# Scientific project

*Piloting an UR10e robot with gesture recognition*

**Done in the laboratory ARS Control of the University of UNIMORE in the field of a Erasmus exchange**


Written by : **BAHEU JUSTIN**


The : 05/01/2022


*Supervised by :*

**Cristian SECCHI** **Davide FERRARI** →Unimore

**Simona D'ATTANASIO** **Maria HERBAUT-AGUIRRE** → ICAM

# Recap of the project

In the field of my last year in the engineering school of ICAM, I wanted to discover new science fields that we don't really see in school . Otherwise, I was also interested in robotics and computer science so naturally my scientific project choice was turned into the University of Unimore and it's mechatronics department.

During the middle of september 2021, I came to Reggio Emilia to work for 5 months in the Technopole and more precisely, in the ARS laboratory.

Before the beginning of my project, I had to learn how to program in ROS, in C++ and in Python. This preliminary step lasted more or less one month.

After that, I was able to begin my project which consists of driving a robot with gestures. One external tool has to be used for this project and that's : Mediapipe.

Mediapipe is a google open-sourced feature which offers you some solutions to detect some part of your body and automatically draw some specific points on it.

So my job during this project will be to manipulate the coordinates of these points to create body positions (like the position "STOP" with your hand opened) and after driving the robot with my hands or my entire body.

# Summary

# Illustration table

# Glossary

**Topic** : A place where we publish and read a type of messages, accessible from files and ubuntu commands

**Publisher** : A function which will publish a message in a topic

**Subscriber** : A function which will read and extract a message from a topic

**Workspace** : A folder where we will create our ROS environment

**Launch file** : A folder which allows to launch differents nodes with launch conditions

**Node** : Executive files of your program

**Package** : Folder where you can find all of what it's required to build a correct ROS program

**Skeleton_tracking** : recognition of the human body and association of points with parts of the body

**Landmark** : Specific point drawed by Mediapipe solution

# I/ Project context

## 1) ARS laboratory

ARS Control, means Automation, Robotics and System Control. In the lab you can find some Robotic, Automation and System Control researcher of the engineering branch of Unimore University (Università degli studi di Modena e Reggio Emilia).

The group ARS Control trains some students in their laboratory in Reggio Emilia and in Gavasseto. Different projects are in partnership with companies, with the Emilie ROmagne region and with Europe. On my side, I worked in Reggio Emilia, in Italy. The building, located next to the train station is called "Technopolo" (Appendices N°1 & 2).

## 2) Professional goals

During this project my final goal was to pilot this UR10E robot with solutions that Mediapipe offer.

My period to build my project was divided into 6 differents part :

| Milestones | Verification |
|---|---|
| Learn how to use Python and/or C++ | Online courses |
| Learn how to use Mediapipe for skeleton tracking solution | Presentation of all mediapipe solutions |
| Learn how to use ROS | Gazebo simulation for moving UR10e robot |
| Create a node to start Mediapipe with ROS | Acquire skeleton tracking data from ROS |
| Gestures recognition | Develop a machine learning module for simple gesture recognition |
| Drive the robot with gestures | Control UR10e with gestures |

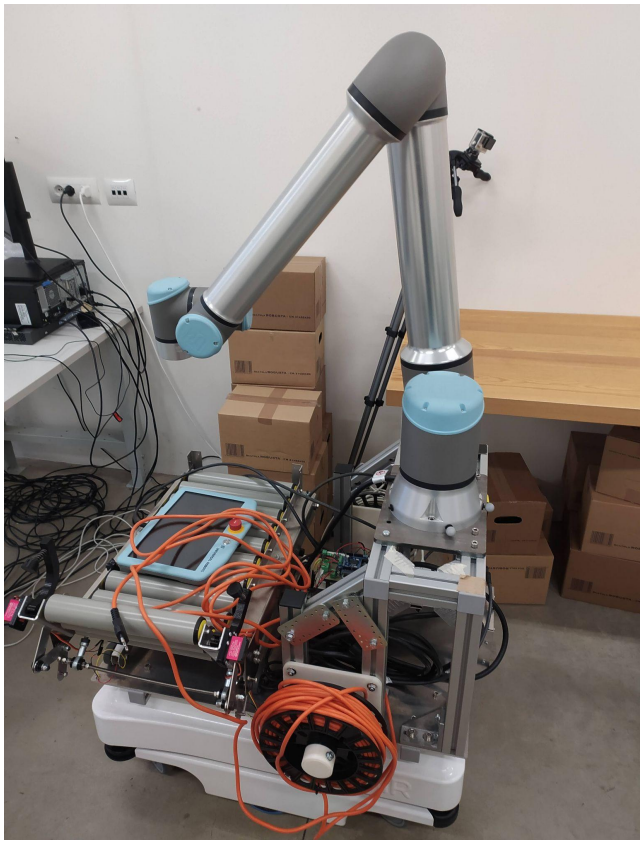
## 3) The robot

On the robot you can find two differents part :

- The UR10E robot which is the arm on the picture
- The mobile base

This robot is a popular collaborative robot, builded by Universals robotics.

A collaborative robot is a robot that is used to work together with human, that's "Cobotics"



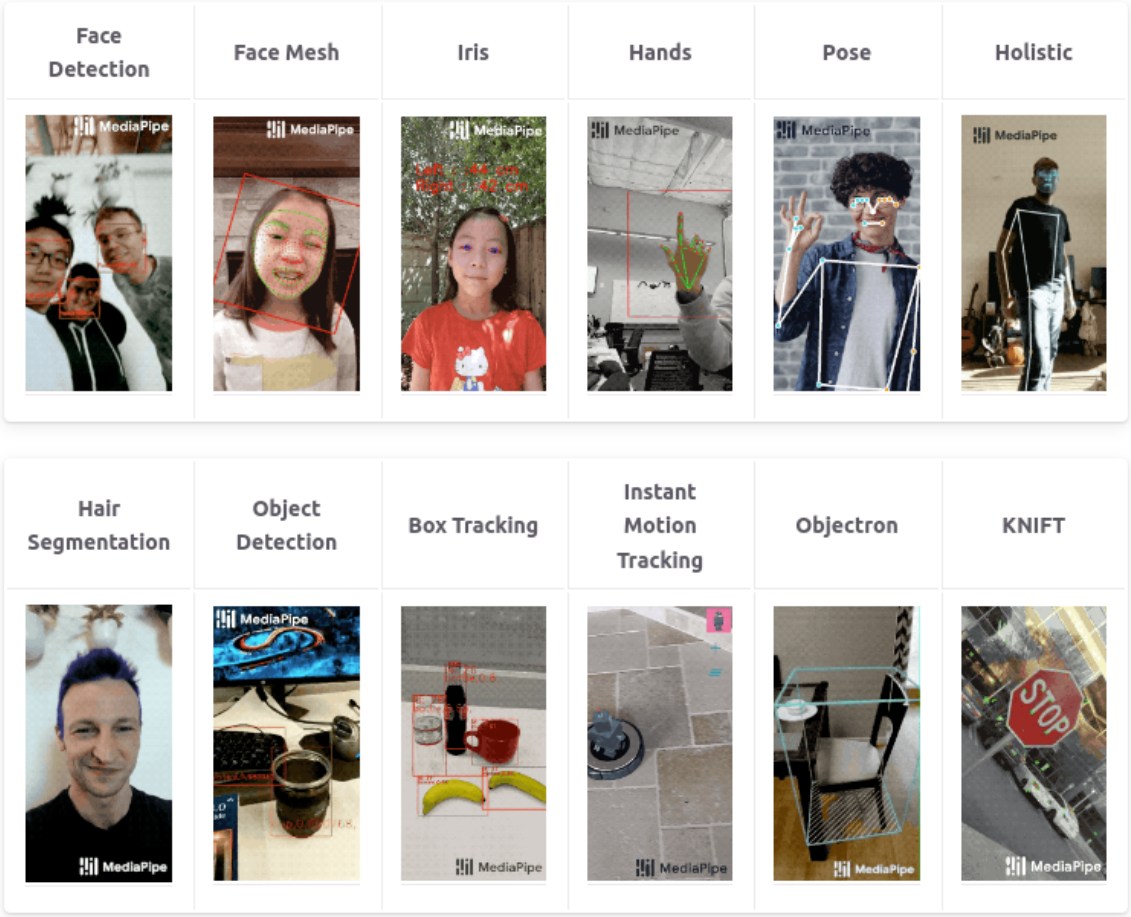
***Picture N°1 : The UR10E robot***

## 4) Presentation of Mediapipe

Mediapipe offers multiple solutions to detect the human body or object.

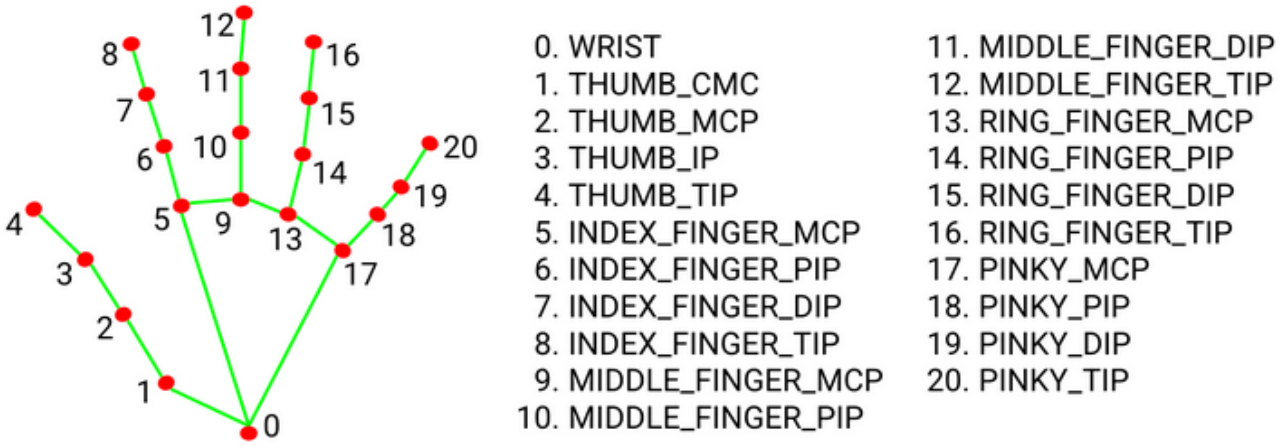
As you can see on the picture below Mediapipe offers 7 human body detections solutions (Face detection, Face Mesh, Iris, Hands, Pose, Holistic, Hair segmentation).

In our cases we'll use the Holistic solution (which regroup Face Mesh, Hands and Pose solutions) to build our project.




***Picture N°2 : All Mediapipe solutions***

As an example and to show you how it works you can show on the picture below the different landmarks which are drawn on a hand.




***Picture N°3 : Hands landmarks***

As you can see, 21 different landmarks will be printed on your hand as long as you show your hand to the webcam.

For the "pose" solutions we're gonna have 33 different landmarks and for the "Face Mesh" solution that's 465 different landmarks.

# II/ Progress of the project

## 1) Understanding the work environment (ROS, Python) :

My project started with a one month period of training and understanding the languages of Python, C++ and ROS. First of all, and to facilitate my work, my tutor asked me to work with Ubuntu. Which was an easier way to work with the ROS environment.

### 1.1) ROS

ROS means Robot Operating System. The interest of using ROS is that it is a set of computer tools allowing to create a programming environment for robotics applications. ROS runs on Windows and Ubuntu.

In order to define a programmation work environment, we first create a workspace. Within this workspace, we can find a source file (Where you'll create our program), a build file ( That one will be fed up automatically when we will compile a code) and a devel file (Where the file's links are placed).

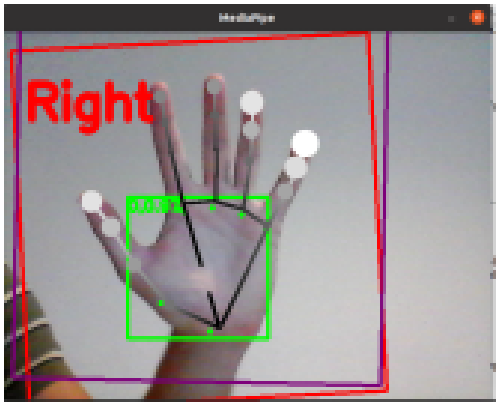
Inside the source folder, we can create our packages. In this package you will automatically have a CMakeList file and a Package file. These two files are in XML and are used to configure the work environment, establish dependencies between files, call other files, call files from other packages, call ROS functionalities …

### 1.2) Python

Python is a very popular programming language. I choice this language because C++ don't really work for the moment with mediapipe solutions. This language will be used to build all of my nodes.

## 2) First use of Mediapipe and manipulation of landmarks

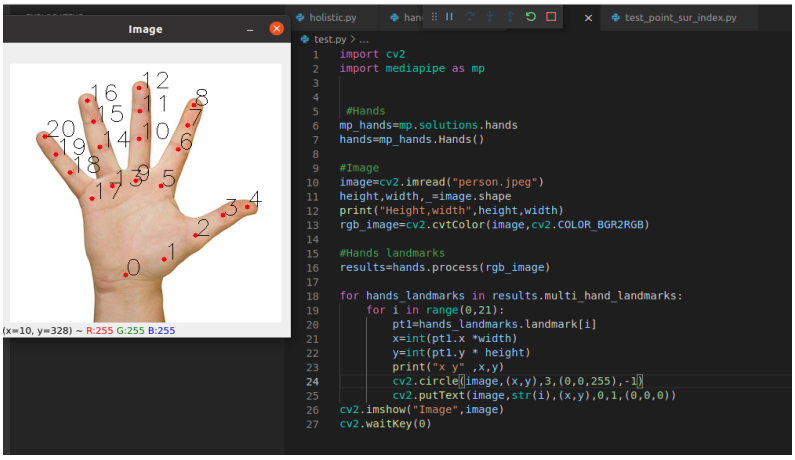
With python, Mediapipe offers you an API (Application Programming Interface) which is some lines of code that will open and start your webcam, detect your body and draw some landmarks on it. That's the base of Mediapipe, of course you can choose if you want to draw some points one your right hand, your left hand, your face or your entire body as you can see on the picture below.




*Picture N°4 : Test with Right Hand*

But for the moment, we don't really know how you can manipulate these points and which one it is. So I created a few lines of code to symbolize where these points are.

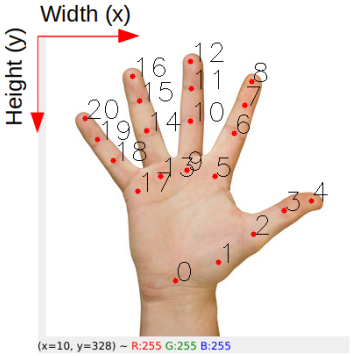
As you can see, that's the same as picture number 3 .




*Picture N°5 : First use of landmarks*

Moreover, Mediapipe offer us the possibilities to extract the coordinates of the points shown on the screen with 4 variables (x,y,z and visibility), that will be very useful for our project.

Indeed, the points coordinates are created with the dimensions of the opened screen on your device. The origin of the vectors are on the top left corner of your webcam screen, like you can see on the picture on the right.




*Picture N°6 : How coordinates works*

So if you say “if the y value of my point 12 is higher than the y value of my point 9” that means that my hand is closed. That’s the first utilization of the coordinates that mediapipe offers to us. But that’s not precise enough, we will have to find another solution to manipulate our coordinates.

## 3) Communication in ROS

As I explained before, a workspace in ROS can be built of multiple packages and a package can have multiple nodes (executive program). If I want my program to be fully understood by anyone who wants to take it after my departure, I need to build my program not in one big block of code, but I need to separate it into different parts. If I split my code into different parts, we still need to have a communication between each of them, that’s why you have to define some Publisher and Subscriber.

### Publisher and Subscriber nodes :

In a ROS package some nodes will publish a message on a topic, and some others will subscribe to this topic to have the message.

Let me explain with an example, you want to detect the postman on a video with a code. How can you do it properly ?

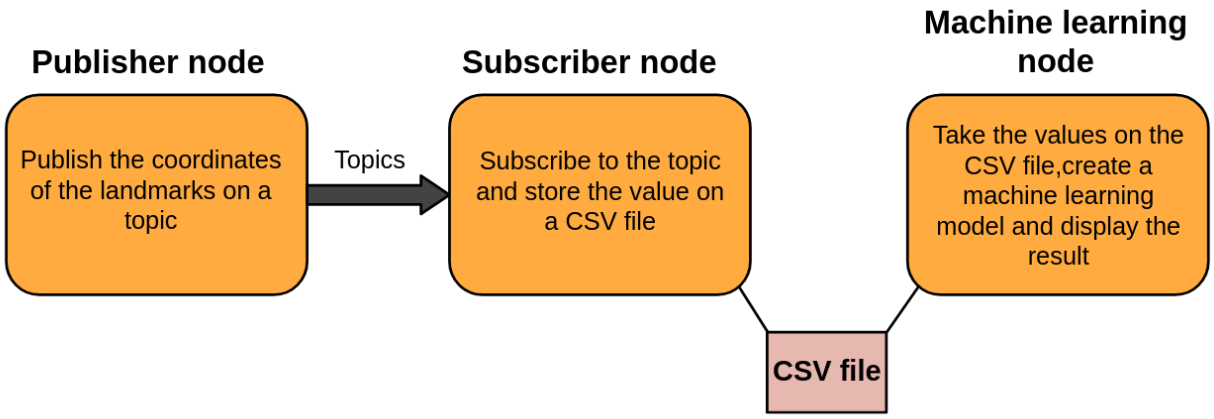
- The first thing that you’ll need to do is to create a node who will publish the video on a topic
- The second one is to create a subscriber node who will subscribe to this topic to take the video frame and detect the postman.

With this composition of two different nodes, if I wanted to take the publisher node to detect a policeman now, you would just have to subscribe to the same topic and detect a

policeman and not a postman. You will not have to recreate all of mypublisher node each time and that's easier.

## 4) How will my program be built ?

As I explained to you in the example before, I will create different nodes. In my case I will create a node who will capture the coordinates of all of my landmarks and publish it on a topic. Another node will subscribe to this topic and publish all of this landmark in a storage file (CSV) and another node will take all the value in this CSV file to create a machine learning model and publish the results on the screen.



## 5) What's in my Publisher node ?

My publisher node is made of 3 functions, all of theses function can be found into the annexes part :

- ***def param() :*** *(Appendices N°3)*

This part is made to set the initials values and to let the user choose what he wants to use as solutions. For example the user can choose if we want to detect his right hand, left hand or/and his body, but he can also choose if he wants to use his webcam or an external camera.

The default values of these variables were set in the launch file. As I explained before with this file you can launch different nodes and set up some parameters before the running of your code.

```
<launch>


<arg name="webcam" default="0" />
<arg name="enable_right_hand" default="enable" />
<arg name="enable_left_hand" default="enable" />
<arg name="enable_pose" default="enable" />
<arg name="enable_face_mesh" default="disable" />

<param name="webcam" value="$(arg webcam)"/>
<param name="enable_right_hand"  value="$(arg enable_right_hand)"/>
<param name="enable_left_hand" value="$(arg enable_left_hand)"/>
<param name="enable_pose"  value="$(arg enable_pose)"/>
<param name="enable_face_mesh"  value="$(arg enable_face_mesh)"/>

<node pkg="sk_tracking" type="madiapipe.py" name="madiapipe" output="screen">

</node>

</launch>
```

***Picture N°7 : Launch file of Publisher node***

```
E_D_Webcam = rospy.get_param("/webcam")
E_D_Right_Hand = rospy.get_param("/enable_right_hand")
E_D_Left_Hand = rospy.get_param("/enable_left_hand")
E_D_Pose = rospy.get_param("/enable_pose")
E_D_Face_Mesh = rospy.get_param("/enable_face_mesh")
```

***Picture N°8 : Communication between Launch File and node***

- ***def createfiles() :*** *(Appendices N°4 & 5)*

This function is used to create external files.

- ➢ The first part of this function will allow us to create TXT files to save some datas. In our case, the saving name of our project or the differents mediapipe solutions that we will use in this project will be saved.
- ➢ The second part would create the structure of my CSV file. As I said before, all my landmarks will have 4 different values (x,y,z and visibility). So this function creates a table. As you may see, the structure is simple. The first column is used for my class name, which is for example my "STOP" position, my "LEFT" position and my "RIGHT" position. The following columns are used to define my x,y,z and v values for my first landmark, my second landmark etc… The value of my landmark will not be stored with this function, just the first line of the CSV file will be created here.

| class | x1 | y1 | z1 | v1 | x2 |
|---|---|---|---|---|---|
| stopit | 0.11496800184249878 | 0.9761984944343567 | 6.387273856489628e-07 | 0.0 | 0.22236233949661255 |
| stopit | 0.11496800184249878 | 0.9761984944343567 | 6.387273856489628e-07 | 0.0 | 0.22236233949661255 |
| stopit | 0.11496800184249878 | 0.9761984944343567 | 6.387273856489628e-07 | 0.0 | 0.22236233949661255 |
| stopit | 0.11496800184249878 | 0.9761984944343567 | 6.387273856489628e-07 | 0.0 | 0.22236233949661255 |
| stopit | 0.11496800184249878 | 0.9761984944343567 | 6.387273856489628e-07 | 0.0 | 0.22236233949661255 |

***Picture N°9 : Example of a CSV file***

- ***def talker() :*** (Appendices N°6)

This is the main function of the node. Basically, this node will (depending of your previous choice) asked you to choice how many position you wanted to set up, the name of each position (That's the value stored in the "class" column in the CSV file), launch the webcam with the mediapipe solutions chosen, record all the landmark in a message, publish this message on a topic until the record of all the position is finish.

## How are my landmarks published on a topic ?

With ROS you can have standard messages as string, int32, int64 etc... But in my case I needed to create a custom message with an array of floats not with a defined number of values in, but a flexible array. I also need to define a message how can store the name of my class (STOP, RIGHT...). So my message looks like that :

```
float64[] H_Key
string Name
```

*Picture N°10 : Msg file*

The structure of my message is created but I need to publish my values in a topic with that message structure. The following picture shows a typical loop in my talker function which is used to display the mediapipe solution on the screen, extract the datas of the landmarks shown on the screen and publish it on a topic.

```python
msg_h = Holistic()
if results.right_hand_landmarks :

   Right_hand_results = results.right_hand_landmarks.landmark
   right_hand_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in Right_hand_results]).flatten())

if results.left_hand_landmarks:

   Left_hand_results= results.left_hand_landmarks.landmark
   left_hand_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in Left_hand_results]).flatten())

if results.pose_landmarks:

   Pose_results= results.pose_landmarks.landmark
   pose_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in Pose_results]).flatten())


results.right_hand_landmarks or results.left_hand_landmarks or results.pose_landmarks
land=right_hand_row+left_hand_row+pose_row
msg_h.Name=class_name
msg_h.H_Key=land
pub_h.publish(msg_h)
```

*Picture N°11 : A loop to extract and publish landmarks*

Here :

- "***msg_h***" is used to store the Holistic message type on a variable.
- The next 3 if loops is used to catch the results of the landmarks and store it on a list
- The "***land***" variable will be a list of all the values of the 3 previous list of landmarks.
- "***msg_h.Name=class_name***" is used to store the data of the "***class_name***" set up before and store it in the "***msg_h***" variable with the Key "***Name***" corresponding to the string value.
- "***msg_h.H_Key=land***" same thing as before, the value stored in the list "***land***" will be stored in the "***msg_h***" variable with the specific key "***H_Key***".
- The last line is to publish the message on a topic.

In the next part we will see how we can subscribe to this topic and store this data in the CSV file.

## 6) What's in my Subscriber node ?

*(Appendices N°7)*

My Publisher node is always wanting for a message to be published on a topic. If nothing is published, nothing happened but if a message is published on a topic that the node subscribed to, then this line of code will work :

```
def H_callback(data):

    var = []
    rh=list(data.H_Key)
    name=data.Name
    var.append(name)
    var.extend(rh)

    with open('/home/baheu/ws_sk_tracking/src/sk_tracking/CSV files/coords.csv', mode='a', newline='') as f:
        csv_writer = csv.writer(f, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
        csv_writer.writerow(var)
```

***Picture N°12 : Callback function of the subscriber node***

That's the callback function of my node, here you have :

- A empty "***var***" list is created
- "***rh***" is used to stored the data stored in the previous variable "***msg_h***" with the key "***H_Key***", so the list of landmarks will be stored in this "***rh***" variable
- Same for the "***name***" variable which will take the previous "***class_name***" variable
- In the next two lines, I will insert into my empty array "var" the value stored iin the "***name***" variable and after the values stored in the variable "***rh***".

Now we have a list called "var" with in the first place the name and after x,y,z and v values for all the landmarks. This structure remind you something ?

You are right, that's the structure of the CSV file that we talk before in the function "***createfiles***".

Now, in the "with" loop, we just insert the value stored in the list "var" into the CSV file that we created before.

## 7) What's in my machine learning node ?

In this node, the main structure will be based on an if/else loop. Firstly the user will have to choose between building a new machine learning model with the position that he just setted up or start the simulation with a machine learning model that is already built.

The first if loop will look like this : (Appendices N°8 & 9)

```
#SETTINGS STEP

print("\nDo you want to build a new solution model or to use a previous one ? ")
Build_or_not=input("\nWrite NEW for a new solution and PREVIOUS for a previous solution :  " )
webcam=0
choice_cam=input("\nWrite YES if you want to use an external camera, otherwise write NO" )
if (choice_cam=="YES" or choice_cam=="yes" or choice_cam=="Yes"or choice_cam=="y"or choice_cam=="Y"):
    webcam=2
if (Build_or_not=="NEW" or Build_or_not=="New" or Build_or_not=="new"):


# LOOP TO CREATE A NEW MACHINE LEARNING MODEL

    Choice_Okay=False
    while (Choice_Okay == False):
        print("\nThe following name are your previous position saving, type the name before the point to use it !")
        dirPath = r"/home/baheu/ws_sk_tracking/src/sk_tracking/CSV files"                                   #Take all files name stored in that path
        result = [f for f in os.listdir(dirPath) if os.path.isfile(os.path.join(dirPath, f))]               #Take all files name stored and create a list
        print(result)
        Solution_Choice=input("\nWhich want do you want to use ? (Type the exact name before the point) :  ")
        M_Solution=open(f'/home/baheu/ws_sk_tracking/src/sk_tracking/TXT file/Solution_Name/Solution_{Solution_Choice}.txt','r')        #Read the Solution TXT file with the name of
        Mediapipe_Solution=M_Solution.read()                                                                #Same
        Pos_Name=open(f'/home/baheu/ws_sk_tracking/src/sk_tracking/TXT file/Position_Name/Position_Name_{Solution_Choice}.txt','r')     #Read the Postion name TXT file with the name
        Position_Name=Pos_Name.read()                                                                       #Same
        print(f'\nYou choose {Solution_Choice} and you set up these following position : {Position_Name}  \n This position will use the following solution : {Mediapipe_Solution} ')

        Setup_of_robot_action()

        Continue=input("\nIf you made a mistake during the setup please write NO, otherwise write YES :  ")
        if (Continue=="YES") or (Continue=="y") or (Continue== "Yes") or (Continue=="yes"):
            Choice_Okay=True

    train_model()

    detection()
```

***Picture N°13 : Machine learning loop***

Here, in the "While" loop the user will have the choice with which project he wants to create a machine learning model and after his choice we are gonna have the "***Setup_of_robot_action***" function and after the "***train_model***" function.

The second loop (Appendices N°10) looks like the first one but we will not call the "***train_model***" function.

- ***def train_model() :*** *(Appendices N°11)*

Now the hard work starts, we have our well structured CSV file with all of our landmarks but that's not enough as you might expect.

As I explained you before in the part 4, this node will be used to take the value stored on the CSV file, build a machine learning model with that values and launch the webcam with the mediapipe solution, the different position that you make on the camera and the probabilities of corresponding will also be displayed on the screen.

The first part of this function is build like this :

```
#3/ TRAIN CUSTOM MODEL USING SCIKIT LEARN
# 3.1/ READ IN COLLECTED DATA AND PROCESS

df = pd.read_csv('/home/baheu/ws_sk_tracking/src/sk_tracking/CSV files/coords.csv')
X = df.drop('class', axis=1)
y = df['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1234)
```

***Picture N°14 : Collect and process data from CSV file***

Here you have :

- The variable "***df***" which will read the CSV file.
- "***X***" variable which will take all the values except the one stored in the column "class".
- "***y***" variable will take only the value stored in the column "***class***"
- The last line will create some random train and test variable with the module "***train_test_split***" imported with **Scikit-learn** which is the bigger machine learning library available in Python.

```
# 3.2/ TRAIN MACHINE LEARNING CLASSIFICATION MODEL

pipelines = {
    'lr':make_pipeline(StandardScaler(), LogisticRegression()),
    'rc':make_pipeline(StandardScaler(), RidgeClassifier()),
    'rf':make_pipeline(StandardScaler(), RandomForestClassifier()),
    'gb':make_pipeline(StandardScaler(), GradientBoostingClassifier()),
        }

fit_models = {}
for algo, pipeline in pipelines.items():
    model = pipeline.fit(X_train, y_train)
    fit_models[algo] = model
```

***Picture N°15 : Create machine learning model***

With Scikit-learn we launch 4 different pipelines which will be 4 different machine learning solutions that will allow us to classify the datas, these machine learning solutions are called “Classifiers” and here we can use it with the functions (LogisticRegression, RidgeClassifier,RandomForestClassifier or GradientBoostingClassifier).

After, a typical model will be created with the random value taking before and the pipeline solutions called before.

```
# 3.3/ EVALUATE AN SERIALIZE MODEL

for algo, model in fit_models.items():
    yhat = model.predict(X_test)
    print(algo, accuracy_score(y_test, yhat))
```

***Picture N°16 : Train the model***

In this loop we create a “***yhat***” variable with the prediction of precision of each of the 4 pipelines. With the values stored in my CSV file, sometimes maybe the first pipelines will be more precise than the other one and maybe another time that will be another.

```
with open('/home/baheu/ws_sk_tracking/src/sk_tracking/PKL files/pose_recognition.pkl', 'wb') as f:
    pickle.dump(fit_models['rf'], f)
```

*Picture N°17 : Publish the model in a PKL file*

These few lines will create a PKL file with the more precise pipeline (Here I choose "***rf***"). The PKL file will be used to store the result in an independent file, as the CSV file. After, it will be possible to have multiple different PKL files with different positions setted up.

- ***def Setup_of robot_action()***

```
def Setup_of_robot_action():
    with open(f"/home/baheu/ws_sk_tracking/src/sk_tracking/TXT file/Position_Name/Position_Name_{Solution_Choice}.txt", "r") as file:
        allText = file.read()
        words = list(map(str, allText.split()))
        print('\nFor the moment the robot can have 5 different information, GO, Turn left, Turn right, stop and back\nIn this next line
        print('\nTo set up a position you just have to write :\n1 for GO\n2 for LEFT\n3 for RIGHT\n4 for STOP\n5 for BACK')

    global liste
    nbr_pos=len(words)
    liste=["P1","P2","P3","P4","P5","P6","P7","P8","P9","P10"]
    for i in range(0,nbr_pos):
        pos=input(f'\nFor which action your position {words[i]} will be used ? ')
        print(pos)
        if (pos=="1"):
            liste[0]=words[i]
        elif (pos=="2"):
            liste[1]=words[i]
        elif (pos=="3"):
            liste[2]=words[i]
        elif (pos=="4"):
            liste[3]=words[i]
        elif (pos=="5"):
            liste[4]=words[i]
        elif (pos=="6"):
            liste[5]=words[i]
        elif (pos=="7"):
            liste[6]=words[i]
        elif (pos=="8"):
            liste[4]=words[i]
        elif (pos=="9"):
            liste[8]=words[i]
        elif (pos=="10"):
            liste[9]=words[i]
```

*Picture N°18 : Setup_of_robot_action() function*

This following function is used to add the name of the position that you called before as (stop, right, left, go, back…) to a list. This list will be used in your "***detection***" function to pilot your loop.

So basically this function will read the TXT file where you stored your position name and split all the words in a list called here "***words***".

After, depending on the number of positions that you setted up will create a list and the user will have to fill up this list with a number (1,2,3...) corresponding with the action that he wanted to do with his position name.

- ***def detection() :*** *(Appendices N° 12)*

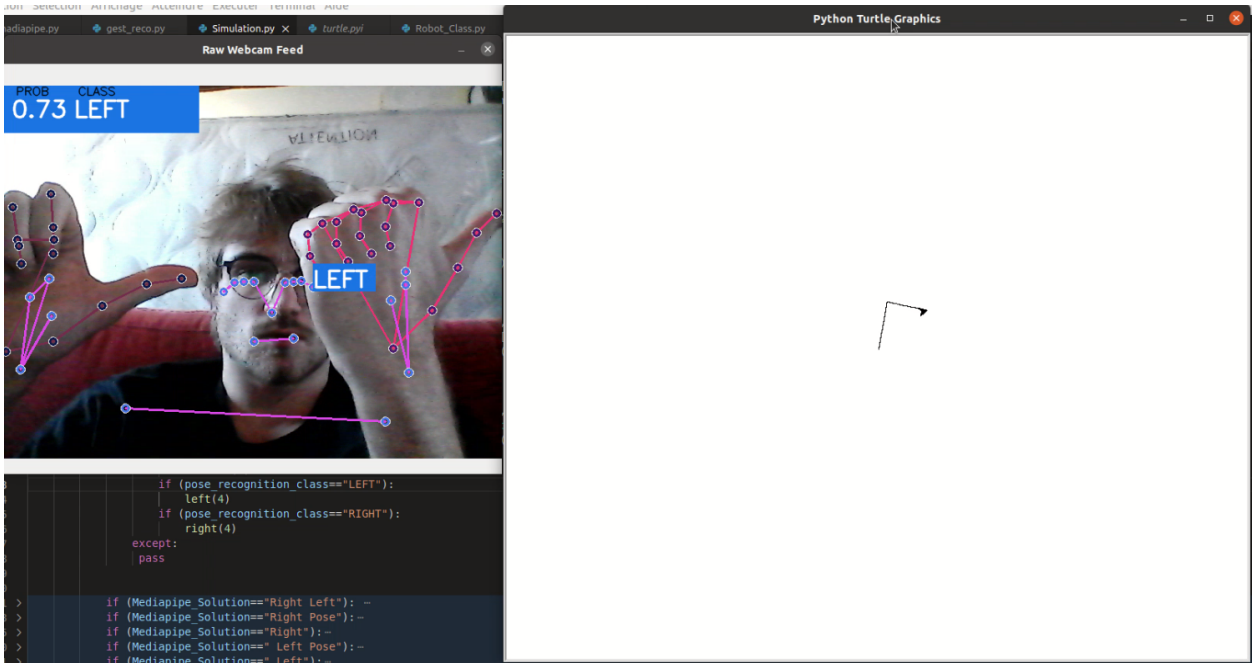
Now the last line of code will start the webcam with the mediapipe solution and will continuously compare the new value of the landmarks with the model, make a prediction of precision and print the result on the screen. That will also be in that loop that we would be able to define (depending on the position making) the instruction to the robot. Like if I make the position (GO STRAIGHT) I will have a loop with a function that say to the robot "Go straight" until the next position or the shutdown of the program.

```
# Here you can set up your robot driving parameters with the name of your class (like STOP, LEFT, RIGHT...) and the probability of precision
Prob=max(pose_recognition_prob)
if (pose_recognition_class==liste[0]) and (Prob>0.7):
    rc.move_straight()
elif (pose_recognition_class==liste[1]) and (Prob>0.7):
    rc.turn_left()
elif (pose_recognition_class==liste[2]) and (Prob>0.6):
    rc.turn_right()
elif (pose_recognition_class==liste[3]) and (Prob>0.7):
    rc.stop()
elif (pose_recognition_class==liste[4]) and (Prob>0.7):
    rc.move_back()
```

***Picture N°19 : Loop to give instruction to the robot depending on the position making***

Here you have an example of this loop. Two variables can be used to define the good instruction to give to the robot, the name of the class (STOP, RIGHT,LEFT...) and the probability of precision. In the picture N°18, you are able to see the basic loop to give information to the robot depending on the position that you make. It will depend if the position is detected and if the probability of precision is good enough. In these if loops you can easily see a function call from the class "***rc***", which is my "RobotControl" class that we will see in the next part.

To try for the first time I imported the turtle module and the function call that you can see in picture N°18 by the function call of the turtle module.




***Picture N°20 : Test of the program with the turtle module***

## 8) What's in my Robot control node ?

*(Appendices N°13)*

This node will be used to set up the information that you want to give to the robot if you want to move it. As you can see on the picture N°19, some functions are called and that's what you can see in the picture N°21. Which are typical functions to give information to the robot.

```python
class RobotControl():

    def __init__(self):
        rospy.init_node('robot_control_node', anonymous=True)
        self.vel_publisher = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
        self.summit_vel_publisher = rospy.Publisher('/summit_xl_control/cmd_vel', Twist, queue_size=1)

        self.cmd = Twist()

        self.ctrl_c = False
        self.rate = rospy.Rate(10)


    def publish_once_in_cmd_vel(self):…


    def move_straight(self):

        # Initilize velocities
        self.cmd.linear.x = 0.05
        self.cmd.linear.y = 0
        self.cmd.linear.z = 0
        self.cmd.angular.x = 0
        self.cmd.angular.y = 0
        self.cmd.angular.z = 0

        # Publish the velocity
        self.publish_once_in_cmd_vel()
```

***Picture N°21 : Some functions of the "RobotControl" class***

In this picture, you see that I created a class named "***RobotControl***" and I call it in my simulation node with the value "***rc***".

Here, you can find firstly the "***__init__***" function, it was used to initialize the node and to set up our publisher.

Secondly, in the "***move_straight***" function you will have the information that we wanted to publish on the topic "cmd_vel" to pilot the robot. Here, "self.cmd.linear.x=0.05" can be translated by "move the robot on the x axis with a velocity of 0.05".

The velocity is a value from 0 to 1, which is similar to the speed of the robot. A velocity of 0 will not move the robot but a velocity of 1 will move the robot on his higher speed value.

# III/ Personal and professional retrospective

In a professional point of view, I succeed in taking and storing datas of the landmarks allowed by Mediapipe and to build a machine learning model with these datas to predict the position that the user makes in real time. I also succeed in controlling a robot with my program during the simulation phase. All the objectives are achieved at the end of this period, so I'm very proud. I heard that my program will be used by the next erasmus student to build it with "Amazon Alexa"

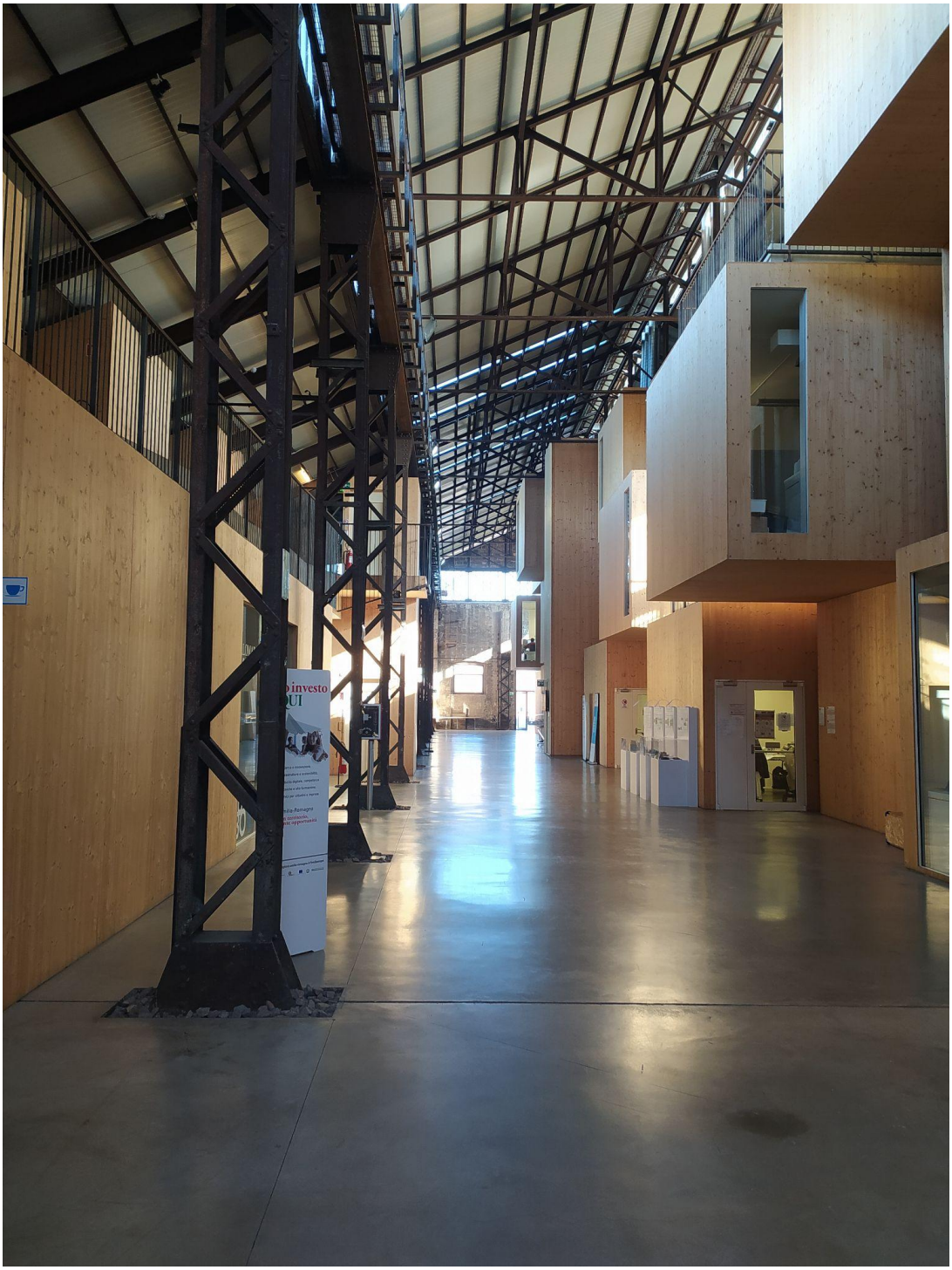
I also sent my package to my tutor with a description of my code and how it works. A list of future things that can be added to my code to try different solutions offered by mediapipe.

In a personal point of view, obviously I will say that I learned a lot during this period and with this project. Before this, I had never done programming, even less robotics and machine learning. Now I can say that I have few knowledge in that field and that's very rewarding. That field of study trained me to structure my work, think to build on a different way to be more understandable to future other students who may use my work to theirs .

My only regret is that with the covid situation I had to stay a lot of time working from home.

# Appendices

## *N°1 : The technopolo*



## *N°2 : The laboratory*

## N°3 : The param() function of the Publisher node

```
def param():
    #Set up some global variables to be used in different function of the node
    global E_D_Webcam
    global E_D_Right_Hand
    global E_D_Left_Hand
    global E_D_Pose


    #Print the value of the default parameters
    print("The default parameters are the following one : ")
    print("Webcam :",E_D_Webcam)
    print("Right Hand :",E_D_Right_Hand)
    print("Left Hand :",E_D_Left_Hand)
    print("Pose :",E_D_Pose)
    print("If you want to use your webcam, 0 is the good value, if you want to use an external camera write webcam after")
    #Create a loop if the user want to change some of the parameters
    choice_okay="False"
    choice=input("Do you want to change something in the setup ?")
    if (choice=="yes" or choice=="Yes" or choice=="YES" or choice=="y"):
            while(choice_okay!="GO" ):
                choice_param=input("Please write the name of the parameters that you want to disable ")
                if (choice_param=="Webcam" or choice_param=="webcam"):
                    E_D_Webcam=2

                elif (choice_param=="Right Hand" or choice_param=="right hand" or choice_param=="Right hand"):

                    E_D_Right_Hand="disable"
                    print(E_D_Right_Hand)

                elif (choice_param=="Left_Hand" or choice_param=="left hand" or choice_param=="Left Hand" or choice_param=="Left hand"):
                    E_D_Left_Hand="disable"

                elif (choice_param=="Pose" or choice_param=="pose"):
                    E_D_Pose="disable"


                else :
                    print("You don't write it corectly, please retry")

                choice_okay=input("Are you done with the setup, if it's okay please write : GO , if it's not write anything ")
```

## *N°4 : The create TXT part of the createfiles() function*

```python
def createfiles():
    # Firstly I create some txt files to save the parameters of this session to use it in the following nodes

    # Write the project name on a TXT file
    File_Name=input("What is your project name ?")
    Project_name_txt=open('/home/baheu/ws_sk_tracking/src/sk_tracking/TXT file/projectname.txt','w')
    Project_name_txt.write(File_Name)
    Project_name_txt.close()

    # Write the different solution used on a TXT file
    Solution=""
    if (E_D_Right_Hand=="enable"):
        Solution="Right"
    if (E_D_Left_Hand=="enable"):
        Solution= Solution + " Left"
    if (E_D_Pose=="enable"):
        Solution= Solution + " Pose"
    Solution_txt=open(f'/home/baheu/ws_sk_tracking/src/sk_tracking/TXT file/Solution_{File_Name}.txt','w')
    Solution_txt.write(Solution)
    Solution_txt.close()
```

## *N°5 : A loop to create CSV in the createfiles() function*

```python
if (E_D_Right_Hand == "enable" and  E_D_Left_Hand=="enable" and E_D_Pose=="enable"):
    landmarks = ['class']
    for val in range(1, 75+1):
        landmarks += ['x{}'.format(val), 'y{}'.format(val), 'z{}'.format(val), 'v{}'.format(val)]

    with open(f'/home/baheu/ws_sk_tracking/src/sk_tracking/CSV files/{File_Name}.csv', mode='w', newline='') as f:
        csv_writer = csv.writer(f, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
        csv_writer.writerow(landmarks)
```

## *N°6 : A loop to extract and publish landmarks in the talker() function*

```
# All the different loop of extracting and publishing the landmarks, the choosen loop will depend with which parameters you will use
 if (E_D_Right_Hand == "enable" and  E_D_Left_Hand=="enable" and E_D_Pose=="enable" ):  #This loop will use 3 mediapipes solutions (Right hand, Left Hand and Pose)
     # Drawing of right hand points on the screen
     mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                                     mp_drawing.DrawingSpec(color=(80,22,10), thickness=2, circle_radius=4),
                                     mp_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2)
                                     )
     #Drawing of the left hand points on the screen
     mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                                     mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                                     mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                                     )
     # Drawing of the Pose points on the screen
     mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS,
                                     mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),
                                     mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                                     )
     msg_h = Holistic()  # Set a variable as the type of my msg file called Holistic
     if results.right_hand_landmarks :

        Right_hand_results = results.right_hand_landmarks.landmark  #Set up a variable with the value of all my right hand landmarks
        right_hand_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in Right_hand_results]).flatten())   #Create a list w

     if results.left_hand_landmarks:

        Left_hand_results= results.left_hand_landmarks.landmark  #Same for left hand
        left_hand_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in Left_hand_results]).flatten()) #Same

     if results.pose_landmarks:

        Pose_results= results.pose_landmarks.landmark
        pose_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in Pose_results]).flatten())



     land=right_hand_row+left_hand_row+pose_row     #Create a unique list with the value of the landmarks of the right hand, left hand and the pose
     msg_h.Name=class_name  #Insert my class name into the Holistic message structure
     msg_h.H_Key=land       #Insert my list of landmarks into the Holistic message structure
     pub_h.publish(msg_h)   #Publish the messages
```

## N°7 : The Subscriber node

```python
#!/usr/bin/env python3
import rospy
from sk_tracking.msg import Holistic
import csv

def H_callback(data):

    var = []
    rh=list(data.H_Key)
    name=data.Name
    var.append(name)
    var.extend(rh)


    with open(f'/home/baheu/ws_sk_tracking/src/sk_tracking/CSV files/{File_Name}.csv', mode='a', newline='') as f:
        csv_writer = csv.writer(f, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
        csv_writer.writerow(var)

def listener():

    rospy.init_node('listener', anonymous=True)
    rospy.Subscriber("H_Topic", Holistic, H_callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    Isitgood=input("Write GO when you have set the project name : ")
    if Isitgood=="go" or "GO" or "Go":
        Fn=open('/home/baheu/ws_sk_tracking/src/sk_tracking/TXT file/projectname.txt','r')
        File_Name=Fn.read()

        listener()
```

## N°8 : Settings step of machine learning node

```
#SETTINGS STEP

print("\nDo you want to build a new solution model or to use a previous one ? ")
Build_or_not=input("\nWrite NEW for a new solution and PREVIOUS for a previous solution :  " )
webcam=0
choice_cam=input("\nWrite YES if you want to use an external camera, otherwise write NO" )
if (choice_cam=="YES" or choice_cam=="yes" or choice_cam=="Yes"or choice_cam=="y"or choice_cam=="Y"):
    webcam=2
if (Build_or_not=="NEW" or Build_or_not=="New" or Build_or_not=="new"):
```

## N°9 : Loop to create a new machine learning model

```
# LOOP TO CREATE A NEW MACHINE LEARNING MODEL

    Choice_Okay=False
    while (Choice_Okay == False):
        print("\nThe following name are your previous position saving, type the name before the point to use it !")
        dirPath = r"/home/baheu/ws_sk_tracking/src/sk_tracking/CSV files"                                   #Take all files name stored in that path
        result = [f for f in os.listdir(dirPath) if os.path.isfile(os.path.join(dirPath, f))]               #Take all files name stored and create a list
        print(result)
        Solution_Choice=input("\nWhich want do you want to use ? (Type the exact name before the point) :  ")
        M_Solution=open(f'/home/baheu/ws_sk_tracking/src/sk_tracking/TXT file/Solution_Name/Solution_{Solution_Choice}.txt','r')       #Read the Solution TXT file with the name of yo
        Mediapipe_Solution=M_Solution.read()                                                                #Same
        Pos_Name=open(f'/home/baheu/ws_sk_tracking/src/sk_tracking/TXT file/Position_Name/Position_Name_{Solution_Choice}.txt','r')    #Read the Postion name TXT file with the name o
        Position_Name=Pos_Name.read()                                                                       #Same
        print(f'\nYou choose {Solution_Choice} and you set up these following position : {Position_Name}  \n This position will use the following solution : {Mediapipe_Solution} ')

        Setup_of_robot_action()

        Continue=input("\nIf you made a mistake during the setup please write NO, otherwise write YES :  ")
        if (Continue=="YES") or (Continue=="y") or (Continue== "Yes") or (Continue=="yes"):
            Choice_Okay=True

    train_model()

    detection()
```

# *N°10 : Loop to start simulation with an existing model*

```
#LOOP TO USE A PREVIOUS MACHINE LEARNING MODEL
else :
    Choice_Okay2=False
    while (Choice_Okay2 == False):
        print("\nThe following name are your previous position saving, type the name before the point to use it !")
        dirPath = r"/home/baheu/ws_sk_tracking/src/sk_tracking/PKL files"
        result = [f for f in os.listdir(dirPath) if os.path.isfile(os.path.join(dirPath, f))]
        print(result)
        Solution_Choice=input("\nWhich want do you want to use ? (Type the exact name before the point) :  ")
        M_Solution=open(f'/home/baheu/ws_sk_tracking/src/sk_tracking/TXT file/Solution_Name/Solution_{Solution_Choice}.txt','r')
        Mediapipe_Solution=M_Solution.read()
        Pos_Name=open(f'/home/baheu/ws_sk_tracking/src/sk_tracking/TXT file/Position_Name/Position_Name_{Solution_Choice}.txt','r')
        Position_Name=Pos_Name.read()
        print(f'\nYou choose {Solution_Choice} and you set up these following position : {Position_Name}  \nThis position will use the following solution : {Mediapipe_Solution} ')

        Setup_of_robot_action()

        Continue=input("\nIf you made a mistake during the setup please write NO, otherwise write YES :  ")
        if (Continue=="YES") or (Continue=="y") or (Continue== "Yes") or (Continue=="yes"):
            Choice_Okay2=True

    detection()
```

## *N°11 : Loop to create a new machine learning model*

```python
def train_model():
    #3/ TRAIN CUSTOM MODEL USING SCIKIT LEARN
    # 3.1/ READ IN COLLECTED DATA AND PROCESS

    df = pd.read_csv(f'/home/baheu/ws_sk_tracking/src/sk_tracking/CSV files/{Solution_Choice}.csv')
    X = df.drop('class', axis=1)
    y = df['class']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1234)


    # 3.2/ TRAIN MACHINE LEARNING CLASSIFICATION MODEL

    pipelines = {
        'lr':make_pipeline(StandardScaler(), LogisticRegression()),
        'rc':make_pipeline(StandardScaler(), RidgeClassifier()),
        'rf':make_pipeline(StandardScaler(), RandomForestClassifier()),
        'gb':make_pipeline(StandardScaler(), GradientBoostingClassifier()),
                }

    fit_models = {}
    for algo, pipeline in pipelines.items():
        model = pipeline.fit(X_train, y_train)
        fit_models[algo] = model

    # 3.3/ EVALUATE AN SERIALIZE MODEL

    for algo, model in fit_models.items():
        yhat = model.predict(X_test)
        print(algo, accuracy_score(y_test, yhat))

    with open(f'/home/baheu/ws_sk_tracking/src/sk_tracking/PKL files/{Solution_Choice}.pkl', 'wb') as f:
        pickle.dump(fit_models['rf'], f)
```

## *N°12 : Loop to start the simulation 1/4*

## *Create and display on the mediapipe solution on the screen*

```
def detection():

    with open(f'/home/baheu/ws_sk_tracking/src/sk_tracking/PKL files/{Solution_Choice}.pkl', 'rb') as f:
        model = pickle.load(f)

    cap = cv2.VideoCapture(0)

    # Initiate holistic model
    with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:

        while cap.isOpened():
            ret, frame = cap.read()

            # Recolor Feed
            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            image.flags.writeable = False

            # Make Detections
            results = holistic.process(image)

            # Recolor image back to BGR for rendering
            image.flags.writeable = True
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)


            if (Mediapipe_Solution=="Right Left Pose"):
                # 2. Right hand
                mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                                          mp_drawing.DrawingSpec(color=(80,22,10), thickness=2, circle_radius=4),
                                          mp_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2)
                                          )
                # 3. Left Hand
                mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                                          mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                                          mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                                          )
                # 4. Pose Detections
                mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS,
                                          mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),
                                          mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                                          )
```

## *N°12 : Loop to start the simulation 2/4*

## *Extract the landmarks and compare them with the model*

```python
# Export coordinates
try:

    # Extract Pose landmarks
    pose = results.pose_landmarks.landmark
    pose_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in pose]).flatten())


    #Extract Right Hand landmarks
    right_hand = results.right_hand_landmarks.landmark
    rh_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in right_hand]).flatten())

    #Extract Left Hand landmarks
    left_hand = results.left_hand_landmarks.landmark
    lh_row = list(np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark in left_hand]).flatten())

    # Concate rows
    row = rh_row+lh_row+pose_row

    X = pd.DataFrame([row])
    pose_recognition_class = model.predict(X)[0]
    pose_recognition_prob = model.predict_proba(X)[0]
    print(pose_recognition_class, pose_recognition_prob)
```

## *N°12 : Loop to start the simulation 3/4*

## *Print the information of the detected class and the probability of precision on the screen*

```
# Grab ear coords
coords = tuple(np.multiply(
                np.array(
                    (results.pose_landmarks.landmark[mp_holistic.PoseLandmark.LEFT_EAR].x,
                    results.pose_landmarks.landmark[mp_holistic.PoseLandmark.LEFT_EAR].y))
            , [640,480]).astype(int))

cv2.rectangle(image,
            (coords[0], coords[1]+5),
            (coords[0]+len(pose_recognition_class)*20, coords[1]-30),
            (245, 117, 16), -1)
cv2.putText(image, pose_recognition_class, coords,
            cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

# Get status box
cv2.rectangle(image, (0,0), (250, 60), (245, 117, 16), -1)

# Display Class
cv2.putText(image, 'CLASS'
            , (95,12), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(image, pose_recognition_class.split(' ')[0]
            , (90,40), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

# Display Probability
cv2.putText(image, 'PROB'
            , (15,12), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE_AA)


cv2.putText(image, str(round(pose_recognition_prob[np.argmax(pose_recognition_prob)],2))
            , (10,40), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
```

***N°12 : Loop to start the simulation 4/4***

***Loop to give information to pilot the robot***

```
# Here you can set up your robot driving parameters with
Prob=max(pose_recognition_prob)
if (pose_recognition_class==liste[0]) and (Prob>0.7):
    rc.move_straight()
elif (pose_recognition_class==liste[1]) and (Prob>0.7):
    rc.turn_left()
elif (pose_recognition_class==liste[2]) and (Prob>0.6):
    rc.turn_right()
elif (pose_recognition_class==liste[3]) and (Prob>0.7):
    rc.stop()
elif (pose_recognition_class==liste[4]) and (Prob>0.7):
    rc.move_back()
```

## *N°13 : Robot_class node 1/2*

## *Function ___init___ and publish_once_in_cmd_vel and move_straight*

```python
#!/usr/bin/env python3

import rospy
from geometry_msgs.msg import Twist



class RobotControl():

    def __init__(self):
        rospy.init_node('robot_control_node', anonymous=True)
        self.vel_publisher = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
        self.summit_vel_publisher = rospy.Publisher('/summit_xl_control/cmd_vel', Twist, queue_size=1)

        self.cmd = Twist()

        self.ctrl_c = False
        self.rate = rospy.Rate(10)


    def publish_once_in_cmd_vel(self):
        """
        This is because publishing in topics sometimes fails the first time you publish.
        In continuos publishing systems there is no big deal but in systems that publish only
        once it IS very important.
        """
        while not self.ctrl_c:
            connections = self.vel_publisher.get_num_connections()
            summit_connections = self.summit_vel_publisher.get_num_connections()
            if connections > 0 or summit_connections > 0:
                self.vel_publisher.publish(self.cmd)
                self.summit_vel_publisher.publish(self.cmd)
                #rospy.loginfo("Cmd Published")
                break
            else:
                self.rate.sleep()


    def move_straight(self):

        # Initilize velocities
        self.cmd.linear.x = 0.1
        self.cmd.linear.y = 0
        self.cmd.linear.z = 0
        self.cmd.angular.x = 0
        self.cmd.angular.y = 0
        self.cmd.angular.z = 0

        # Publish the velocity
        self.publish_once_in_cmd_vel()
```

## *N°13 : Robot_Class node 2/2*

## *Other functions to pilot the robot*

```
def move_back(self):

    # Initilize velocities
    self.cmd.linear.x = -0.1
    self.cmd.linear.y = 0
    self.cmd.linear.z = 0
    self.cmd.angular.x = 0
    self.cmd.angular.y = 0
    self.cmd.angular.z = 0

    # Publish the velocity
    self.publish_once_in_cmd_vel()

def turn_right(self):
    self.cmd.linear.x = 0
    self.cmd.linear.y = 0
    self.cmd.linear.z = 0
    self.cmd.angular.x = 0
    self.cmd.angular.y = 0
    self.cmd.angular.z = 0.1

    # Publish the velocity
    self.publish_once_in_cmd_vel()

def turn_left(self):
    self.cmd.linear.x = 0
    self.cmd.linear.y = 0
    self.cmd.linear.z = 0
    self.cmd.angular.x = 0
    self.cmd.angular.y = 0
    self.cmd.angular.z = -0.1

    # Publish the velocity
    self.publish_once_in_cmd_vel()

def stop(self):
    self.cmd.linear.x = 0
    self.cmd.linear.y = 0
    self.cmd.linear.z = 0
    self.cmd.angular.x = 0
    self.cmd.angular.y = 0
    self.cmd.angular.z = 0

    # Publish the velocity
    self.publish_once_in_cmd_vel()
```