Une image contenant texte, Police, logo, Graphique

Le contenu généré par l’IA peut être incorrect.

|  |  |
| --- | --- |
| 2nd year internship report | |
| presented by | |
| Matteo PROVERBIO | |
| INLG 2017 | Enterprise :  **USC CiTIUS**  Enterprise tutor :  **Juan Antonio CORRALES RAMON**  SIGMA Clermont tutor:  **Chedli Belhassen BOUZGARROU**  Date : September 2025 |
| Référence du fichier : Proverbio\_ Matteo\_RapportStage2A.pdf | |

**SIGMA Clermont**   
Campus de Clermont-Ferrand – CS 20265 – 63178 Aubière– France  
Tél. +33 (0)4 73 28 80 00 – Fax +33 (0)4 73 28 81 00 –https://www.sigma-clermont.fr/

**Une image contenant texte, Police, logo, Graphique

Description générée automatiquementFICHE D’IDENTIFICATION DU DOCUMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Rapport de Stage 2ème Année** | | | |
| **TITRE DU DOCUMENT :** | | Rapport de stage 2ème année | |
|  | | *2nd year internship report* | |
| **AUTEUR(S) :** | | Matteo PROVERBIO –MRS – Pôle mécanique | |
| **Date du document** | **Nbre de pages** | | **Référence du document** |
| 19/09/2025 |  | | Proverbio\_ Matteo\_RapportStage2A.pdf |
| **ABSTRACT:**  Presentation of the second-year internship as a researcher at the technology research center affiliated with the University of Santiago de Compostela CiTIUS. Installation and handling of a new work environment on a Linux distribution in order to manipulate an ur30 robotic arm through SensONE T80 force sensors and motion planning within the ROBOTA-SUDOE project. | | | |
| **Keywords:** report, project, robotic arm, effort sensor, motion planning, ur30 | | | |
| **RESUME :**  Présentation du stage de deuxième année en tant que chercheur au centre de recherche technologique affiliée à l’université de Saint Jacques de Compostelle CiTIUS. Installation et prise en main d’un nouvel environnement de travail sur une distribution Linux afin de manipuler un bras robotique ur30 au travers de capteurs d’efforts SensONE T80 et de planification de mouvements dans le cadre du projet ROBOTA-SUDOE. | | | |
| **Mots clés :** rapport, projet, bras robotique, capteurs d’efforts, planification de mouvements, ur30 | | | |
| **Visa de l’entreprise :**  Ce rapport de stage a été visé par (Nom, Prénom, Qualité) : ……………………………………..  Date :  Signature :  Je demande que ce rapport soit confidentiel (non diffusé sur l’intranet de l’école) : ð OUI ð NON | | | |

**SIGMA Clermont**   
Campus de Clermont-Ferrand – CS 20265 – 63178 Aubière– France  
Tél. +33 (0)4 73 28 80 00 – Fax +33 (0)4 73 28 81 00 –http://www.sigma-clermont.fr/

# Summary

[Summary 4](#_Toc206510511)

[Acknowledgments 5](#_Toc206510512)

[Introduction 7](#_Toc206510513)

[1.1. CiTIUS 7](#_Toc206510514)

[1.2. The project 8](#_Toc206510515)

[1.3. ROS 8](#_Toc206510516)

[1.4. Moveit 9](#_Toc206510517)

[The robotic arm ur 10](#_Toc206510518)

[1.5. UR30 10](#_Toc206510519)

[1.6. UR5e 11](#_Toc206510520)

[SensONE T80 from BOTA SYSTEMS 12](#_Toc206510521)

[1.7. Serial 12](#_Toc206510522)

[1.8. EtherCAT 12](#_Toc206510523)

[The knife 14](#_Toc206510524)

[1.9. 3D print 14](#_Toc206510525)

[1.10. The handle 15](#_Toc206510526)

[1.10.1. Arduino 17](#_Toc206510527)

[1.10.2. Python 18](#_Toc206510528)

[1.10.3. Neural network 19](#_Toc206510529)

[1.11. Alternative with OnRobot HEX-E v2 20](#_Toc206510530)

[1.12. Flexible handle 21](#_Toc206510531)

[Side tasks 22](#_Toc206510532)

[1.13. UR30 adaptor 22](#_Toc206510533)

[1.14. 3D printed serial sensor 23](#_Toc206510534)

[1.15. 3D simulation of the handle 24](#_Toc206510535)

[1.15.1. Sensors view 24](#_Toc206510536)

[1.15.2. Manipulability ellipse 25](#_Toc206510537)

[Conclusion 26](#_Toc206510538)

[Annexe 27](#_Toc206510539)

# Acknowledgments

I express my sincerest acknowledgements to Mr Corrales Ramon for the opportunity of this internship and the support he provided during those 4 months.

I address my acknowledgements to Saltanat SEITZHAN for the help she gave us during our internship and during her PhD.

Finaly, I thank warmly the whole CiTIUS team for their welcome and for providing us with all the help we needed.

# Introduction

## CiTIUS

My internship took place at CiTIUS (Centro Singular de Investigación en Tecnoloxías Intelixentes), located at the University of Santiago de Compostela in Galicia, Spain. CiTIUS is a leading interdisciplinary research center created in 2009 that focuses on the development and application of intelligent technologies about computer science, artificial intelligence, robotics, mathematics and engineering. For four months, I worked on a robotic project with Universal Robots.

Furthermore, CiTIUS is linked with the academical universe on one hand, and on the other hand with the industrial world. It has several academic partners such as the University of Santiago de Compostela, the university of Vigo and A coruña.

But it has also several industrial partners as we can see on Figure 1.



Figure 1 : CiTIUS' partners



Figure 2 : Front of CiTIUS

Inside, there are some places dedicated to several activities :

The Figure 4 shows the laboratory were the interns work. I passed the most of my time here.

The Figure 3 shows the Maker Lab, the place were are used the 3D printers and machine tool.

The Figure 5 shows the laboratory where are stored the microelectronics equipment, such as solder for instance.



Figure 3 : Maker Lab



Figure 4 : Laboratory



Figure 5 : Microelectronics lab

## The project

This project, called ROBOTA-SUDOE, is divided into two different parts :

The first one is about the cut of meat.

The second one is about the assembly of the different parts of a doll.

All the previous tasks are realized with the help of a seriel robotic arm UR30. During this internship, I was charged to program the trajectory of the arm and to adapt the different sensors in it.

At the end, the robotic arm will have to support a human user in the cut of meat by providing enough power to cut meat without any effort.

The ur30 will also be autonomous in the doll assembly and will reduce the assembling time thanks to computer vision, trajectory program and tactile sensors.

The link for the Github where is the entire content of this report is at Annexe A.

## ROS

First, I had to learn to use ROS (Robot Operating System), a new environment made for this kind of missions, on a Linux distribution called Ubuntu 20.04.

Then, I became familiar with ROS and its different applications, such as Moveit, used for the arm movements planification.

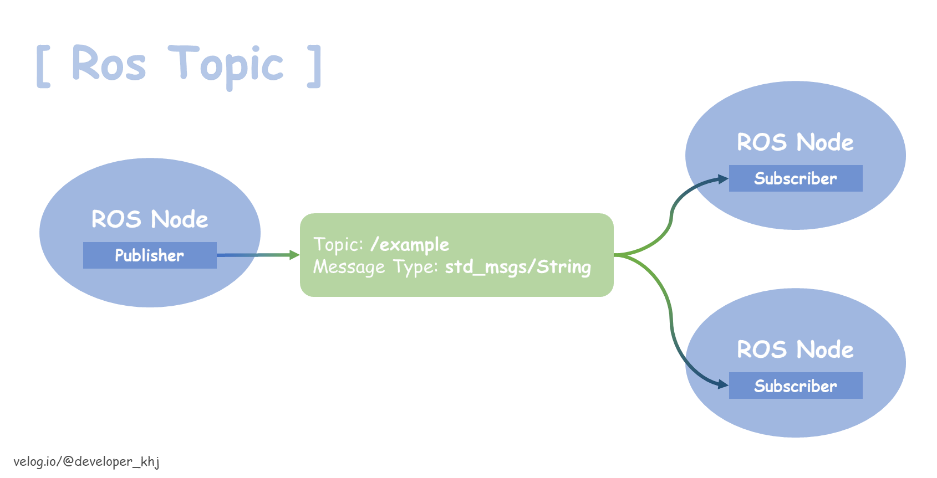


Figure 6 : ROS scheme

Here is how ROS is working. A program (in python or C++ language) is created. It can be a publisher or a subscriber, their role on the robotic moves depends on it. The publisher will publish some data to a topic. It can be everything : force, torque, temperature, acceleration… Then, another program (the subscriber) will get those data from this topic to use them (for moving the robotic arm for example). Those two programs are called nodes, and they can’t communicate, they have to pass through a dedicated topic.

For instance :

force\_data

In this case, a node called force\_sensor get the data from the force sensor and publishes them to the topic called force\_data. Then, another node called moving\_arm gets the data from the topic force\_data in order to move the robotic arm according to what the sensor gets.

## Moveit

Moveit is one of the most widely used motion planning frameworks in ROS. It contains a powerful set of tools for developing applications involving robot motion planning, manipulation, 3D perceptin, kinematics, control and collision detection. At the beginning, we first used Moveit and Rviz to move the robotic arm through our computer. We moved the arm inside the simulation and the arm moved also in the reality. Then, we used Moveit to calculate the inverse of a Jacobian matrix to move the arm with inverse kinematics. We also implemented some volumes to create a security space where the arm can’t go. Finally, we needed to be faster and to work with a high frequency, which were impossible to do with Moveit. That’s why we found a way to make the calculus ourselves and to not use Moveit.

Furthermore, we also used Moveit to implement the gripper we will use in the doll assembly, to predict which move the arm will do and to avoid a collision between the gripper and an arm’s joint.

# The robotic arm ur

During my internship, I worked with two collaborative robotic arms from Universal Robots : the UR5e and the UR30. Both of them belong to the e-series family and are used in industrial and research environments due to their flexibility, safety features and ease to integration.

## UR30



Figure 7 : UR30

The UR30 is a recent addition to the UR e-series lineup. It offers a higher payload capacity of 30kg while maintaining a reach of 1300mm with 6 degrees of freedom and a repeatability of 0.05mm more or less. It is more often used for heavy payload applications, and it will be used for the two aspect of the ROBOTA SUDOE project. But first, to make some test before using the UR30, we used a smaller Universal Robots : the UR5e.

## UR5e



Figure 8 : UR5e

The UR5e is a versatile 6-axis collaborative robot with a payload capacity of 5kg and a reach of 850mm. It is well-suited for tasks such as pick-and-place or testing and has a repeatability of 0.03mm more or less. That’s why we used it for testing our nodes and the security on our program.

However, during a test, the UR30 had a malfunction and we couldn’t use it then. There was a security message about a mechanical break who broke, which was not the case, but the arm were in security mode and we couldn’t use it anymore. That’s why we used the UR5e to run all our tests after that.

# SensONE T80 from BOTA SYSTEMS

The main goal of this project is to get the forces through sensors in order to help a human user to cut meat without any efforts. In that purpose, we are using two different captors SensONE T80 from BOTA SYSTEMS, one serial and the other etherCAT.

## Serial

The serial sensor is the first sensor we succeeded to use. Actually, it is easier to use it than the etherCAT one. All we needed to do was to download the right files and packages from the official website and then, we were able to get the data, it means the 3 forces and the 3 torques through a USB link. But in order to create an enslavement of our robotic arm, it is not enough. We have to publish those data in a topic we called /force\_sensor dedicated to this task : it’s the publisher. Then, we created another code to get those data and to adapt the UR30 behavior : it’s the subscriber who knows where to move the arm.



Figure 9 : Serial force sensor from BOTA SYSTEM

## EtherCAT

The etherCAT sensor is more complex than the serial one. First, it is linked through ethernet, which enable to be faster (about 10µs). To exchange data with the sensor, the Master (the computer) send a bus through the wire. Then, a slave (our sensor) fulfil the bus with its data (3 forces and 3 torques), and sends them back to the Master. Finaly, the Master read the data show them to the user. To do that, the sensor must have the access to the network card, which is quite complicated to get because of the computer’s safety.



Figure 10 : EtherCAT force sensor from BOTA SYSTEMS

However, we had an issue with those two sensors. At first, we used 3D printed supports on which were mounted the sensors. It worked but there were some slacks, that’s why we asked to a partner company, CENTIMFE, to make us the same support with metal. We claimed them two months after we arrived. Then, we decided to mount the sensors on those new supports, but they didn’t work anymore without knowing why. They sent None signal, so we unmounted them to try to get data, but the issue didn’t change. We contacted the support, and there is no explanation for this behaviour. We are assuming that maybe, when we mounted the sensor on the metallic supports, there was some static electricity which broke the inside of the two sensors, which is quite strange because it is industrial sensors, with electric norms, so this would be quite astonishing.

To keep doing some test with the UR5e, we decided to use the integrated sensor of the Universal Robots:



Figure 11 : Integrated sensor

The light grey part is an integrated sensor we can find in every Universal Robots. However, they are less accurate and it is harder to move the arm like that, that’s why we were using two others sensors from BOTA SYSTEMS. Finally, we succeeded to use it and keep doing some experiments with the arm.

# The knife

## 3D print



Figure 12 : 3D printed knife

To realize some tests, we needed to use a fake knife for more safety. That’s why we used 3D printed supports and 3D printed handle and blade. Later, we would need a more complex handle, but, at the beginning we used a simple and rigid handle to manipulate the robotic arm. Even after the issue with the metallic supports, we used this handle with this support. It was a lot easier to apply torques on the sensors then. We also decided to make the yellow support in one piece instead of three like it was the case before. Even if it is still plastic, it reduced the slacks.

Then, when we got the metallic ones from CENTIMFE, we just needed to print an adapted handle to mount it on the metallic support, since we had to change the shape of the handle to add a button and several sensors on it.

We also wanted to add a security button on the handle : when the user presses the button, the robotic arm turns on or turns off.

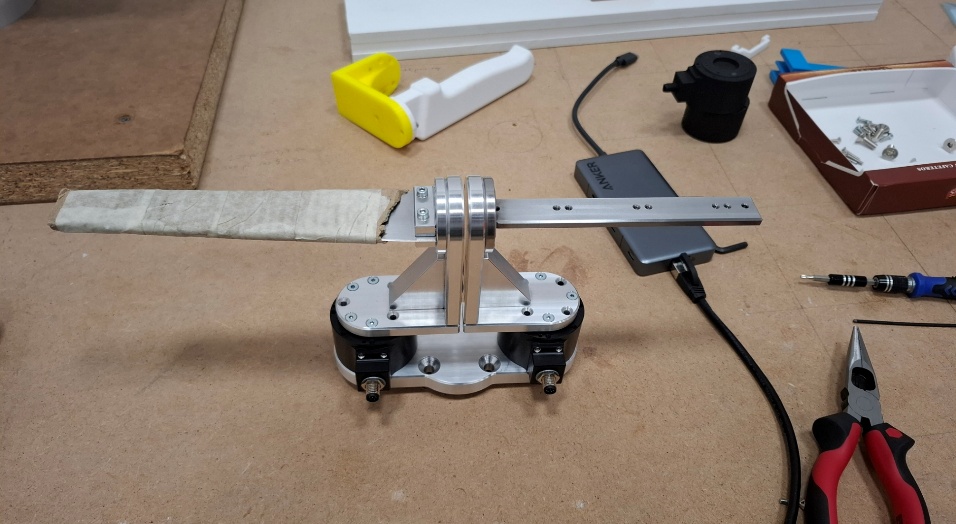


Figure 13 : Metallic supports from CENTIMFE

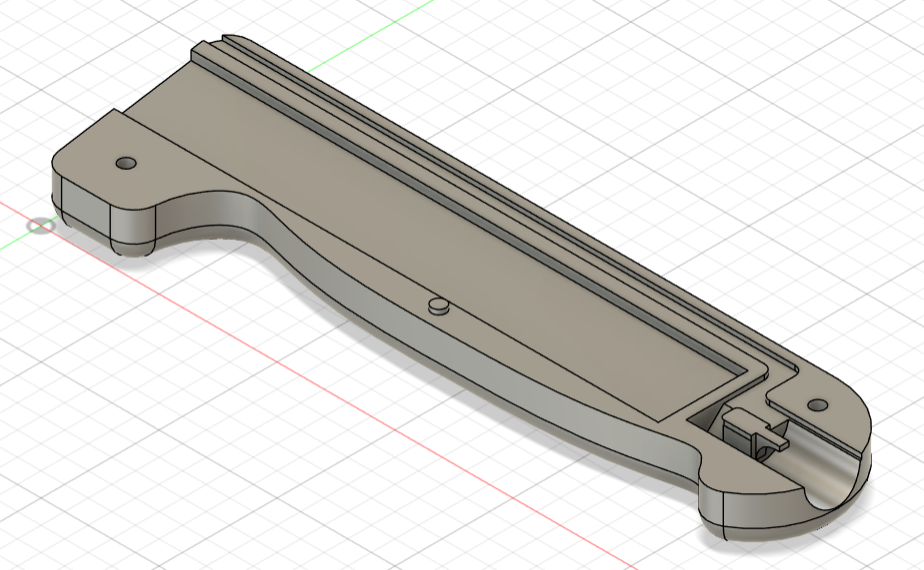


Figure 14 : Adapted handle for button and wires

## The handle

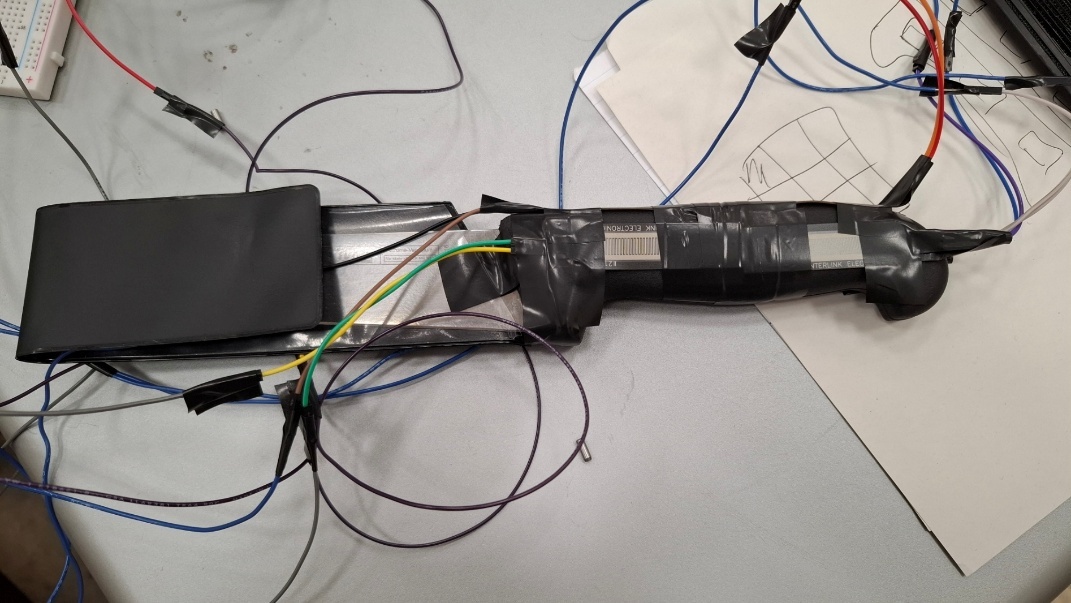


Figure 15 : The real knife

Once we succeeded to manipulate the robotic arm with the 3D printed knife, we were searching for a way to predict how we are holding the knife : in a standard way or in a reverse way. For that, we needed to put several tactile sensors on the handle. We put six with tape, two on a side, two on the other side and two above as we can see on the 1.10Figure 10. The purpose was to train a neural network to succeed to guess how we are holding it. At first, we created a csv file to study the data, and we noticed that on the six sensors, only two were required. But, to confirm that and to have more data, we kept the six sensors and we trained the neural network with them.



Figure 16 : Reverse mode



Figure 17 : Standard mode

However, there were some issues with the sensors we used on the handle fixed on the robotic arm : the wires were too much tough. That’s why, when we were mounting them, some sensors ‘pins broke as we can see on Figure 15.



Figure 18 : Sensors on the fixed handle

To fix that, we replaced the tough wires by flexible ones, so it was easier to move the sensors without damaging the pins. In addition to that, we oriented the wires in the same way (to the blade) to avoid to damage the pins.

### Arduino

In order to get the data from the sensors, we used the Arduino IDE with a Teensy card. We put a resistor and a sensor in a series, and we made this six times in parallel (Figure 12). Then, we could get and see the data on the serial monitor (Figure 11).

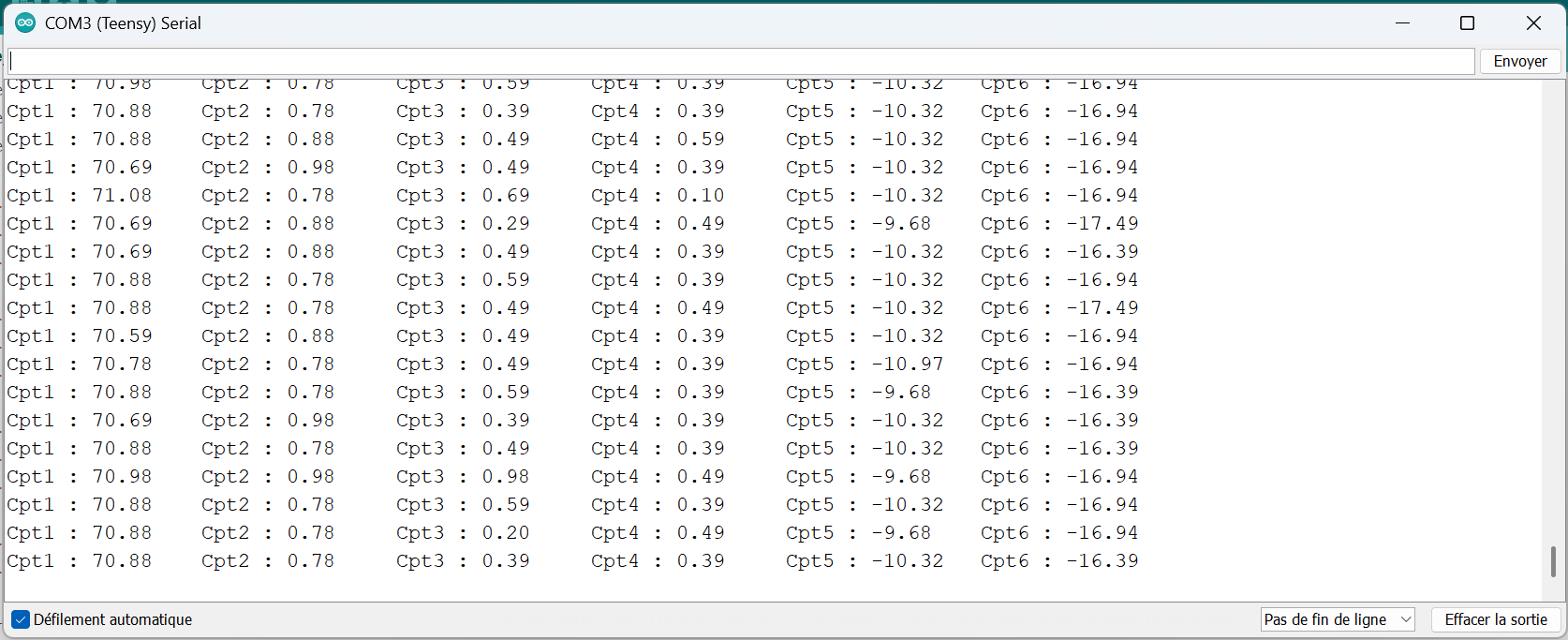


Figure 19 : Data from the sensors on the handle

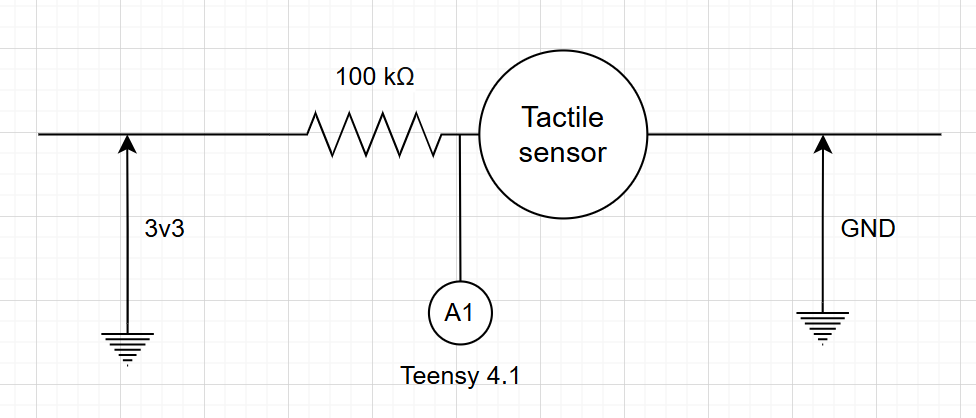


Figure 20 : Scheme for one tactile sensor

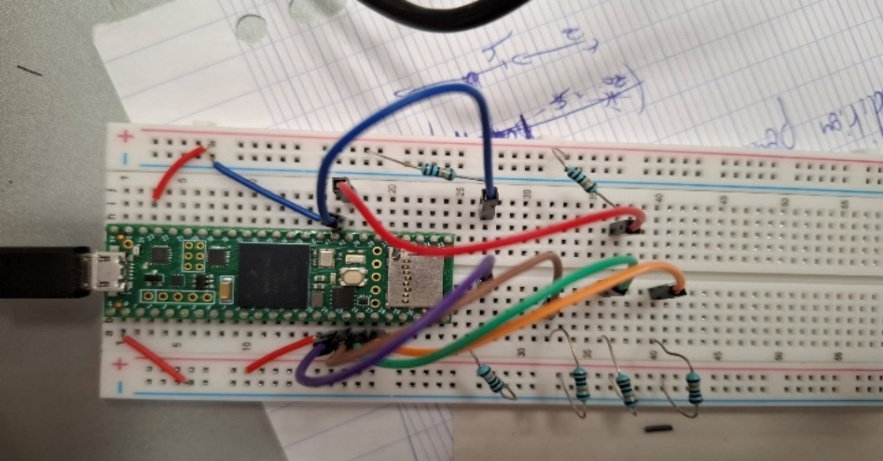


Figure 21 : Teensy set-up

First, we were using small sensors, but then we noticed that it didn’t fit according to the person holding the knife. Some would touch the sensors, and others wouldn’t. So, we decided to put longer sensors on it to become independent from the person holding the handle.

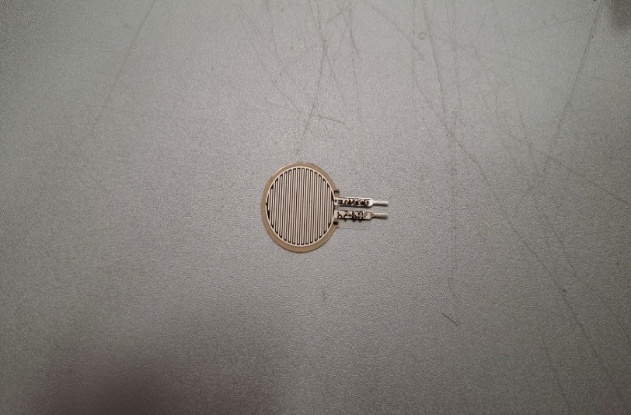


Figure 22 : Small tactile sensor

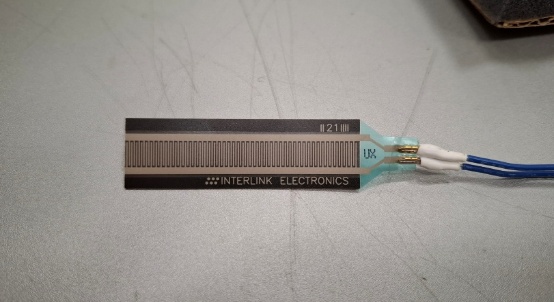


Figure 23 : Long tactile sensor

Then, we needed to make the data readable for the python code, that’s why the data send by the Teensy card were in this form : value1,value2,value3,value4,value5,value6.

### Python

At first, we tried to get the data on python to study them. In that purpose, we made two windows : one with the real time acquisition (Figure 14) to be sure that everything worked well, and a second one with a histogram where we could see the frequency of each value for each sensor every 5 seconds (Figure 13).

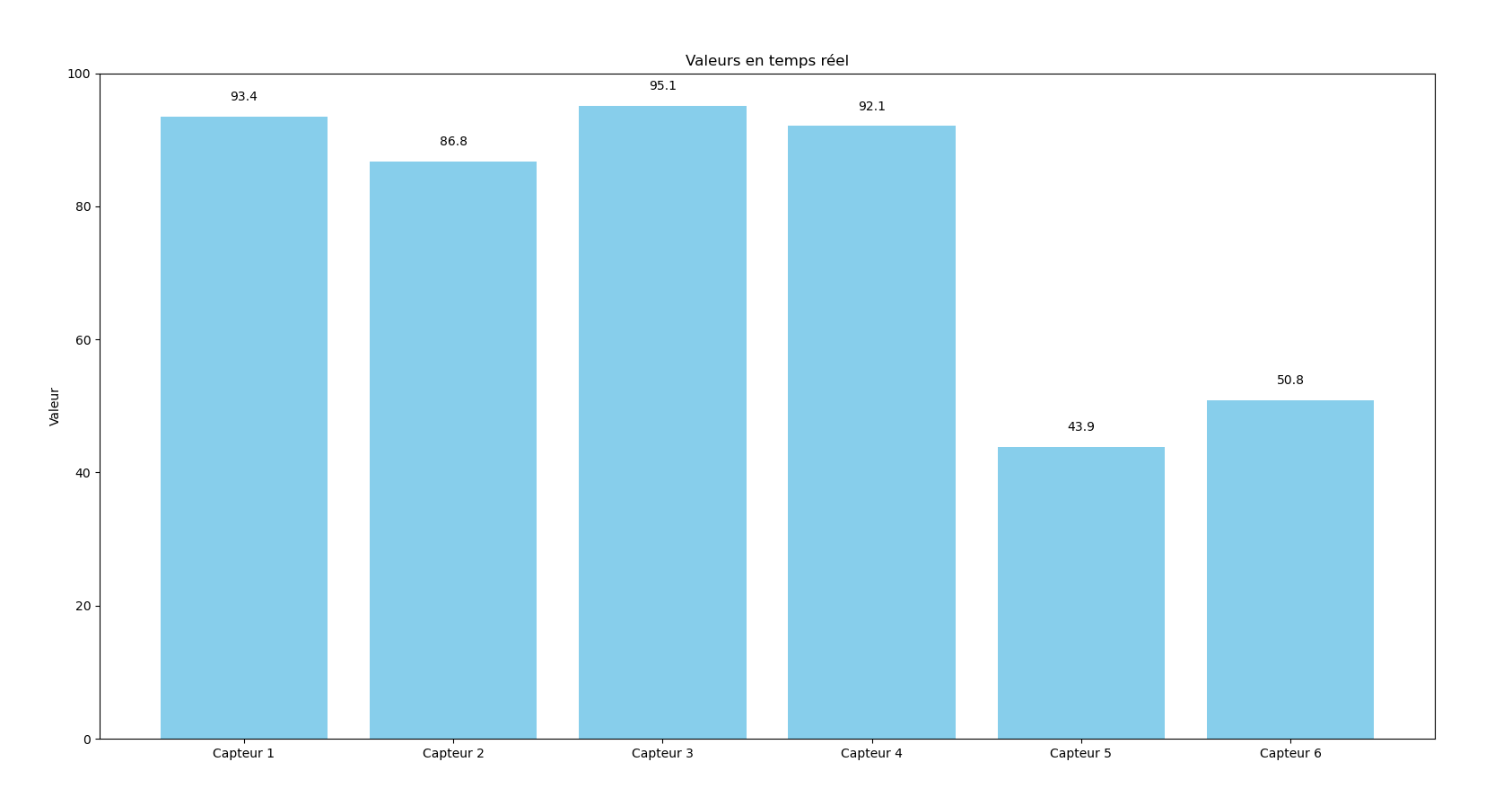


Figure 24 : Real time sensors

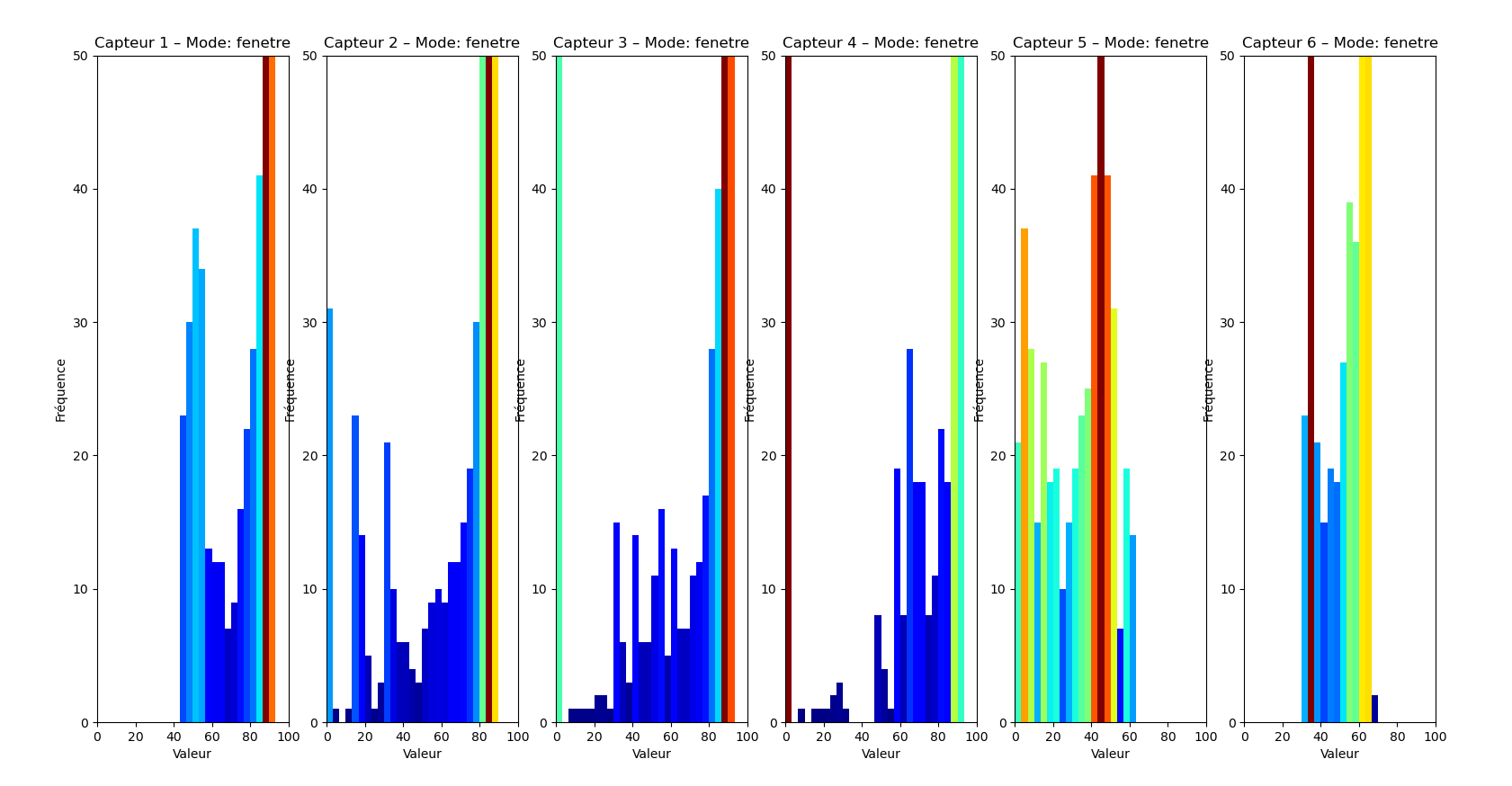


Figure 25 : Frequency of the sensors' figures

That’s how we noticed that there was a problem : the sensors were disconnected very easily, we couldn’t manipulate the handle properly. So, we soldered them.

The whole code is available at Annexe A. We have to take care of closing the serial port in the code before restarting it, otherwise Python couldn’t read the serial port.

### Neural network

In order to guess how we are holding the handle, we used a neural network to make the difference between three positions : when we are not holding the handle, when we’re holding it in a standard way and when we’re holding it in a reverse way.

The code provided in Annexe B creates a csv file in which there are the data for the three positions.

At the beginning, the program asks the user what he wants to do : the user can do an acquisition, a prediction or a training.

First, the user has to do an acquisition. The program asks the user if he wants to reinitialize the csv file (if it exists), then it asks how long the acquisition is for each position. Finally, the user can hold the handle through the three positions and complete the csv file automatically.

After the acquisition, the user has to train the neural network model.

Once those two steps done, the neural network can start to predict the new positions. First, we had several bad guesses because the set-up with the sensors was not optimal. Then, we soldered the sensors and the data were cleaner.

But the purpose evolved. We didn’t have to only guess in which position we are holding the handle, we also had to predict the movement right after. That’s why we decided to put the handle in real conditions on the UR5e to try to guess the movement.

So, we first tried with a flexible handle described at 1.15. The purpose was to guess if we are holding the handle in standard mode or in reverse mode, with a straight trajectory or a curved trajectory to the right or to the left. But we didn’t succeed to guess all the positions.

That’s why we tried with the 3D printed handle described at 1.9 and the six tactile sensors. There, the results were better, but there were still some positions we didn’t succeed to guess correctly. The code for this neural network is available at Annexe F.

Furthermore, since we were guessing a movement and not a position, we couldn’t just look at the sensors’ values one by one. That’s why the neural network is trained on a window of several consecutive values for each sensors.

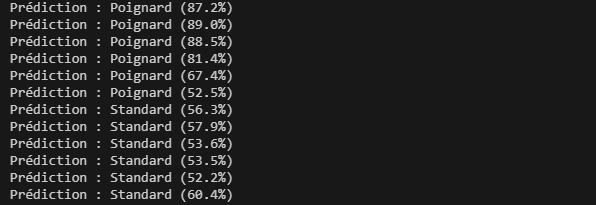


Figure 26 : Predictions of the neural network

Unfortunately, we didn’t succeed to use it properly. When I trained the neural network, it guessed perfectly how I was holding the handle, but when another person tried, it was unable to give a correct prediction. I assume that maybe there were not enough sensors on the handle, or maybe we should train the neural network with the butcher who would use it.

## Alternative with OnRobot HEX-E v2

Since the two Rokubi sensors broke, we didn’t have any sensors able to get data from the handle and from the blade, which was the point to use two different sensors. So, to get those missing data, we just used a system to prove that the arm is efficient. For the handle part, we used the sensors described in 1.10. Furthermore, we needed to be sure that the transmitted force was the same in the handle and the blade.

For the blade part, we used another sensor : the OnRobot HEX-E v2, fixed on the table. However, to fix it we needed to print an adaptor since it was made for a Universal Robot effector.

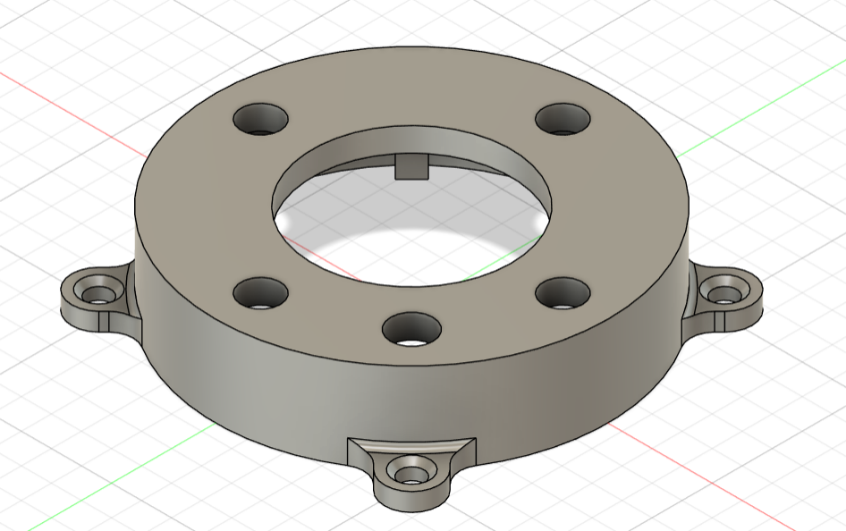


Figure 27 : Adaptor for the OnRobot sensor

I was surprised by the hole in the middle. I knew that the printer was not very accurate, so I designed it by adding 1mm to its diameter. But it didn’t fit yet. That’s how I learned that for common 3D printer, we need to add about 3mm in order to make fit holes correctly.



Figure 28 : Set-up to get the data from the blade

However, we encountered several issues with this sensor. There weren’t official user manual to provide us the code and the driver to use this sensor, that’s why we didn’t have a good frequency by using it.

## Flexible handle

To get data from the handle, we also used another kind of 3D printed material : a flexible plastic. Inside the handle, there are six sensors connected to a ESP32-WROOM-32D card. The code to see the result is available at Annexe C.

However, it is the first model and it is still a prototype, that’s why we noticed several issues by using it. First, we couldn’t identify which sensor is which. With the python program described at 1.12, we can see in real time the sensors’ values, but we couldn’t find which rectangle is associated to which sensor.

Furthermore, two of the six sensors were sending the value 0, like they didn’t work at all.

Finally, to try using this handle, we decided to test it with the neural network. Since we were not able to find which sensor was doing what, we thought that the neural network could make the difference to guess the way we were holding the handle. The code used is given at Annexe D. Unfortunately, even if the neural network could make the difference between the three positions, its accuracy was quite low. In fact, the way we are holding the handle in a classical way or in reverse way are very similar. That’s why we thought to rise the size of the training data and to take different persons to take the handle. In that case, we could access to several ways to hold the knife for a same position in order to improve the accuracy of the neural network prediction.

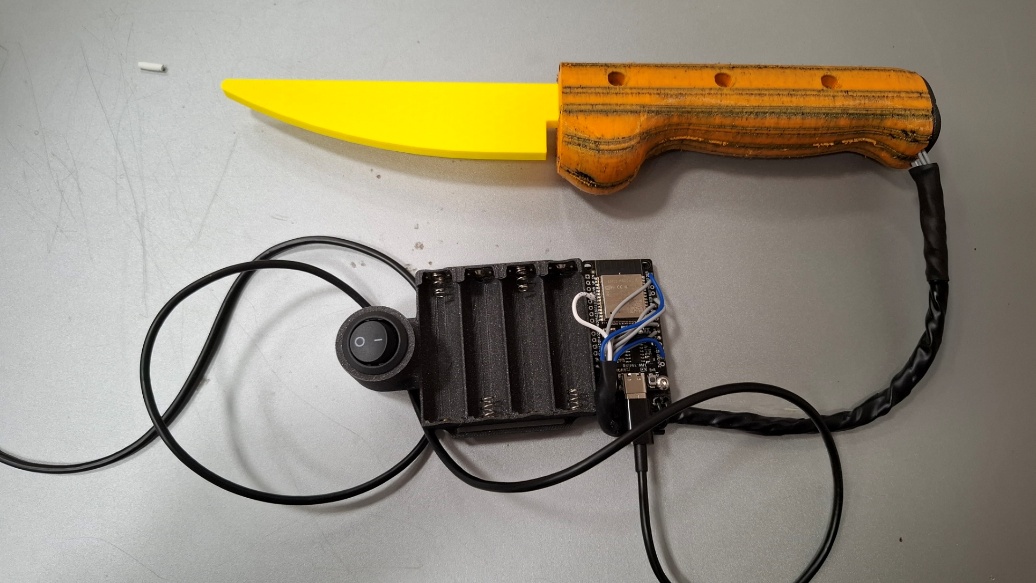


Figure 29 : Flexible handle with integrated sensors

At first, we used the neural network with this handle, but the low accuracy didn’t enable us to guess correctly the way we are holding it.

# Side tasks

## UR30 adaptor

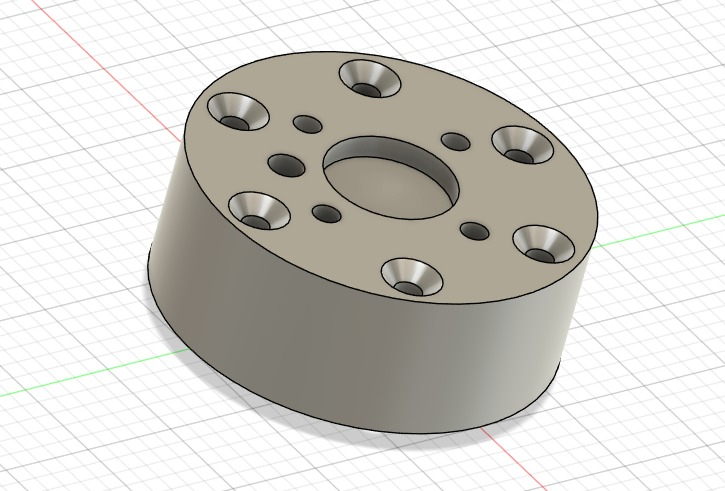


Figure 30 : UR30 adaptor

When we arrived, there was a sensor from BOTA SYSTEM mounted on the UR30, and on this sensor, there was the 3D printed supports of the knife. The purpose was to use it in addition to the integrated sensor and the two Rokubi sensors, to see which one is more efficient. However, the sensor has been broken. At first, we kept it to keep using the 3D printed support, but then we send it back to make it fix. So, I created this piece to mount the supports on the UR30 to proceed some tests on it instead of the UR5e.

The final mounting is shown in Figure 32.

## 3D printed serial sensor



Figure 31 : 3D printed serial sensor

In order to realize some tests, we needed to manipulate the robotic arm UR5e. However, without the serial sensor mounted on it, we first used a plastic model sent by CENTIMFE to mount the blade on the metallic support. But quickly we found that the blade was not stable, so the data got weren’t accurate. That’s why printed the same model as the serial sensor, but with placements we could use to screw with bolts (Figure 31).

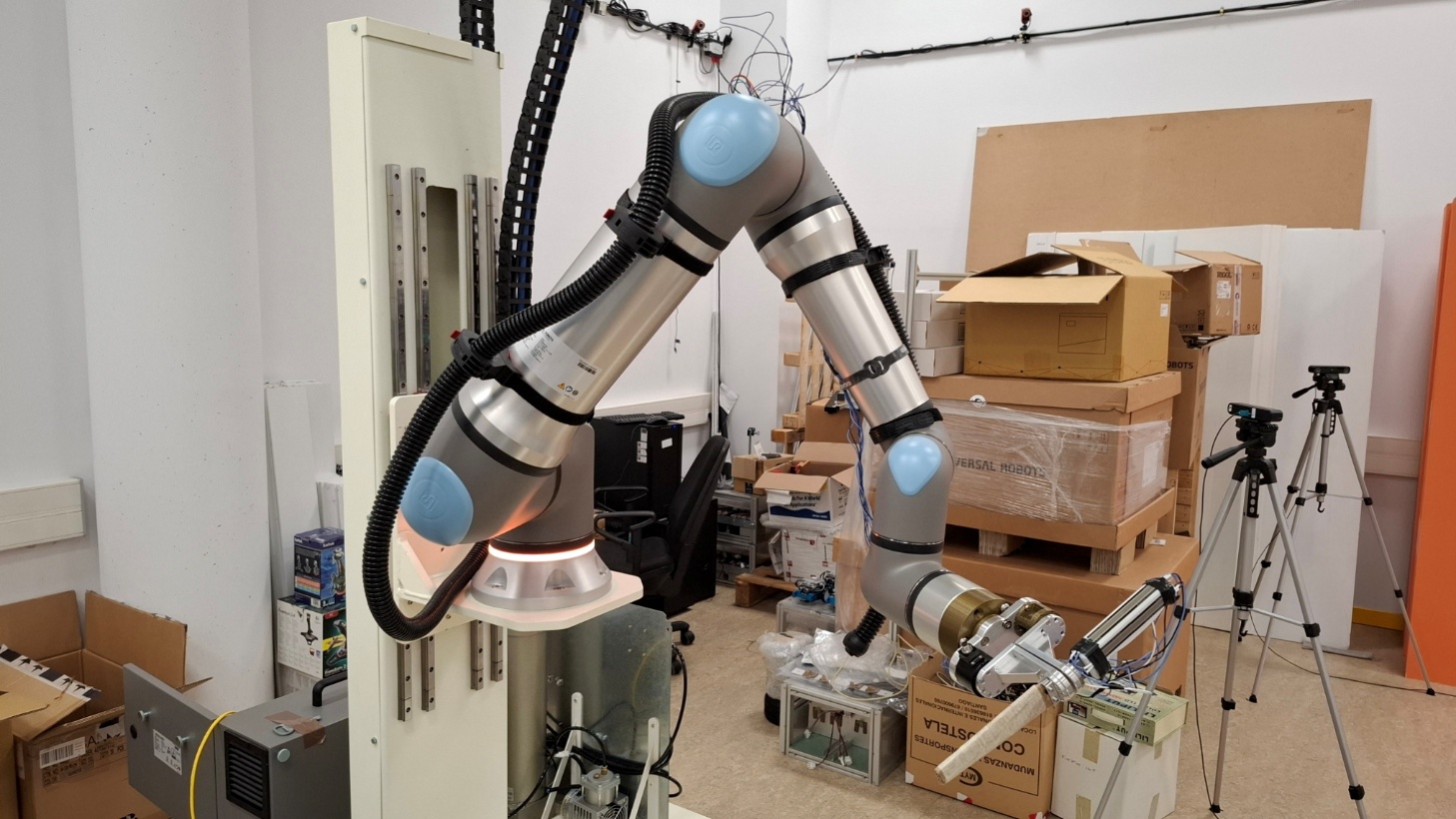


Figure 32 : Final mounting

## 3D simulation of the handle

### Sensors view

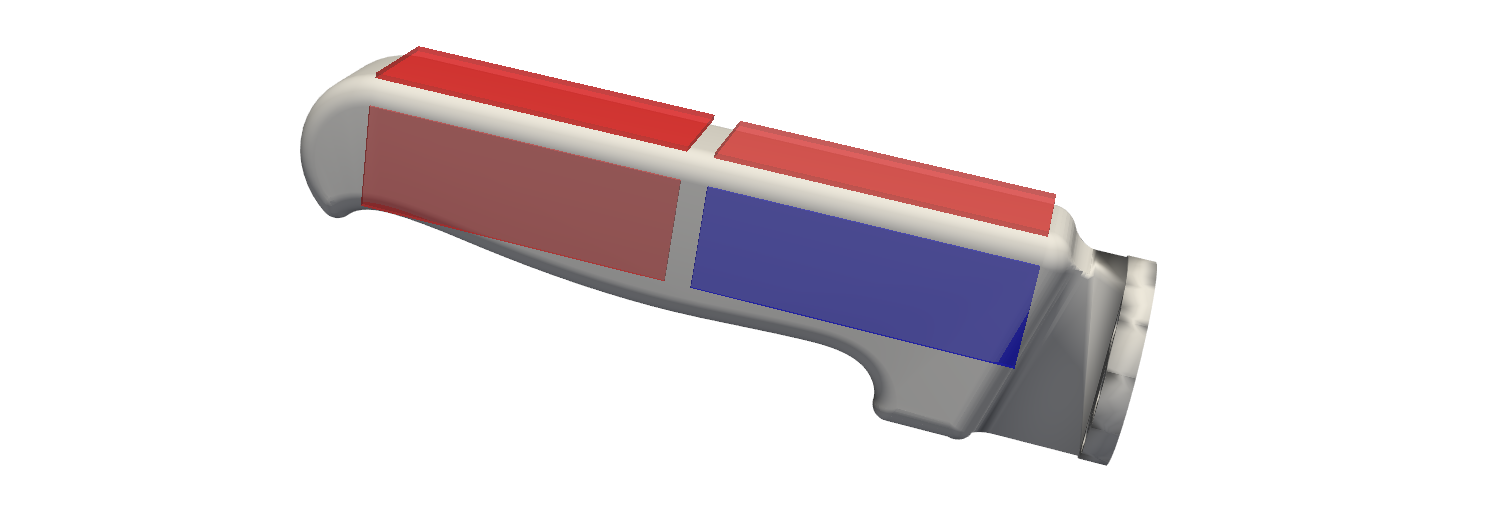


Figure 33 : 3D simulation with the handle and the sensors

With python, I could import the STL file of the handle to display it on a screen with the library **pyvista**. Above this model, I added several thin rectangles to imitate the six sensors. Then, I used it in tseveral ways. First, I connected the rectangles with the real ones in real time through a serial port and the Arduino card connected to the computer. Then, when we manipulate the handle, we can see thanks to a colour gradient which sensor is the most solicitated (dark blue when it is not and dark red when it is). The code is available at Annexe G.

Then, after doing several tests, we collected several data on csv files. I used those csv files to recreate the simulation with this model. So, if we want to replay the moment and just see the way sensor are solicitated through the test, we could use this code available in Annexe H.

### Manipulability ellipse

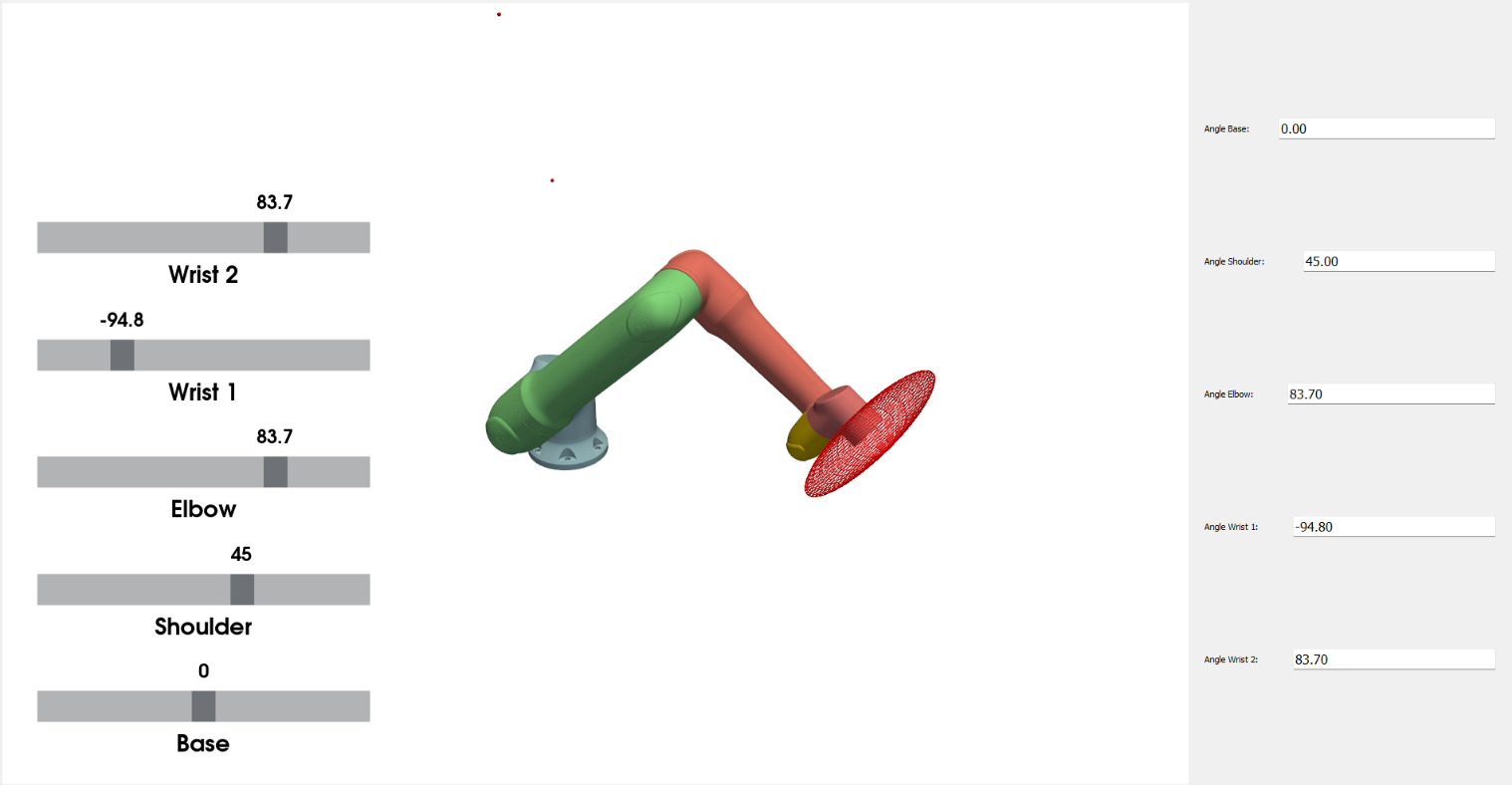


Figure 34 : Simulation of the robotic arm UR30 with its manipulability ellipse

Then, we thought it could be useful to have a visibility upon the manipulability ellipse of the robotic arm. This ellipse shows the arm’s ability to move in the space according to the way it is. For example, if all the joints are straight, the ellipse will be flat because the arm could move in only one direction. It is useful to see the singularity for instance. First, I assembled the robotic arm in the graphic window. I downloaded the official fusion file from OnRobot website, then I isolated the different parts to turn them into a STL file. Then I placed them in the graphic window to display the UR30. To be able to move the arm, I associated each part to a cursor to move them. It is the direct kinematic. In order to be more precise, I also put an input window to set the joint angles as we want. After doing that, we needed to see the manipulability ellipse. I wrote the code at Annexe I to calculate the Jacobian matrix in order to know in which direction the robot can move, to display the ellipse. Then with the code provided at Annexe J, We can display the ellipse associated to the end effector as we can see in Figure 34.

# Conclusion

# Annexe

Annexe A : Project GitHub link

[Protteo/Capteurs\_tactiles](https://github.com/Protteo/Capteurs_tactiles)

Annexe B : Python code to get data from the sensors

import serial

import matplotlib.pyplot as plt

import matplotlib.animation as animation

from collections import deque

import threading

import numpy as np

import time

from matplotlib.colors import LinearSegmentedColormap

from matplotlib.patches import Rectangle

from mpl\_toolkits.mplot3d import Axes3D

import matplotlib.widgets as widgets

# --------------------- CONFIGURATION ---------------------

mode = "fenetre" # "continu" ou "fenetre"

interval\_update\_sec = 1 # fréquence de mise à jour histogramme (en secondes)

num\_capteurs = 6

port\_serial = "COM3"

baudrate = 9600

acquisition\_active = True

# --------------------- INITIALISATION ---------------------

ser = serial.Serial(port\_serial, baudrate, timeout=1)

def on\_close(event):

global fenetres\_ouvertes, acquisition\_active, thread\_acq

fenetres\_ouvertes -= 1

if fenetres\_ouvertes == 0:

print("Toutes les fenêtres sont fermées. Fermeture du port série...")

acquisition\_active = False

if thread\_acq is not None:

thread\_acq.join(timeout=2) # Attend la fin du thread (max 2s)

try:

ser.close()

print("Port série fermé proprement.")

except:

print("Erreur lors de la fermeture du port série.")

labels = [f"Capteur {i+1}" for i in range(num\_capteurs)]

data\_buffers = [deque() for \_ in range(num\_capteurs)] # Historique complet

data\_fenetre = [deque() for \_ in range(num\_capteurs)] # Fenêtre glissante 5s

valeurs\_en\_temps\_reel = [0.0 for \_ in range(num\_capteurs)] # Dernières valeurs

# Nouveau : pour gérer la fermeture propre des deux fenêtres

fenetres\_ouvertes = 4 # nombre total de fenêtres à gérer

lock = threading.Lock()

# --------------------- THREAD DE LECTURE SERIE ---------------------

def lire\_serial():

global acquisition\_active

while acquisition\_active:

try:

line = ser.readline().decode(errors='ignore').strip()

valeurs = list(map(float, line.split(",")))

if len(valeurs) != num\_capteurs:

continue

now = time.time()

with lock:

for i in range(num\_capteurs):

val = valeurs[i]

valeurs\_en\_temps\_reel[i] = val

data\_buffers[i].append(val)

data\_fenetre[i].append((now, val))

except Exception:

continue

thread\_acq = threading.Thread(target=lire\_serial, daemon=True)

thread\_acq.start()

# Petit délai pour laisser l'acquisition série démarrer

# time.sleep(1)

# --------------------- FENETRE 1 : HISTOGRAMMES ---------------------

fig\_freq, axs\_freq = plt.subplots(1, num\_capteurs, figsize=(6 \* num\_capteurs, 4))

if num\_capteurs == 1:

axs\_freq = [axs\_freq]

def update\_histogram(frame):

with lock:

now = time.time()

snapshots = []

for i in range(num\_capteurs):

if mode == "fenetre":

while data\_fenetre[i] and now - data\_fenetre[i][0][0] > 5:

data\_fenetre[i].popleft()

valeurs = [v for (t, v) in data\_fenetre[i]]

else:

valeurs = list(data\_buffers[i])

snapshots.append(valeurs)

for i in range(num\_capteurs):

ax = axs\_freq[i]

ax.clear()

ax.set\_title(f"Capteur {i+1} – Mode: {mode}")

ax.set\_xlim(0, 100)

ax.set\_ylim(0, 50)

ax.set\_xlabel("Valeur")

ax.set\_ylabel("Fréquence")

valeurs = snapshots[i]

if valeurs:

counts, bins = np.histogram(valeurs, bins=30, range=(0, 100))

colors = plt.cm.jet(counts / counts.max()) if counts.max() > 0 else 'gray'

ax.bar(bins[:-1], counts, width=(bins[1] - bins[0]), color=colors, align='edge')

else:

ax.text(0.5, 0.5, "Aucune donnée", ha='center', va='center', transform=ax.transAxes)

ani\_freq = animation.FuncAnimation(fig\_freq, update\_histogram, interval=interval\_update\_sec \* 1000)

# --------------------- FENETRE 2 : TEMPS REEL ---------------------

fig\_rt, ax\_rt = plt.subplots()

bars = ax\_rt.bar(labels, [0] \* num\_capteurs, color='skyblue')

ax\_rt.set\_ylim(0, 4096)

ax\_rt.set\_title("Valeurs en temps réel")

ax\_rt.set\_ylabel("Valeur")

text\_labels = [ax\_rt.text(i, 0, "", ha='center', va='bottom') for i in range(num\_capteurs)]

def update\_realtime(frame):

with lock:

snapshot = list(valeurs\_en\_temps\_reel)

for i, bar in enumerate(bars):

val = snapshot[i]

bar.set\_height(val)

text\_labels[i].set\_text(f"{val:.1f}")

text\_labels[i].set\_y(val + 2)

ani\_rt = animation.FuncAnimation(fig\_rt, update\_realtime, interval=100)

#----------------------Ferme le port série proprement

# Attache cette fonction aux deux fenêtres

fig\_freq.canvas.mpl\_connect("close\_event", on\_close)

fig\_rt.canvas.mpl\_connect("close\_event", on\_close)

# --------------------- FENETRE 3 : 3D MANCHE + CENTRE DES FORCES

fig\_3d = plt.figure()

ax\_3d = fig\_3d.add\_subplot(111, projection='3d')

fig\_3d.canvas.mpl\_connect("close\_event", on\_close)

# Coordonnées fixes des capteurs [x, y, z] en cm

capteurs\_coords = np.array([

[2, -1.5, 0.5], # Capteur 1

[2, +1.5, 0.5], # Capteur 2

[8, -1.5, 0.5], # Capteur 3

[8, +1.5, 0.5], # Capteur 4

[8, 0.0, 2.0], # Capteur 5

[2, 0.0, 2.0], # Capteur 6

])

# Manche dimensions (pour affichage)

manche\_length = 10

manche\_width = 3

manche\_height = 2

# Boîte du manche pour visuel

def draw\_manche(ax):

# Crée les coins d'une boîte

from itertools import product, combinations

r = [0, manche\_length]

w = [-manche\_width/2, manche\_width/2]

h = [0, manche\_height]

points = list(product(r, w, h))

for s, e in combinations(points, 2):

if sum([s[i] != e[i] for i in range(3)]) == 1:

ax.plot3D(\*zip(s, e), color="gray", alpha=0.4)

def update\_3d(frame):

with lock:

forces = np.array(valeurs\_en\_temps\_reel)

ax\_3d.clear()

draw\_manche(ax\_3d)

ax\_3d.set\_xlim(0, manche\_length)

ax\_3d.set\_ylim(-manche\_width/2 - 1, manche\_width/2 + 1)

ax\_3d.set\_zlim(0, manche\_height + 1)

ax\_3d.set\_title("Manche + Centre des forces")

ax\_3d.set\_xlabel("X (longueur)")

ax\_3d.set\_ylabel("Y (largeur)")

ax\_3d.set\_zlabel("Z (hauteur)")

# Affiche capteurs

for i, (x, y, z) in enumerate(capteurs\_coords):

ax\_3d.scatter(x, y, z, color='blue')

ax\_3d.text(x, y, z + 0.2, f"{i+1}", color='blue')

# Calcul du centre de force

if np.sum(forces) > 0:

center\_force = np.average(capteurs\_coords, axis=0, weights=forces)

ax\_3d.scatter(\*center\_force, color='red', s=100, label='Centre des forces')

ax\_3d.legend()

ani\_3d = animation.FuncAnimation(fig\_3d, update\_3d, interval=200)

# --------------------- FENETRE 4 : AFFICHAGE DES 3 FLANCS ---------------------

fig\_color, ax\_color = plt.subplots()

fig\_color.canvas.mpl\_connect("close\_event", on\_close)

fig\_color.subplots\_adjust(bottom=0.2)

# Rectangle + 2 zones par face (gauche, haut, droite)

rects = {}

positions = [0.2, 0.5, 0.8]

capteurs\_zones = [(0, 2), (5, 4), (1, 3)] # Gauche, Haut, Droite

# Couleur personnalisée vert -> rouge

green\_red = LinearSegmentedColormap.from\_list("green\_red", ["blue", "red"])

acquisition\_en\_cours = True

start\_time = time.time()

donnees\_10s = [[] for \_ in range(num\_capteurs)]

# Zones dessinées

for i, x in enumerate(positions):

rects[f"{i}\_top"] = ax\_color.add\_patch(Rectangle((x - 0.05, 0.6), 0.1, 0.3, color='gray'))

rects[f"{i}\_bot"] = ax\_color.add\_patch(Rectangle((x - 0.05, 0.2), 0.1, 0.3, color='gray'))

ax\_color.set\_xlim(0, 1)

ax\_color.set\_ylim(0, 1)

ax\_color.axis('off')

ax\_color.set\_title("Zones de pression sur le manche (10s)")

# Bouton réinitialiser

ax\_button = plt.axes([0.4, 0.05, 0.2, 0.075])

button = widgets.Button(ax\_button, 'Réinitialiser')

def reset\_acquisition(event):

global acquisition\_en\_cours, start\_time, donnees\_10s

acquisition\_en\_cours = True

start\_time = time.time()

donnees\_10s = [[] for \_ in range(num\_capteurs)]

for r in rects.values():

r.set\_color("gray")

button.on\_clicked(reset\_acquisition)

def update\_color(frame):

global acquisition\_en\_cours

if not acquisition\_en\_cours:

return

now = time.time()

with lock:

for i in range(num\_capteurs):

donnees\_10s[i].append(valeurs\_en\_temps\_reel[i])

if now - start\_time >= 10:

acquisition\_en\_cours = False

for idx, (top\_id, bot\_id) in enumerate(capteurs\_zones):

moyenne\_top = np.mean(donnees\_10s[top\_id])

moyenne\_bot = np.mean(donnees\_10s[bot\_id])

rects[f"{idx}\_top"].set\_color(green\_red(moyenne\_top / 100))

rects[f"{idx}\_bot"].set\_color(green\_red(moyenne\_bot / 100))

ani\_color = animation.FuncAnimation(fig\_color, update\_color, interval=200)

# --------------------- AFFICHAGE FINAL

plt.show()

Annexe C : Neural network code

import serial

import pandas as pd

import time

import os

import numpy as np

from sklearn.neural\_network import MLPClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

import joblib

PORT\_SERIE = 'COM3'

BAUDRATE = 9600

N\_CAPTEURS = 2

CSV\_FILENAME = "donnees\_manche\_2\_cpt.csv"

SCALER\_FILENAME = "scaler\_manche.pkl"

MODEL\_FILENAME = "modele\_manche.pkl"

TEMPS\_ACQUISITION\_PAR\_POSITION = 60 # secondes

def initialiser\_csv():

if os.path.exists(CSV\_FILENAME):

choix = input("Souhaitez-vous réinitialiser le fichier CSV ? (o/n) : ").lower()

if choix == 'o':

os.remove(CSV\_FILENAME)

print("Fichier réinitialisé.")

else:

print("Les nouvelles données seront ajoutées au fichier existant.")

else:

print("Le fichier n'existe pas, il sera créé.")

def lire\_donnees\_serie(ser):

try:

ligne = ser.readline().decode('utf-8').strip()

valeurs = ligne.split(',')

if len(valeurs) == N\_CAPTEURS:

return [float(v) for v in valeurs]

except Exception:

pass

return None

def acquisition\_par\_positions():

try:

ser = serial.Serial(PORT\_SERIE, BAUDRATE, timeout=1)

print("Connexion série ouverte.")

time.sleep(2)

position = 0

data\_total = []

print("\nDébut de l'acquisition pour la position 0.")

print("Appuyez sur Entrée pour passer à la position suivante.")

while position <= 2:

print(f"\n➡ Acquisition pour la position {position}.")

input("→ Appuyez sur Entrée pour commencer l'acquisition pour cette position.")

try:

duree = float(input(f"⏱️ Entrez la durée d'acquisition en secondes pour la position {position} : "))

except ValueError:

print("Entrée invalide. Utilisation de 60 secondes par défaut.")

duree = 60

start\_time = time.time()

while True:

if ser.in\_waiting:

donnees = lire\_donnees\_serie(ser)

if donnees:

donnees.append(position)

data\_total.append(donnees)

print(f"Position {position} : {donnees[:-1]}")

if time.time() - start\_time > duree:

break

position += 1

ser.close()

colonnes = [f"capteur\_{i+1}" for i in range(N\_CAPTEURS)] + ["classe"]

df = pd.DataFrame(data\_total, columns=colonnes)

if os.path.exists(CSV\_FILENAME):

df.to\_csv(CSV\_FILENAME, mode='a', index=False, header=False)

else:

df.to\_csv(CSV\_FILENAME, index=False)

print(f"\n📁 Données enregistrées dans {CSV\_FILENAME}")

except Exception as e:

print(f"Erreur : {e}")

def entrainer\_modele():

if not os.path.exists(CSV\_FILENAME):

print("Aucune donnée d'entraînement trouvée.")

return None, None

df = pd.read\_csv(CSV\_FILENAME)

X = df.iloc[:, :-1].values

y = df.iloc[:, -1].values

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

joblib.dump(scaler, SCALER\_FILENAME)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2)

model = MLPClassifier(hidden\_layer\_sizes=(12, 12), max\_iter=300)

model.fit(X\_train, y\_train)

print(f"Précision sur test : {model.score(X\_test, y\_test)\*100:.2f}%")

joblib.dump(model, MODEL\_FILENAME)

print("🧠 Modèle entraîné et sauvegardé.")

return model, scaler

def prediction\_temps\_reel():

if not os.path.exists(MODEL\_FILENAME) or not os.path.exists(SCALER\_FILENAME):

model, scaler = entrainer\_modele()

if model is None:

print("Impossible de lancer la prédiction sans modèle.")

return

else:

model = joblib.load(MODEL\_FILENAME)

scaler = joblib.load(SCALER\_FILENAME)

ser = serial.Serial(PORT\_SERIE, BAUDRATE, timeout=1)

print("🔮 Prédictions en temps réel. Appuyez sur Ctrl+C pour arrêter.\n")

try:

while True:

if ser.in\_waiting:

donnees = lire\_donnees\_serie(ser)

if donnees:

entree = scaler.transform([donnees])

pred = model.predict(entree)[0]

proba = np.max(model.predict\_proba(entree)) \* 100

print(f"Prédiction : position {pred} (confiance : {proba:.1f}%)")

except KeyboardInterrupt:

print("\n⛔ Arrêt des prédictions.")

ser.close()

if \_\_name\_\_ == "\_\_main\_\_":

mode = input("Choisissez le mode : (a)cquisition, (p)rédiction ou (e)ntraînement ? : ").lower()

if mode == 'a':

initialiser\_csv()

acquisition\_par\_positions()

elif mode == 'p':

prediction\_temps\_reel()

elif mode == 'e':

entrainer\_modele()

else:

print("Mode inconnu.")

Annexe D : Flexible handle code

import serial

import matplotlib.pyplot as plt

import matplotlib.animation as animation

import re

# Paramètres de connexion série

SERIAL\_PORT = 'COM5' # À adapter

BAUDRATE = 115200

# Initialisation de la liaison série

ser = serial.Serial(SERIAL\_PORT, BAUDRATE, timeout=1)

# Initialisation du graphique

fig, ax = plt.subplots()

bar\_labels = [f'R{i+1}' for i in range(6)] # 6 capteurs

bar\_values = [0] \* 6

bars = ax.bar(bar\_labels, bar\_values, color='skyblue')

value\_texts = [ax.text(i, 0, '', ha='center', va='bottom') for i in range(6)]

ax.set\_ylim(0, 2000)

ax.set\_title("Valeurs brutes des capteurs R1 à R6")

ax.set\_ylabel("Valeur brute")

# Fonction pour parser une ligne de type : "R1:565 R2:343 ... R6:XXX BTN:0"

def parse\_line(line):

matches = re.findall(r'R(\d):(\d+)', line)

values = [0] \* 6

for sensor, val in matches:

index = int(sensor) - 1

if 0 <= index < 6:

values[index] = int(val)

return values

# Mise à jour du graphique

def update(frame):

try:

line = ser.readline().decode('utf-8').strip()

if line:

values = parse\_line(line)

for i, val in enumerate(values):

bars[i].set\_height(val)

value\_texts[i].set\_text(str(val))

value\_texts[i].set\_y(val + 20)

except Exception as e:

print("Erreur :", e)

# Fonction appelée à la fermeture de la figure

def on\_close(event):

print("Fermeture de la fenêtre. Libération du port série...")

if ser.is\_open:

ser.close()

# Connexion de l'événement de fermeture

fig.canvas.mpl\_connect('close\_event', on\_close)

ani = animation.FuncAnimation(fig, update, interval=50)

plt.show()

Annexe E : Neural network program for flexible handle

import serial

import pandas as pd

import time

import os

import numpy as np

from sklearn.neural\_network import MLPClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

import joblib

import re

PORT\_SERIE = 'COM4'

BAUDRATE = 115200

N\_CAPTEURS = 6

CSV\_FILENAME = "donnees\_manche\_souple.csv"

SCALER\_FILENAME = "scaler\_manche.pkl"

MODEL\_FILENAME = "modele\_manche.pkl"

def initialiser\_csv():

if os.path.exists(CSV\_FILENAME):

choix = input("Souhaitez-vous réinitialiser le fichier CSV ? (o/n) : ").lower()

if choix == 'o':

os.remove(CSV\_FILENAME)

print("Fichier réinitialisé.")

else:

print("Les nouvelles données seront ajoutées au fichier existant.")

else:

print("Le fichier n'existe pas, il sera créé.")

def lire\_donnees\_serie(ser):

try:

ligne = ser.readline().decode('utf-8').strip()

matches = re.findall(r'R(\d):(\d+)', ligne)

values = [0] \* N\_CAPTEURS

for sensor, val in matches:

idx = int(sensor) - 1

if 0 <= idx < N\_CAPTEURS:

values[idx] = int(val)

return values

except Exception:

pass

return None

def acquisition\_par\_positions():

try:

ser = serial.Serial(PORT\_SERIE, BAUDRATE, timeout=1)

print("Connexion série ouverte.")

time.sleep(2)

position = 0

data\_total = []

print("\nDébut de l'acquisition pour la position 0.")

print("Appuyez sur Entrée pour passer à la position suivante.")

while position <= 2:

print(f"\n➡ Acquisition pour la position {position}.")

input("→ Appuyez sur Entrée pour commencer l'acquisition pour cette position.")

try:

duree = float(input(f"⏱️ Entrez la durée d'acquisition en secondes pour la position {position} : "))

except ValueError:

print("Entrée invalide. Utilisation de 60 secondes par défaut.")

duree = 60

start\_time = time.time()

while True:

if ser.in\_waiting:

donnees = lire\_donnees\_serie(ser)

if donnees:

donnees.append(position)

data\_total.append(donnees)

print(f"Position {position} : {donnees[:-1]}")

if time.time() - start\_time > duree:

break

position += 1

ser.close()

colonnes = [f"capteur\_{i+1}" for i in range(N\_CAPTEURS)] + ["classe"]

df = pd.DataFrame(data\_total, columns=colonnes)

if os.path.exists(CSV\_FILENAME):

df.to\_csv(CSV\_FILENAME, mode='a', index=False, header=False)

else:

df.to\_csv(CSV\_FILENAME, index=False)

print(f"\n📁 Données enregistrées dans {CSV\_FILENAME}")

except Exception as e:

print(f"Erreur : {e}")

def entrainer\_modele():

if not os.path.exists(CSV\_FILENAME):

print("Aucune donnée d'entraînement trouvée.")

return None, None

df = pd.read\_csv(CSV\_FILENAME)

X = df.iloc[:, :-1].values

y = df.iloc[:, -1].values

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

joblib.dump(scaler, SCALER\_FILENAME)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2)

model = MLPClassifier(hidden\_layer\_sizes=(12, 12), max\_iter=300)

model.fit(X\_train, y\_train)

print(f"Précision sur test : {model.score(X\_test, y\_test)\*100:.2f}%")

joblib.dump(model, MODEL\_FILENAME)

print("🧠 Modèle entraîné et sauvegardé.")

return model, scaler

def prediction\_temps\_reel():

if not os.path.exists(MODEL\_FILENAME) or not os.path.exists(SCALER\_FILENAME):

model, scaler = entrainer\_modele()

if model is None:

print("Impossible de lancer la prédiction sans modèle.")

return

else:

model = joblib.load(MODEL\_FILENAME)

scaler = joblib.load(SCALER\_FILENAME)

ser = serial.Serial(PORT\_SERIE, BAUDRATE, timeout=1)

print("🔮 Prédictions en temps réel. Appuyez sur Ctrl+C pour arrêter.\n")

try:

while True:

if ser.in\_waiting:

donnees = lire\_donnees\_serie(ser)

if donnees and len(donnees) == N\_CAPTEURS:

entree = scaler.transform([donnees])

pred = model.predict(entree)[0]

proba = np.max(model.predict\_proba(entree)) \* 100

print(f"Prédiction : position {pred} (confiance : {proba:.1f}%)")

except KeyboardInterrupt:

print("\n⛔ Arrêt des prédictions.")

ser.close()

if \_\_name\_\_ == "\_\_main\_\_":

mode = input("Choisissez le mode : (a)cquisition, (p)rédiction ou (e)ntraînement ? : ").lower()

if mode == 'a':

initialiser\_csv()

acquisition\_par\_positions()

elif mode == 'p':

prediction\_temps\_reel()

elif mode == 'e':

entrainer\_modele()

else:

print("Mode inconnu.")

Annexe F : Neural network for movements

import serial

import pandas as pd

import time

import os

import numpy as np

from sklearn.neural\_network import MLPClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

import joblib

# === Paramètres à personnaliser ===

DOSSIER\_TRAVAIL = r"C:\Users\matte\OneDrive\Documents\Scolaire\Sigma\2A\Stage\ROBOTA SUDOE\Tactile\_sensor\Capteurs\_tactiles\PPorPS"

PORT\_SERIE = 'COM3'

BAUDRATE = 9600

N\_CAPTEURS = 6

NB\_VALEURS\_GLISSANTES = 5  # Taille fenêtre glissante

HIDDEN\_LAYER = (30, 30)

MAX\_ITER = 2500

CLASSES = [

    "neutre",

    "rectiligne\_std",

    "courbe\_droite\_std",

    "courbe\_gauche\_std",

    "rectiligne\_poignard",

    "courbe\_gauche\_poignard",

    "courbe\_droite\_poignard"

]

CSV\_FILENAME = os.path.join(DOSSIER\_TRAVAIL, "donnees\_mouvements\_UR5\_statique.csv")

SCALER\_FILENAME = os.path.join(DOSSIER\_TRAVAIL, "scaler.pkl")

MODEL\_FILENAME = os.path.join(DOSSIER\_TRAVAIL, "modele.pkl")

# === Fonctions ===

def lire\_donnees\_serie(ser, num\_capteurs):

    """Lit une ligne du port série et extrait les valeurs des capteurs."""

    try:

        line = ser.readline().decode(errors='ignore').strip()

        valeurs = list(map(float, line.split(",")))

        if len(valeurs) != num\_capteurs:

            return None

        return valeurs

    except Exception:

        return None

def initialiser\_csv():

    if os.path.exists(CSV\_FILENAME):

        choix = input("Réinitialiser le fichier CSV ? (o/n) : ").lower()

        if choix == 'o':

            os.remove(CSV\_FILENAME)

            print("Fichier CSV réinitialisé.")

        else:

            print("Les nouvelles données seront ajoutées.")

    else:

        print("Le fichier sera créé.")

def acquisition\_par\_classe():

    ser = serial.Serial(PORT\_SERIE, BAUDRATE, timeout=1)

    print("Connexion série établie.")

    time.sleep(2)

    data\_total = []

    for classe in CLASSES:

        input(f"\nPréparez la classe '{classe}'. Appuyez sur Entrée pour démarrer l'acquisition.")

        try:

            duree = float(input("Durée d'acquisition (sec) : "))

        except ValueError:

            duree = 60

            print("Durée par défaut : 60s.")

        buffer = []

        start = time.time()

        while time.time() - start < duree:

            if ser.in\_waiting:

                donnees = lire\_donnees\_serie(ser, N\_CAPTEURS)

                if donnees:

                    buffer.append(donnees)

                    if len(buffer) >= NB\_VALEURS\_GLISSANTES:

                        fenetre = buffer[-NB\_VALEURS\_GLISSANTES:]

                        ligne = sum(fenetre, [])

                        ligne.append(classe)

                        data\_total.append(ligne)

                        print(f"{classe} : {ligne[:-1]}")

        print(f"Acquisition terminée pour {classe}.")

    ser.close()

    colonnes = [f"capteur\_{i+1}\_t{j+1}" for j in range(NB\_VALEURS\_GLISSANTES) for i in range(N\_CAPTEURS)] + ["classe"]

    df = pd.DataFrame(data\_total, columns=colonnes)

    if os.path.exists(CSV\_FILENAME):

        df.to\_csv(CSV\_FILENAME, mode='a', index=False, header=False)

    else:

        df.to\_csv(CSV\_FILENAME, index=False)

    print(f"✅ Données enregistrées dans {CSV\_FILENAME}")

def entrainer\_modele(HIDDEN\_LAYER, MAX\_ITER):

    if not os.path.exists(CSV\_FILENAME):

        print("Fichier CSV non trouvé.")

        return

    df = pd.read\_csv(CSV\_FILENAME)

    X = df.iloc[:, :-1].values

    y = df.iloc[:, -1].values

    scaler = StandardScaler()

    X\_scaled = scaler.fit\_transform(X)

    joblib.dump(scaler, SCALER\_FILENAME)

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2)

    model = MLPClassifier(hidden\_layer\_sizes=HIDDEN\_LAYER, max\_iter=MAX\_ITER, solver='adam', activation='logistic', verbose=True)

    model.fit(X\_train, y\_train)

    print(f"🎯 Précision : {model.score(X\_test, y\_test)\*100:.2f}%")

    joblib.dump(model, MODEL\_FILENAME)

    print("🧠 Modèle sauvegardé.")

def prediction\_temps\_reel():

    if not os.path.exists(MODEL\_FILENAME) or not os.path.exists(SCALER\_FILENAME):

        print("Modèle non trouvé.")

        return

    model = joblib.load(MODEL\_FILENAME)

    scaler = joblib.load(SCALER\_FILENAME)

    ser = serial.Serial(PORT\_SERIE, BAUDRATE, timeout=1)

    print("🔮 Prédictions en cours. Ctrl+C pour arrêter.")

    buffer = []

    try:

        while True:

            if ser.in\_waiting:

                donnees = lire\_donnees\_serie(ser, N\_CAPTEURS)

                if donnees:

                    buffer.append(donnees)

                    if len(buffer) >= NB\_VALEURS\_GLISSANTES:

                        fenetre = buffer[-NB\_VALEURS\_GLISSANTES:]

                        entree = sum(fenetre, [])

                        X\_input = scaler.transform([entree])

                        pred = model.predict(X\_input)[0]

                        proba = np.max(model.predict\_proba(X\_input)) \* 100

                        print(f"Prédiction : {pred} ({proba:.1f}%)")

    except KeyboardInterrupt:

        print("⛔ Arrêt.")

        ser.close()

# === Menu principal ===

if \_\_name\_\_ == "\_\_main\_\_":

    mode = input("Mode : (a)cquisition, (e)ntrainement, (p)rédiction ? ").lower()

    if mode == 'a':

        initialiser\_csv()

        acquisition\_par\_classe()

    elif mode == 'e':

        entrainer\_modele(HIDDEN\_LAYER, MAX\_ITER)

    elif mode == 'p':

        prediction\_temps\_reel()

    else:

        print("Mode inconnu.")

Annexe G : 3D simulation of the handle in real time

import serial

import time

import numpy as np

import matplotlib.pyplot as plt

import pyvista as pv

from pyvistaqt import BackgroundPlotter

from stl import mesh

import threading

from queue import Queue

# --- Paramètres de configuration ---

SERIAL\_PORT = 'COM3'

BAUD\_RATE = 9600

STL\_FILE\_PATH = r"C:\Users\matte\OneDrive\Documents\Scolaire\Sigma\2A\Stage\ROBOTA SUDOE\ROBOTA SUDOE 2025 BRIANCON-PROVERBIO\CAD\STL\Handle\Manche\_couteau.STL"

rect\_params = [

    {'x': 20, 'y': 20, 'z': 31, 'width': 60, 'height': 20, 'depth': 1},

    {'x': 20, 'y': 20, 'z': 7, 'width': 60, 'height': 20, 'depth': 1},

    {'x': 85, 'y': 20, 'z': 31, 'width': 60, 'height': 20, 'depth': 1},

    {'x': 85, 'y': 20, 'z': 7, 'width': 60, 'height': 20, 'depth': 1},

    {'x': 85, 'y': 45, 'z': 8.8, 'width': 60, 'height': 1, 'depth': 20},

    {'x': 20, 'y': 45, 'z': 8.8, 'width': 60, 'height': 1, 'depth': 20}

]

# File d'attente pour la communication entre les threads

data\_queue = Queue()

stop\_thread = threading.Event()

# --- Fonction de lecture du port série dans un thread séparé ---

def read\_serial\_data():

    """Lit les données du port série et met à jour la file d'attente."""

    try:

        ser = serial.Serial(SERIAL\_PORT, BAUD\_RATE, timeout=0.1)

        time.sleep(2)

        ser.reset\_input\_buffer()

        print(f"Thread de lecture série démarré sur {SERIAL\_PORT}")

        while not stop\_thread.is\_set():

            # Vidage du buffer si des données s'y accumulent

            if ser.in\_waiting > 0:

                ser.reset\_input\_buffer()

            line = ser.readline().decode('utf-8').strip()

            if line:

                try:

                    new\_values = [float(val) for val in line.split(',') if val]

                    if len(new\_values) == 6:

                        data\_queue.put(new\_values)

                except (ValueError, IndexError) as e:

                    print(f"Erreur de format de données: {line} - {e}")

            time.sleep(0.01)

    except serial.SerialException as e:

        print(f"Erreur fatale dans le thread série: {e}")

    finally:

        if 'ser' in locals() and ser.is\_open:

            ser.close()

# --- Fonction principale ---

def main():

    serial\_thread = threading.Thread(target=read\_serial\_data, daemon=True)

    serial\_thread.start()

    plotter = BackgroundPlotter(title="Modélisation 3D du manche de couteau")

    try:

        pyvista\_mesh = pv.read(STL\_FILE\_PATH)

        plotter.add\_mesh(pyvista\_mesh, color='white', smooth\_shading=True)

    except FileNotFoundError:

        print(f"Erreur: Le fichier STL '{STL\_FILE\_PATH}' est introuvable.")

        stop\_thread.set()

        serial\_thread.join()

        return

    sensors\_actors = []

    for params in rect\_params:

        box = pv.Box(bounds=(params['x'], params['x'] + params['width'],

                             params['y'], params['y'] + params['height'],

                             params['z'], params['z'] + params['depth']))

        actor = plotter.add\_mesh(box, color='blue', opacity=0.5)

        sensors\_actors.append(actor)

    color\_map = plt.get\_cmap("bwr")

    norm = plt.Normalize(vmin=0, vmax=4096)

    def update\_colors():

        last\_values = None

        while not data\_queue.empty():

            last\_values = data\_queue.get\_nowait()

        if last\_values:

            print(f"Données mises à jour: {last\_values}")

            try:

                for i, value in enumerate(last\_values):

                    color = color\_map(norm(value))

                    sensors\_actors[i].prop.color = color

            except Exception as e:

                print(f"Erreur lors de la mise à jour des couleurs: {e}")

    plotter.add\_callback(update\_colors, interval=50)

    try:

        plotter.app.exec()

    except KeyboardInterrupt:

        print("Fermeture de l'application.")

    finally:

        stop\_thread.set()

        serial\_thread.join()

        if 'ser' in locals() and ser.is\_open:

            ser.close()

        plotter.close()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

Annexe H : 3D simulation of the handle with csv files

import pandas as pd

import numpy as np

import pyvista as pv

from pyvistaqt import BackgroundPlotter

import matplotlib.pyplot as plt

import os

import time

# --- Paramètres de configuration (à ajuster) ---

CSV\_FILE\_PATH = r"C:\Users\matte\OneDrive\Documents\Scolaire\Sigma\2A\Stage\ROBOTA SUDOE\Tactile\_sensor\ROBOTA\_SUDOE\Plot\_pos\wrench\_camera\_data\wrench\_camera\_data\linear\_y\_hp1\wrench\_data\_2025-08-04\_17-43-46.csv"

STL\_FILE\_PATH = r"C:\Users\matte\OneDrive\Documents\Scolaire\Sigma\2A\Stage\ROBOTA SUDOE\ROBOTA SUDOE 2025 BRIANCON-PROVERBIO\CAD\STL\Handle\Manche\_couteau.STL"

# Correspondance entre les capteurs et les colonnes du fichier CSV

sensor\_mapping = [21, 22, 23, 24, 25, 26]

rect\_params = [

    {'x': 20, 'y': 20, 'z': 31, 'width': 60, 'height': 20, 'depth': 1},

    {'x': 20, 'y': 20, 'z': 7, 'width': 60, 'height': 20, 'depth': 1},

    {'x': 85, 'y': 20, 'z': 31, 'width': 60, 'height': 20, 'depth': 1},

    {'x': 85, 'y': 20, 'z': 7, 'width': 60, 'height': 20, 'depth': 1},

    {'x': 85, 'y': 45, 'z': 8.8, 'width': 60, 'height': 1, 'depth': 20},

    {'x': 20, 'y': 45, 'z': 8.8, 'width': 60, 'height': 1, 'depth': 20}

]

# --- Paramètres de l'animation (à ajuster) ---

ANIMATION\_DURATION\_SECONDS = 30 # Durée totale de l'animation en secondes

# --- Variables de contrôle de l'animation ---

animation\_running = False

current\_frame\_index = 0

data\_to\_plot = None

sensors\_actors = []

color\_map = None

norm = None

# --- Fonction de rappel pour l'animation ---

def update\_animation():

    global current\_frame\_index, animation\_running

    if not animation\_running or data\_to\_plot is None:

        return

    if current\_frame\_index < len(data\_to\_plot):

        try:

            current\_values = data\_to\_plot.iloc[current\_frame\_index, :].fillna(0).tolist()

            # Mise à jour des couleurs des capteurs

            for i, value in enumerate(current\_values):

                color = color\_map(norm(value))

                sensors\_actors[i].prop.color = color

            print(f"Frame {current\_frame\_index + 1}/{len(data\_to\_plot)} - Valeurs: {current\_values}")

            current\_frame\_index += 1

        except Exception as e:

            print(f"Erreur lors de la mise à jour de l'animation: {e}")

            animation\_running = False

    else:

        print("Animation terminée. Appuyez sur Espace pour recommencer.")

        current\_frame\_index = 0

        animation\_running = False

# --- Fonction pour démarrer/arrêter l'animation ---

def toggle\_animation():

    global animation\_running

    animation\_running = not animation\_running

    if animation\_running:

        print("Animation démarrée. Appuyez sur Espace pour mettre en pause.")

    else:

        print("Animation en pause. Appuyez sur Espace pour reprendre.")

# --- Fonction principale ---

def main():

    global data\_to\_plot, sensors\_actors, color\_map, norm

    try:

        # 1. Lecture des données à partir du fichier CSV

        df = pd.read\_csv(CSV\_FILE\_PATH, header=None).apply(pd.to\_numeric, errors='coerce')

        data\_to\_plot = df.iloc[:, sensor\_mapping]

        num\_frames = len(data\_to\_plot)

        interval\_ms = int(np.ceil((ANIMATION\_DURATION\_SECONDS \* 1000) / num\_frames))

        print(f"Total de données chargées : {num\_frames}.")

        print(f"Intervalle d'affichage calculé : {interval\_ms} ms par frame.")

        print(f"Durée totale de l'animation : {ANIMATION\_DURATION\_SECONDS} secondes.")

        print("Appuyez sur la touche ESPACE pour démarrer l'animation.")

    except FileNotFoundError:

        print(f"Erreur: Le fichier CSV '{CSV\_FILE\_PATH}' est introuvable.")

        return

    except Exception as e:

        print(f"Erreur lors de la lecture du fichier CSV : {e}")

        return

    # 2. Initialisation du plotter PyVista

    plotter = BackgroundPlotter(title="Modélisation 3D du manche de couteau")

    try:

        pyvista\_mesh = pv.read(STL\_FILE\_PATH)

        plotter.add\_mesh(pyvista\_mesh, color='white', smooth\_shading=True)

    except FileNotFoundError:

        print(f"Erreur: Le fichier STL '{STL\_FILE\_PATH}' est introuvable.")

        plotter.close()

        return

    for params in rect\_params:

        box = pv.Box(bounds=(params['x'], params['x'] + params['width'],

                             params['y'], params['y'] + params['height'],

                             params['z'], params['z'] + params['depth']))

        actor = plotter.add\_mesh(box, color='blue', opacity=0.5)

        sensors\_actors.append(actor)

    color\_map = plt.get\_cmap("bwr")

    norm = plt.Normalize(vmin=0, vmax=4096)

    # 3. Ajout du contrôle de l'animation

    plotter.add\_key\_event('space', toggle\_animation)

    # Exécuter la fonction de mise à jour à l'intervalle calculé

    plotter.add\_callback(update\_animation, interval=interval\_ms)

    try:

        plotter.app.exec()

    except KeyboardInterrupt:

        print("Fermeture de l'application.")

    finally:

        plotter.close()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

Annexe I : Calculation of the Jacobian

import numpy as np

# Paramètres de Denavit-Hartenberg (DH)

# Les longueurs et offsets sont en mm, les angles en degrés.

a = np.array([0, 638, 502, 13, 0])

d = np.array([237.0, 0, 0, 0, -201.0])

alpha = np.array([90, 0, 0, 90, -90])

def calculate\_transform\_matrix(a\_i, d\_i, alpha\_i, theta\_i):

    """

    Calculates the transformation matrix for a single joint using DH parameters.

    """

    theta\_rad = np.deg2rad(theta\_i)

    alpha\_rad = np.deg2rad(alpha\_i)

    T = np.array([

        [np.cos(theta\_rad), -np.sin(theta\_rad) \* np.cos(alpha\_rad), np.sin(theta\_rad) \* np.sin(alpha\_rad), a\_i \* np.cos(theta\_rad)],

        [np.sin(theta\_rad), np.cos(theta\_rad) \* np.cos(alpha\_rad), -np.cos(theta\_rad) \* np.sin(alpha\_rad), a\_i \* np.sin(theta\_rad)],

        [0, np.sin(alpha\_rad), np.cos(alpha\_rad), d\_i],

        [0, 0, 0, 1]

    ])

    return T

def calculate\_forward\_kinematics\_matrix(q, a, d, alpha):

    """

    Calculates the final transformation matrix for the end-effector.

    """

    T\_final = np.identity(4)

    for i in range(len(q)):

        T\_link = calculate\_transform\_matrix(a[i], d[i], alpha[i], q[i])

        T\_final = np.dot(T\_final, T\_link)

    return T\_final

def calculate\_jacobian(q, a, d, alpha):

    """

    Calculates the geometric Jacobian matrix.

    """

    n\_dof = len(q)

    J = np.zeros((6, n\_dof))

    T\_list = [np.identity(4)]

    T\_current = np.identity(4)

    for i in range(n\_dof):

        T\_current = np.dot(T\_current, calculate\_transform\_matrix(a[i], d[i], alpha[i], q[i]))

        T\_list.append(T\_current)

    p\_end\_effector = T\_list[-1][:3, 3]

    for i in range(n\_dof):

        T\_i\_minus\_1 = T\_list[i]

        z\_i\_minus\_1 = T\_i\_minus\_1[:3, 2]

        p\_i\_minus\_1 = T\_i\_minus\_1[:3, 3]

        J[3:, i] = np.cross(z\_i\_minus\_1, p\_end\_effector - p\_i\_minus\_1)

        J[:3, i] = z\_i\_minus\_1

    return J

Annexe J : Manipulability ellipse code

import pyvista as pv

from pyvistaqt import QtInteractor

from PyQt5 import QtWidgets, QtCore

import sys

import os

import numpy as np

# Ajoute le répertoire où se trouve robot\_kinematics.py

chemin\_du\_repertoire = r"C:\Users\matte\OneDrive\Documents\Scolaire\Sigma\2A\Stage\ROBOTA SUDOE\Tactile\_sensor\ROBOTA\_SUDOE\Plot\_pos\Ellipse"

sys.path.append(os.path.abspath(chemin\_du\_repertoire))

from robot\_kinematics import calculate\_jacobian, a, d, alpha

# Chemins de vos fichiers STL

base\_path = r"C:\Users\matte\OneDrive\Documents\Scolaire\Sigma\2A\Stage\ROBOTA SUDOE\Tactile\_sensor\ROBOTA\_SUDOE\Plot\_pos\Ellipse\704-244-01\_filled\_A\Base.stl"

shoulder\_path = r"C:\Users\matte\OneDrive\Documents\Scolaire\Sigma\2A\Stage\ROBOTA SUDOE\Tactile\_sensor\ROBOTA\_SUDOE\Plot\_pos\Ellipse\704-244-01\_filled\_A\Shoulder.stl"

elbow\_path = r"C:\Users\matte\OneDrive\Documents\Scolaire\Sigma\2A\Stage\ROBOTA SUDOE\Tactile\_sensor\ROBOTA\_SUDOE\Plot\_pos\Ellipse\704-244-01\_filled\_A\Elbow.stl"

wrist1\_path = r"C:\Users\matte\OneDrive\Documents\Scolaire\Sigma\2A\Stage\ROBOTA SUDOE\Tactile\_sensor\ROBOTA\_SUDOE\Plot\_pos\Ellipse\704-244-01\_filled\_A\Wrist1.stl"

wrist2\_path = r"C:\Users\matte\OneDrive\Documents\Scolaire\Sigma\2A\Stage\ROBOTA SUDOE\Tactile\_sensor\ROBOTA\_SUDOE\Plot\_pos\Ellipse\704-244-01\_filled\_A\Wrist2.stl"

# Définition des points de pivot (inchangé)

link\_pivots = {

    'Shoulder': np.array([0.0, 237.0, -135.0]),

    'Elbow': np.array([0.0, 875.0, 0.0]),

    'Wrist 1': np.array([0.0, 1377.0, 0.0]),

    'Wrist 2': np.array([0.0, 1390.0, -201.0]),

}

# Charger et décaler les maillages

original\_meshes = {

    'Base': pv.read(base\_path),

    'Shoulder': pv.read(shoulder\_path),

    'Elbow': pv.read(elbow\_path),

    'Wrist 1': pv.read(wrist1\_path),

    'Wrist 2': pv.read(wrist2\_path)

}

shoulder\_offset = -link\_pivots['Shoulder']

elbow\_offset = -link\_pivots['Elbow']

wrist1\_offset = -link\_pivots['Wrist 1']

wrist2\_offset = -link\_pivots['Wrist 2']

initial\_meshes = {

    'Base': original\_meshes['Base'],

    'Shoulder': original\_meshes['Shoulder'].translate(shoulder\_offset),

    'Elbow': original\_meshes['Elbow'].translate(elbow\_offset),

    'Wrist 1': original\_meshes['Wrist 1'].translate(wrist1\_offset),

    'Wrist 2': original\_meshes['Wrist 2'].translate(wrist2\_offset)

}

class RobotApp(QtWidgets.QMainWindow):

    def \_\_init\_\_(self, parent=None):

        super(RobotApp, self).\_\_init\_\_(parent)

        self.setWindowTitle("Contrôle du robot avec PyVista et PyQt")

        self.joint\_angles = {

            'Base': 0.0, 'Shoulder': 45.0, 'Elbow': 45.0, 'Wrist 1': 0.0, 'Wrist 2': 0.0

        }

        main\_widget = QtWidgets.QWidget()

        self.setCentralWidget(main\_widget)

        main\_layout = QtWidgets.QHBoxLayout(main\_widget)

        self.plotter = QtInteractor(self)

        control\_panel = QtWidgets.QWidget()

        control\_layout = QtWidgets.QVBoxLayout(control\_panel)

        splitter = QtWidgets.QSplitter(QtCore.Qt.Horizontal)

        splitter.addWidget(self.plotter)

        splitter.addWidget(control\_panel)

        splitter.setStretchFactor(0, 3)

        splitter.setStretchFactor(1, 1)

        main\_layout.addWidget(splitter)

        controllable\_joints = ['Base', 'Shoulder', 'Elbow', 'Wrist 1', 'Wrist 2']

        self.entries = {}

        for name in controllable\_joints:

            entry\_layout = QtWidgets.QHBoxLayout()

            label = QtWidgets.QLabel(f"Angle {name}:")

            entry = QtWidgets.QLineEdit(str(self.joint\_angles[name]))

            entry.returnPressed.connect(lambda n=name, e=entry: self.on\_text\_entry(n, e))

            self.entries[name] = entry

            entry\_layout.addWidget(label)

            entry\_layout.addWidget(entry)

            control\_layout.addLayout(entry\_layout)

        self.manip\_ellipsoid\_actor = None

        self.effector\_marker = None

        self.setup\_robot\_scene()

    def setup\_robot\_scene(self):

        self.plotter.background\_color = 'white'

        self.robot\_actors = {

            'Base': self.plotter.add\_mesh(initial\_meshes['Base'], color='lightblue', label='Base'),

            'Shoulder': self.plotter.add\_mesh(initial\_meshes['Shoulder'], color='lightgreen', label='Shoulder'),

            'Elbow': self.plotter.add\_mesh(initial\_meshes['Elbow'], color='salmon', label='Elbow'),

            'Wrist 1': self.plotter.add\_mesh(initial\_meshes['Wrist 1'], color='gold', label='Wrist 1'),

            'Wrist 2': self.plotter.add\_mesh(initial\_meshes['Wrist 2'], color='lightcoral', label='Wrist 2')

        }

        self.pivot\_meshes = {}

        for name, pos in link\_pivots.items():

            self.pivot\_meshes[name] = pv.Sphere(radius=5).translate(pos)

            self.plotter.add\_mesh(self.pivot\_meshes[name], color='red', render\_points\_as\_spheres=True, point\_size=10.0)

        slider\_params = [

            ('Base', [-180, 180], self.joint\_angles['Base']),

            ('Shoulder', [-180, 180], self.joint\_angles['Shoulder']),

            ('Elbow', [-180, 180], self.joint\_angles['Elbow']),

            ('Wrist 1', [-180, 180], self.joint\_angles['Wrist 1']),

            ('Wrist 2', [-180, 180], self.joint\_angles['Wrist 2'])

        ]

        spacing = 0.15

        start\_y = 0.1

        for i, (name, rng, value) in enumerate(slider\_params):

            self.plotter.add\_slider\_widget(

                callback=lambda val, n=name: self.on\_slider\_change(val, n),

                rng=rng, title=name, pointa=(0.02, start\_y + i \* spacing), pointb=(0.32, start\_y + i \* spacing),

                value=value, style='modern'

            )

        self.update\_robot()

    def update\_robot(self):

        t\_base = pv.Transform().rotate\_y(self.joint\_angles['Base'])

        t\_shoulder = pv.Transform().rotate\_z(self.joint\_angles['Shoulder'])

        t\_elbow = pv.Transform().rotate\_z(self.joint\_angles['Elbow'])

        t\_wrist1 = pv.Transform().rotate\_z(self.joint\_angles['Wrist 1'])

        t\_wrist2 = pv.Transform().rotate\_y(self.joint\_angles['Wrist 2'])

        final\_transform\_base = t\_base.matrix

        final\_transform\_shoulder = final\_transform\_base @ pv.Transform().translate(link\_pivots['Shoulder']).matrix @ t\_shoulder.matrix

        final\_transform\_elbow = final\_transform\_shoulder @ pv.Transform().translate(link\_pivots['Elbow'] - link\_pivots['Shoulder']).matrix @ t\_elbow.matrix

        final\_transform\_wrist1 = final\_transform\_elbow @ pv.Transform().translate(link\_pivots['Wrist 1'] - link\_pivots['Elbow']).matrix @ t\_wrist1.matrix

        final\_transform\_wrist2 = final\_transform\_wrist1 @ pv.Transform().translate(link\_pivots['Wrist 2'] - link\_pivots['Wrist 1']).matrix @ t\_wrist2.matrix

        self.robot\_actors['Base'].user\_matrix = final\_transform\_base

        self.robot\_actors['Shoulder'].user\_matrix = final\_transform\_shoulder

        self.robot\_actors['Elbow'].user\_matrix = final\_transform\_elbow

        self.robot\_actors['Wrist 1'].user\_matrix = final\_transform\_wrist1

        self.robot\_actors['Wrist 2'].user\_matrix = final\_transform\_wrist2

        q\_angles\_deg = np.array(list(self.joint\_angles.values()))

        try:

            jacobien = calculate\_jacobian(q\_angles\_deg, a, d, alpha)

            wrist2\_transform\_matrix = self.robot\_actors['Wrist 2'].user\_matrix

            self.calculate\_and\_plot\_ellipsoid(jacobien, wrist2\_transform\_matrix, scale\_factor=250)

        except Exception as e:

            print(f"Erreur lors du calcul de la cinématique ou de l'affichage de l'ellipsoïde : {e}")

        self.plotter.render()

    def on\_slider\_change(self, value, name):

        self.joint\_angles[name] = value

        self.entries[name].setText(f"{value:.2f}")

        self.update\_robot()

    def on\_text\_entry(self, name, entry):

        try:

            value = float(entry.text())

            self.joint\_angles[name] = value

            self.update\_robot()

        except ValueError:

            print(f"Valeur invalide pour {name}. Veuillez entrer un nombre.")

    def calculate\_and\_plot\_ellipsoid(self, jacobien, wrist2\_transform\_matrix, scale\_factor=250):

        Jv = jacobien[:3, :]

        try:

            U, s, Vt = np.linalg.svd(Jv)

            if np.isclose(s[0], 0, atol=1e-6):

                if self.manip\_ellipsoid\_actor:

                    self.plotter.remove\_actor(self.manip\_ellipsoid\_actor)

                print("Le robot est dans une configuration singulière. L'ellipsoïde de manipulabilité ne peut pas être affiché.")

                return

            s\_scaled = s / s[0] \* scale\_factor

            if self.manip\_ellipsoid\_actor:

                self.plotter.remove\_actor(self.manip\_ellipsoid\_actor)

            ellipsoid\_mesh = pv.Sphere(radius=1.0, phi\_resolution=30, theta\_resolution=30)

            # Paramètre pour le positionnement manuel de l'ellipse

            # decalage\_local\_ellipse = np.array([0.0, 1535.0, -300.0])

            decalage\_local\_ellipse = np.array([0.0, 150.0, -100.0])

            ellipsoid\_mesh.scale(s\_scaled, inplace=True)

            ellipsoid\_mesh.translate(decalage\_local\_ellipse, inplace=True)

            ellipsoid\_mesh.transform(wrist2\_transform\_matrix, inplace=True)

            self.manip\_ellipsoid\_actor = self.plotter.add\_mesh(ellipsoid\_mesh, color='red', opacity=0.5, style='wireframe')

        except np.linalg.LinAlgError:

            if self.manip\_ellipsoid\_actor:

                self.plotter.remove\_actor(self.manip\_ellipsoid\_actor)

            print("Erreur de calcul SVD : Le robot est dans une configuration singulière.")

        except Exception as e:

            print(f"Erreur lors de l'affichage de l'ellipsoïde : {e}")

if \_\_name\_\_ == '\_\_main\_\_':

    app = QtWidgets.QApplication(sys.argv)

    window = RobotApp()

    window.show()

    sys.exit(app.exec\_())