ACKNOWLEDGEMENT

First, we would like to thank ALLAH the almighty, for giving us the strength, knowledge and the ability to undertake this project study and complete it satisfactorily. Without his blessings, this achievement would not have been possible.

We would like to express our deepest sense of gratitude to Dr. MAACHE Ahmed for his guidance, and to all and every teacher who taught and guide us from primary school to the last year of study.

DEDICATION

Every challenging work needs self-efforts as well as guidance of Elders those who were very close to our heart.

Our humble efforts we dedicate to our loving

***Parents** and our **family members**, whose affection, love, encouragement and prays of day and night make me able to get such success and honor.*

*Along with all **my friends**, **hardworking** and respected **Teachers.***

*We also dedicate this work to **XERISE studio** and Ahmed Bentaleb for helping along this entire journey.*

ABSTRACT

This project consists of designing and building a hand gesture-controlled robot arm platform. The appropriate hardware equipment suitable for the desired tasks and for the robot arm movement are selected. Moreover, the robot arm architecture is designed. Then, the Robot Operating System (ROS) is used for the software implementation, synchronization and communication of the system. The first ROS node which recognize the hand gestures is coded on the laptop whereas the second node which controls the servos of the robot is build on the raspberry pi. The data is sent to second node in real time to generate appropriate control signals for the robot servos. The robot arm is 3D printed with 3 axis of rotation and a claw hand. We tried to control each servo alone according to some gestures so that we can move the arm wherever we want without any predefined coordinates.

Table of Contents

List of Figures:

GLOSSARY

3D : Three-dimensional
API: Application Programming Interface.
ARM :Advanced RISC (Reduced Instruction Set Computer) Machine
FPS: Frame Per Second
GPIO: General Purpose Input Output.
GPU: Graphics Processing Unit
ML : Machine Learning
OS : Operatin system
PWM : Pulse Width Modulation
RAM : Random Access memory
ROS:Robotic Operating system.
SGML : The Standard Generalized Markup Language
SPI : Service Provider Interface
SRV: Ros service.
XML: eXtensible Markup Language.

GENERAL INTRODUCTION

Modern applications of industrial automation and robotics are now heavily reliable on image processing techniques to solve many actual problems that face robotics and instrumentation. Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. There exists two type of image processing, it can be either analog or digital processing. It consists of importing the images via image acquisition tools, analyzing and performing some operations on them then output either altered images or report that is based on image study. In the field of industrial robotics, the interaction between man and machine typically consists of programming and maintaining the machine by the human operator. For safety reasons, a direct contact between the working robot and the human has to be prevented. If the human assistance is needed in the control, then real time communication and speed in exchanging information plays a vital role in this case. The classical computer devices like keyboard, mouse and monitor requires encoding and decoding of data. In this manner the programmer needs to give the coordinates of the object to be grasped or moved by typing them. Using image processing can give more flexibility and speed: if, for instance the human wants to control a robot with a camera, it would be much better to just do the hand movements to the camera.

Therefore, the hand movement can be used as a communication channel to provide information about the behavior and the movement of the robot.

It is necessary that the control of the robot and the sending of data from multiple machines to be very fast, to prevent lagging and unexpected movements.

This report is organized as follows. Chapter 1 Background contains an overview of the industrial robotics, the media pipe for the hand detection and the explanation of Robot operating system.

Chapter 2 is a combination of both the design and the implementation, it shows a general picture of the Mediapipe based gesture control system, and shows all the steps of implementing a successful hand gesture control. First the Mediapipe library detects the hand landmarks then a gesture is perceived accordingly, then the ROS sends the data to the microcontroller (raspberry pi) that controls the robotic arm.

Chapter 3 is results and discussion in which we talked about the results and issues related to the hand detection and the robot control.

CHAPTER 01: BACKGROUND

1.1.Robotics

1.1.1. Definition

Robotics is the study of design, building, and use of machines (robots) to do jobs that are traditionally performed by humans. Robots are commonly utilized in areas such as vehicle manufacturing to do basic repetitive jobs, as well as in areas where workers must work in dangerous situations. Artificial intelligence is used in many parts of robotics robots may be endowed with the equivalent of human senses including vision, touch, and the capacity to perceive temperature. Some are even capable of making rudimentary decisions, and contemporary robotics research is focused on developing robots with a degree of self-sufficiency that will allow them to move and make decisions in an unstructured environment [1].

1.1.2. Industrial robotics

Industrial robots are being employed in a range of sectors and applications because they can be taught to execute dangerous, filthy, and repetitive jobs with constant precision and accuracy. They come in a variety of types, with the most frequent differentiating qualities being the reach distance, payload capacity, and the number of axes of motion (up to six) of its jointed arm.

1.1.3. ROS

The Robot Operating System (ROS) is a software development platform for robots. It's a set of tools, libraries, and protocols that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

The ability to exchange and reuse libraries and packages on another platform, as well as the modularity and encapsulation given by this platform, allowing the system to operate even if a subsystem fails, is a benefit of adopting ROS over traditional robotic development approaches.

1.1.3.1. ROS nodes

In ROS all of the functions or subsystems are represented by nodes, each node should be responsible for a single, module purpose (e.g. hand capture). Every single node has the ability to send and receive data from other nodes via topics, services, actions or parameters [6].

1.1.3.2. ROS topics

Nodes publish information over topics, which allows any number of other nodes to subscribe to and access that information. If a node wants to publish a message it needs to send it through a topic, whereas subscribing to that topic lets nodes be able to access to that message.

1.1.3.3. ROS messages

ROS provides over 200 predefined messages and the ability to create custom ones. Messages are exchanged between ROS nodes using publish/subscribe mechanism. One ROS node would publish the ROS message to certain ROS topic, while the other ROS node would subscribe to that ROS topic and obtain the sent ROS message. ROS Messages are typically described in text files inside msgs folder under ROS folder structure. These text files are following certain standards for description of ROS messages. The description format of ROS messages is fairly simple. Each ROS message is a data structure which contains primitive types (integers, floats or Booleans) or an array of primitive types. Additionally, ROS message can contain the other ROS message or an array of ROS messages as a data type. ROS messages can be also exchanged in direct communication between nodes, called ROS Services and in this case, the messages should be inside of the srv folder [14].

1.1.3.4. ROS services

Another mean of communication between nodes in ROS is services. Services use a call and response model contrasted with topics' model. In services, data is provided only when requested while in topics data is published in streams [6].

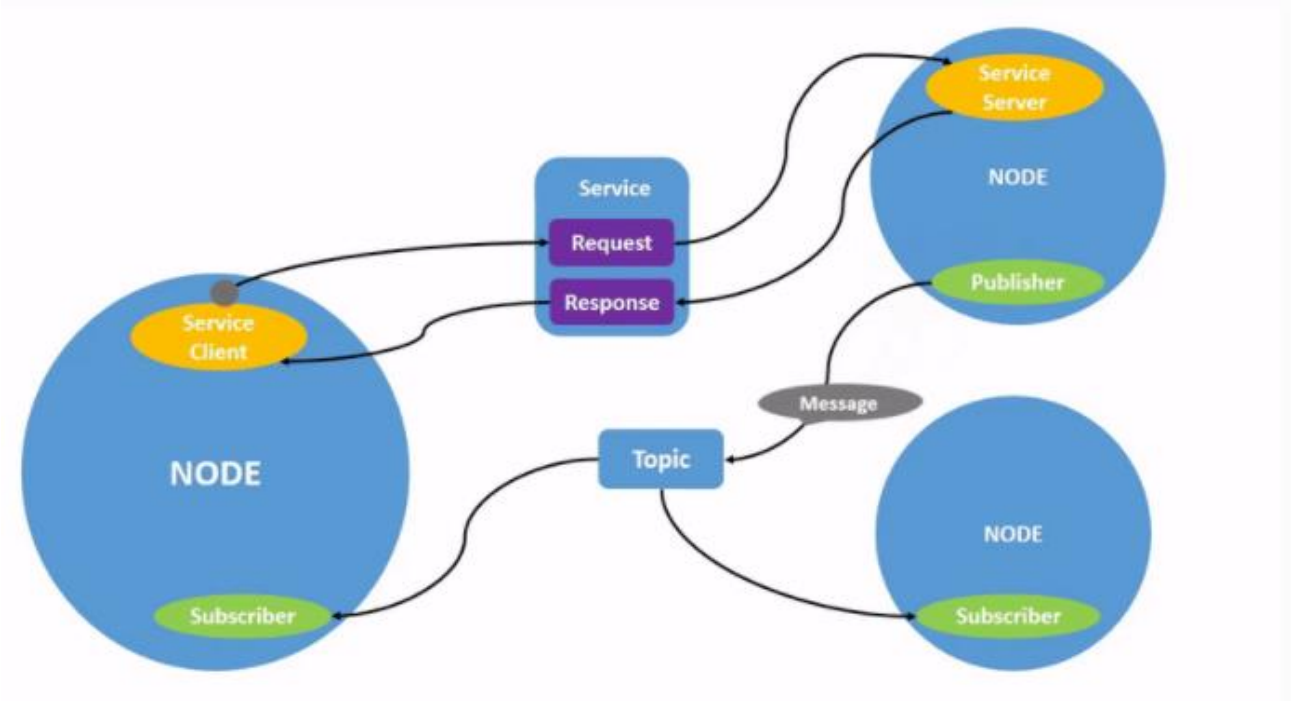


Figure 1.1: ROS nodes, topics and services [9].

1.1.4. Motion control

Motion control is a specialization in automated control systems, and its use is not limited to fundamental machine operation. It allows the machine tooling or the component itself to be moved in a controlled and frequently precise rotational or linear way [15].

1.2.Mediapipe

1.2.1. Definition

MediaPipe is a framework for creating applied ML pipelines that are multimodal (e.g. video, audio, any time series data) and cross platform (i.e. Android, iOS, web, edge devices). A perception pipeline may be constructed using MediaPipe as a network of modular components, such as implication models (e.g., TensorFlow, TFLite) and media processing functions [3].

1.2.2. Why Mediapipe

Mediapipe is reliably fast when it comes to performance, it is able to speed up thanks to the use of GPU acceleration and multithreading. Such approaches are inherently challenging, but MediaPipe takes rein and handles them for you, as long as you follow appropriate graph-making procedures. Newer devices may run away with fps thanks to multi-threading and GPU acceleration, frequently being at FPS that are too high to see with the naked eye [4].

1.2.3. ML pipeline

Pipelines have become increasingly popular in data science, and it can be found everywhere, ranging from basic data pipelines to complicated machine learning pipelines. A pipeline's main goal is to make data analytics and machine learning operations more efficient.

ML pipeline refers to the means of automating the machine learning workflow by allowing data to be processed and associated into a model, which can then be examined to provide outputs. The process of feeding data into the ML model is totally automated using this form of ML pipeline.

Another form of ML pipeline is the process of breaking down your machine learning operations into separate, reusable, modular components that can then be pipelined together to generate models. Building models becomes more efficient and simpler with this form of ML pipeline, cutting out redundant work [5].

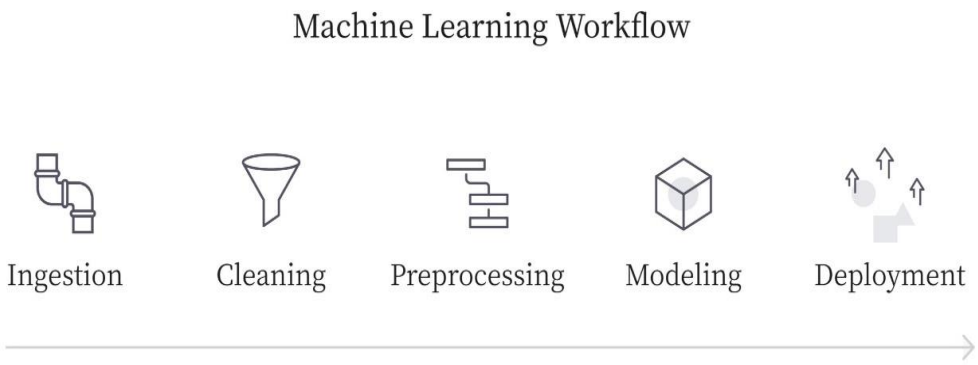


Figure 1.2: Machine learning workflow [5].

In a mainstream ML system design all the tasks shown above in Figure 1.2 are run together as a cluster. This means the same script will extract the data, clean and prepare it, model it, and deploy it. Since machine learning models usually consist of far less code than other software applications, the approach to keep all of the assets in one place makes sense.

However, each step of your workflow is abstracted into its own service using the ML pipeline. Then, whenever you create a new process, you can pick and choose the pieces you want to utilize

and place them where you want them, while any modifications to that service will be done at a higher level [5]

1.2.4. Applications

MediaPipe is used by many internal Google products and teams including: Nest, Gmail, Lens, Maps, Android Auto, Photos, Google Home, and YouTube [3].

Cutting edge ML models:

- Face Detection
- Multi-hand Tracking
- Hair Segmentation
- Object Detection and Tracking
- Objectron: 3D Object Detection and Tracking

Figure 1.3: Pipeline structure [16].

- AutoFlip: Automatic video cropping pipeline.

1.3. Technology and Terminology

1.3.1. Python

Python is an interpreted high-level general-purpose, object-oriented programming language which combines remarkable power with very clear syntax.

It's also expandable in C or C++, with interfaces to a variety of system functions, libraries, and window systems.

1.3.2. XML

Extensible Mark-up Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.

1.3.3. OpenCV

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today’s

systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features [7].

1.3.4. Raspberry pi 4:

The raspberry pi4 is a small, affordable microcontroller with a 40-pin GPIO, which can be programmed to function as an input or an output. This model comes with a 2GB RAM and a 1.5GHz quad core ARM microprocessor, and the range of connectivity options such as Wi-Fi (2.4GHz and 5GHz). The figure 1.4 below shows the GPIO pin list.

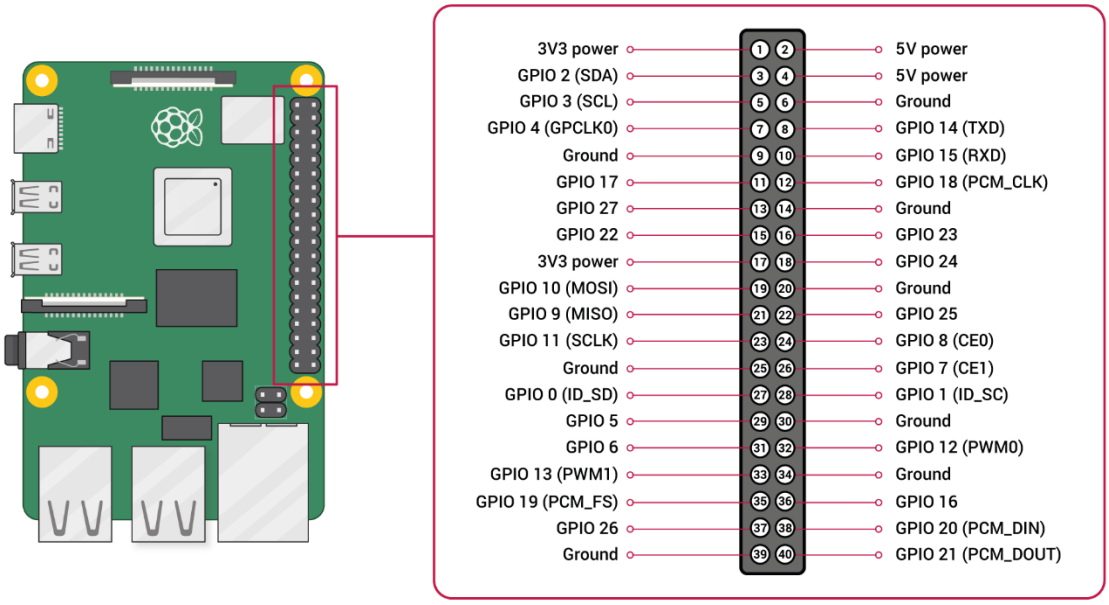


Figure 1.4: Raspberry pi 4 GPIO pins [17].

1.4. Summary

This chapter covers the essential ideas and language connected to Mediapipe, robotics technology and ROS. Starting with the fundamental procedures and principles of ML pipeline, descriptions and explanations of the technologies and terminology are provided.

CHAPTER 02: SYSTEM DESIGN AND IMPLIMENTATION

This chapter will go through the design and structure of the entire system, as well as the components that were chosen, in order to create a gesture-controlled robot that can integrate the ROS system and test it.

2.1. Overall System structure

The system could be divided into two main components

I. **Hand capture and landmarks detection** this section deals with the detection of the position of the hands and the manipulation of the landmarks to recognize hand gestures needed for the robot control
II. **Robot control** refers to the raspberry pi microcontroller and the 3D printed robotic arm circuit, the last which has three axes of rotation and a claw hand that's use to pick objects.

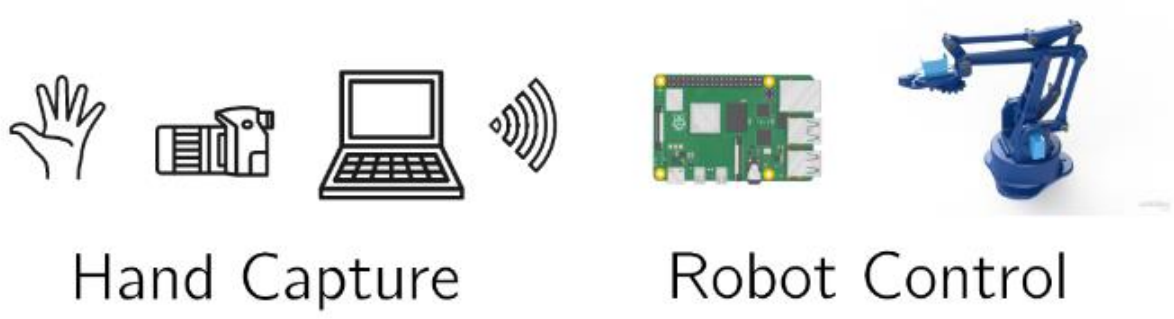


Figure2.1: System diagram

2.1.1. Hand capture platform

MediaPipe Hands utilizes an ML pipeline consisting of multiple models working together. A palm detection model that operates on the entire image and returns an oriented hand bounding box. A hand landmark model that returns high-fidelity 3D hand key point from the cropped image region defined by the palm detector.

Providing the hand landmark model with a correctly cropped hand image eliminates the requirement for data augmentation (e.g., rotations, translations, and scale) and allows the network to focus its resources on accuracy in coordinate prediction. In addition, in the pipeline the crops can also be generated based on the hand landmarks identified in the previous frame, and only when

the landmark model could no longer identify hand presence is palm detection invoked to relocate the hand [8].

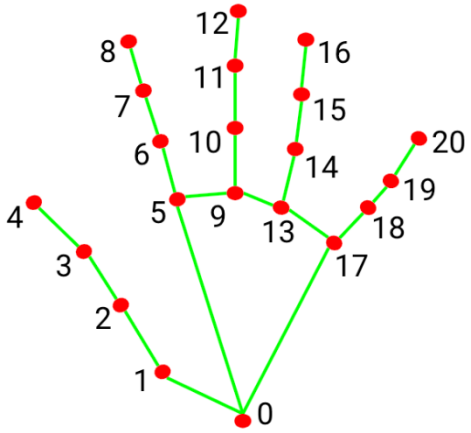
2.1.1.1. Palm Detection Model

To detect initial hand locations, a single-shot detector model optimized for mobile real-time. Detecting hands is a decidedly complex task, this model has to work across a variety of hand sizes with a large-scale span (~20x) relative to the image frame and be able to detect occluded and self-occluded hands.

The above challenge has been addressed using different strategies. First a palm detector is trained instead of a hand detector, because recognizing hands with flexible fingers is much more difficult than calculating bounding boxes of rigid objects like palms and fists. Moreover, as palms are smaller objects, the non-maximum suppression algorithm works well even for two-hand self-occlusion cases, like handshakes. Moreover, palms can be modelled using square bounding boxes (anchors in ML terminology) ignoring other aspect ratios, and therefore reducing the number of anchors by a factor of 3-5. Second, an encoder-decoder feature extractor is used for bigger scene context awareness even for small objects. Lastly, the focal loss is minimized during training to support a large number of anchors resulting from the high scale variance [8].

2.1.1.2. Hand Landmark Model

After the palm detection over the whole image the subsequent hand landmark model performs precise key point localization of 21 3D hand-knuckle coordinates inside the detected hand regions via regression, that is direct coordinate prediction. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions. The figure 2.2 shows the 21 landmarks.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 2.2: hand landmarks [8].

To start detecting the hand and obtaining its position and all the landmarks. We build the ROS packages and execute the publisher node.

After getting the image from the video capture, we call the find hands method that takes the image process it and draw the landmarks.

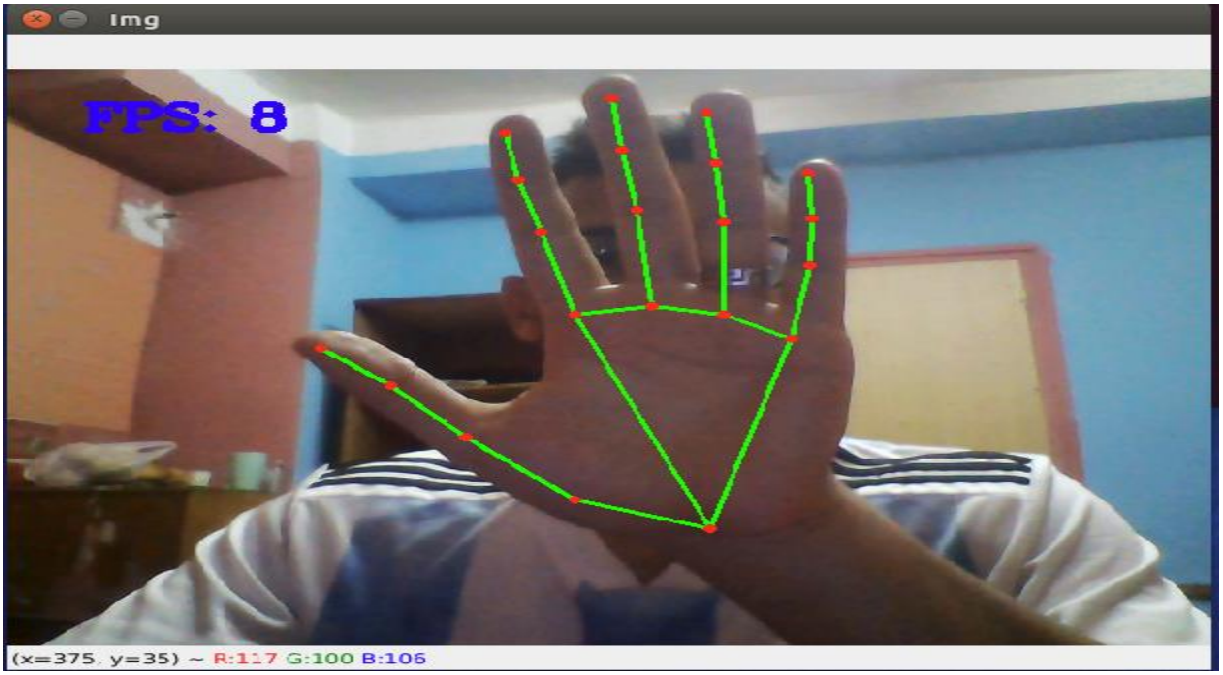


Figure 2.3: hand capture.

To better, tune the use of the media pipe library some parameters should be taken into consideration.

2.1.1.3. Static image mode

It is set to false, so solution treats the input images as a video stream. It will try to detect hands in the first input images, and upon a successful detection further localizes the hand landmarks. In subsequent images, it simply tracks those landmarks without invoking another detection until it loses track of any of the hands. This reduces latency and is ideal for processing video frames. If set to `true`, hand detection runs on every input image, ideal for processing a batch of static, possibly unrelated, images. Default to false.

2.1.1.4. Min tracking confidence:

Minimum confidence value ([0.0, 1.0]) from the landmark-tracking model for the hand landmarks to be considered tracked successfully, or otherwise hand detection will be invoked automatically on the next input image. Setting it to a higher value can increase robustness of the solution, at the expense of a higher latency. Ignored if **static image mode** is true, where hand detection simply runs on every image. Default to 0.5.

2.1.1.5. Gesture recognition:

We use the **LANDMARKS** output which contains a landmark list that contains 21 landmark. In the Figure2.4 below you can see the landmarks in a 3d space. Each landmark have **x**, **y** and **z** values. Only **x**, **y** values are sufficient for our Goal.

Figure 2.4: landmarks indexes

We have finger states:

1.thumbState

2.indexFingerState

3.middleFingerState

4.ringFingerState

5.littleFingerState

we declare the states as unknown and then we check if each finger is open or closed.

Thumb:

Thumb finger is closed if the x value of the landmark 3 is less than the x value of landmark 2 and the value of landmark 4 is less than the one of landmark 3. else we check if x of landmark 2 is less than x of landmark 3 and x of landmark 3 is less than the one of landmark 4.

for the thumb finger we used the x values since the finger points horizontally when close or open.

Index Finger:

For this finger to be open we checked if Y value of landmark 7 is less than the Y value of landmark 6 and Y value of landmark 7 is less then Y value of landmark 8. else we check if Y of 6 is less than Y of 7 and Y of 7 is less than Y of 8 and the finger is closed.

Middle finger:

for this finger to be open we check If the Y value of landmark 11 is less than 10 then Y of 12 is less than the one of landmark 11 else we see if the fixed key point 10 is less than the 11 and 11 is less than 12 than the finger is closed.

Ring finger:

In the same way we see if Y value of landmark 15 is less than Y of 14 and Y of 16 is less than Y of 15 than the finger is open else if the Y value of the fixed key point is less than Y value of landmark 15 and Y of 15 is less than Y 16 than the finger is closed

Little finger:

Lastly, we do the same thing for the little finger by checking the Y value of landmark 19 and see if its less than 18 and if Y of 20 is less than Y of 19 than the finger is open. Else it is closed.

So now after we know whether a finger is closed or open, we can distinguish between different hand gestures. In our case there is 6 hand gestures when the thumb is closed and three another gestures when the thumb is open. All the of six gestures are stated in the table below, the three other gestures are discussed after.

3.2.1 **Thumb closed**

Table 1: hand gestures

Hand gestures	Index	middle	ring	little
1	Open	Close	Close	Close
2	Open	Open	Close	Close
3	Open	Open	Open	Close
4	Close	Close	Close	Open
5	Close	Close	Open	Open
6	Close	Open	Open	Open

3.2.2 **Thumb open:**

When the thumb is open now, we applied some image processing. First, we draw a circle on both landmark 4 and 8 using cv2.circle function. Then we made a line between them using cv2.line and calculated the distance between the two landmarks when different hand movements are applied.

x1, y1 = lmList[4][1], lmList[4][2]

x2, y2 = lmList[8][1], lmList[8][2]

cx, cy = (x1+x2)//2 , (y1+y2)//2

cv2.circle(img,(x1,y1),15,(255,0,255), cv2.FILLED)

cv2.circle(img,(x2,y2),15,(255,0,255), cv2.FILLED)

cv2.line(img,(x1,y1),(x2,y2),(255,0,255),3)

cv2.circle(img,(cx,cy),15,(255,0,255), cv2.FILLED)

Also, we calculated the midpoint of the distance and draw a circle also to distinguish between the different states and make the things clear.

Figure 2.5 below shows the distance between the fingers in purple.

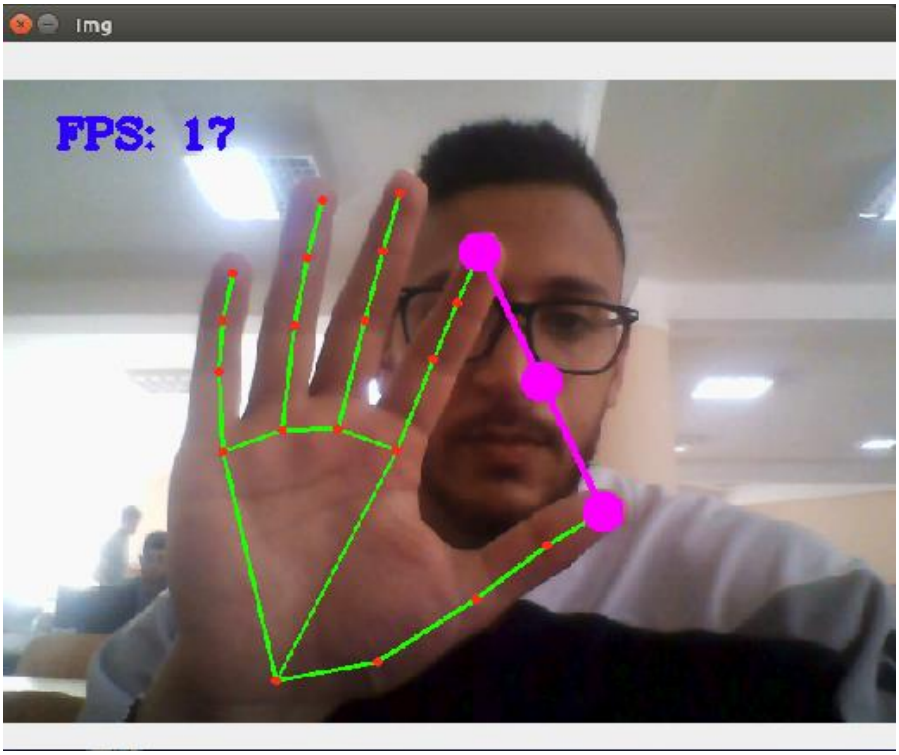


Figure 2.5: Thumb open state

2.2.Robot control

We controlled a 3D printed robotic arm that has three axes of rotation and a claw hand as shown below:

Figure 2.6: Robotic arm design [16].

This robot uses four SG90 servo motors, one for the claw hand, another for the base and two for the main arms.

The SG90 micro servo can rotate approximately 180 degrees (90 in each direction) and can move a max of 1.6kg of load.

In order to control the robotic arm, we used the raspberry pi 4 model B microcontroller that has a 40-pin GPIO, which can be programmed to function as an input or an output. This model comes with a 2GB ram and a 1.5GHz quad core arm microprocessor, and the range of connectivity options such as WiFi (2.4GHz and 5GHz), Gigabit Ethernet, Bluetooth and USB means you can code with convenience anytime, anywhere.

As well as simple input and output devices, the GPIO pins can be used with a variety of alternative functions, some are available on all pins, others on specific pins.

After getting the recognition of the different gestures we now integrated that in ROS in order to control the robot servos.
we created the talker node in ROS2 build on top of Ubuntu 20.04 in the laptop. This node publishes Commands to the above cmd_vel topic.

We created a simple publisher that publishes Twist messages of type geometry_msgs to the topic cmd_vel as follows:

p1 = self.create_publisher(Twist, '/cmd_vel', 10)
In the subscriber node we handled all the hardware setup of the raspberry pi 4. We used RPI.GPIO library.

We started by giving servo pin variables different values of GPIO pins.
servoPIN1 = 13
servoPIN2 = 18
servoPIN3 = 19 #FOR BOTTOM SERVO
servoPIN4 = 12 #FOR HEAD SERVO

we used those pins because of their hardware pwm support.

next we set the GPIO mode to BCM as follows:

GPIO.setmode(GPIO.BCM). To tell the library which pin numbering system we are going to use.BCM signifies the Broadcom SOC channel designation.

After that we set all the GPIO pins to be output pins:

GPIO.setup(servoPIN1, GPIO.OUT)
GPIO.setup(servoPIN2, GPIO.OUT)
GPIO.setup(servoPIN3, GPIO.OUT)
GPIO.setup(servoPIN4, GPIO.OUT)

we gave the pwm signal generated by the GPIO pins a frequency of 50 HZ:
p1 = GPIO.PWM(servoPIN1, 50)
p2 = GPIO.PWM(servoPIN2, 50)
p3 = GPIO.PWM(servoPIN3, 50)
p4 = GPIO.PWM(servoPIN4, 50)

Now we move to the part where we change duty cycles of the PWM signals of each pin. All the values are shown in the table 2 below.

Duty Cycle

"Duty cycle" is the width of positive pulse (**square wave**) and a deciding factor for servo's angular position. For example, if you have a servo with 180° turn, then 90° is the center position of the servo with 0° being minimum, and 180°, being the maximum. Now, if a positive pulse of 1.5ms is sent, then the servo stays at 90° (servo center) as long as it receives the same pulse. If another pulse of 1ms is sent, the circuit tries to move the shaft to 0°, and a pulse of 2ms tries to move the output shaft to 180°. This means, a pulse shorter than 1.5ms moves the servo in one direction and wider than 1.5ms moves it in another direction.

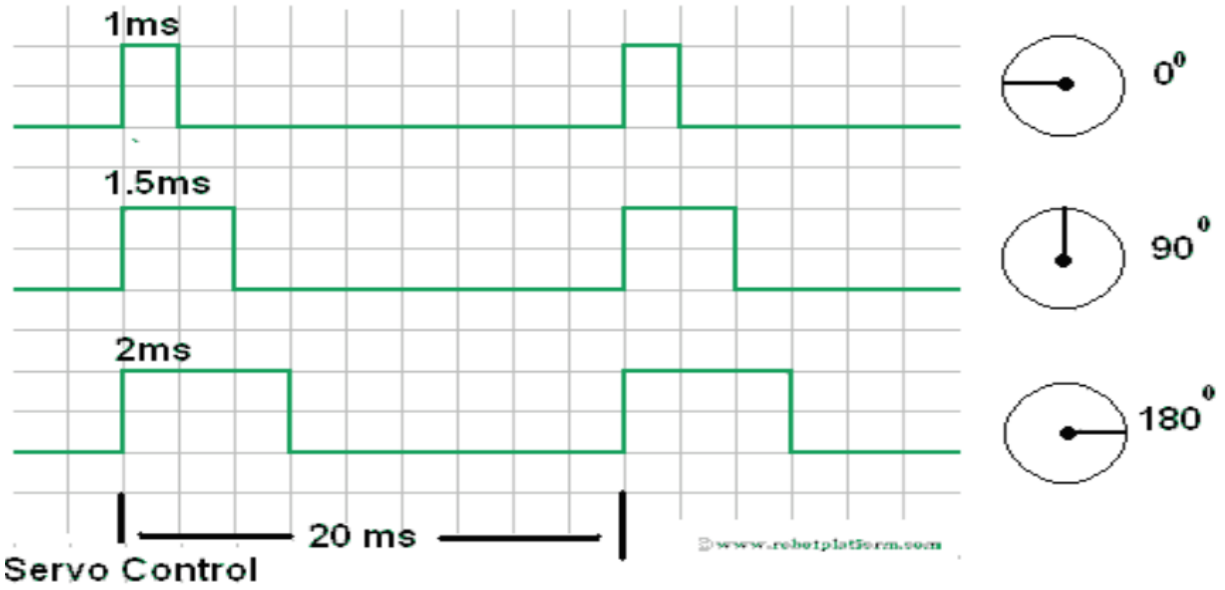


Figure2.7: Servo motor control signals.

Different servo models have different minimum and maximum pulse requirements. For example, a Hextronik servo used has a minimum pulse requirement of 0.5ms to move to 0° and maximum pulse duration of 2.5ms to move to 180°. Sending a pulse of 1ms moves it to 45° and 2ms moves it to 135°. Another servo requires 1ms pulse to move to 0°, 1.5ms to move to 45° and 2ms to move to 90° and maximum angular rotation being 90°. The de-facto standard is 1ms for minimal angle, 1.5ms for servo center and 2ms for maximum angle. In our case we used the SG90 servo motor.

Whenever the listener gets a value on the cmd_vel topic it fires a listener callback function. The callback function checks which value of linear.x is received and change duty cycle accordingly as it is shown in table 2:

Table 2: Duty Cycles

Gestures	1	2	3	4	5	6	7	8	9
Pwm 12							2.5	7.5	12.5
Pwm 13				2.5	7.5	12.5			
Pwm 18				2.5	7.5	12.5			
Pwm 19	2.5	7.5	12.5						

The table above contains the gestures we defined from 1 to 9 each 3 gestures control a certain pin For example : gesture 7 gives a 25 percent duty cycle signal to the PWM pin 12 to move to the initial position, 8 moves it to 90 degrees by giving a 75 duty cycle and the gestures 9 turn it to the maximum positon .

We tried to control each servo separately rather than predefining each movement to control the robot freely.
The gestures 7,8 and 9 refer to the thumb open states.

Table 3 Claw control using distance between fingers

Distance between thumb and index(x)	x than 50	50<x<150	x>150
PWM duty cycle for pin 12	2.5		
PWM duty cycle for pin 12		7.5	
PWM duty cycle for pin 12			12.5

Using rqt graph we visualize the ROS graph of our application to see all the running nodes, as well as the communication between them. The nodes and topics are displayed inside their namespace as illustrated in the figure below:

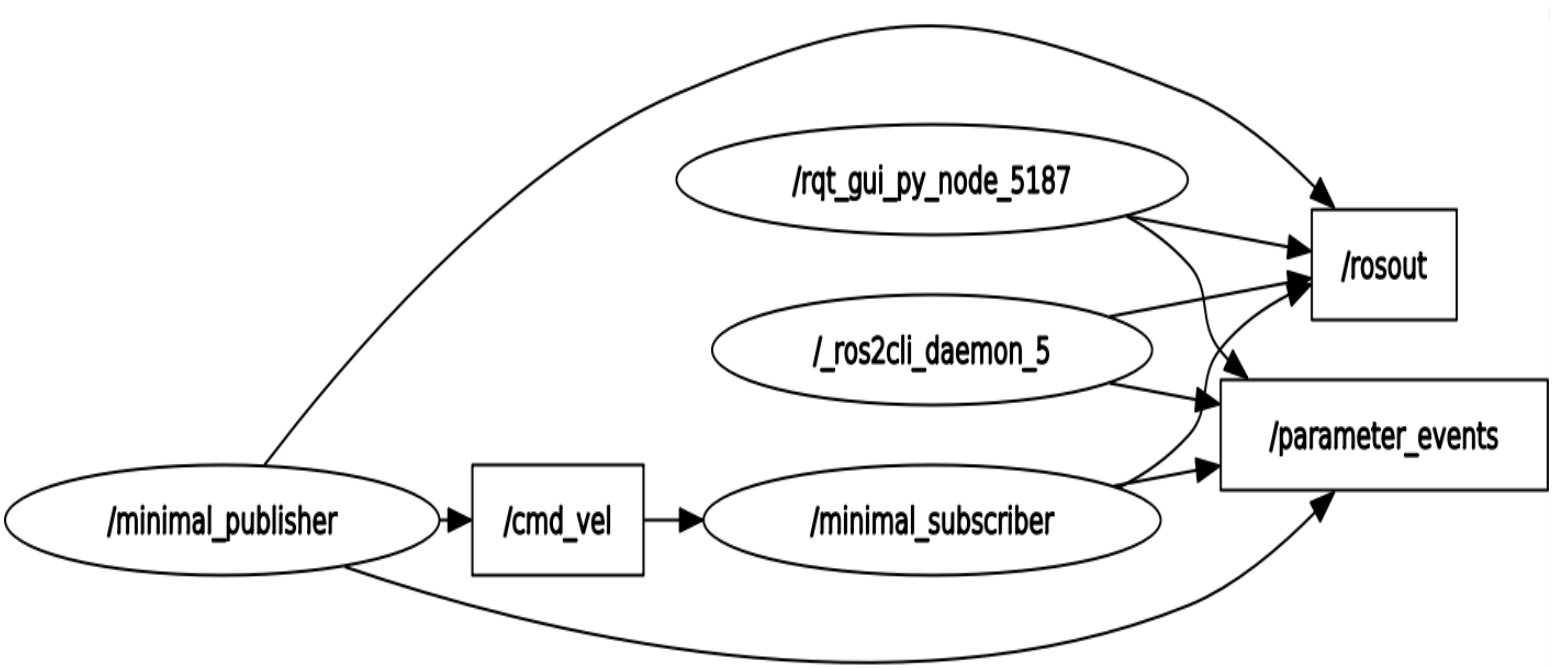


Figure 2.8: ROS rqt graph.

an overall flowchart can explain the logic behind the control:
The minimal_publisher node is the node responsible for defining the hand gestures and sending data accordingly to the minimal_subscriber node the cmd_vel topic receives the data sent and check which gestures is sent. The minimal subscriber node check that data and sent PWM signals to the pins.The daemon node is running all the time to track all the nodes it has information about nodes lifespan, dead nodes, and the new launched nodes.

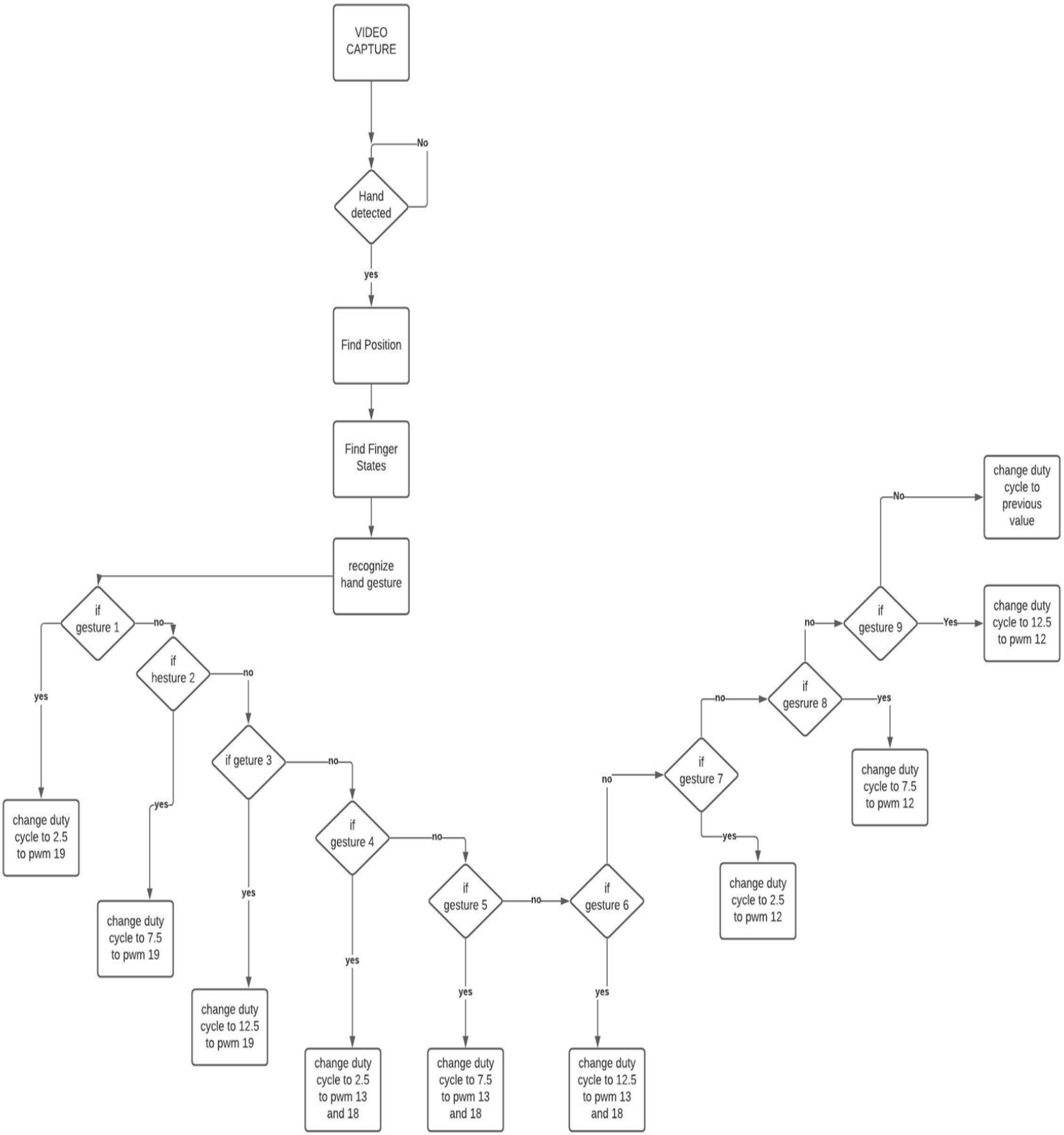


Figure 2.9: General flow chart of the hand capture

2.3.Summary

This chapter covers the implementation of the Mediapipe library and robot control using the raspberry pi microcontroller.

All the necessary configuration and design of the system has been presented and clarified including the Mediapipe library and ROS environment for control and communication.

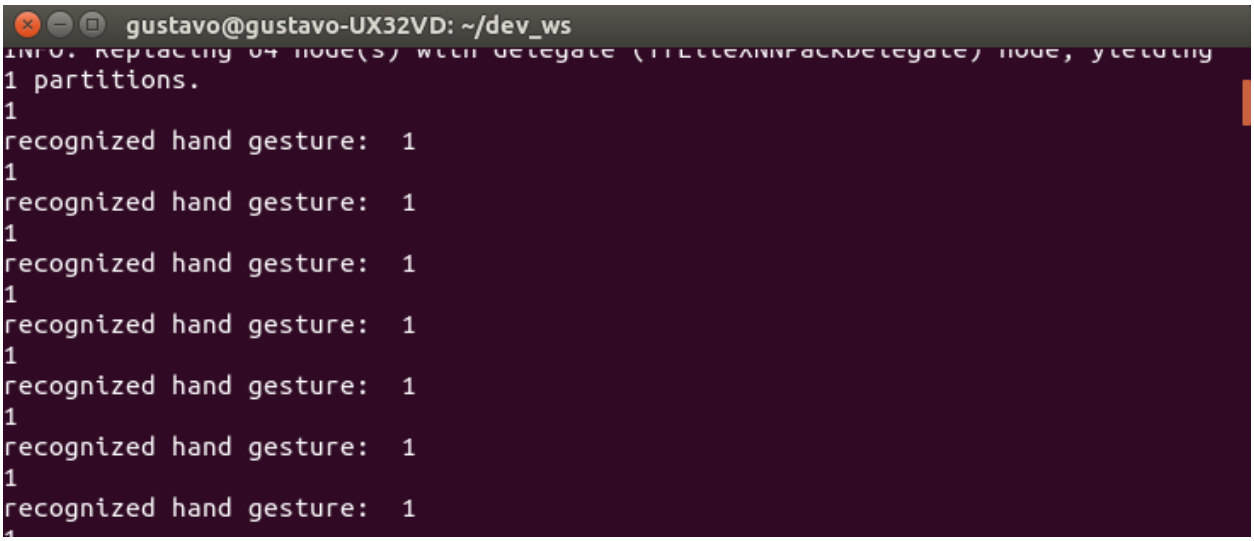
CHAPTER 03: RESULTS AND DISCUSSION

In this chapter we will discuss the results, problems and suggestions that came around during our system testing and implementation.

3.1.Results

This section will present the results of the gesture control using the ROS:

As we can see in figure 3.1 and 3.2, we could recognize the gesture and move the robot arm accordingly. First, we open a terminal and we start a talker node for the hand capture.



After detecting the gesture, the talker node publishes the data to the topic where our listener

Figure 3.1: Gesture 1 recognition.

node reads it and controls the robot consequently

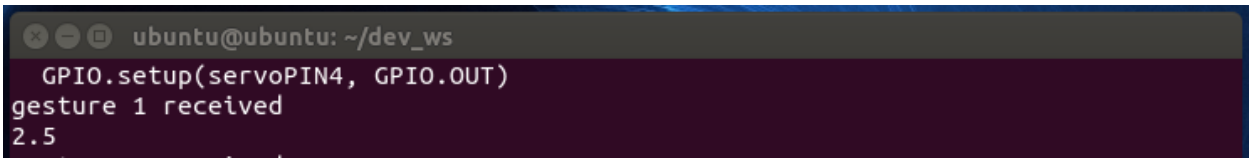


Figure 3.2: Gesture 1 reception.

This works for different gestures, a total of 9 gestures was declared and used

Figure 3.3: Gesture 2 recognition.

Figure 3.4: Gesture 2 reception.

Other gestures were detected successfully

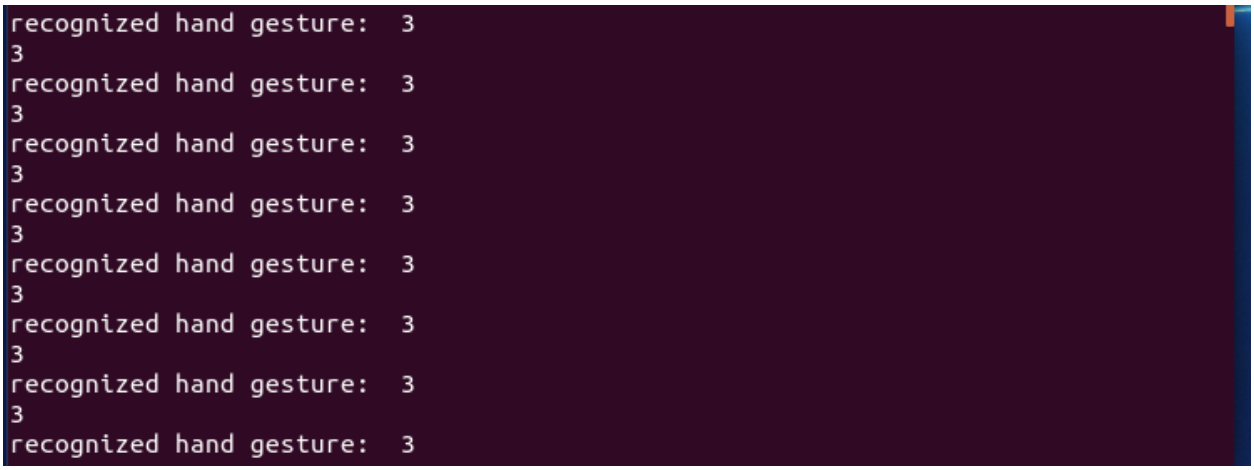


Figure 3.5: Gesture 3 recognition.

.

3.2.Discussion

Using Mediapipe made hand detection very easy and reliable through its multi-threading and GPU acceleration. The library supports Multi-platform usage that helped us in working on ubuntu because ROS environment is recommended in ubuntu OS.

We found that recognition was very fast because of the high fps and hence it kept publishing infinitely the data to the cmd_vel topic and that the microcontroller was struggling and couldn't control the servos properly. So, we overcame this problem by checking whether the recognized hand gesture was the same then we don't have to publish any new data.Often when launching the video capture when get a loss and a low fps that causes problems when moving the hand to a different gesture that was caused by low light that means there is more noise in the video. With more noise the encoding engine has to put far more effort to get the compression it needs. Unless the encoder has a denoiser the encoding engine has far more noise to deal with than normal conditions.we could also improve our FPS simply by creating a new thread that does nothing but poll the camera for new frames while our main thread handles processing the current frame.This is a simple concept, but it's one that's rarely seen in OpenCV examples since it does add a few extra lines of code (or sometimes *a lot* of lines, depending on your threading library) to the project. Multithreading can also make your program harder to debug, but once you get it right, you can dramatically improve your FPS.

General Conclusion:

The aim of this project was to implement a Mediapipe based hand gesture control for a robotic arm. The robot arm must able to rotate in different possible directions holding certain weight.

In the beginning, we started talking about robotics in general and their role in the industry either by reducing the human struggles when doing high precision tasks or by proving fast product delivery, then a clear description of the overall system structure where the details of the implementation are stated.

We started by the gesture detection procedure where we get a video from the camera, process the frames to detect the hand landmarks, then we manipulated them to know first if a finger is open or closed then by knowing that we can define different hand gesture. We used mediapipe library as it has high accuracy for hand detection.

All the work done before for the hand detection was inside a ROS publisher node, where we get the camera frames process them and recognize the hand gestures. After that we created a publisher that publishes data to specific topic to help the subscriber node to know which duty cycle to give to the servo motor.

The Hardware for controlling the robot is a raspberry pi. The subscriber node was created on the raspberry whereas the publisher was on the laptop in order to get full performance of the hand detection using the GPU multithreading.

The communication between the two sides of the system was done through Wi-Fi, the data sent through cmd_vel topic.

Finally, a successful robot arm control is presented where hand gesture inputs give the robot movement output.

References:

[1]"Robotics", https://www.britannica.com/technology/robotics.[online][accessed on 27th June 2021]

[2]"Robots", https://www.mhi.org/fundamentals/robots,[online][accessed on 27th June 2021]

[3] "mediapipe", https://opensource.google/projects/mediapipe.[online][accessed on 27th June 2021]

[4] "mediapipe", https://medium.com/swlh/a-review-of-googles-new-mobile-friendly-ai-framework-mediapipe-25d62cd482a1.[online] [accessed on 27th June 2021]

[5] "ML pipeline", https://algorithmia.com/blog/ml-pipeline.[online] [accessed on 27th June 2021]

[6] "ROS', https://www.ros.org/about-ros/. [online][accessed on 27th June 2021]

[7] "opencv', https://www.geeksforgeeks.org/opencv-overview/. [online][accessed on 27th June 2021]

[8] "mediapipe"https://google.github.io/mediapipe/solutions/hands. [online][accessed on 27th June 2021]

[9]"ROS2",https://docs.ros.org/en/foxy/Tutorials/Understanding-ROS2-Nodes.html.[online] [accessed on 27th June 2021]

[10] "image processing", https://sisu.ut.ee/imageprocessing/book/1. [online][accessed on 27th June 2021]

[11] "lighting effect on fps "https://stackoverflow.com/questions/12099845/low-lighting-leads-to-low-fps-when-recording-video. [online][accessed on 27th June 2021]

[12] "fps "https//stackoverflow.com/questions/12099845/low-lighting-leads-to-low-fps-when-recording-video. [online][accessed on 27th June 2021]

[13] "ros1 vs ros2 ", https://roboticsbackend.com/ros1-vs-ros2-practical-overview/.[online][accessed on 27th June 2021]

[14] Aleksandar Zivkovic, Development of Autonomous Driving using Robot Operating System, Master thesis, Madrid, May 2018.

[15]"motion control"https://www.controldesign.com/articles/2018/what-is-motion-control/.[online][accessed on 27th June 2021]

[16] "machine learningpipeline"https://valohai.com/machine-learning-pipeline/.[online][accessed on 27th June 2021]

[17] “raspberry gpio ”, https://www.raspberrypi.org/documentation/usage/gpio/.[online][accessed on 27th June 2021]

[18] “robot model ”https://www.thingiverse.com/thing:1015238. [online][accessed on 27th June 2021]

I. APPENDIX A

Introduction to ros2

The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools.

Since ROS was started in 2007, a lot has changed in the robotics and ROS community. The goal of the ROS 2 project is to adapt to these changes, leveraging what is great about ROS 1 and improving what isn't.

The ROS2 was developed from scratch rather than Improving the previous version because the modifications can cause instability in ROS1. The goal of ROS2 is to make compatible with industrial applications by adding some requirements like real time, safety, certification and security.

ROS Noetic's EOL (End of Life) is scheduled for 2025. After that, no more ROS1! So, if you have a big code base in ROS1 today, it's totally OK, but don't wait until 2024 to start making changes.For ROS2, from the LTS (Long Term Support) version Foxy Fitzroy (release date: 2020), a new ROS2 version is released every year. Same as what was previously done for ROS1.

ROS2 vs ROS1

Writing nodes

In ROS1, for Cpp you use roscpp, and for Python, rospy. Both libraries are completely independent and built from scratch. It means that the API is not necessarily the same between roscpp and rospy, and some features are developed for one, and not the other.

ROS2 has more layers. There is only one base library, named rcl, and implemented in C. This is the foundation which contains all of the ROS2 core features.

You won't use the rcl library directly in your programs. You'll use another client library built on top of rcl. For example: rclcpp for Cpp, rclpy for Python.

What's great about it? Well any new functionality only needs to be implemented with rcl. Then, the client libraries on top of rcl just need to provide the binding.

Python and Cpp versions

Python2 is not supported anymore. It is still supported only for Ubuntu 18 and ROS1 Melodic until their EOL (2023).
ROS1 Noetic targets Python3, as well as all ROS2 versions.
Now, for Cpp, there is some great progress. ROS1 was targeting Cpp 98, and you could use Cpp 11/14 in later ROS1 versions, provided that it didn't break other dependencies.
In ROS2 you can now use Cpp 11 and 14 by default. Cpp 17 is also on the roadmap. That's great because new versions of Cpp introduce many useful functionalities, making development easier,

quicker, and safer. Also, it makes Cpp more fun, and maybe this will help democratize this powerful and great language (well it seems I'm biased).

OOP for node writing

In ROS1 there is no specific structure for writing the node. The callbacks functions can be called anywhere in the program. In ROS2 there is a convention about writing nodes by creating a class which inherits from Node object (for example: rclcpp::Node in Cpp, rclpy.node.Node in Python). In this class all ROS2 functionalities are present.

Multiple nodes in the same executable

Using components many nodes can be handled from the same executable. A component is a slightly modified node class. The components can be started from a launch file, terminal or executables.

Writing a launch file

In ROS2 launch files are written in python rather than xml in ROS1. Launch files make you able to launch different nodes from one file. There is an API that allows to start nodes, retrieve configuration file and add parameters.

Communication

In ROS1, we have always to start ROS master before starting nodes, it acts as a DNS server for other nodes. ROS2 to is a decentralized system which allow nodes to discover each other. Each node is independent of the master which gives a fully distributed system.

Network configuration for ROS2 multi-machines

For the multiple machine communication the same distribution must be installed on both machines. The machines should be connected to the same network to allow UDP multicasting and the discovery of packets.

If the two devices can ping each other successfully then we can start some nodes in Machine 1, some other nodes in Machine 2, and they will all be able to communicate through topics, services and actions. Just like they were all in the same machine.

To run two different ROS applications on the same network you need to export a new environment variable, named ROS_DOMAIN_ID, using a number for the value (preferably a low number, between 1 and 232). Then, only the nodes started in sessions with the same ROS_DOMAIN_ID will be able to communicate with each other.

Machine 1 – session (terminal) A:

$ export ROS_DOMAIN_ID=5

$ source /opt/ros/your_ros2_distribution/setup.bash

$ ros2 run package_name executable

Machine 1 – session (terminal) B:

$ export ROS_DOMAIN_ID=5

$ source /opt/ros/your_ros2_distribution/setup.bash

$ ros2 run package_name executable

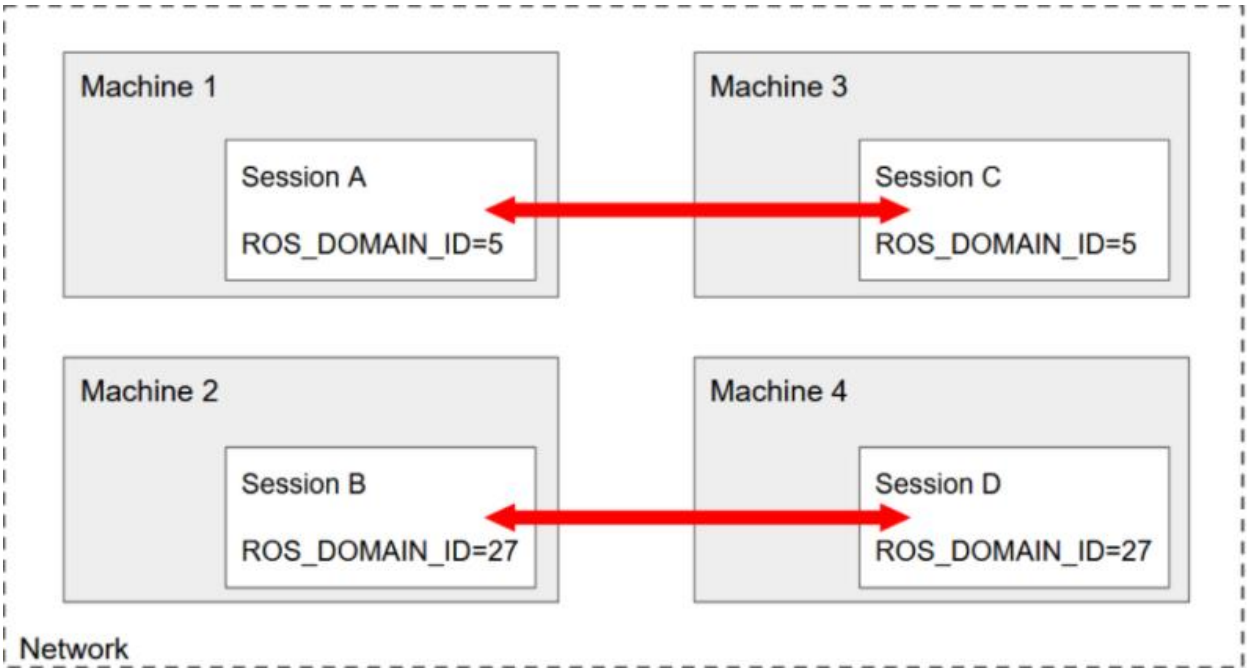


Figure I.1: Ros multimachine support

Here you have 4 different machines, each starting one session.

Ros2 installation

Ros2 is installed on top of Ubuntu Linux - Focal Fossa (20.04).

Set locale

Make sure you have a locale which supports utf-8. If you are in a minimal environment (such as a docker container), the locale may be something minimal like posix. We test with the following settings. However, it should be fine if you're using a different UTF-8 supported locale.

Locale #check for utf-8

sudo apt update && sudo apt install locales

sudo locale-gen en_US en_US.UTF-8

sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8

locale # verify settings

Setup sources

Ros2 repositories addition to system, first by authorizing the GPG key

sudo apt update && sudo apt install curl gnupg2 lsb-release
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg

then adding the repository to the source list

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null

install ros2 packages

Update your apt repository caches after setting up the repositories.

Sudo apt update

ROS-Base Install (Bare Bones): Communication libraries, message packages, command line tools. No GUI tools

sudo apt install ros-foxy-ros-base

Environment setup

Sourcing the setup script

Setup the environment by sourcing the file

source /opt/ros/foxy/setup.bash

An End-to-End Middleware (DDS)

DDS provides a publish-subscribe transport which is very similar to ROS's publish-subscribe transport. DDS uses the "Interface Description Language (IDL)" as defined by the Object management group for message definition and serialization. DDS has a request-response style transport, which would be like ROS's service system, in beta 2 as of June 2016 (called DDS RPC).

The default discovery system provided by DDS, which is required to use DDSs publish-subscribe transport, is a distributed discovery system. This allows any two DDS programs to communicate without the need for a tool like the ROS master. This makes the system more fault tolerant and flexible. It is not required to use the dynamic discovery mechanism, however, as multiple DDS vendors provide options for static discovery.

When exploring options for the next generation communication system of ROS, the initial options were to either improve the ROS 1 transport or build a new middleware using component libraries such as zero MO Protocol Buffers, and zeroconf (Bonjour/Avahi). However, in addition to those options, both of which involved us building a middleware from parts or scratch, other end-to-end middlewares were considered. During our research, one middleware that stood out was DDS.

ROS file system

Similar to an operating system, ROS files are also organized on the hard disk in a particular fashion. In this level, we can see how these files are organized on the disk. The following graph shows how ROS files and folder are organized on the disk:

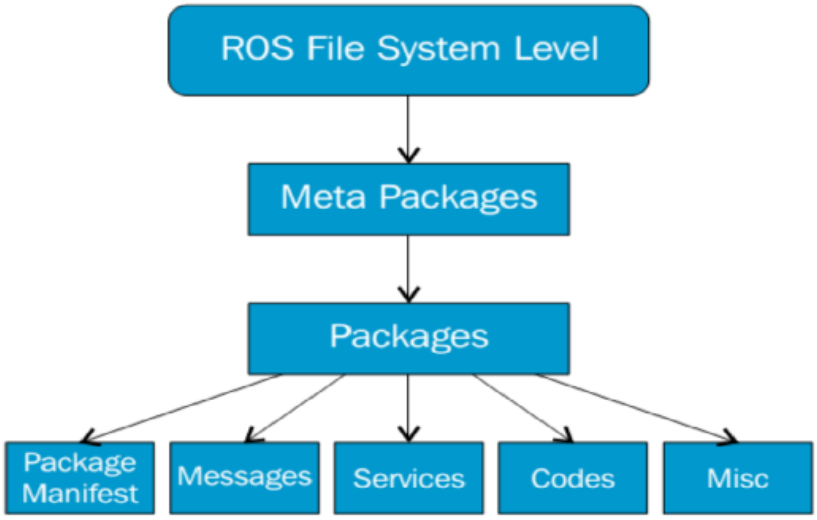


Figure I.2: ROS file system level.

A typical structure of a ROS package is shown here:

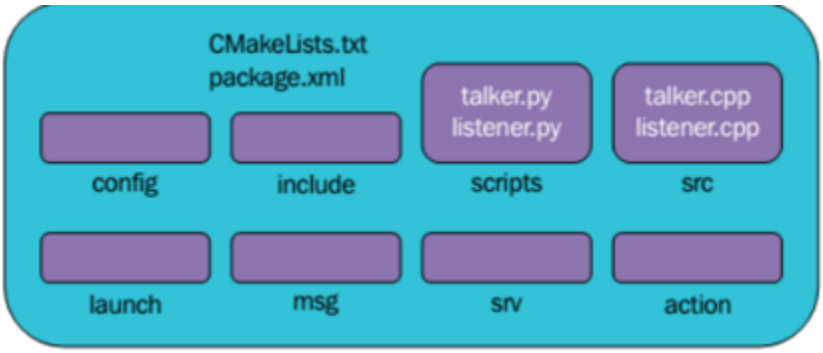


Figure I.3: Structure of a typical ROS package.

I. APPENDIX B

Raspberry pi setup

The operating system installed on the raspberry pi is ubuntu server 32bit. It is the version compatible with ROS2.

Network configuration

To connect the raspberry to the wifi network we have to change the SSID of the network config file in the SD card, then change them in another file called

```
# This file is generated from information provided by the datasource.  Changes
# to it will not persist across an instance reboot.  To disable cloud-init's
# network configuration capabilities, write a file
# /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg with the following:
# network: {config: disabled}
network:
    ethernets:
        eth0:
            dhcp4: true
            optional: true
    version: 2
    wifis:
        wlan0:
            optional: true
            access-points:
                "MyWiFi":
                    password: "MyPass"
            dhcp4: true
```

Then change them in /etc/netplan/50-cloud-init.yaml file

Raspberry PWM

Fully hardware PWM

This type of PWM is generated by the Pi's PWM peripheral.

The timing of the pulses is controlled by the PWM peripheral.

It is the most accurate and arguably the most flexible.

It can be generated on GPIO 12/13/18/19. However there are only two channels, so only two different PWM streams can be generated at a time. GPIO 12/18 are on one channel, GPIO 13/19 on the other.

Suitable for jitter free servos, glitch free LED brightness control, motor speed control.

DMA timed PWM

This type of PWM is generated by the Pi's DMA peripheral.

The timing of the pulses is controlled by DMA. It is not as timing accurate as fully hardware PWM but appreciably more accurate than software timed PWM. Depending on the implementation it is not as flexible as fully hardware PWM, e.g. the number of frequencies is much more limited and the number of steps between on and off is much more limited.

This type of PWM may be generated on any GPIO on the expansion header. All GPIO may have different settings.

Suitable for jitter free servos, glitch free LED brightness control, motor speed control.

Software Timed PWM

This type of PWM is generated by software.

The timing of the pulses is controlled by the (Linux) scheduler. It is appreciably less timing accurate than fully hardware PWM or DMA timed PWM. It is much more flexible than DMA timed PWM and just as flexible as fully hardware PWM, e.g. the number of frequencies is unlimited and the number of steps between on and off is unlimited.

This type of PWM may be generated on any GPIO on the expansion header. All GPIO may have different settings. The timing accuracy will vary according to the number of GPIO being used for PWM.

Not really suitable for servos, will control LED brightness but will suffer from glitches, suitable for motor speed control.