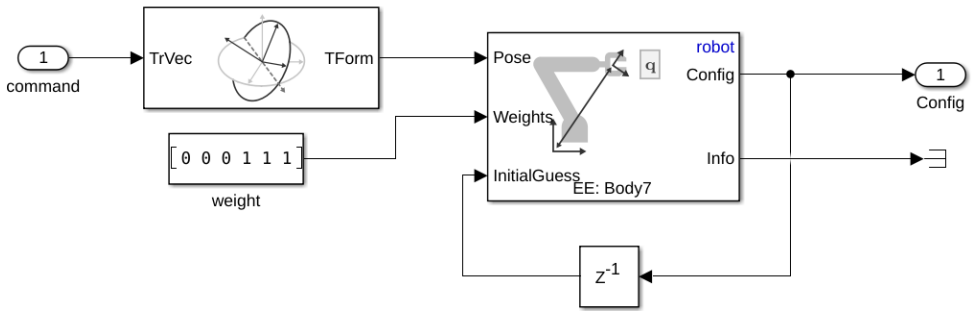


Figure 5.4: Inverse kinematics simulink

We are only interested in the position of the hand. Indeed, the hand being symmetrical the orientation of the fingers to catch objects is not important. Thus, the weights for the orientation are null and the ones for the position are at the maximum.

To verify the validity of the method, we retrieved the values of the link coordinates found by this block for desired end effector positions. Then, we submitted the robot in forward kinematics to these angles and verified that the positions are the same. Even if we know a set of angles corresponding to this position, depending on the initial hypothesis, the angles found can be different. Thus, comparing the value of the angles is not a good way to make sure that the resolution is working properly. As we do not take into account the orientation, we have only compared the positions.

desired position (m)	**real position (m)**	**angles (rad)**
[-0.4527 0.00063 0.3455]	[-0.4542 0.00063 0.3455]	[9.10^{-6} 9.10^{-3} 9.10^{-3} 0]
[-0.1286 0.06948 0.07537]	[-0.1288 0.06968 0.0739]	[0.7854 0.6981 0.7006 1.377]
[-0.2259 0.1668 0.4583]	[-0.2269 0.1678 0.4585]	[0.7855 0.6981 -0.1312 0.6766]

Table 2: Inverse kinematics results with Matlab

In the case of the table above, the initial assumption is always [0 0 0 0]. The desired position and the actual position are identical at $10^{-3}m$. We can therefore validate the model. Nevertheless, the closer the initial value is to the final result, the smaller the error is. As we will see later, in our case, the arm will start in its zero configuration. We, therefore, know the value of these angles. In order to be as accurate as possible, this will be our starting point for calculating the inverse kinematics in all cases. As we can see from the table, this is sufficient to obtain a satisfactory result in all cases.

5.3.2 Python simulation

To perform the inverse kinematics with python, we relied again on the library *modern robotic*. Inverting the kinematic forward matrix may not be impossible in some cases (redundancy[19] for example), so it is necessary to start from an initial guess and then iteratively calculate the result. It is based on the Newton-Raphson method to compute the solution.

Newton-Raphson method : To solve the equation $g(\theta) = 0$ numerically for a given differentiable function $g : \mathbb{R} \to \mathbb{R}$, assume θ^0 is an initial guess for the solution. Write Taylor's expansion of $g(\theta)$ at θ^0 and truncate it at first order. Keeping only these terms, set $g(\theta) = 0$ and solve for θ. Using this value of θ as the new guess for the solution and repeating the above, we get the following iteration :

$$g(\theta) = g(\theta^0) + \frac{\partial g}{\partial \theta}(\theta^0)(\theta - \theta^0)$$
$$\theta = \theta^0 - (\frac{\partial g}{\partial \theta}(\theta^0))^{-1} g(\theta^0)$$
$$\theta^{k+1} = \theta^k - (\frac{\partial g}{\partial \theta}(\theta^k))^{-1} g(\theta^k)$$

This iteration is repeated until some stopping criterion is satisfied: $|g(\theta^k) - g(\theta^{k+1}|/|g(\theta^k)| \leq \epsilon$ for some user-prescribed threshold value ϵ.

If we apply this to our case, we call f the forward kinematics function: $f(\theta) = x$. We define J the jacobian of f, $J(\theta^k) = \frac{\partial f}{\partial \theta}|_{\theta^k}$ and $\Delta\theta = \theta^{k+1} - \theta^k$. Using the Newton-Raphson method on the problem : $f(\theta) = x_d$, we will solve $f(\theta) - x_d = 0$:

$$x_d - f(\theta^k) = J(\theta^k)\Delta\theta$$

Depending on the fact that J is invertible, we will invert it or use the pseudo inverse to solve it.

[19]The fact that a robot can reach a specified position with more than one joints configuration

All this algorithm is already implemented in the *modern robotic* library. To compute the inverse kinematics in the base frame we need to specify: The joint screws S expressed in the base frame, the end effector home configuration M, the desired end effector configuration T, an initial guess θ^0 and the tolerances e_ω and e_v on the final error.

$$[\theta,\text{success}] = \text{IKinSpace}(\text{S},\text{M},\text{T},\theta^0,e_\omega,e_v)$$

As we explained for *Matlab*, we are only interested in the final position of the end effector. So we put a large value for e_ω (1) the orientation error and a very small value for e_v (0.005) the position error.

Unfortunately, we didn't have enough time to develop this part and we never managed to make the solutions converge. However, with a little more time, using python would speed up the development of the software by avoiding the use of *Matlab* and the code generator.

6 Torque Study

6.1 General principal

As soon as we were able to create the first model, we were interested in the necessary motor torque[20]. The objective was to determine the characteristics of each joint. Even if it was not the final robot, the different prototypes allowed us to have an estimation that we updated as soon as a choice was made.

In order to determine the necessary motors, we measured the following variables:

- torque
- rotation speed
- power

Figure 6.1: Joint sensing block

All these measurements were made in simulation with *Matlab*. In the simscape model, we can modify the linkage blocks to return the torque and rotational speed. The Power is measured by multiplying the two. The units are not specified and depend on the units defined in the URDF file. In our case :

- length: m
- time: s
- mass: kg
- angle: rad
- force: N
- power: W

We remind the relationship between torque, speed, and power:

$$P \quad = \quad CW \text{ with } P \text{ in } W, C \text{ in } N.m, \text{ and } W \text{ in } rad.s^{-1}$$

To make these measurements, we built the following model: a block to send a trajectory, a block for inverse kinematics, and a block of simulation on which we come to carry out the measurements. It was important not to control the position of the links directly to avoid any discontinuity. Indeed, it is necessary that all the functions are derivable twice as a function of time to calculate the torque and the speed of rotation. This is why we make our robot follow a given trajectory.

[20]The amount of rotational force that the motor develops

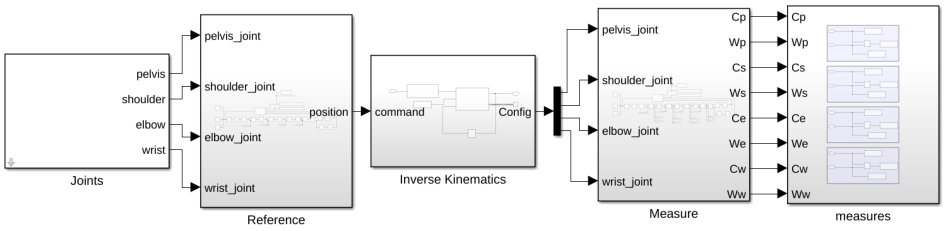


Figure 6.2: Joint analysis model

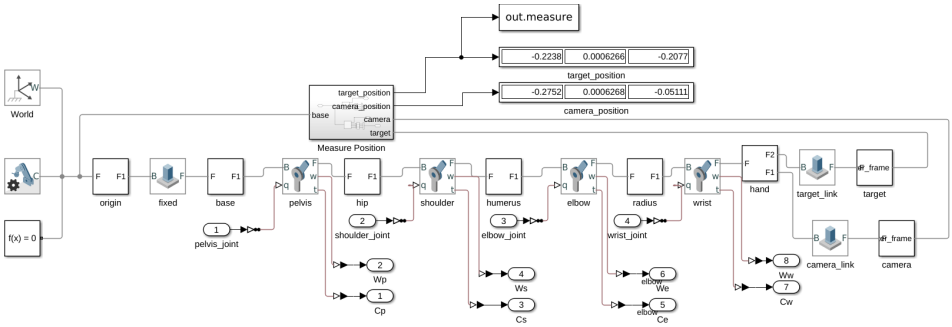


Figure 6.3: Joint analysis robot measure

We develop in the following sub-sections the results for each link. After having estimated the characteristics a margin of 30% was taken. Our consultant explained to us that it was a security.

In reality, only the power delivered interested us to select the engines. Indeed, if an engine is able to provide a certain power, we must look at the tuple (torque, speed) allowing this power. If one of the two does not meet the necessary values, a gear can be added. Let's take a gear of speed r, $\omega' = r\omega$ and $C' = \frac{C}{r}$ or $P' = \omega' C' = r\omega\frac{C}{r} = \omega C = P$. The gear allows changing the torque and the speed without modifying the power. Therefore, a corresponding gear must exist.

A static study was also conducted. We placed the robot in the "worst" configuration for each link and then we measured the torque required to maintain the robot in this position. The torques indicated on the motors' data sheet do not usually guarantee a static hold. To make sure that our motors are powerful enough, we multiplied this value by 3 to choose them.

The table below lists the characteristics required for each linkage as well as the estimates obtained on the first prototype. It is important to note that this one, although farther from reality, is still close. Moreover, the final values are all lower because we have reduced the mass of the robot. In order to anticipate the delivery time, we based ourselves on results obtained on intermediate prototypes. We can then validate this method.

Joint	**Power/estimate (W)**	**Torque/estimate (N)**	**Angulare speed/estimate (rad/s)**
Pelvis	0.2/1	0.2/4.2	0.9/0.23
Shoulder	1.8/4	5.4/9.8	0.77/0.4
Elbow	0.7/2.3	2.07/2.4	0.78/0.94
Wrist	0.23/0.4	0.38/1.1	0.94/0.35

Table 1: Power, torque, and speed rotation needed per joint

6.2 Static

For the static study, we put the robot in the position that requires the most effort to hold it in place. It is thus in a horizontal position with its arm stretched out. This is indeed the position where the torque is the most important for the shoulder, the elbow, and the wrist. In the static position, there is never any effort on the pelvis.

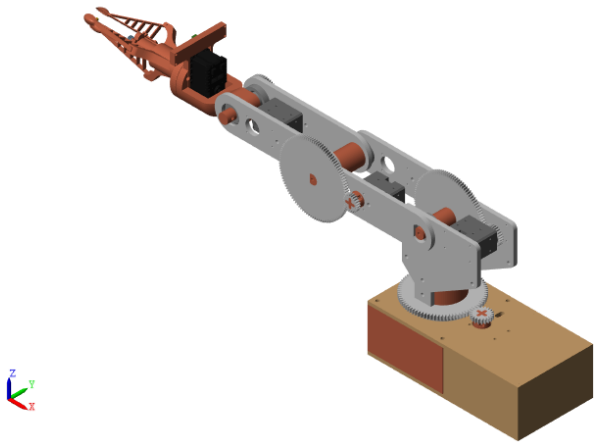


Figure 6.4: Static position

We obtain the following results. We have plotted in blue the measured value and in red the value that our motor must be able to provide to ensure that it will be able to maintain this position. The torques indicated on the data sheets are generally dynamic torques and not static. According to the advice given to us, we take three times the value obtained in static to be sure the motor will be able to give enough torque.

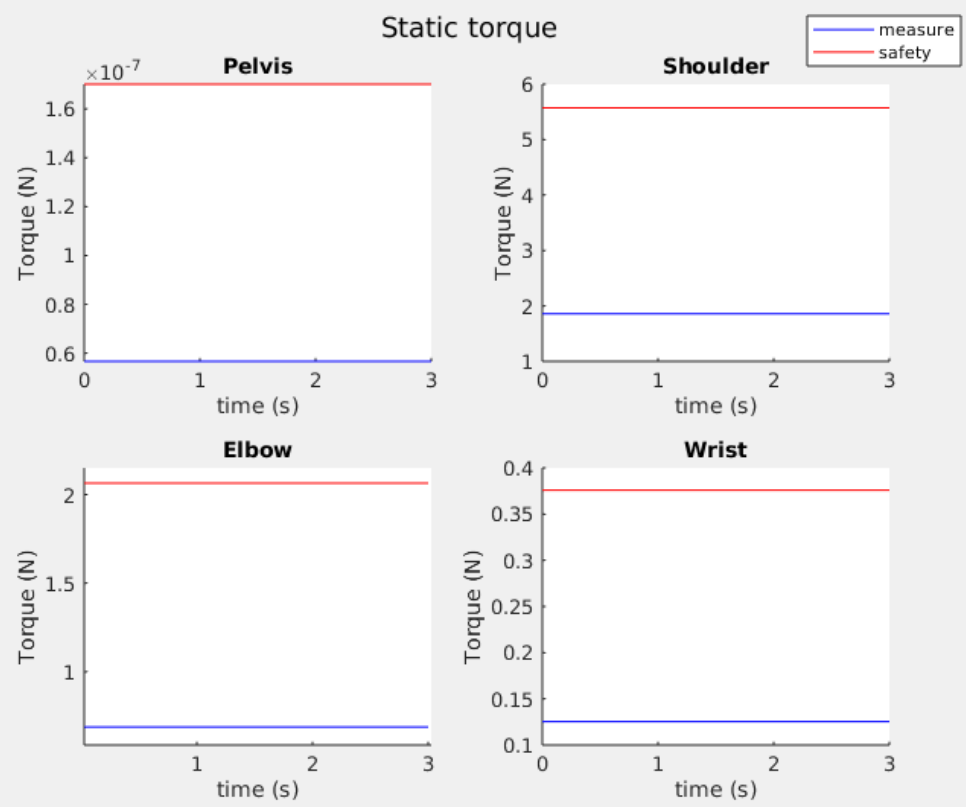


Figure 6.5: Static torque

These values could have been calculated manually. ONSHAPE allows finding the mass and the position of the center of gravity for a given set of parts. Thus, by retrieving the mass and the distance between the center of gravity and the link, we could have calculated the torque by assimilating the set of selected parts to a point of mass m at a distance d from the link.

$$C = m * g * d$$

Joint	**Mass (kg)**	**Distance (m)**	**Torque (N.m)**	**Matlab (N.m)**
Shoulder	0.852	0.222	1.8	1.8
Elbow	0.460	0.152	0.68	0.68
Wrist	0.165	0.78	0.13	0.13

Table 2: Static torque

It can be noted that the values calculated and measured via *Matlab* are almost identical. This also allows us to validate our measurement model and to make sure that we can rely on the values found with the simulation to choose the motors.

6.3 Joint movement

For each of the links, we conjectured the most "difficult" trajectory to achieve. We then tested it as well as different trajectories to make sure that we were right. The duration was also chosen according to the amplitude of the movement in order to meet the speed criterion defined in the specifications. We present here only the trajectory having presented the highest values. A margin of 30 % was then retained following the advice of our consultant to take into account the frictions as well as the possible differences between the simulation and the reality.

6.3.1 Pelvis

In the case of the pelvis, the motor needs the most power when the arm is stretched horizontally and rotates around the base. So we made the robot make a half circle in 2s to measure the torque, the power, and the rotation speed.

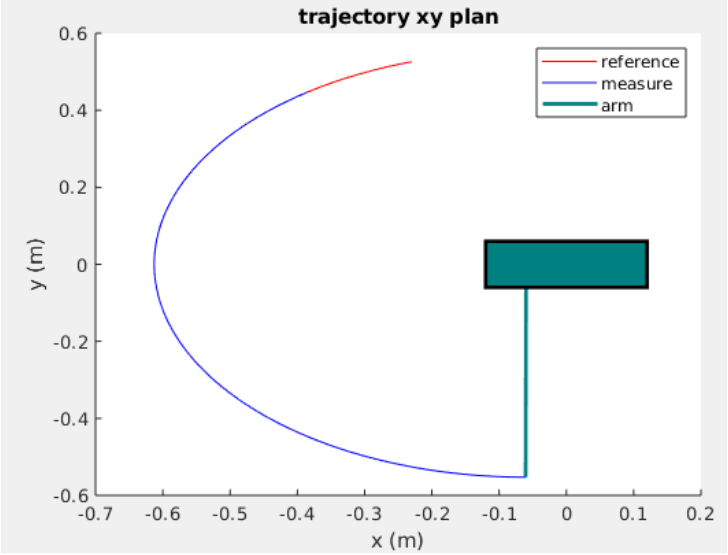


Figure 6.6: Pelvis trajectory seen from top

As we can see on the graph below, in this case, the maximum power is $0.15W$. For this power, the torque is $0.16N.m$ and the rotation speed is $0.95rad.s^{-1}$. By adding 30% we obtain a power of $0.2W$ and a torque of $0.21N.m$. The torque being superior to the one found to maintain the arm in static, finding a motor capable of delivering such power will be sufficient.

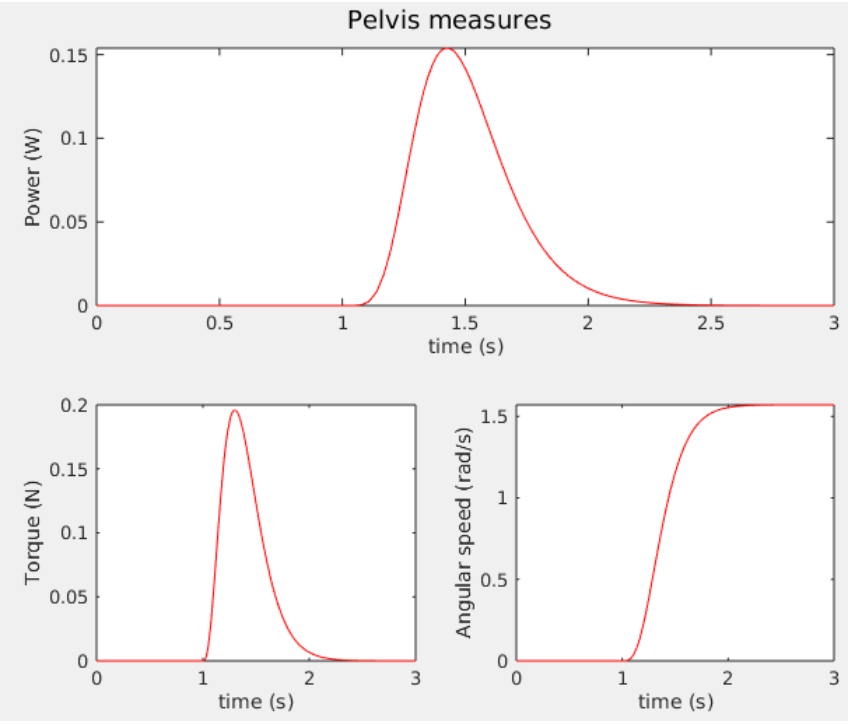


Figure 6.7: Pelvis characteristics

6.3.2 Shoulder

In the case of the shoulder, the motor needs the most power when the arm starts stretched horizontally and rises only around this axis. So we made the robot make a quarter circle in 2s to measure the torque, the power, and the rotation speed.

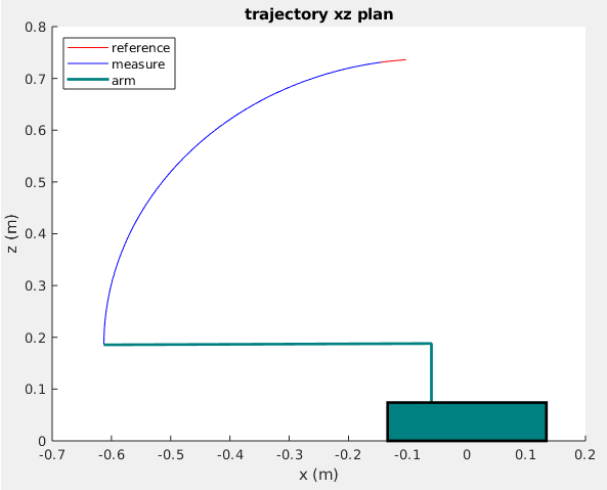


Figure 6.8: Shoulder trajectory seen from side

As we can see on the graph below, in this case, the maximum power is $1.41W$. For this power, the torque is $1.83N.m$ and the rotation speed is $0.77rad.s^{-1}$. By adding 30% we obtain a power of $1.83W$ and a torque of $2.34N.m$. The torque found is lower than the one needed to keep the robot static, so care must be taken to deliver both the power and the torque needed.

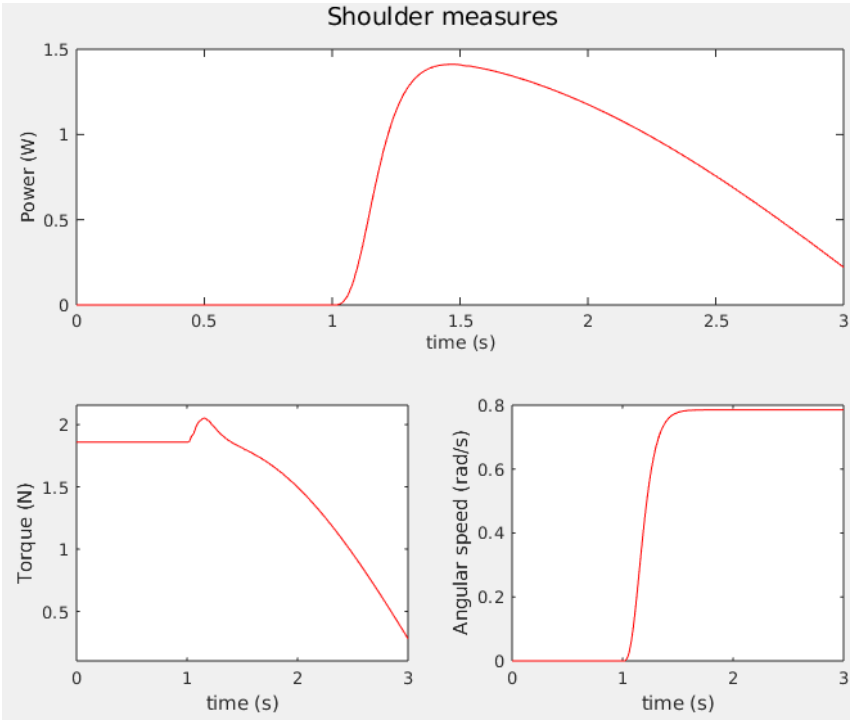


Figure 6.9: Shoulder characteristics

6.3.3 Elbow

In the case of the elbow, the motor needs the most power when the arm starts at 90 degrees at this link and rotates to a horizontal position. So we made the robot make a quarter circle in 2s to measure the torque, the power, and the rotation speed.

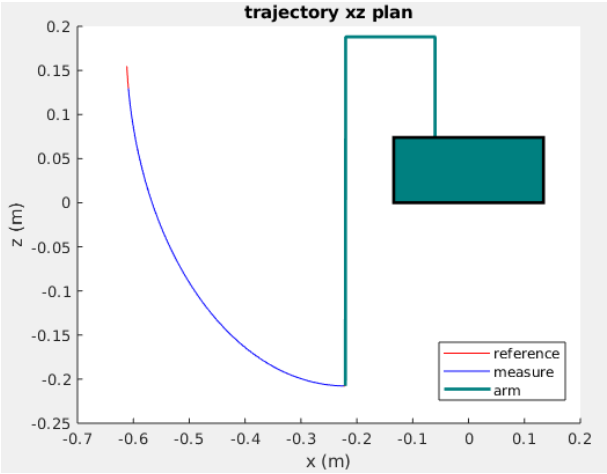


Figure 6.10: Elbow trajectory

As we can see on the graph below, in this case, the maximum power is $0.51W$. For this power, the torque is $0.66N.m$ and the rotation speed is $0.77rad.s^{-1}$. By adding 30% we obtain a power of $0.67W$ and a torque of $0.86N.m$. The torque found is lower than the one needed to keep the robot static, so care must be taken to deliver both the power and the torque needed.

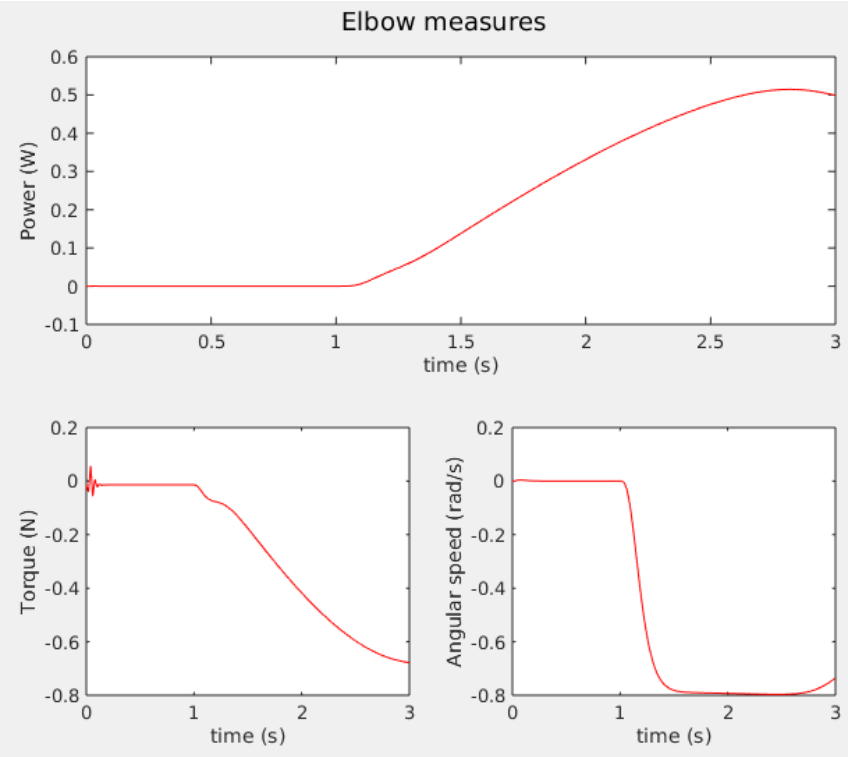


Figure 6.11: Elbow characteristics

6.3.4 Wrist

In the case of the wrist, the motor needs the most power when the arm starts at 45 degrees at this link and rotates around. So we made the robot make a quarter circle in 2s to measure the torque, the power, and the rotation speed.

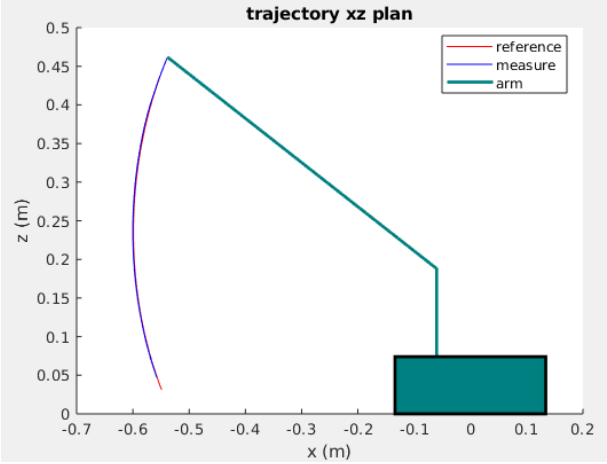


Figure 6.12: Wrist trajectory

As we can see on the graph below, in this case, the maximum power is $0.18W$. For this power, the torque is $0.19N.m$ and the rotation speed is $0.94rad.s^{-1}$. By adding 30% we obtain a power of $0.23W$ and a torque of $0.24N.m$. The torque found is lower than the one needed to keep the robot static, so care must be taken to deliver both the power and the torque needed.

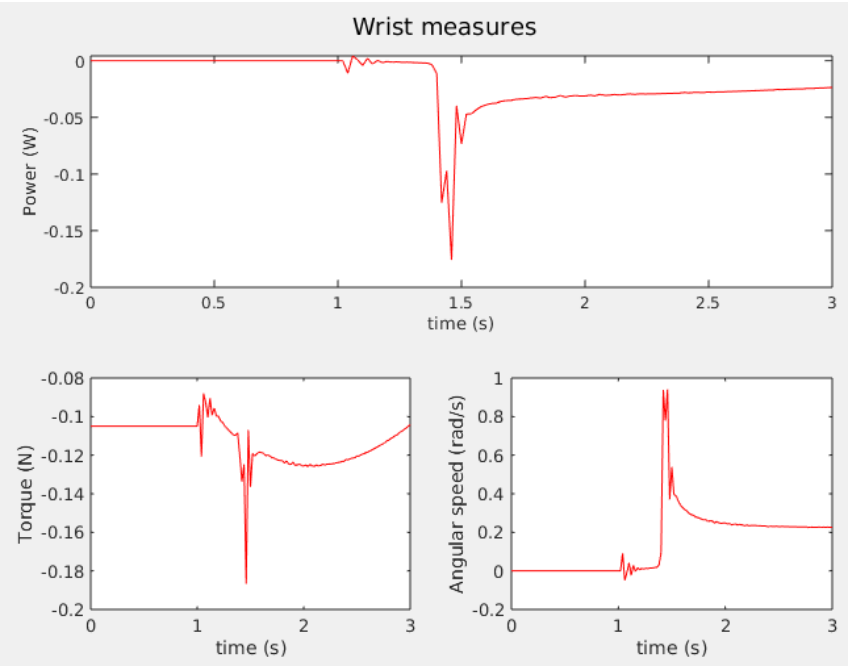


Figure 6.13: Wrist characteristics

6.4 Hardware choices

Just as we presented the results on the final robot for the torque, we also continue here. So the gears we will calculate will not be the ones we have mounted on the robot because we had

based ourselves on the intermediate model. To speed up the manufacturing of the robot, we decided to buy servomotors[21] so that we don't have to do any electronics. Dynamixel [Robc] motors were thus recommended. First, because they are widely used in robotics and very reliable but also because a ROS package has already been created with many examples to integrate them very quickly into our software. We have only selected the motors provided by this company by respecting two criteria: a reasonable price and the necessary characteristics respected by adding a gear if necessary.

Figure 6.14: Dynamixel servomotor principle

6.4.1 Pelvis

In view of the different models available at dynamixel, we have selected the following engine: **dynamixel XL430-W250**. The following graph shows the performance of the motor. In particular, in black, the maximum torque that can be delivered according to the speed of rotation of the motor. By multiplying the two we obtain the maximum power.

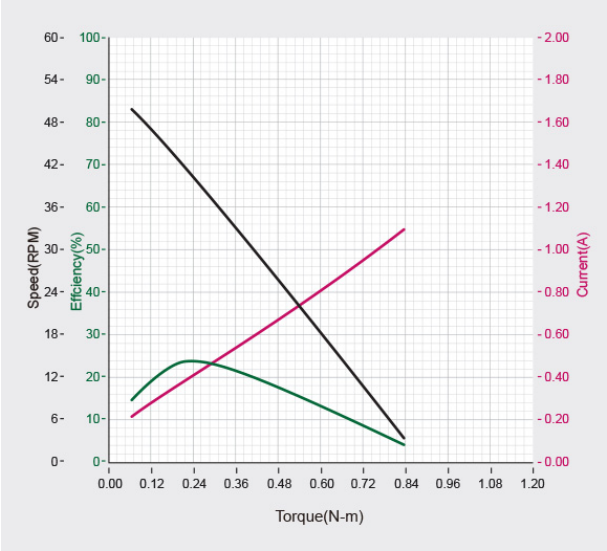


Figure 6.15: Pelvis motor performance

The necessary characteristics measured at the time as well as the one measured with the last version and the one of the engine are summarized below.

[21] A rotary or linear actuator that allows precise control of angular or linear position, velocity, and acceleration thanks to a position feedback sensor

Joint	Power (W)	Torque (N.m)	Angular Speed (rad/s)	Static torque (N.m)
previous need	1	4.2	0.1	0
actual need	0.2	0.2	0.9	1.4
motor	0.65	0.73	0.9	1.5

Table 3: Pelvis motor specification

The motor is able to deliver directly the necessary dynamic and static values. No gearbox is really necessary. However, at the time we made the measurements, because of a more approximate model and our less good mastery of *Matlab*, we had found much higher values. So we calculated that we needed a gear with a ratio of 1/7.

6.4.2 Shoulder

We have selected the following engine: **dynamixel XM430-W350**. The following graph shows the performance of the motor. In particular, in black, the maximum torque that can be delivered according to the speed of rotation of the motor. By multiplying the two we obtain the maximum power.

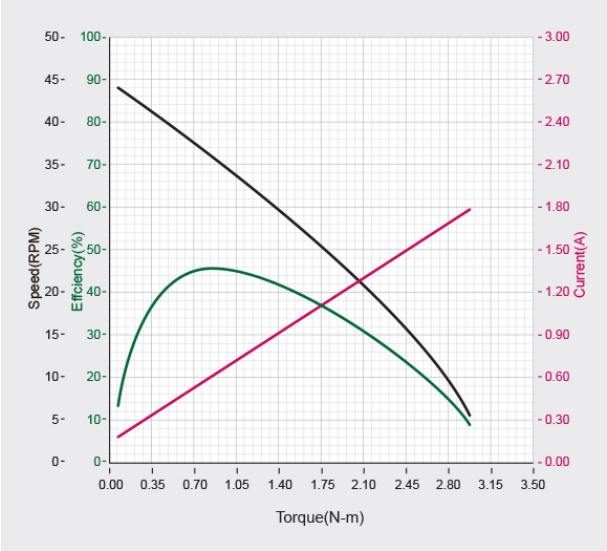


Figure 6.16: Shoulder motor performance

The necessary characteristics measured at the time as well as the ones measured with the last version and the one of the engine are summarized below.

Joint	Power (W)	Torque (N.m)	Angular Speed (rad/s)	Static torque (N.m)
previous need	2.8	7.6	0.4	5.2
actual need	1.8	2.4	0.77	5.1
motor	2.3	2.87	0.83	4.1

Table 4: Shoulder motor specification

The motor is able to deliver directly the necessary dynamic values. However, the static torque is not sufficient and a gearbox will then be necessary with a ratio $r = \frac{4.1}{5.1} = 1.3$. At the time we made the measurements, we had found higher values. So we calculated that we needed a gear with a ratio of 1/4.8.

After mounting the robot, we measured the percentage of the maximum torque used by the motor to make the same trajectory. We realized that we are not even using 10% of the maximum value. This confirms that the simulation results on the last prototype are much closer to reality and that we could replace the present gears.

6.4.3 Elbow

We have selected the following engine: **dynamixel XC430-W150**. The following graph shows the performance of the motor. In particular, in black, the maximum torque that can be delivered according to the speed of rotation of the motor. By multiplying the two we obtain the maximum power.

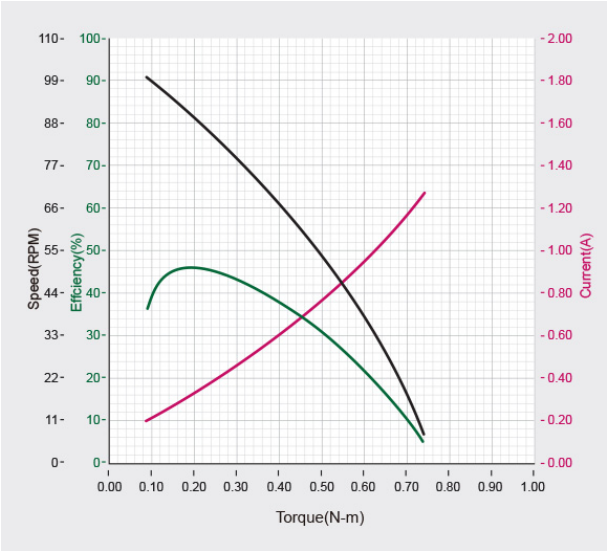


Figure 6.17: Elbow motor performance

The necessary characteristics measured at the time as well as the ones measured with the last version and the one of the engine are summarized below.

Joint	Power (W)	Torque (N.m)	Angular Speed (rad/s)	Static torque (N.m)
previous need	2.26	2.4	1	1.86
actual need	0.7	0.86	0.78	2.07
motor	1.2	0.7	0.83	1.6

Table 5: Elbow motor specification

The motor is able to deliver directly the necessary dynamic power. However, the static and dynamic torques are not sufficient and a gearbox will then be necessary with a ratio $r = \frac{2.07}{1.6} = 1.3$. The dynamic torque needs a ratio of $r = \frac{0.86}{0.7} = 1.23$, a ratio of 1.3 is then perfect. At the time we made the measurements, we had found higher values. So we calculated that we needed a gear with a ratio of 1/5.

After mounting the robot, we measured the percentage of the maximum torque used by the motor to make the same trajectory. We realized that we are not even using 20% of the maximum value. This confirms that the simulation results on the last prototype are much closer to reality and that we could replace the present gears.

6.4.4 Wrist

In view of the different models available at dynamixel, we have selected the following engine: **dynamixel XL430-W250**. The following graph shows the performance of the motor. In particular, in black, the maximum torque that can be delivered according to the speed of rotation of the motor. By multiplying the two we obtain the maximum power.

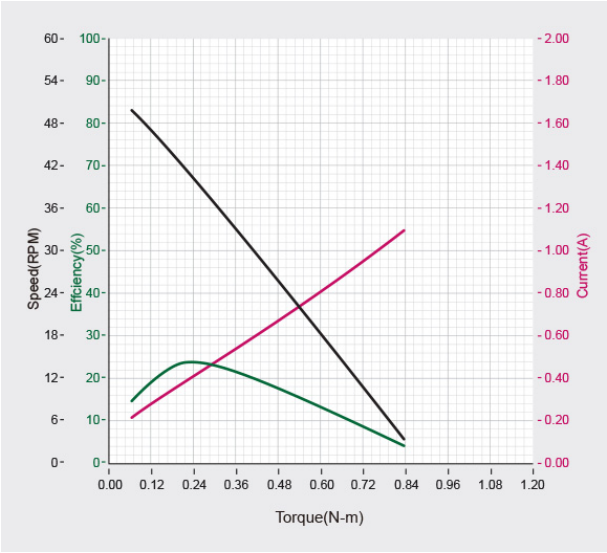


Figure 6.18: Wrist motor performance

The necessary characteristics measured at the time as well as the ones measured with the last version and the one of the engine are summarized below.

Joint	**Power (W)**	**Torque (N.m)**	**Angular Speed (rad/s)**	**Static torque (N.m)**
previous need	0.36	1.04	0.35	0
actual need	0.23	0.24	0.95	0.38
motor	0.65	0.73	0.9	1.5

Table 6: Wrist motor specification

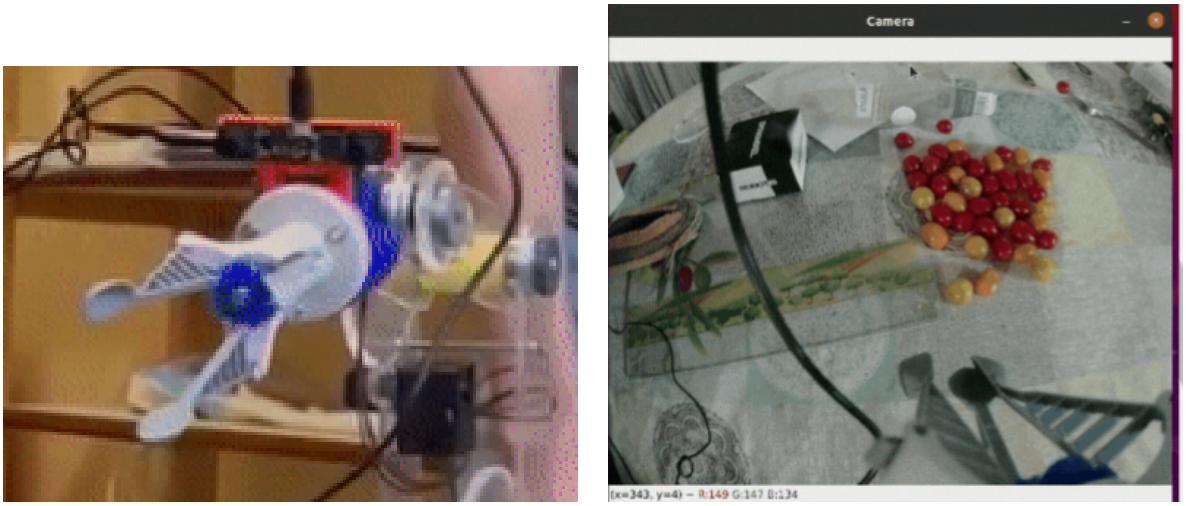
The motor is able to deliver directly the necessary dynamic and static values. No gearbox is really necessary. However, at the time we made the measurements, because of a more approximate model and our less good mastery of *Matlab*, we had found much higher values. So we calculated that we needed a gear with a ratio of 1/2.2.

7 Joystick Control

7.1 Principle

The initial request of our customer was a remote-controlled arm. This was the easiest and fastest solution to implement. Mr. Kedziora also wanted to avoid using artificial intelligence in order not to lengthen the prototype manufacturing time. Despite our fears about the difficulties that this type of order can pose, he really insisted that we use this method, which is the first one that we have put in place.

In this configuration, a camera, fixed on the arm at the level of the hand, allows the user to have a video return. An Xbox controller in our case is used to control the arm. The commands sent by the user are done in the camera's frame.



(a) Position (b) View

Figure 7.1: Camera

The control of the three axes in the camera frame is done with the two joysticks on the controller. The left joystick allows to move according to the plane (x,y) of the camera (see 4.1 and the right joystick manages the depth. This movement is then transformed into a command to move along the desired axis. Some features listed below have also been added to the buttons to facilitate the work of the user and the proper functioning of the arm.

- authorize the movement: button A
- prohibit the movement: button B
- return to zero position : button X
- return to deposit position : button Y

Figure 7.2: Xbox Controller

As explained in the previous section, the inverse kinematics calculates the necessary angles for a given position in the base frame {s}. However, it seemed more natural to us to make the user work in the camera reference frame {c} and then to change the reference frame. For this, we reused the modern robotics library. In the same way that we can calculate the transformation matrix between the base and the end effector from the position of the links, we can calculate the one between the base and the camera.

$$T_{sc} = \prod_{n=1}^{4} e^{[S_i]\theta_i} M_c$$

Then the relation between the position in the reference frame and the position in the reference frame is the following:

$$p_s = T_{sc} p_c$$

Attention, T_{sc} being a 4x4 matrix, p_s and p_c are homogenous coordinates[22]:$p = [p \ 1]$. At each command sent by the user, we retrieve the position of each of the links and then we apply the following code that puts into practice the previous equations. We obtain the position in the base frame.

```
    # import kinematics parameters
    from parameters import *
    import modern_robotics as mr
    # receive angles and desired position from ros
    thetalist = get_angles()
    p_c = get_position()
    p_c = np.array([p_c 1])
    # get transformation matrix and change the position
    t = mr.FKinSpace(m_c,screw_list,thetalist)
    p_s = np.dot(t,p_c)
    p_s = p[:3]
```

The instruction sent by the user is added to an offset. Indeed, we want to control the position of the center of the hand. This one is fixed in the camera frame since there is no motor link between the camera and the hand. Thus, we can calculate the position of the center of the hand

[22] 4×1 vector representing coordinates x, $[x1]$

in the camera frame {c} when the robot is in its zero configuration. This offset will then be added to the command. An absence of command will not correspond to the position (0,0,0) but to the position of the end effector in camera frame so that the robot does not move. The calculation of this offset is done in the following way. The transformation matrix from the base of the camera M_c and of the end effector M_e in the zero configuration have been given in section 4. By multiplying these two matrices, we can obtain the transformation between the end effector and the hand T_{ce}. Then in the same way as we changed the reference frame from camera to base, we can pass the point (0,0,0) in the end effector frame into the camera frame.

$$T_{ce} = T_{cs}T_{se} = T_{sc}^{-1}T_{se} = M_c^{-1}M_e$$
$$p_c = T_{ce}p_e$$
$$\text{offset} = T_{ce} \cdot [0\ 0\ 0\ 1]^T$$

```
    # import kinematics parameters
    from parameters import *
    import modern_robotics as mr
    # receive angles and desired position from ros
    p_e = np.array([0, 0, 0, 1])
    # get transformation matrix and change the position
    t_ce = np.dot(np.linalg.inv(m_c),m_e)
    translation = np.dot(t,p_e)[:3]
```

7.2 Open loop

In this section, we only present the operation and results of the open loop using the tools explained above. The implementation of each block as well as the information exchanges are done with the help of ROS. Everything is detailed in the dedicated part.

The user sends a desired position with the joystick, a change of reference frame is performed and then the necessary angles are calculated. The diagram above explains the different blocks used and the information exchanged between them. The camera block is separate because it does not really intervene in the loop. It is just used to obtain a video return.

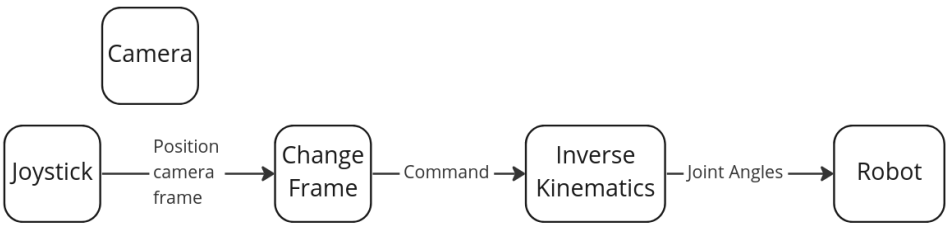
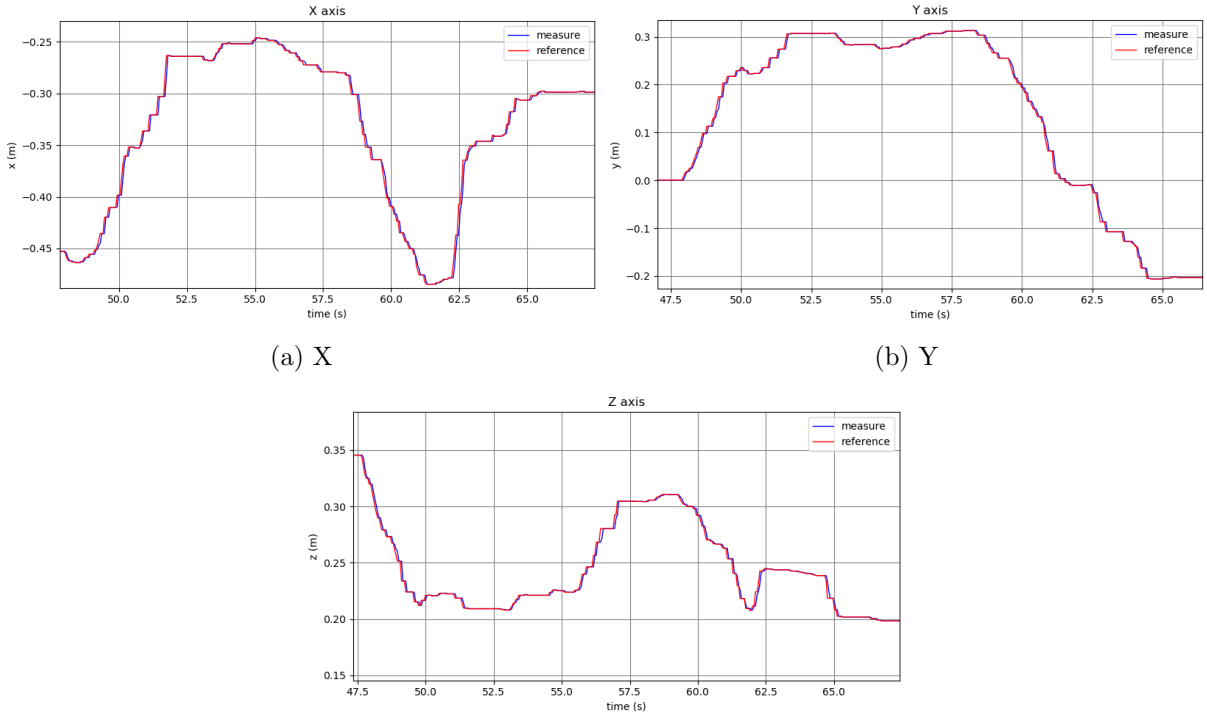


Figure 7.3: Open loop schema

To check the correct operation, we have made our arm follow a random trajectory using the joystick. As we can see on figure 7.5, the robot follows well the requested trajectory along the 3 axes.



(a) X

(b) Y

(c) Z

Figure 7.4: Measure in base frame for random trajectory

After visually checking the correct operation of the control, we subjected the robot to a step function[23] according to each axis of the base. Even if we control the joystick in the camera frame, it is transformed and expressed in the base frame. We will see later, the corrector will be applied to this command, that's why we study the open loop in this reference frame. We measured the error, response time, and overshoot. The results are detailed below. All the results presented were obtained in simulation. We did not buy a sensor to measure the position of the real robot. We have therefore based ourselves on the simulation results and on the observations of the arm movements to validate the control.

Axis	**Step (mm)**	**Time response (s)**	**error (mm)**	**Overshoot (mm)**
X	50	0.24	1.4	7.6
Y	50	0.5	0.2	0
Z	50	0.2	0.5	0

Table 1: Open loop result

[23] A discontinuous function, whose value is zero until x reaches a certain value t and a given value y when x is greater than t

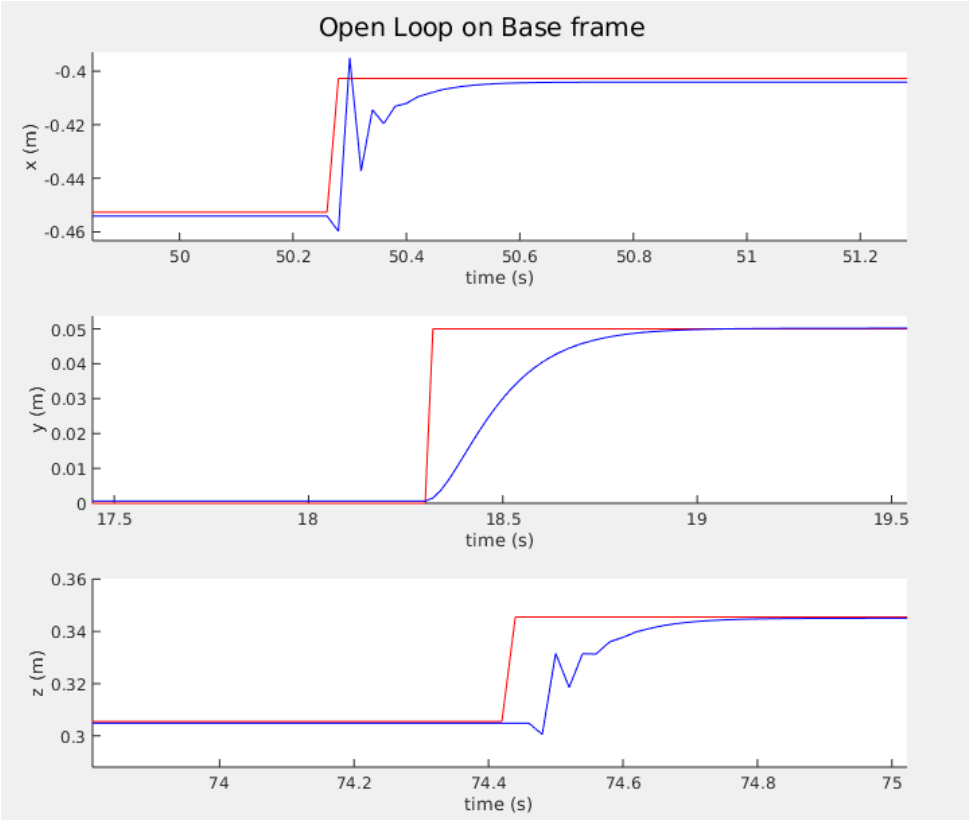


Figure 7.5: Open loop response in base frame for step input

Using the joystick, the user gives an approximate position and corrects the desired position thanks to the camera feedback. As the results show, the open loop is accurate and allows to follow a position. Thus, it was not necessary to set up a closed loop on the real robot. We thus avoided buying a position sensor and saved development and delivery time.

7.3 Closed loop

However, we still servoed the robot in position in simulation. To stick to reality, we have modified a bit the robot in simulation compared to the reference robot used. Thus, we were closer to a real case and we were able to test a positional servoing using a proportional integral corrector. All this was done in simulation and was never tested on the real robot because we didn't have the time, we didn't have a sensor and the open loop was enough for a joystick control.

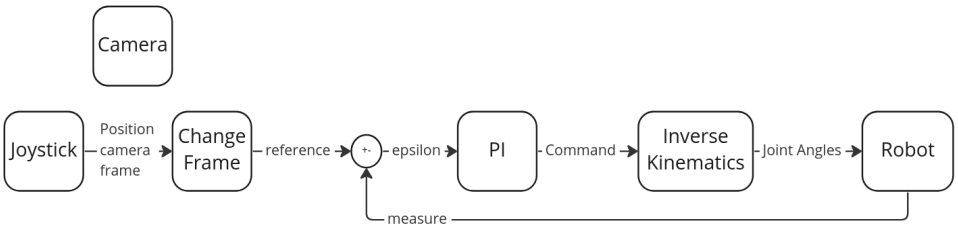


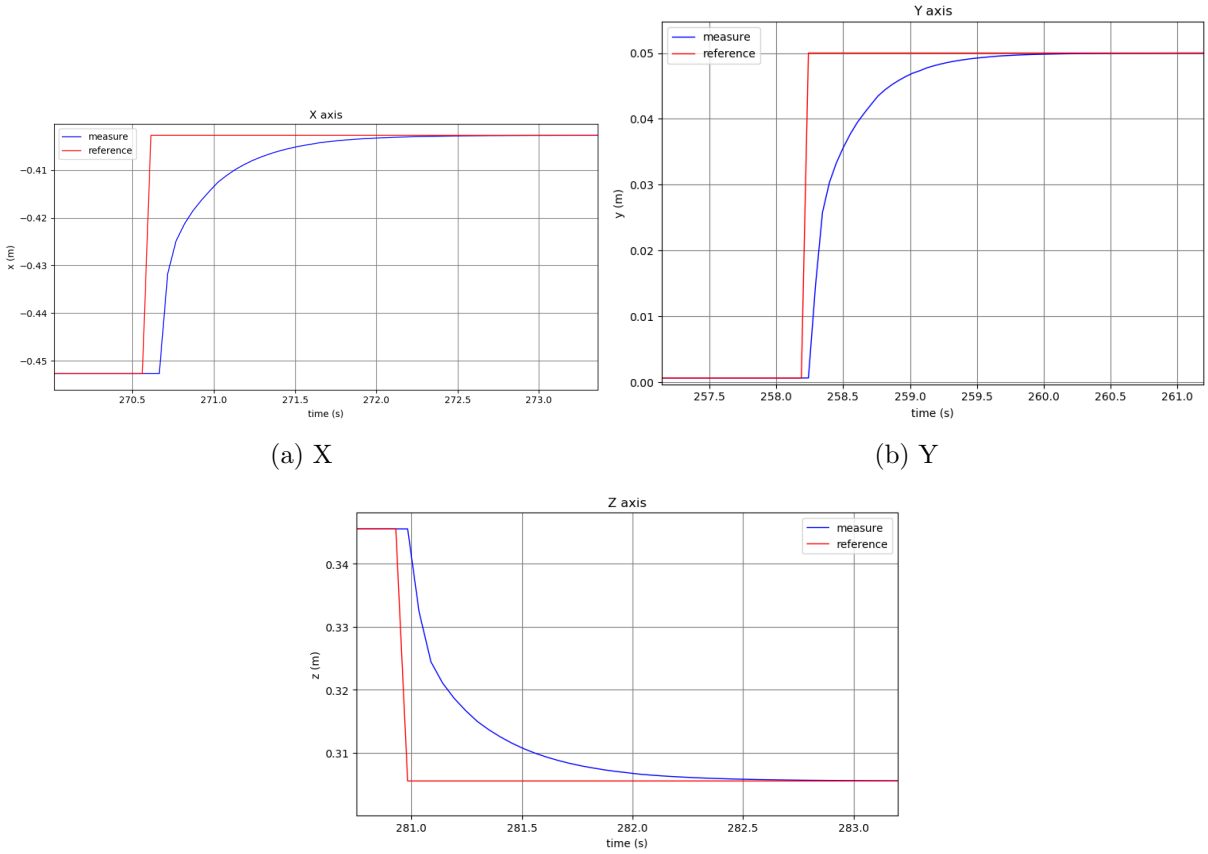
Figure 7.6: Closed loop Schema

For the corrector, we have chosen a proportional integral to suppress the error and the possible overshoot. For the adjustment, we did not calculate a theoretical value. We started with a value that works $k_p < 1$ and $k_i < 1$ to avoid that the solution diverges. Then we varied

the parameters to increase the time response[24] while avoiding oscillations and overshoot. An anti-windup[25] was also implemented at the output of the corrector to avoid that the control diverges. Everything was done using python. Even if it is faster to implement the PID using *Matlab*, the deployment on ROS is longer. Thus, python allowed us to test different values for the corrector parameters more quickly. Finally, we subjected the arm to the same step as for the open loop and we obtained the results listed in the table and figure below.

Axis	**Step (mm)**	**Time response (s)**	**error (mm)**	**Overshoot (mm)**
X	50	1.3	0	0
Y	50	1.2	0	0
Z	50	1	0	0

Table 2: Closed loop result



(a) X

(b) Y

(c) Z

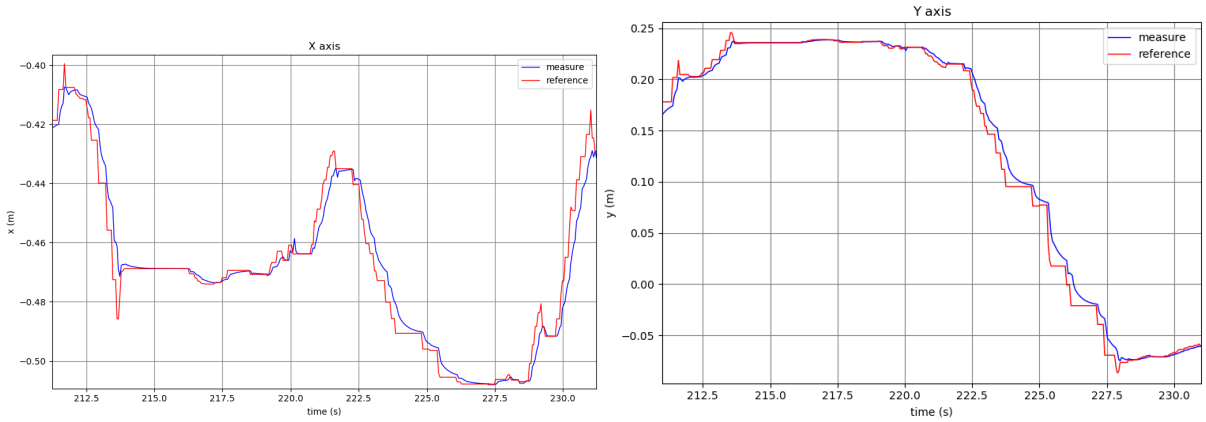
Figure 7.7: Closed loop response in base frame for step input

As we only realized this part in simulation, we did not really insist on finding the best parameters to optimize the response time. The objective here was rather to set up a closed loop and to show that it works. In view of the results, we can say that it is the case and in the case where we would like to set up a corrector on the real robot, it will be enough to spend time to find the good parameters.

[24] Also called setting time, it is the time required for the response curve to reach and stay within a range of a certain percentage of the final value

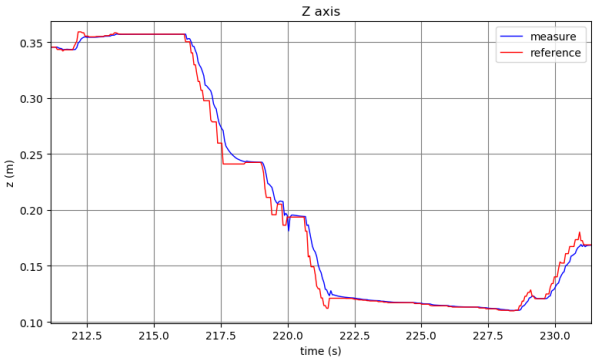
[25] The situation in a PID feedback controller where a large change in setpoint occurs (say a positive change) and the integral term accumulates a significant error during the rise

Finally, we have also studied the behavior of the arm when we control it with the Xbox controller in the camera's frame. We can see that the error when we stop moving the joystick is zero. However, since the response time is higher, the tracking is a little slower.



(a) X

(b) Y



(c) Z

Figure 7.8: Measure in base frame for random trajectory with closed loop

8 Vision Controle

The joystick was an explicit request from our client that he wanted us to implement. However, he wanted to relocate the workforce from the United States to Asian countries such as the Philippines. We, therefore, warned him about possible problems, especially the latency time for such a system. Indeed, the distance between Manila (the capital of the Philippines) and Los Angeles is 11,759 km. Assuming that we have a cable connecting them, which is very unlikely, the information can travel at the speed of light. A command will therefore take $\frac{11759}{3.10^5} = 0.04s$. The duration between the sending of a command and the video return would be 0.08s which is considerable and would make the operation very complicated.

We then looked for solutions that did not involve artificial intelligence to address this problem. However, since we still had to develop a solution using a joystick, we ran out of time. It is nevertheless a promising solution. To implement these solutions, we used the python openCV library[Vil19]. We trained ourself thanks to a python book and youtube channels.

8.1 3D position

Our first one consisted in making the user select the desired tomato and then estimating its 3D position using a camera. We kept the user without the need to send the order live.

The depth cameras being relatively expensive, we decided to buy only a stereo camera and to build a software able to determine the depth. All this process is done in the camera frame. It is a camera with two lenses that can retrieve two synchronized images.

Using the youtube channel and the GitHub page of *Nicolai Nielsen*, we calibrated the lenses to standardize the images received. Once this was done, we were able to calculate the disparity[26]. OpenCv has a class that allows to perform such a calcul for a calibrated stereo camera[27]. Some parameters needed to be tune, and we did it thanks to sidebar until we considered we obtained the best disparity image. The code can be found on appendix E.

(a) Disparity

(b) Camera view

Figure 8.1: Stereo camera image

[26]The apparent motion of objects between a pair of stereo images
[27]A camera with two or more lenses with a separate image sensor or film frame for each lens

The figure below shows an image and its disparity. You can see that the tomato stands out a little, in a white mass. We can also note that the farther the elements on the image are, the darker they appear on the disparity image. There is indeed a correlation between disparity and distance. These are inversely proportional. To find the coefficient, we then took photos of an object at different distances known then we calculated the disparity. We then obtain the curves below which give the relationship between depth and disparity.

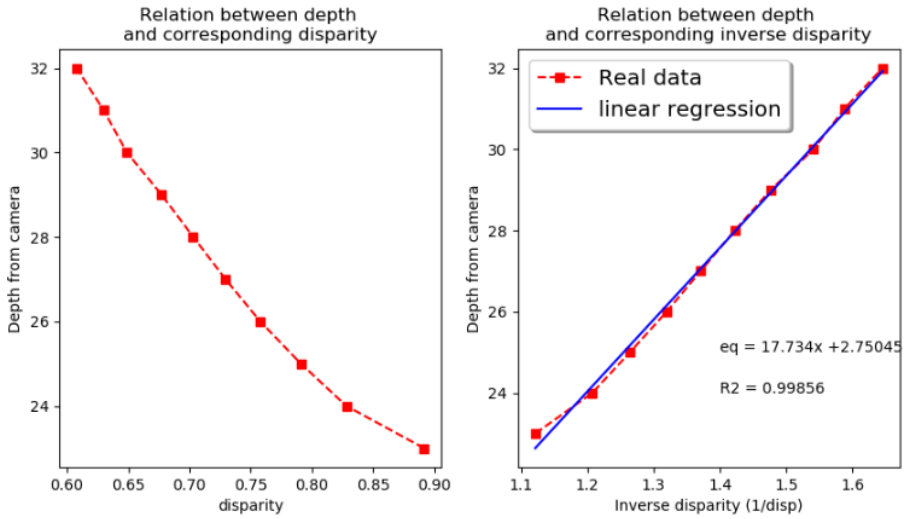


Figure 8.2: Disparity depth relation

Similarly, there is a link between depth and size of a pixel. Indeed, the closer an image will be, the more it will take a large place in the image. Once we were able to obtain the distance along the z axis, we wanted to calculate the position in x,y. We then sought the relationship in depth and size of a pixel. We have measured the size in pixel of the same object of known size for different depths and then we have drawn the curve. A polynomial of degree 3 seems to stick to the curve on the desired interval (10-40cm).

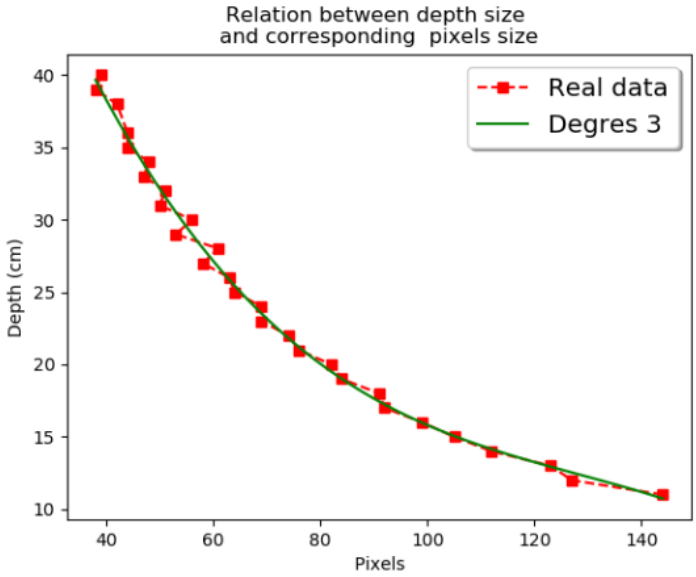


Figure 8.3: Depth pixel size relation

The center of the image corresponds to the position (0,0). The user clicks on the center of

the tomato, to get the central pixel. Using the first relation we are able to provide the distance to the camera. Then we calculate the pixel distance between the center and this point and with the second relation we calculate the position in m. We then obtain the 3D position of the center of the tomato. We then calculate the position in the base frame and the inverse kinematics as shown in the diagram below.

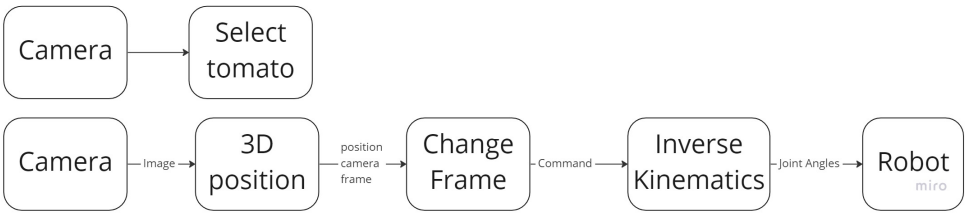


Figure 8.4: 3D position schema

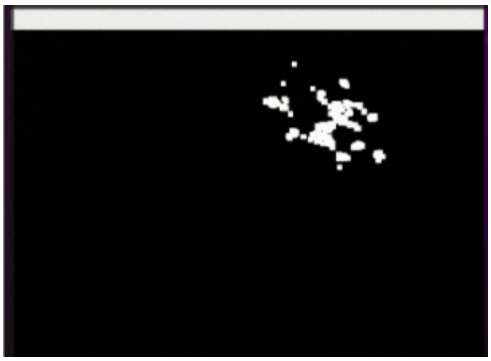
Unfortunately, our solution was not precise enough. Setting up the disparity calculation with openCV would have required more time and for the same tomato, we could vary by several centimeters.

8.2 Tracking

The second idea consisted in object tracking. As in the previous solution, the user selects the tomato and a software developed with openCV allows to track it.

(a) Camera view

(b) no tracking



(c) tracking

Figure 8.5: Computer tracking view

Once the tomato has been selected, a filter is applied to our image: only elements similar to

the one chosen are kept. We can see that the chair in front of the camera disappears when the tracking is activated. Visually, a square highlighting the selected tomato appears on the image too.

To transform this into a command, the green dot on the image in the center of the met corresponds to the lens. We must therefore center the tomato on this point. To do this, we calculate the distance in pixels between the two and we apply a command proportional to this distance on the xy plane in the camera frame. Once the tomato is centered, we can move the arm to the tomato. The overall operation is summarized in the diagram below.

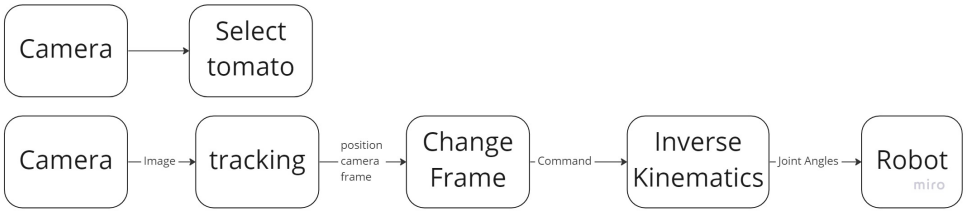


Figure 8.6: Object tracking schema

Two problems then arose for us. First, our tracking system was still relatively sensitive and the arm had to move slowly at the risk of losing the tracking. Finally, we had not yet solved the question of how to stop the arm once it moves forward centered on the tomato.

9 ROS

For our project, we used *ROS*1 noetic [Robb].

9.1 Definition

As the name suggests, *ROS* (Robot Operating System) is an operating system for robots. Like operating systems for PCs, servers, or stand-alone devices, *ROS* is a complete operating system for service robotics.

It is composed of a set of open-source software[28] allowing to develop software for robotics. *ROS* is a meta operating system, something between the operating system and the middleware.

ROS is therefore positioned as a facilitator of robotics projects. Researchers or engineers in R&D departments no longer spend time creating a new ecosystem for each new robotics project. *ROS* has an accelerating effect on R&D by reducing costs and time to market. It is therefore very interesting for the rapid development of a prototype

Master: The master is a *ROS* process that helps nodes locate each other and establish communication channels based on their publisher subscriber relationships and any services. The master also manages the parameter server. The master is typically started through the roscore command-line tool, or it is automatically started via a roslaunch call.

Topic: A topic is a many-to-many information transport system. It is typed, it is necessary to specify which type of messages we transport and based on the system of subscription / publication

Node: A node is an instance of an executable; it can be linked to an engine, a sensor, or purely software. A node can post messages to a topic or subscribe to messages on a topic. They are independent processes, the crash of a node does not crash the whole *ROS* (Notion of micro-kernel on which *ROS* is based)

Launch file: Launch files are very common in *ROS* to both users and developers. They provide a convenient way to start up multiple nodes and a master, as well as other initialization requirements such as setting parameters.

Package: A collection of source code, configuration files, and other resources that implements functionality in *ROS*. By dividing related functionality into packages, it can be shared with others and reused across projects. All packages must have a package.xml file defining their manifest.

Workspace: A workspace is a folder where you modify, build, and install packages.

9.2 Training

Following the advice of our consultant, we quickly understood the advantage of using *ROS*. For that, the members of the automatic pole followed training to understand how to use it, understand the principle, the functioning, and the possible applications. Fortunately, the site offers many tutorials to learn through practice. This training was necessary and should be done as soon as possible to become familiar with totally foreign tools. We did everything using python because we had developed everything with it.

[28] A software with source code that anyone can inspect, modify, and enhance

This training on *ROS* is over. We knew how to create a workspace, packages, and nodes. We also decided to link *Matlab* and *ROS* to accelerate the development. So we followed the tutorials proposed by *Matlab*[Matb] for the use of *ROS* with *Simulink* and the development of a package from the model *Simulink*. This allowed us to accelerate the tests but especially the handling of *ROS* because the commands were relatively automated by *Matlab*. However, we were less free for the package creation. Indeed, once the model is generated, it is difficult to directly modify the package code. You have to go back to the *Simulink* model and regenerate the package. Once we understood how python packages work, we limited the export of *Simulink* to packages to use more python nodes which are more easily and faster to modify. We still continued to use the connection between *ROS* and *Matlab* for testing.

9.3 Packages

For this project, we decided to separate the tools into categories. Four packages have been created and one is from an existing *ROS* package.

- **arm**: It contains all the nodes concerning the arm (forward kinematics, change the frame, joystick conversion). It also contains all the information about the arm (kinematics parameters, rviz file, gazebo file, stl file) allowing the simulation with *ROS* only.
- **inverse kinematics**: The inverse kinematics having been generated from *Simulink*, constitutes a separate package containing only the inverse kinematics node.
- **motors**: It contains all the codes related to the motors. The node listens to the instructions and sends them to the engines.
- **camera**: It contains all nodes related to the camera (webcam display, object tracking, object detection)
- **joy**: It allows us to get Xbox controller data.

9.4 Topics

We list here all the topics that will be used in the command chain. Be careful, inside the nodes, they can have different names to be more generic and reusable in other projects without having to modify them. But as we will see, we can change the name of these topics at the time of launch to give them a new value linked to the project. It is therefore this name that we make explicit.

- /Joystick/ref/joy : [Joy (sensor_msgs)] joystick movement and button pressed
- /Camera/ref/position : [Point (geometry_msgs)] command position in the camera frame
- /Robot/ref/position : [Point (geometry_msgs)] command position in the base frame
- /Robot/state/joint : [JointState (sensor_msgs)] position of each joint
- /Robot/state/position : [PoseStamped (geometry_msgs)] end effector position in the base frame
- /Robot/state/reference : [PointStamped (geometry_msgs)] end effector target position in the base frame
- /EndEffector/state/position : [Point (geometry_msgs)] end effector state position in the base frame (forward kinematics)
- /Motors/state/current : [JointState (sensor_msgs)] current delivered by each motor
- /Motors/state/present_position : [JointState (sensor_msgs)] real position of each motors

9.5 Nodes

To start a node, you have to source *ROS*, source the workspace that contains the package if this has not been done in the terminal, and then launch the node.

```
~$    source /opt/ros/noetic/setup.bash
~$    roscore
~$    rosrun <package_name> <node_name>
```

For all python nodes, a class has been created. It allows to initialize the node and the variables used but also indicates the name, the frequency, and the topics to subscribe to and publish. An update function is also present which takes care of publishing and listening at the requested frequency. An example is given in the appendix F.

9.5.1 Forward kinematics

The forward kinematics worked very well with python with results equivalent to $10^{-4}m$ compared to the *Simulink* model. So we started from this file to create the node. This node is part of the arm's package.

It subscribes to the topic /Robot/ref/joint and publishes /EndEffector/state/position. The calculation is done exactly the same way as explained in part 5.2

9.5.2 Inverse kinematics

The inverse kinematics node was created from *Simulink*. The python model did not provide any results, so it was easier to use the code generator of *Matlab*.

The node subscribes to the topic */Robot/ref/position* and publishes the topic */Robot/ref/joint*. It publishes the x,y,z coordinates of the desired position. The output topic publishes a list containing the position of each link as well as another one with their names and a header indicating the time.

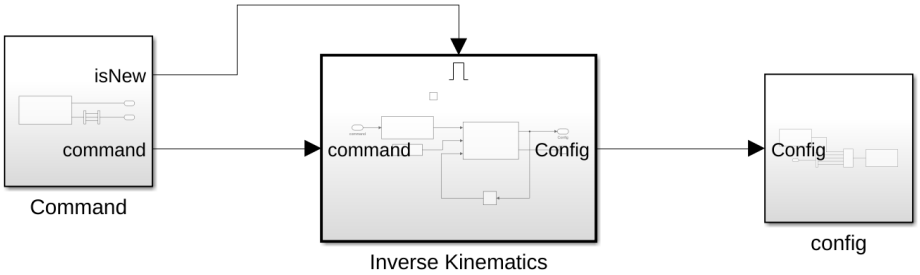


Figure 9.1: Inverse kinematics node

To connect to *ROS* with *Matlab* you will need to configure *ROS* Network address. You can access this in the **Simulation tab** by selecting **ROS Toolbox** > **ROS Network**. You will need to set :

- ROS Master
- Network Adress

To run files that contain *ROS* block, you need a *ROS* master. You can run it on a terminal with the following command :

```
~$    source /opt/ros/noetic/setup.bash
~$    roscore
```

The *ROS* folder contains files that are generated by *Matlab* to create *ROS* package from a *Simulink* file. To generate them you need set parameters. You can access this in the **modeling tab** > **Model Settings** > **Hardware Implementation** > **Hardware board settings** > **Target hardware resources** > **Groups** > **Build Options**. You need to set :

- Device Adress (127.0.0.1 if localhost)
- Username
- password
- Ros folder (already set)
- path to catkin workspace

Then you just have to generate the files. It will create :

- build_ros_model.sh
- < model_name > .tgz

From the folder where are the files, you can now create your package with the following command. The package and the node will have the same name as the *Simulink* model.

```
~$    ./build_ros_model.sh <model>.tgz <catkin_workspace_path>
```

9.5.3 Joystick

To get the information from a controller, we used an existing package available on the *ROS* wiki site. This one allows getting the joysticks movements and the preset buttons for any controller. Only the order of the information can change depending on the controller.

In our case, we used an Xbox controller which is the one presented in the package. Then we created a node that converts the actions of the controller into the robot's command. The buttons used have already been specified in section 6.1 as well as the use of joysticks. The joysticks take values between -1 and 1 depending on the inclination. For a smooth movement and to avoid too big jumps between two topic sends (100Hz), we multiply this value by 0.03. Thus, the robot can move only 3cm along each axis between two sends.

9.5.4 Change frame

As explained in section 6.1, the command is sent to the camera frame. It is then necessary to change the reference frame using the formula and the code seen previously.

The node listens to the topics */Robot/state/joint* and */Camera/ref/position* in order to calculate the position in the base frame. It then publishes the topic */Robot/ref/position*. This one is sent only if there is a new command.

9.5.5 Motors

In addition to the advantages mentioned above, there is a Dynamixel python library allowing you to send commands and to get information from the motors. A *ROS* package also exists, which allowed us to start from this one to create our own. All the examples can be found on the Dynamixel website. (dynamixel SDK and Dynamixel workbench). The *ROS* examples are in C++ but are easily transferable to python.

The node for motors allows to transform the value of each link into a motor command. It listens to the node /Robot/state/joint. Then it converts this value according to the present reducer and passes it to the indicated unit of the motors.

The motor angles are not in radian, the documentation indicates that they can take values between $\pm 1,048,575$ and that one turn (2π) corresponds to $4,095$. At startup, the robot must be in its zero configuration. We then get the value of each motor at time zero which is then used as an offset for the command. We also multiply the desired position by a variable called orientation worth ± 1 to ensure the correct direction of rotation of the motor.

To make sure that the robot is working properly and that it will not break, the node also publishes the current and the position of each motor. These are obtained directly via functions of the dynamixel library.

Below are the two functions that control the robot and publish the current and position of the motors.

```
    def getGoalPositions(self):
        """ transform joint position into a motor position"""
        pos = self.joints*self.gearRatio
        pos = pos*4095/(2*math.pi)
        pos = self.initPosition+self.orientation*pos
        self.goalPositions = pos.astype(int)

    def actuate(self):
        while not rospy.is_shutdown():
            # write position
            self.getGoalPositions()
            self.motors.write_position(self.goalPositions)
            # read and publish current and position
            self.present_position = self.motors.read_position()
            self.present_current = self.motors.read_current()
            self.publish_current()
            self.publish_present_position()
            self.rosRate.sleep()
```

9.5.6 Camera

As it is, the camera node is a bit special. It doesn't listen to and publish any topic but only displays a camera return from a webcam on the computer. Pour cela, nous avons utilisé la bibliothèque openCV

9.6 Launch files

Each of the created launch files allows launching a complete function of the robot. Their content is described as a block diagram. A square represents a node and an arrow a topic.

To launch a launch file, you have to source *ROS*, source the workspace and launch it :

```
~$    source /opt/ros/noetic/setup.bash
~$    source /catkin_ws/devel/setup.zsh
~$    roslaunch <package_name> <file>
```

9.6.1 Rviz

rviz is a 3D visualizer for the Robot Operating System (*ROS*) framework. It is what we use to simulate our robot and visualize the movement.

Figure 9.2: Rviz launch file

It gives the following screen which is a simulation of the robot. Thanks to that we can get the position of the joints and with the forward kinematics, we can compute the position of the end effector in each frame.

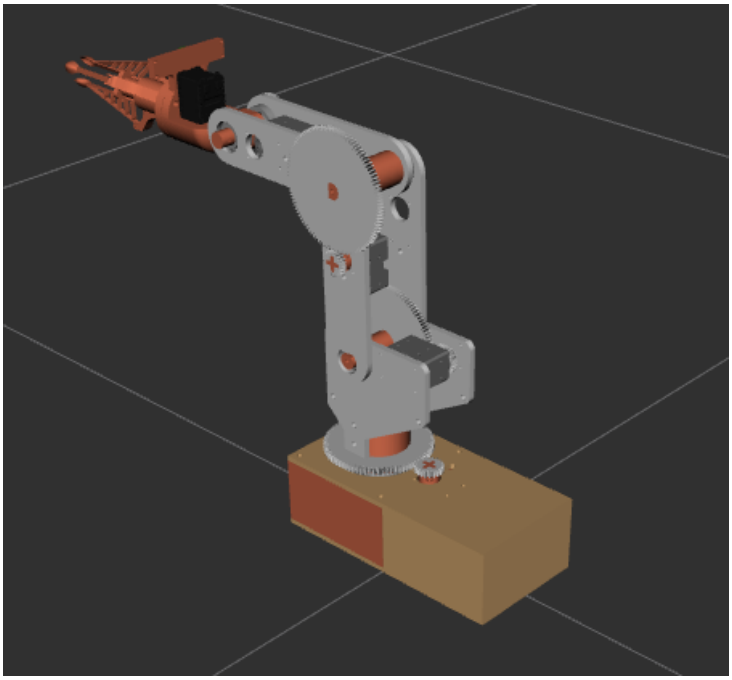


Figure 9.3: Rviz simulation

9.6.2 Inverse kinematics

Figure 9.4: Inverse kinematics launch file

9.6.3 Open loop

For this one, we have two launch files. One where the robot node is the Rviz simulation and the other one where the robot node is the motors. In the second case, You have to connect the real robot to your computer.

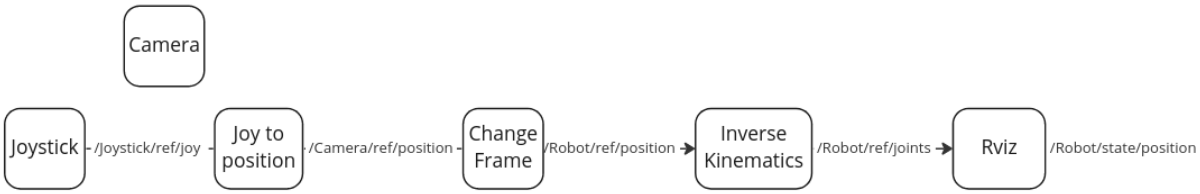


Figure 9.5: Open loop launch file

9.6.4 Closed loop

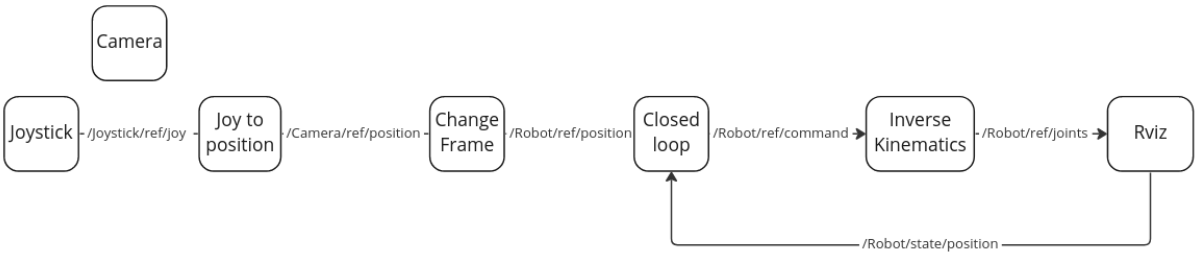


Figure 9.6: Closed loop launch file

As we don't have a sensor for the position in real life, this is only in simulation.

Conclusion

This project offered us a first practical experience. Almost all of us came from preparatory classes, we had never known of such a project. It was however very complete from the exact definition with our customer until its realization. It required various technical skills such as design, automation, and programming that we did not necessarily possess at the beginning. This project was a perfect example of self-training. We learned to find answers online or to ask the right people. We were able to understand that even if we didn't know the exact answer at the beginning, we can still answer a problem.

It is a very good overview of what will be our life in the professional world. We will have to combine technical skills and project management. Defining specifications, following a project, being accountable to a client, and adapting to their demands are all qualities that we have been able to develop.

The objective of the project was to design a remote-controlled robot capable of picking tomatoes. We were able to discover the difficulties that we could encounter in such a realization such as delays or lack of communication. However, we still managed to meet a large part of our customer's requests. Our robot can indeed be controlled by a user with a remote control and is able to pick up tomatoes from its plant without damaging it. Certains points non malheureusement pas pu être abordé, par exemple la mise en ligne de notre solution afin de commander ce robot à distance.

Finally, from a personal point of view, this project was a real discovery for me. In my first year, I didn't know in which sector I should go. Robotics was then a great discovery and this project fascinated me. I loved the diversity of the technical tasks, but also as a project manager to face complications and look for solutions. I was also able to make contacts in the world of robotics and research which then allowed me to do an internship in research.

Glossary

CAD the use of computers to aid in the creation, modification, analysis, or optimization of a design. 17

disparity The apparent motion of objects between a pair of stereo images. 56

forward kinematics The use of a robot's kinematic equations to compute the end-effector's position from specified values for the joint parameters. 29

frame An abstract coordinate system whose origin, orientation, and scale are specified by a set of reference points. 23

Gantt diagram Graphical representation of a schedule. 13

home configuration Corresponds to the rest position, usually all joints or links are set to zero. 23

homogenous coordinates 4×1 vector representing coordinates x, $[x1]$. 50

inverse kinematics The use of kinematic equations to determine a robot's motion (appropriate joints' configuration) to reach a desired position. 29

kinematic diagram Illustates connectivity of links and joints of a mechanism. 23

market study Analyze, understand, and measure the real functioning of the forces at work in a market. 8

milestone Siginificant stage or event in the development of something. 13

motor torque The amount of rotational force that the motor develops. 36

octopus diagram Represents the relationship between a product and its environment. 10

open-source software A software with source code that anyone can inspect, modify, and enhance. 60

payload Maximum mass a robot can support. 27

redundancy The fact that a robot can reach a specified position with more than one joints configuration. 34

scope statement Defines the purpose of the project, the stages of its realization, and the elements necessary to carry it out. 9

screw axis Is used to describe rigid body motions and forces, be its axes of motion (such as the rotation axis of a pure rotation) or force (such as the line of action of a force). 25

servomotor A rotary or linear actuator that allows precise control of angular or linear position, velocity, and acceleration thanks to a position feedback sensor. 44

step function A discontinuous function, whose value is zero until x reaches a certain value t and a given value y when x is greater than t. 52

stereo camera A camera with two or more lenses with a separate image sensor or film frame for each lens. 56

time response Also called setting time, it is the time required for the response curve to reach and stay within a range of a certain percentage of the final value. 54

transformation matrix Transforms one vector into another vector by the process of matrix multiplication The transformation matrix alters the cartesian system and maps the coordinates of the vector to the new coordinates. 24

tunnel effect case where the sponsor receives little or no information during implementation. 12

WBS Hierarchical breakdown of the necessary work. 13

windup The situation in a PID feedback controller where a large change in setpoint occurs (say a positive change) and the integral term accumulates a significant error during the rise. 54

Workspace A specification of the reachable configuration of the end effector. 26

References

Articles

[Dav+16] Joseph Davidson et al. "Hand-picking dynamic analysis for undersensed robotic apple harvesting". In: *Transactions of the ASABE* 59.4 (July 2016), pp. 745–758.

Books

[Bar10] Mathieu Baril. *Conception d'un préhenseur sous-actionné pour les prothèses de membre supérieur et analyse de stabilité*. 2010.

[LP17] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017. ISBN: 978-1-10715-630-2.

[Vil19] Alberto Fernandez Villan. *OpenCV 4 with Python*. Packt Publishing Ltd, 2019. ISBN: 978-1-78934-491-2.

Websites

[FFR] FFRobotics. *FFRobotics*. URL: https://www.ffrobotics.com/.

[Fra] FranceInfo. *Cueillette de pommes : les saisonniers manquent à l'appel*. URL: https://www.francetvinfo.fr/economie/emploi/metiers/agriculture/cueillette-de-pommes-les-saisonniers-manquent-a-l-appel_2349153.html. (accessed: 05.10.2020).

[Ind] Thought Industries. *Onshape Learning Center*. URL: https://learn.onshape.com/.

[Mata] MathWorks. *Matlab and Simulink tutorial*. URL: https://www.mathworks.com/support/learn-with-matlab-tutorials.html.

[Matb] MathWorks. *Matlab and Simulink tutorial*. URL: https://in.mathworks.com/help/ros/ug/get-started-with-ros.html.

[Roba] Abundant Robotics. *Abundant Robitics*. URL: https://waxinvest.com/projects/abundant-robots/.

[Robb] Open Robotics. *ROS Documentation*. URL: http://wiki.ros.org/.

[Robc] Robotis. *Dynamixel Documentation*. URL: https://emanual.robotis.com/. (accessed: 01.06.2021).

A CDC

Functional Specifications

Eden Robotics Project

Document Objectives:

Propose a functional specification for the G1G2 Eden Robotics project.

Actors:

Name Surname	Role	Email
Antoine ALESSANDRINI	Project Manager	antoine.alessandrini@centrale.centralelille.fr
Noé LUXEMBOURGER	Member of the mechanical division	noe.luxembourger@centrale.centralelille.fr
Simon KURNEY	Equipment manager	simon.kurney@centrale.centralelille.fr
Héloïse BOYER-VIDAL	Treasurer	heloise.boyer-vidal@centrale.centralelille.fr
Anna BERGER	Responsible for the mechanics department and training	anna.berger@centrale.centralelille.fr
Aya SKHOUN	Member of the IT department	aya.skhoun@centrale.centrallelille.fr
Thomas JAOUËN	Member of the mechanical division	thomas.jaouen@centrale.centralelille.fr
Maxime BACQUET	Robotics and documentation manager	maxime.bacquet@centrale.centralelille.fr

David KIROV	Head of the IT department	david.kirov@centrale.centralelille.fr
Simon DAHY	Risk manager	simon.dahy@centrale.centralelille.fr
Victor Guinebertiere	External and internal communication manager	victor.guinebertiere@centrale.centralelille.fr
Anna DUCROS	Planning manager	anna.ducros@centrale.centralelille.fr

History of changes

N° version	Date	Description of the change
1	03/12/2020	Creation of the first version of the CDCF
2	22/02/2021	Description of the functions related to the arms/hands.
3	09/06/2021	Clarification of validity criteria
4	12/10/2021	Adaptation to pigeon heart tomatoes
5	02/02/2022	Addition of the "comparison with reality" part in the table of main and constraint functions

1. Study case

Provide a solution to the labor problem in agriculture. Workers must be able to pick up tomatoes from other countries. The solution must be easily fixable and affordable for farmers.

2. Definition of the studied system.

2.1. Utilisateur final

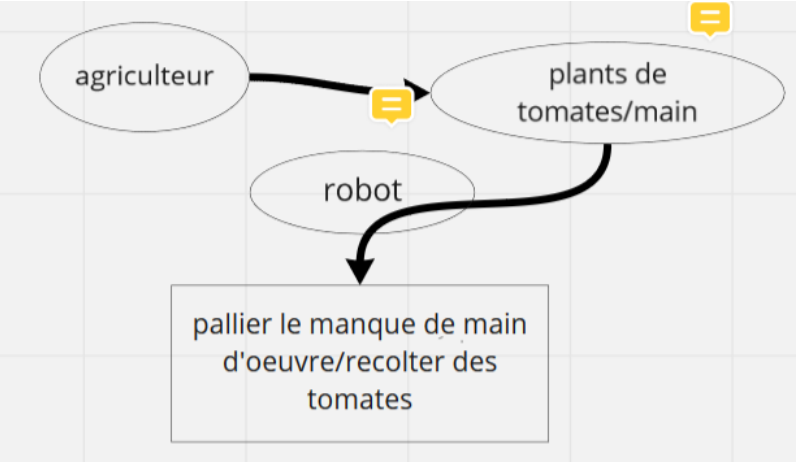
The system is a service to farmers with tomato greenhouses.

2.1. Variété de tomate considérée

The robot is designed to pick cherry tomatoes of the pigeon heart variety, grown in greenhouses.

3. Definition of the need

3.1 Identification of the need



3.2. Validation of the need

- Why the need exists:
  - Lack of cheap labor in agriculture.
- Why the project exists:
  - To address this lack of labor
- Risks of the need changing or disappearing:
  - Stopping the consumption of tomatoes

- Disease that would ravage the tomato plants
- Disappearance of the lack of labor

- Validate the need

4. Identification of functions

4.1. List of life situations studied

Picking up tomatoes

Storage of tomatoes

Storing the robot

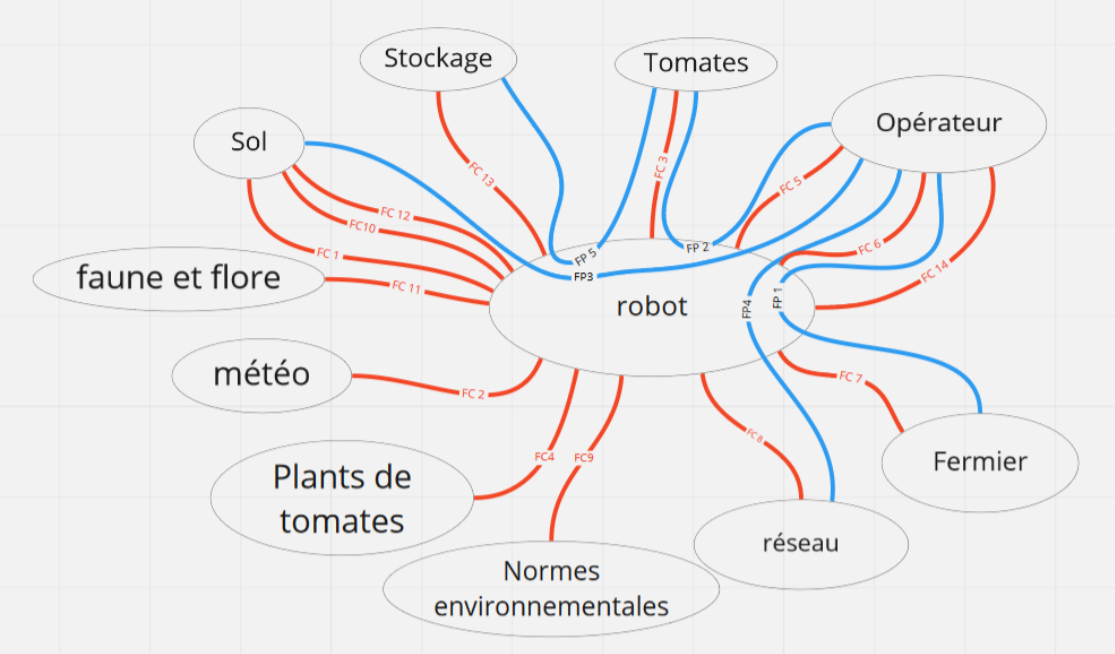
Moving the robot

Recycling the robot

Cleaning the robot

4.2. Life situation: Use

4.2.1. Octopus



4.2.2. Transfer functions

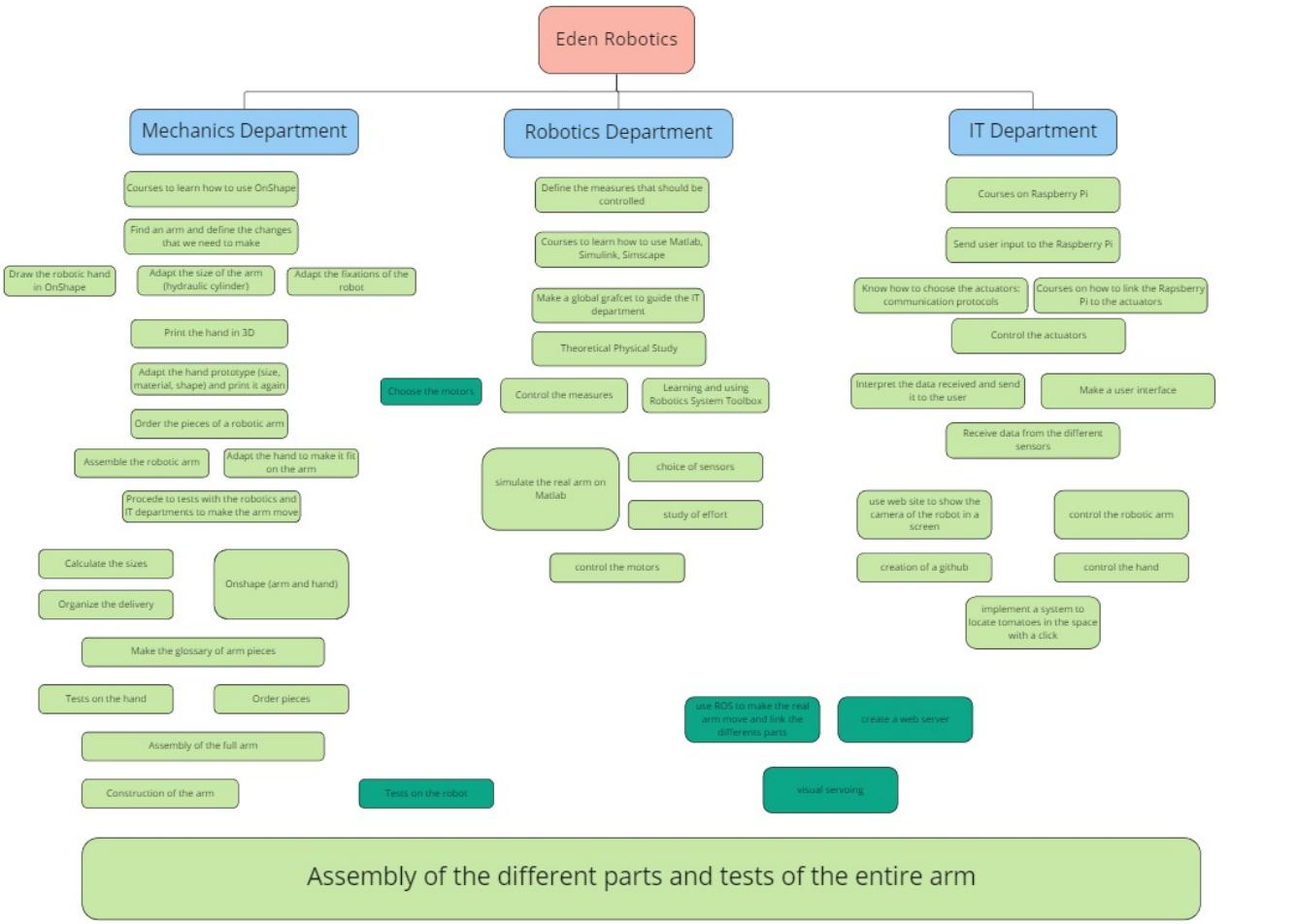
Transfer function	Assessment criterion	Level	Flexibility	Reality
PS 1: Be at least as efficient as a human	Picking time of a tomato	Moins de 10 secondes	F2	8-10 s
FP 2: Picking tomatoes	The tomatoes are detached from the plant	90% of the tomatoes from the plant	F1	75-80% of the time the tomato is not lost
PS 3: Being directed by the operator	Use of a joystick and correct movement	The robot obeys 90% of the commands	F1	Yes (99% of the time)
FP 4: Be remotely controlled	The operator can control the whole robot	500 m distance	F2	No

4.2.3. Constraint functions

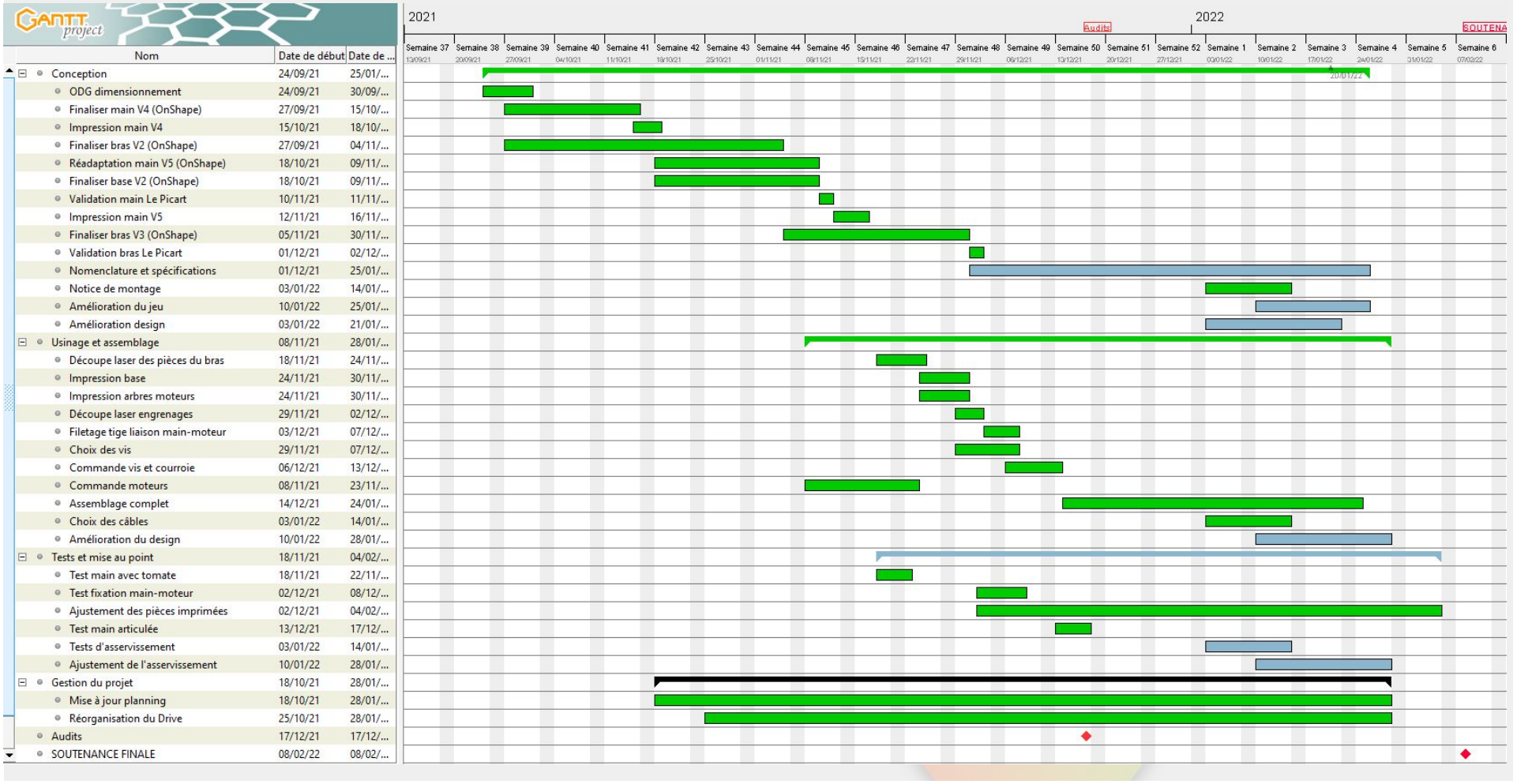
Constraint function	Assessment criterion	Level	Flexibility	Reality
CF 1: Be able to reach all the tomatoes in a tree	be able to reach every tomato	98% of tomatoes reachable for a "standard" orchard 50cm high arm	F1	Accessibility zone achieved
FC 2 : Keep the tomatoes intact	Force exerted on the tomato	Fmax = 5 N	F1	F<5N
FC 3 : Avoid branches and tomatoes (fragility)	Degree of accuracy of the arm	Relative deviation < 1cm	F2	1 to 2 cm
FC 4 : Be easy to control	User's feeling	Handling in less than 2 hours	F3	Yes
FC 5: Be removable and adaptable	To be able to carry it in a medium size suitcase	Dimensions included in a cube of 1m50*1m*1m	F2	Yes

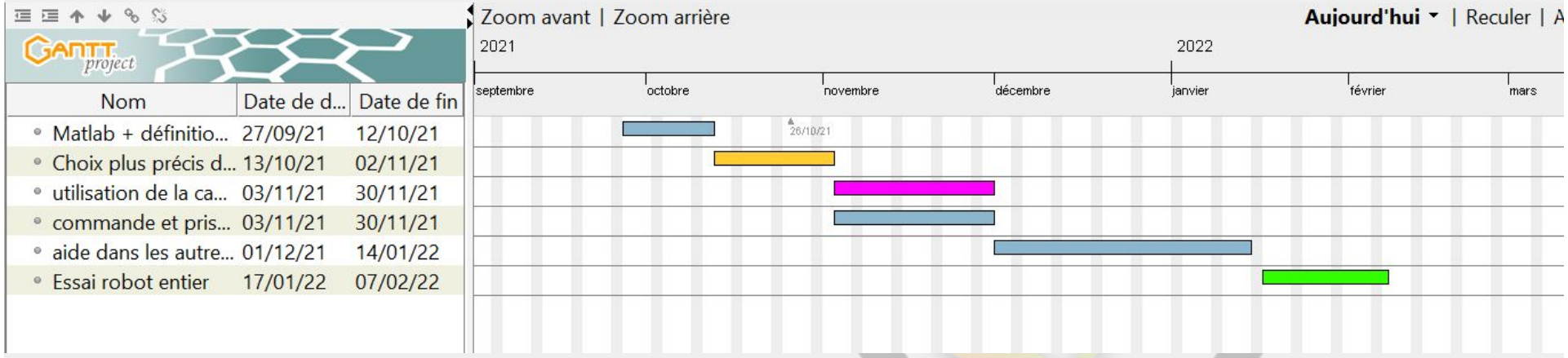
FC 8: Transmit Information Correctly and Quickly	Response time to the transmission of information	Rep time <200ms	F1	Instantaneous local response time
FC 12 : The robot must be stable	The robot must not tip over	The outstretched arm must not tip over	F0	Yes
FC 13: Storage must be accessible by the robot	The robot can put the tomatoes in a storage device	Unloads tomatoes without injuring them	F2	Yes
FC 14: The robot must allow the operator to have a good viewing	Front view angle	Angle > 110	F1	Angles >110
FC 15 : Fit the shape of the tomato	Contact of all parts of the hand with the tomato	80% of the surface of the fingers in contact with the tomato	F0	Fingers have a good shape

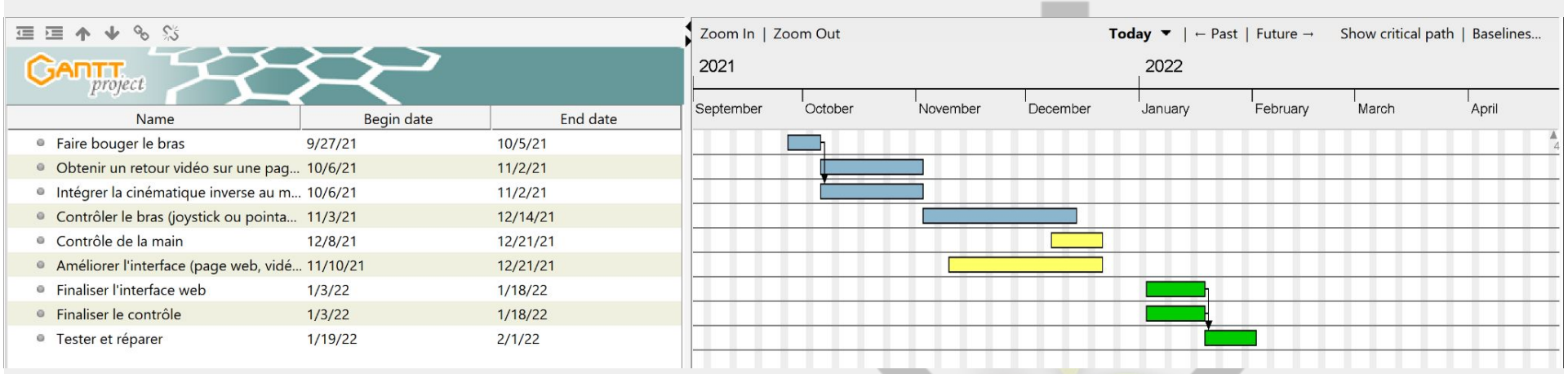
B WBS


Eden Robotics
Mechanics Department
Robotics Department
IT Department
Courses to learn how to use OnShape
Find an arm and define the changes that we need to make
Draw the robotic hand in OnShape
Adapt the size of the arm (hydraulic cylinder)
Adapt the fixations of the robot
Print the hand in 3D
Adapt the hand prototype (size, material, shape) and print it again
Order the pieces of a robotic arm
Assemble the robotic arm
Adapt the hand to make it fit on the arm
Procede to tests with the robotics and IT departments to make the arm move
Calculate the sizes
Organize the delivery
Onshape (arm and hand)
Make the glossary of arm pieces
Tests on the hand
Order pieces
Assembly of the full arm
Construction of the arm
Tests on the robot
Define the measures that should be controlled
Courses to learn how to use Matlab, Simulink, Simscape
Make a global grafcet to guide the IT department
Theoretical Physical Study
Choose the motors
Control the measures
Learning and using Robotics System Toolbox
simulate the real arm on Matlab
choice of sensors
study of effort
control the motors
Courses on Raspberry Pi
Send user input to the Raspberry Pi
Know how to choose the actuators: communication protocols
Courses on how to link the Rapsberry Pi to the actuators
Control the actuators
Interpret the data received and send it to the user
Make a user interface
Receive data from the different sensors
use web site to show the camera of the robot in a screen
control the robotic arm
creation of a github
control the hand
implement a system to locate tomatoes in the space with a click
use ROS to make the real arm move and link the differents parts
create a web server
visual servoing
Assembly of the different parts and tests of the entire arm
miro

C Gantt Diagram


GANTT project
2021
2022
Audits
SOUTENA
Semaine 37
Semaine 38
Semaine 39
Semaine 40
Semaine 41
Semaine 42
Semaine 43
Semaine 44
Semaine 45
Semaine 46
Semaine 47
Semaine 48
Semaine 49
Semaine 50
Semaine 51
Semaine 52
Semaine 1
Semaine 2
Semaine 3
Semaine 4
Semaine 5
Semaine 6
Nom
Date de début
Date de ...
Conception 24/09/21 25/01/...
ODG dimensionnement 24/09/21 30/09/...
Finaliser main V4 (OnShape) 27/09/21 15/10/...
Impression main V4 15/10/21 18/10/...
Finaliser bras V2 (OnShape) 27/09/21 04/11/...
Réadaptation main V5 (OnShape) 18/10/21 09/11/...
Finaliser base V2 (OnShape) 18/10/21 09/11/...
Validation main Le Picart 10/11/21 11/11/...
Impression main V5 12/11/21 16/11/...
Finaliser bras V3 (OnShape) 05/11/21 30/11/...
Validation bras Le Picart 01/12/21 02/12/...
Nomenclature et spécifications 01/12/21 25/01/...
Notice de montage 03/01/22 14/01/...
Amélioration du jeu 10/01/22 25/01/...
Amélioration design 03/01/22 21/01/...
Usinage et assemblage 08/11/21 28/01/...
Découpe laser des pièces du bras 18/11/21 24/11/...
Impression base 24/11/21 30/11/...
Impression arbres moteurs 24/11/21 30/11/...
Découpe laser engrenages 29/11/21 02/12/...
Filetage tige liaison main-moteur 03/12/21 07/12/...
Choix des vis 29/11/21 07/12/...
Commande vis et courroie 06/12/21 13/12/...
Commande moteurs 08/11/21 23/11/...
Assemblage complet 14/12/21 24/01/...
Choix des câbles 03/01/22 14/01/...
Amélioration du design 10/01/22 28/01/...
Tests et mise au point 18/11/21 04/02/...
Test main avec tomate 18/11/21 22/11/...
Test fixation main-moteur 02/12/21 08/12/...
Ajustement des pièces imprimées 02/12/21 04/02/...
Test main articulée 13/12/21 17/12/...
Tests d'asservissement 03/01/22 14/01/...
Ajustement de l'asservissement 10/01/22 28/01/...
Gestion du projet 18/10/21 28/01/...
Mise à jour planning 18/10/21 28/01/...
Réorganisation du Drive 25/10/21 28/01/...
Audits 17/12/21 17/12/...
SOUTENANCE FINALE 08/02/22 08/02/...


Zoom avant | Zoom arrière
Aujourd'hui | Reculer | A
2021
2022
septembre
octobre
novembre
décembre
janvier
février
mars
26/10/21
Nom
Date de d...
Date de fin
Matlab + définitio... 27/09/21 12/10/21
Choix plus précis d... 13/10/21 02/11/21
utilisation de la ca... 03/11/21 30/11/21
commande et pris... 03/11/21 30/11/21
aide dans les autre... 01/12/21 14/01/22
Essai robot entier 17/01/22 07/02/22



Name	Begin date	End date
Faire bouger le bras	9/27/21	10/5/21
Obtenir un retour vidéo sur une pag...	10/6/21	11/2/21
Intégrer la cinématique inverse au m...	10/6/21	11/2/21
Contrôler le bras (joystick ou pointa...	11/3/21	12/14/21
Contrôle de la main	12/8/21	12/21/21
Améliorer l'interface (page web, vidé...	11/10/21	12/21/21
Finaliser l'interface web	1/3/22	1/18/22
Finaliser le contrôle	1/3/22	1/18/22
Tester et réparer	1/19/22	2/1/22

D Assembly Instructions

Material

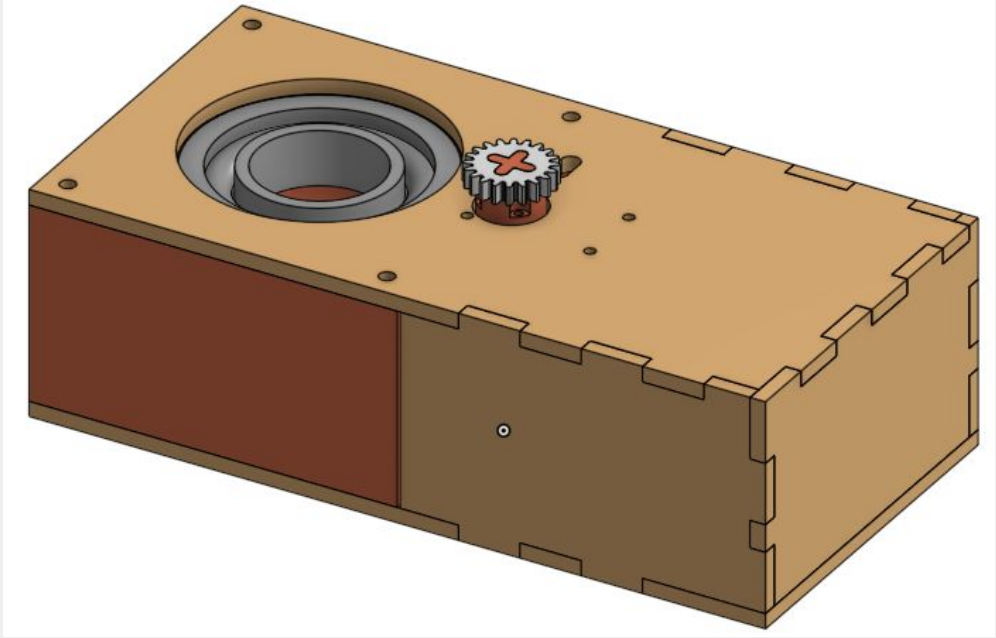
Laser cutting plywood

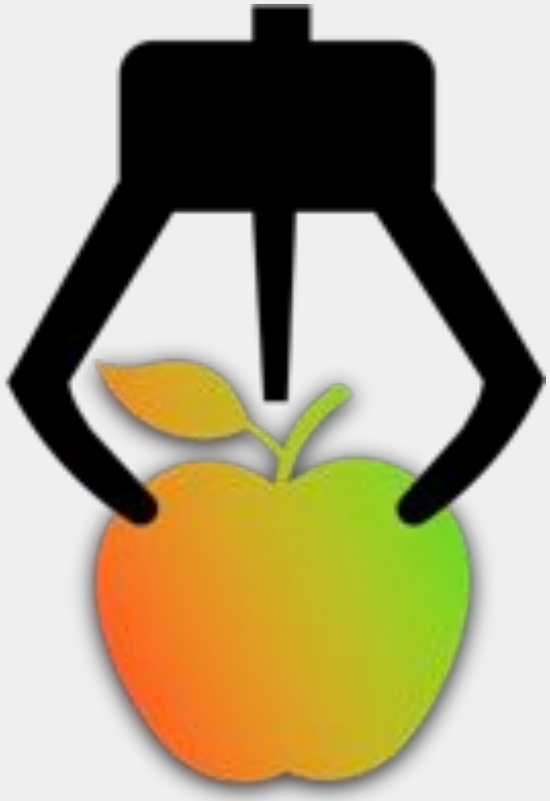
3d printed PLA

Laser cutting PVC

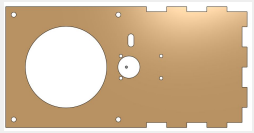
steel

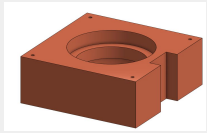
Step 1





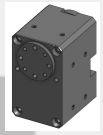
Parts


top x1

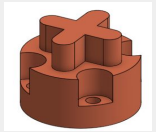

adapteur x1

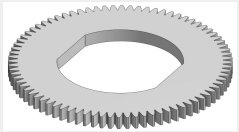

small_gear_pelvis (20 teeths) x1


SKF 7208 x2

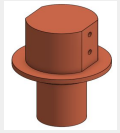

XL430-W250-T x1

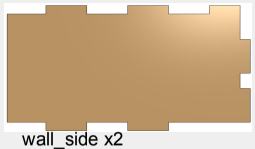

wall_front x1

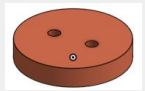

motor_link_xl x1


big_gear_pelvis (80 teeths) x1


bottom x1

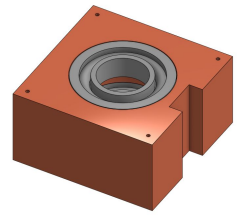

tree_base x1


wall_side x2


fixing_base_tree x1

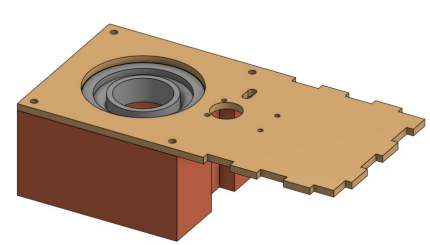
1

adaptation
SKF 2708 x1



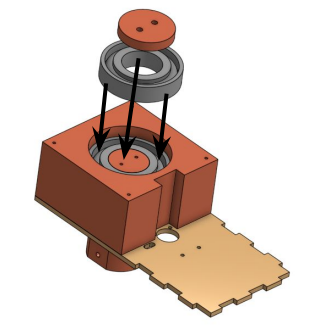
2

top
Screws :
- M3 (20) x4



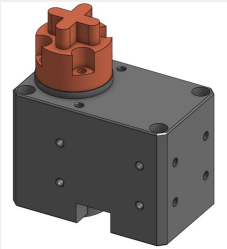
3

tree
fixing tree
SKF 2708 x1
Screws :
- M3 (20) x2



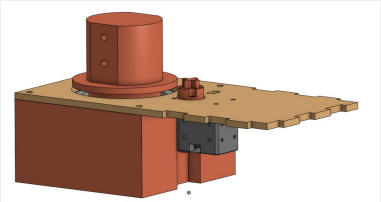
4

motor
link
Screws :
- M2 (12) x4



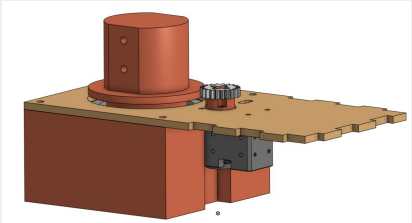
5

Screws :
- M2 (12) x4



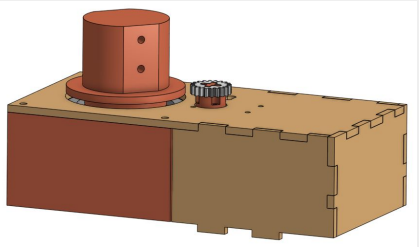
6

Small gear



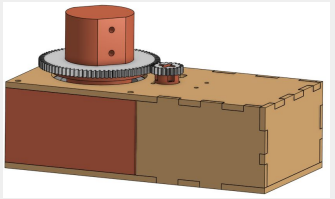
7

side
front

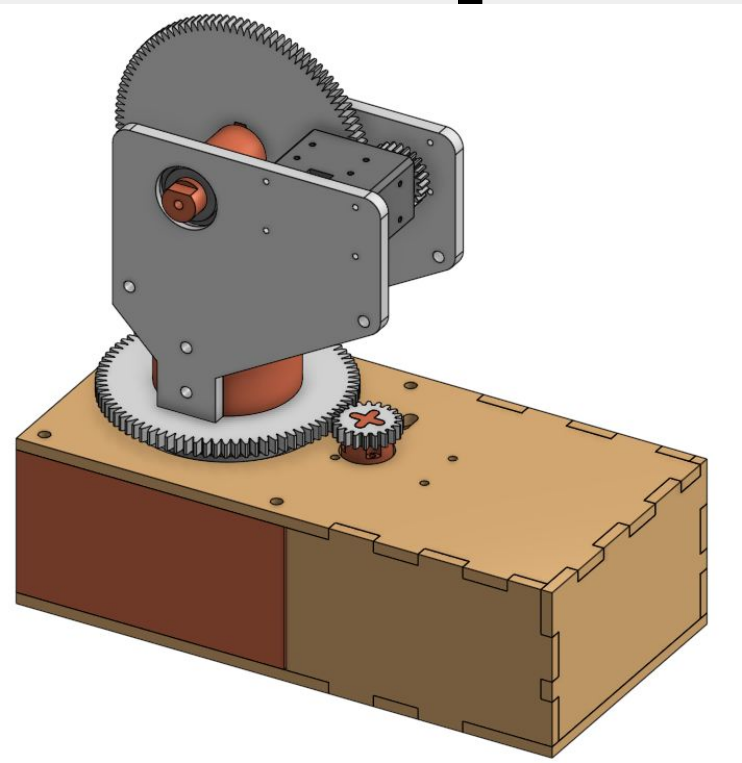


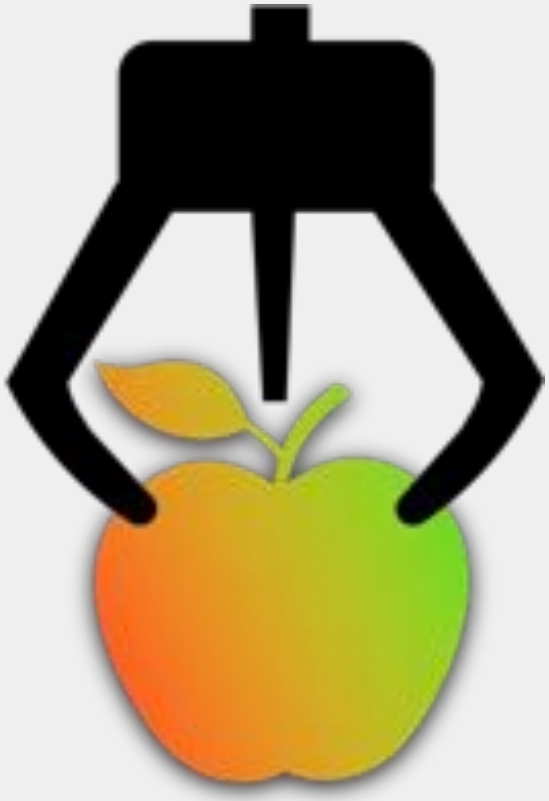
8

bottom
big gear

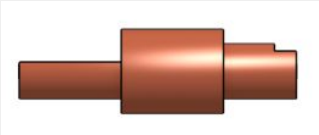


Step 2





Parts

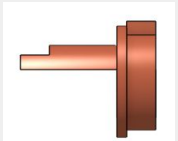


tree_shoulder x1

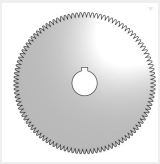
small_gear_shoulder (25 teeths) x1

SKF 7202 x2

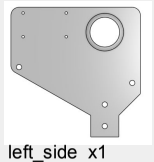
XM430-W350-T x1



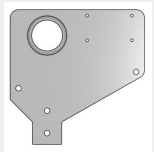
fixing_shoulder_tree x1



big_gear_shoulder (110 teeths) x1



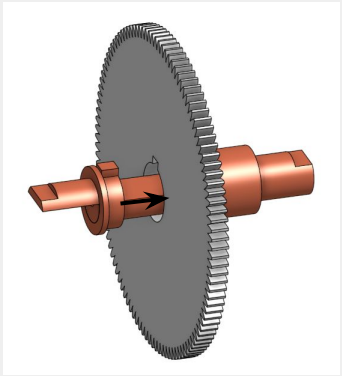
left_side x1



right_side x1

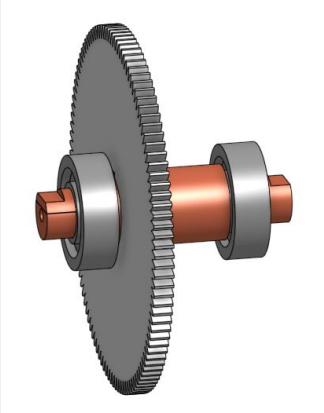
1

beag gear
tree
fixing tree



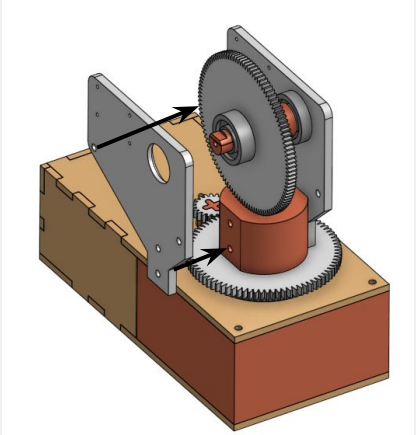
2

SKF 7202



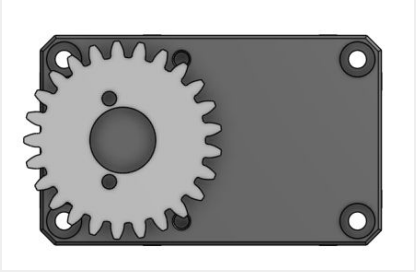
3

left side
right side
Screws :
- M5 (70) x3



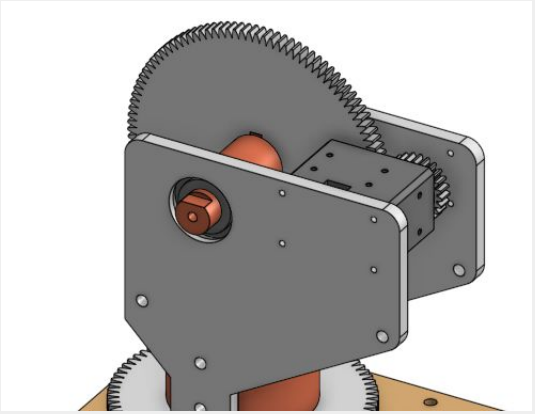
4

motor
small gear
Screws :
- M2 (12) x2

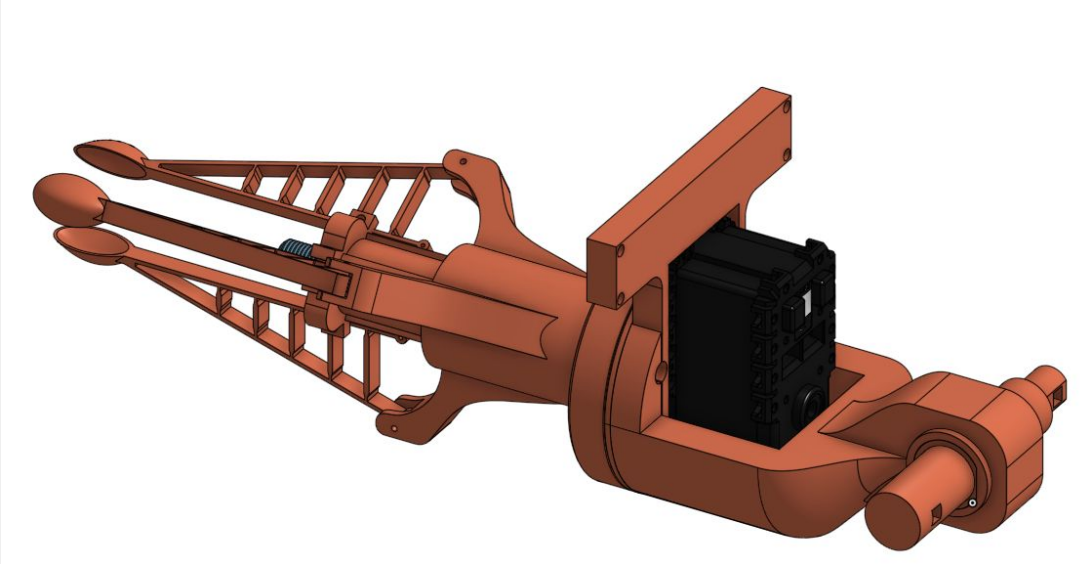


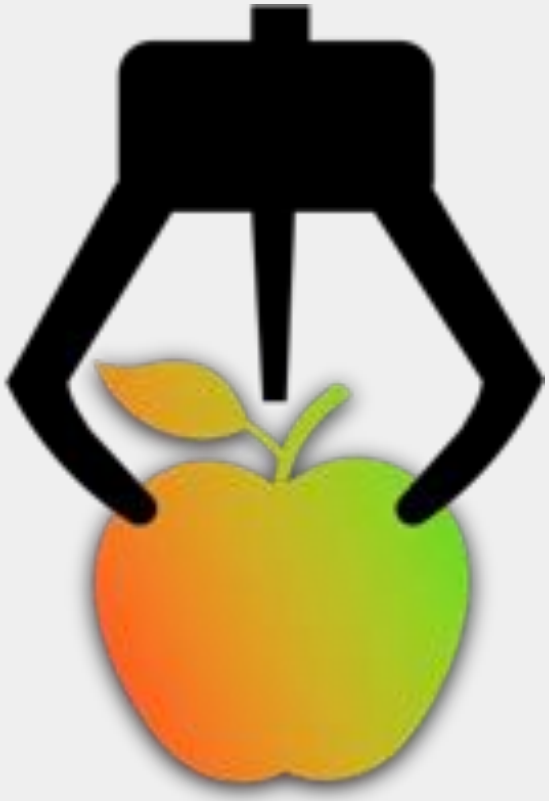
5

Screws :
- M2 (12) x4

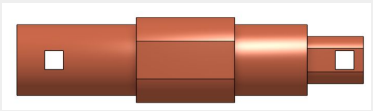


Step 3

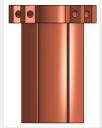




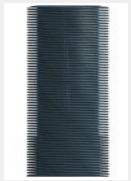
Parts



tree_wrist x1

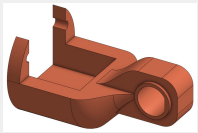


fixing_rod x1



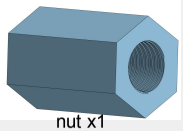
threaded_rod x1

AX-12W x1



fixing_hand x1

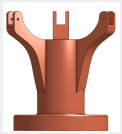
cover_rod x1



nut x1

fixing_camera x1

finger x3



fixing_finger x1

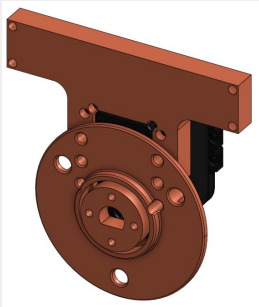
fixing_AX

1

motor
fixing ax
Screws :
- M2 (8) x4

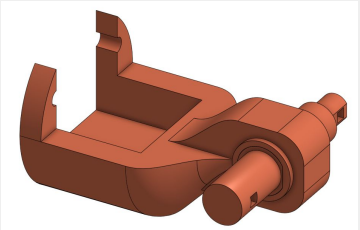
2

fixing camera
Screws :
- M2 (10) x4



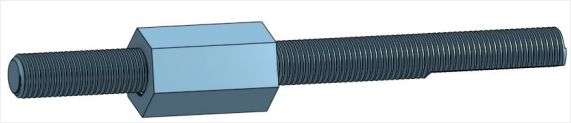
3

fixing hand
tree



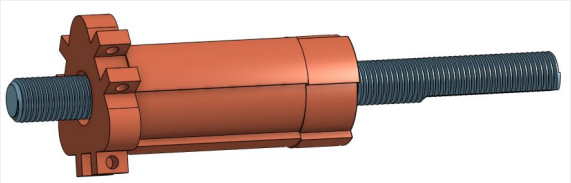
4

threaded rod
nut



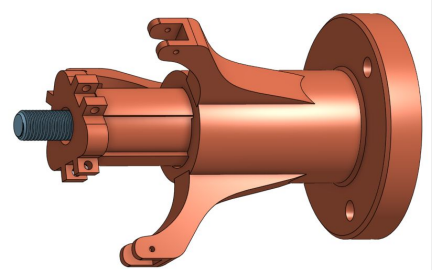
5

fixing rod
cover rod



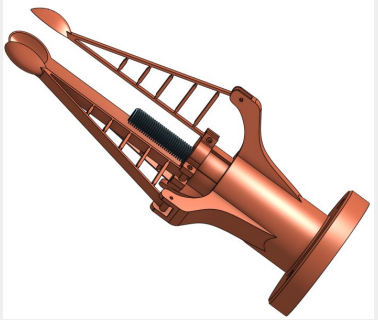
6

fixing finger



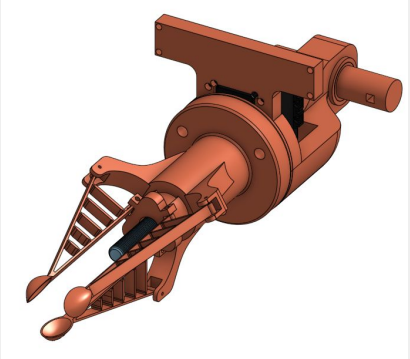
7

finger x3
Screws :
- M2 (15) x6

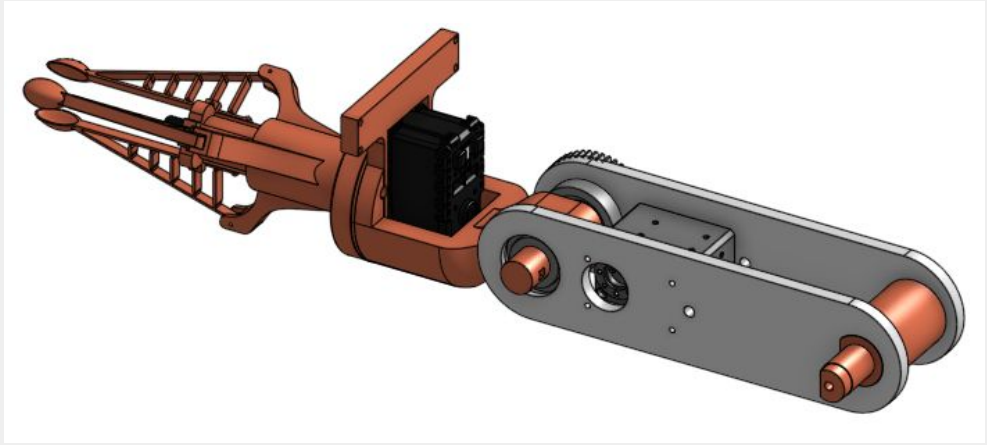


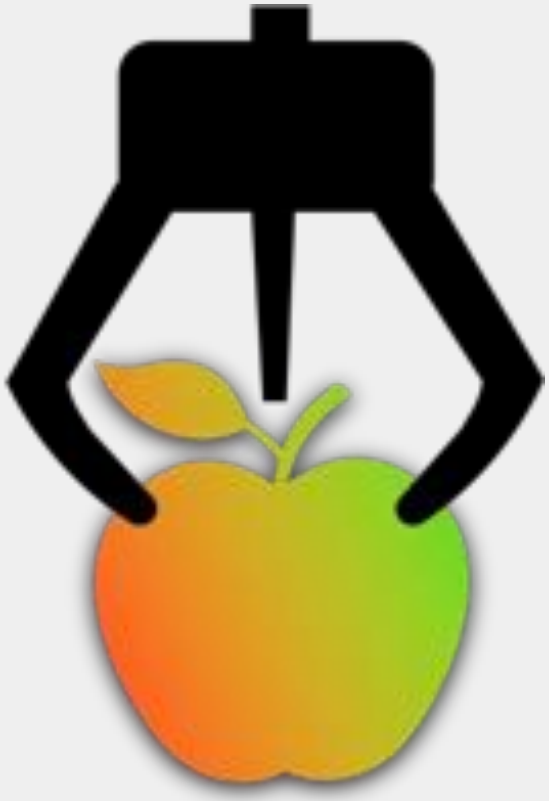
8

Screws :
- M5 (25) x3



Step 4





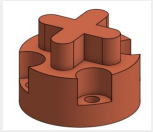
Parts

tree_elbow x1

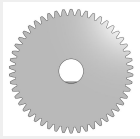
small _gear_wrist (25 teeths) x1

SKF 7202 x2

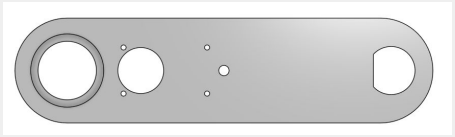
XL430-W250-T x1



motor_link_xl x1



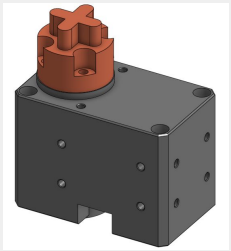
big_gear_wrist (50 teeths) x1



radius x2

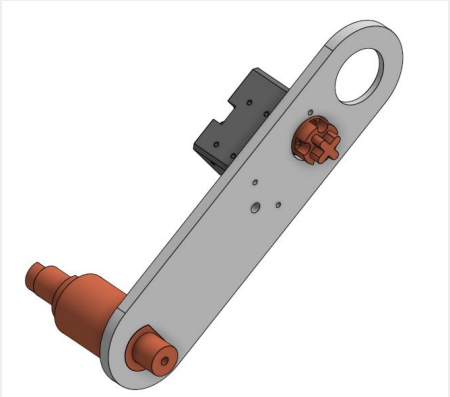
1

motor
motor link
Screws :
- M2 (12) x4

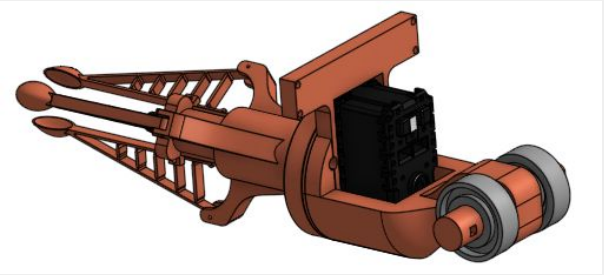


2

tree
arm x1
Screws :
- M2 (12) x4

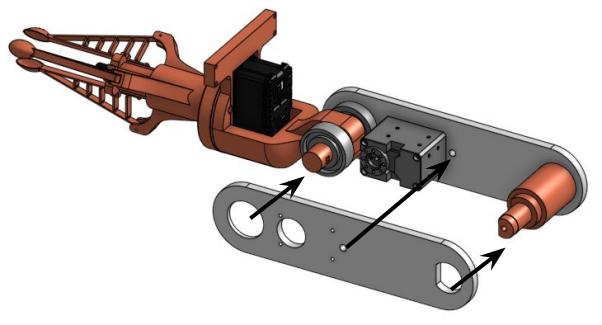


3 SKF 7202 x2



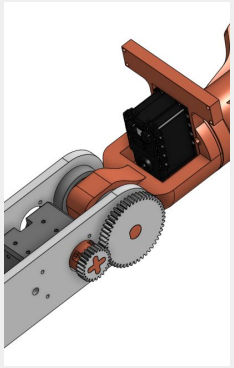
4

arm x1
Screw :
- M5 (60) x1

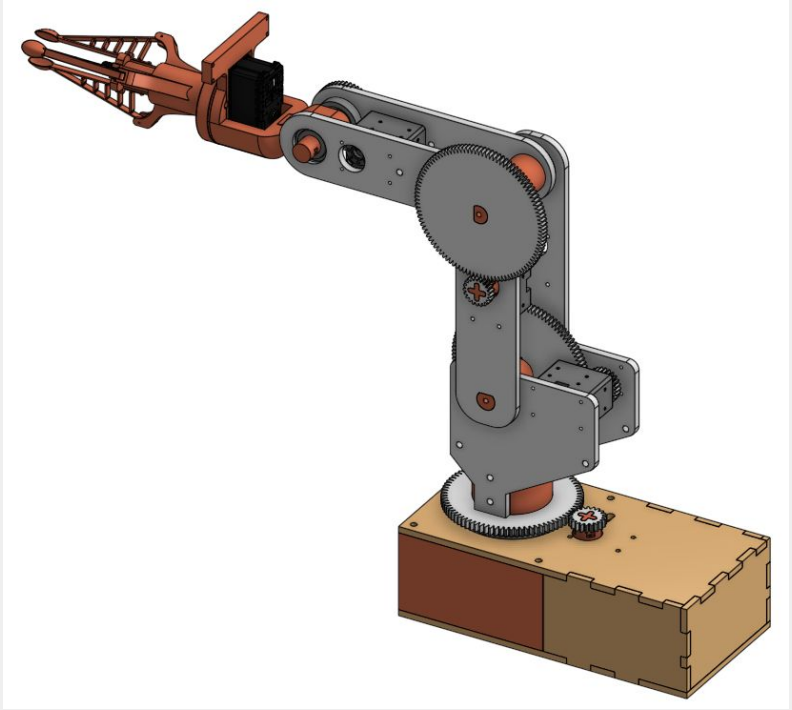


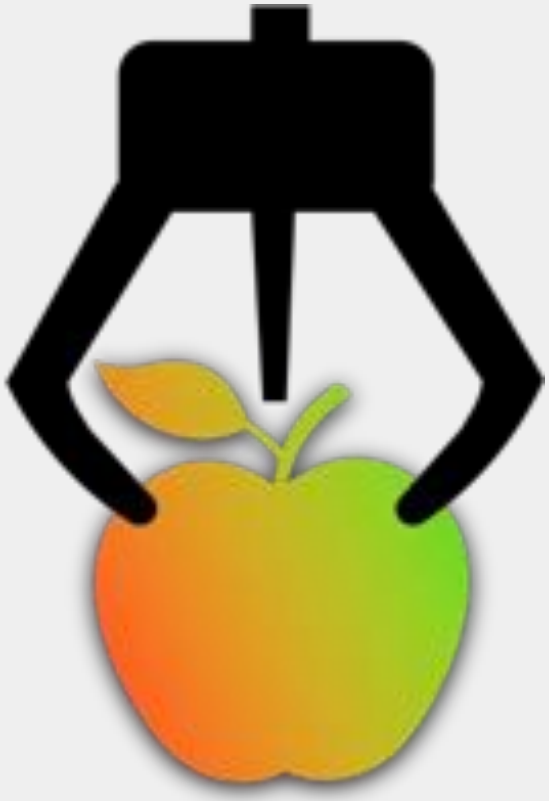
5

Small gear
Big gear

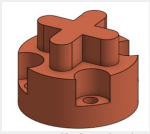


Step 5

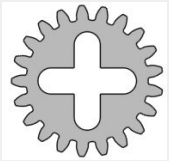




Parts



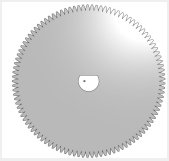
motor_link_xl x1



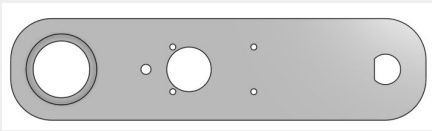
small_gear_elbow (21 teeths) x1

SKF 7202 x2

XC430-W150-T x1



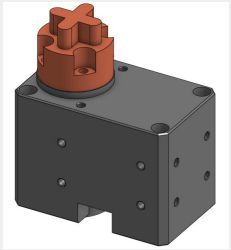
big_gear_elbow (105 teeths) x1



humerus x2

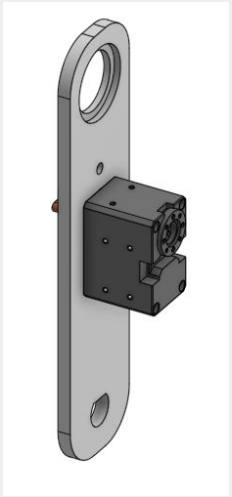
1

motor
motor link
Screws :
- M2 (12) x4

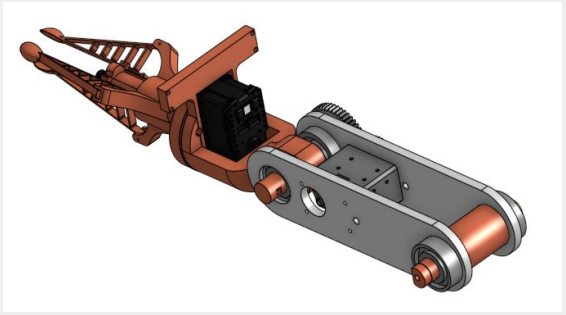


2

arm x1
Screws :
- M2 (12) x4

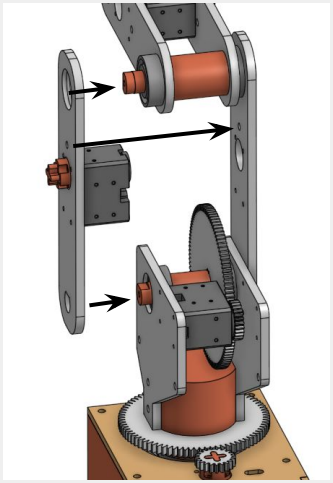


3 SKF 7202



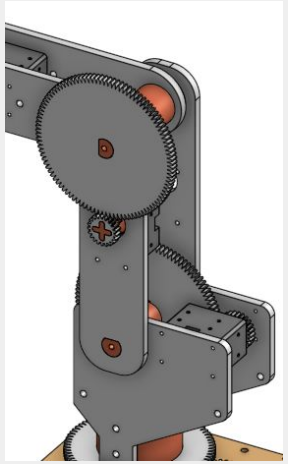
4

arm x1
Screws :
- M5 (80) x1



5

small gear
big gear



E Disparity And Depth

```
import numpy as np
import cv2
import time

###### Parameters to determine position ######

# proportionnal coefficiant between depth and 1/disparity
M1 =  14.49177553
M2 = 10.63325109
# R2 = 0.9989

Camera_id = 4 # Camera ID for left camera
frame_size = [1280,480]

###### Initialise Cameras ######

# Check for left and right camera IDs
# These values can change depending on the system

#open both cameras
Camera = cv2.VideoCapture(Camera_id)
Camera.set(3, frame_size[0])
Camera.set(4, frame_size[1])

###### Mouse clicking ######

# Define the pixel we want to study
row = 0
column = 0

#function for detecting left mouse click
def mousePoints(event, x,y, flags, param):
    global ligne, colonne
    if event == cv2.EVENT_LBUTTONDOWN:
        print("Pressed", x,y)
        ligne = y
        colonne = x

###### Camera paramters to undistort and rectify images ######

# Camera paramters to undistort and rectify images
cv_file =  cv2.FileStorage()
cv_file.open('stereoMap.xml',cv2.FILE_STORAGE_READ)

stereoMapL_x = cv_file.getNode('stereoMapL_x').mat()
stereoMapL_y = cv_file.getNode('stereoMapL_y').mat()
stereoMapR_x = cv_file.getNode('stereoMapR_x').mat()
stereoMapR_y = cv_file.getNode('stereoMapR_y').mat()
cv_file.release()
```

```
50
51
52      ###### Create stereo object ######
53      # create side bar for disparity computation parameters
54
55      def nothing(x):
56          pass
57      cv2.namedWindow('Parameters',cv2.WINDOW_NORMAL)
58      cv2.resizeWindow('Parameters',400,600)
59
60      cv2.createTrackbar('numDisparities','Parameters',6,17,nothing)
61      cv2.createTrackbar('blockSize','Parameters',2,20,nothing)
62      cv2.createTrackbar('uniquenessRatio','Parameters',0,50,nothing)
63      cv2.createTrackbar('speckleRange','Parameters',22,50,nothing)
64      cv2.createTrackbar('speckleWindowSize','Parameters',5,50,nothing)
65      cv2.createTrackbar('disp12MaxDiff','Parameters',15,50,nothing)
66      cv2.createTrackbar('minDisparity','Parameters',18,100,nothing)
67
68
69      stereo = cv2.StereoSGBM_create()
70
71      ###### Start image processing ######
72
73      while True:
74          succes, frame = Camera.read()
75
76          if succes :
77              frame_left = frame[:,:int(frame_size[0]/2)]
78              frame_right = frame[:,int(frame_size[0]/2):frame_size[0]]
79
80              # Undistort and rectify images
81              frame_right = cv2.remap(frame_right, stereoMapR_x, stereoMapR_y,
82                                      cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)
83              frame_left = cv2.remap(frame_left, stereoMapL_x, stereoMapL_y,
84                                      cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)
85
86          ############### Stereo to disparity ######
87              # Updating the parameters based on the trackbar positions
88              numDisparities = cv2.getTrackbarPos('numDisparities','Parameters')*16
89              blockSize = cv2.getTrackbarPos('blockSize','Parameters')*2 + 5
90              uniquenessRatio = cv2.getTrackbarPos('uniquenessRatio','Parameters')
91              speckleRange = cv2.getTrackbarPos('speckleRange','Parameters')
92              speckleWindowSize = cv2.getTrackbarPos('speckleWindowSize','Parameters')*2
93              disp12MaxDiff = cv2.getTrackbarPos('disp12MaxDiff','Parameters')
94              minDisparity = cv2.getTrackbarPos('minDisparity','Parameters')
95
96              # Setting the updated parameters before computing disparity map
97              stereo.setNumDisparities(numDisparities)
98              stereo.setBlockSize(blockSize)
99              stereo.setUniquenessRatio(uniquenessRatio)
100             stereo.setSpeckleRange(speckleRange)
```

```
            stereo.setSpeckleWindowSize(speckleWindowSize)
            stereo.setDisp12MaxDiff(disp12MaxDiff)
            stereo.setMinDisparity(minDisparity)

            # Calculating disparity using the StereoSGBM algorithm
            disparity = stereo.compute(frame_left, frame_right)


            # Converting to float32
            disparity = disparity.astype(np.float32)

            # Scaling down the disparity values and normalizing them
            disparity = (disparity/16.0 - minDisparity)/numDisparities

        ###### Disparity to deth ######

            depth = M1/disparity[ligne][colonne]+M2


        ###### Displaying the disparity map ######

            cv2.imshow("Disparity",disparity)
            cv2.imshow("Left",frame_left)
            # cv2.imshow("Right",frame_right)

            cv2.setMouseCallback("Left", mousePoints)

        ###### print position ######

            if ligne !=0 and colonne!=0:
                print("z = ",depth)
                ligne = 0
                colonne = 0

        ###### Stop ######

            # Close window using typing q
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        else:
            Camera= cv2.VideoCapture(Camera_id)

    ###### Release and destroy all windows before termination ######

    Camera.release()

    cv2.destroyAllWindows()
```

F Forward Kinematics Python Node

```python
#!/usr/bin/env python3
import rospy
from geometry_msgs.msg import Point
from sensor_msgs.msg import JointState
from kinematics.parameters import *

class Forward_Node:
    """
    Node for the forward kinematics of the arm
    """
    def __init__(self,rosName="forward_node"):
        # Init ROS node
        rospy.init_node(rosName, anonymous=True)
        try:
            rate = rospy.get_param('/rate')
        except :
            rate = 100
        self.rosRate = rospy.Rate(rate)
        # init variables
        self.config = m_e
        self.thetalist = np.array([0,0,0,0])
        self.point = np.array([0,0,0,0])
        # init parameters
        self.screw_list = screw_list
        self.config_init = m_e
        # init publisher and subscriber
        self.initGraph()

    def initGraph(self):
        self.pub = rospy.Publisher('/EndEffector/state/position',
                                   Point,
                                   queue_size=10)
        self.sub = rospy.Subscriber('/Robot/ref/joint',
                                   JointState,
                                   self.on_receive_callback,
                                   queue_size=10)

    def on_receive_callback(self,data):
        self.thetalist = np.array(data.position)

    def publish(self):
        msg = Point()
        msg.x = self.point[0]
        msg.y = self.point[1]
        msg.z = self.point[2]
        self.pub.publish(msg)

    def updateConfig(self):
        while not rospy.is_shutdown():
```

```
50                 self.config = mr.FKinSpace(self.config_init,
51                                            self.screw_list,
52                                            self.thetalist)
53                 self.point = self.config.dot(np.array([0,0,0,1]))[:-1]
54                 self.publish()
55                 self.rosRate.sleep()
56 
57     def main():
58         forward = Forward_Node()
59         forward.updateConfig()
60 
61     if __name__ == '__main__':
62         try:
63             main()
64         except rospy.ROSInterruptException:
65             pass
```