ROBOKINESIS - CONTROLLING ROBOTIC ARM WITH COMPUTER VISION

by

MICHAL SITARZ
URN: 6594200

A dissertation submitted in partial fulfilment of the
requirements for the award of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

May 2023

Department of Computer Science
University of Surrey
Guildford GU2 7XH

Supervised by: Dr. Zhenhua Feng, Dr. Oscar Mendez Maldonado

I declare that this dissertation is my own work and that the work of others is acknowledged and indicated by explicit references.

Michal Sitarz
May 2023

Abstract

The aim of the project is to build a fully integrated Robotic Operating System library which will grant users the ability to train, experiment with, and control a robotic arm by mimicking the human arm motions. The project uses Computer vision to capture the arm movement, and maps the motions to the robotic arm. The mapped spacial points are used to gather corresponding joint angles with the use of Inverse Kinematics. The keypoints and the joints are used to train an optimally selected Artificial Neural Network, which learns to predict the joint angles for smooth control of the arm. Currently, there seems to be a lack of openly available libraries for Machine Learning control, despite it being a large research area, and hence the project's contribution is that it provides a strong framework for it. A strong literature review was conducted to find research that could be implemented into the framework for the best solution. The trained Machine Learning model was attempted to be used to substitute Inverse Kinematics in simulation, but the outcome of it felt slightly short as the predictions being made did not cover the full range of motion. The accuracy of the predicted model however looks good, and efficiency of the implemented method is thoroughly evaluated, to determine its usability and future directions.

Repository link:

`https://gitlab.surrey.ac.uk/ms02264/manipulator-control-with-computer-vision`

Acknowledgements

I would like to thank everybody who has supported me on this journey. A lot of different people have helped me and motivated me to complete this project.

I would especially like to thank Dr. Mendez Maldonado, who took time from his busy schedule to offer me help on the development of this project and who introduced me into this topic. Without him I would not have been able to do this project, and learn as much as I did along the way. Without him, I also would not have had access to the resources available at CVSSP which would have made the creation of this project much more difficult.

Similarly, I would like to thank Dr. Feng for giving me a lot of insights into writing a good report and for taking time to read my draft and give me insightful feedback and pointers.

Contents

List of Figures

List of Tables

Abbreviations

CV	Computer Vision
DL	Deep Learning
ML	Machine Learning
ANN	Artificial Neural Network
FK	Forward Kinematics
IK	Inverse Kinematics
ROS	Robotic Operating System
SOTA	State of the art
GUI	Graphical User Interface
DOF	Degrees of Freedom
NAS	Neural Architecture Search

Chapter 1

Introduction

1.1 Problem Background

The problem of moving a robotic arm (a.k.a. robotic manipulator) has been present for decades. Ever since Unimate, the first ever industrial robot, the research into robotic arm movement begun [11]. The early robots had been largely pre-programmed for a specific set of tasks and could only be working on very simple, and repetitive tasks. Moreover, the control of those arms is often quite difficult. For instance, to control an OpenManipulator-X with a keyboard, 8 keys are required, and 2 for the gripper. This scales up with more DOF, with more available joints.

Nowadays, with large strides taken in DL and CV, there is a growing focus on controlling manipulators with the help of those techniques. As it will be shown in Chapter 2, the research into these topics is constant, and new developments are emerging all the time. However, it might be relatively difficult to join the research due to a lack of supporting, openly available, libraries for this task, or even simply using such system without creating it from scratch.

Therefore, this project will aim to engineer a training and a control library, which will in turn be used create a ML model that can accurately control a robotic arm (thus creating a full pipeline for controlling, training and testing). The motivation behind using ML (over more prominent technique such as IK) is explained in Chapter 2. In short, the proposed framework provides an opportunity for collecting data, and in turn, allows for a supervised training experimentation, which can lead to some interesting research.

This area of research is crucial for the improvement of different industries within our society,

and is a step forward, towards automating various tasks - ranging from robots picking up and placing objects to robots performing surgeries. In most of the tasks where human replacement or improvement is sought, the tasks require a lot of dexterity, and thus, the robotic arm has to be extremely accurate and precise.

1.2 Aims and Objectives

The aim of the project is to develop a pipeline, which can be used to collect data, train, and control a manipulator arm. This project is not aiming to be a specific end-product, but rather a tool which can boost future research, and which provides a good ML model for smooth and accurate control of a robotic arm. Therefore, accessibility is largely important, in order to help democratize robotic research and make it openly accessible. ROS is a great place for that as it facilitates package sharing between developers and robotic research (the choice will be further elaborated in Chapter 2).

1.2.1 Aims

- Develop a user-friendly and low-cost data gathering system to be utilized for training of ML algorithms
- Construct a robust AI-based control pipeline for a robotic arm

1.2.2 Objectives

- Create a system which will gather the human arm movement from a camera and save them to a file. The data can be used for training. Complete within two weeks after setting up the libraries.
- Implement a method that will map the gathered arm movements onto the robotic arm in simulation. The changes in human arm movement should be reflected accurately on the robotic arm. The output of this objective can be used to train a supervised ML algorithm. Should be completed before the end of Semester 1.
- Train and visualize a supervised learning model which will predict unseen movements. The movements should be smooth and accurate. This should be completed by the third week

of Semester 2.

- Revise the implementation based on performed usability testing of the software, and the accuracy of arm movement. Evaluate the system based on evaluation metrics which were discovered in Section 2. Research into possible methodologies to improve on the design. Should be done within 3 weeks of finishing the training.
- Apply the trained model to the hardware by the end of the Easter break.

1.3 Project Scope and Success

The project scope has been indirectly addressed in Section 1.2, but this section will elaborated it in more detail, whilst addressing the scope of the project. The main objectives are to engineer a system for controlling OpenManipulator-X. This can be broken down into two main success criteria. The first one being a good library for training and controlling of the robotic arm, and the other one is having an accurate model developed for control.

The control framework should be developed in a way that will be user friendly and easy to use, as well as flexible to changes being made (in case experiments would be performed on it). Visualizing everything to the user is within the scope, as it will make the library more usable and the control more understandable. Using it to develop the model for control will in itself be usability testing.

Furthermore, in order to deem the project as successful, the arm should move smoothly and accurately. This is important as without having those characteristics, the manipulator will not be applicable anywhere, as most places require the control to be flawless. For this purpose of this project, the manipulator should be able to mimic human arm movements. The evaluation strategies for a successful system and control are discussed in depth in Chapter 3.

In general, meeting the aims from Section 1.2 will result in a successful project, but there are also some limitations associated with the objectives.

1.3.1 Limitations

With having a good library developed, there are a lot of possibilities for research to be done. However, it will be difficult to go into any specific research topic within the time frame available.

Therefore, this project will use tools and research available to engineer a good and accurate solution. During the early stages of development, time will also have to be taken to learn how robotic systems works and how to implement them effectively. Furthermore, the aim is to manipulate it in real-time, but beyond estimating the position of the arm, another limitation might be motion planning. With enough time, motion planning can be explored, but it will not be the main focus of this project, as this is a whole other research topic, and including it in this project would spread the scope to wide. Therefore, the project will have to rely on available libraries.

1.4 Report Structure

- **Chapter 1 (Introduction)** - The section explores the background and motivation for the project, the objectives and the project scope.
- **Chapter 2 (Literature Review)** - Detailed investigation of different techniques that are used for robotic manipulation and different useful additional resources.
- **Chapter 3 (Problem Analysis & System Design)** - Will start with the plan for the system, will explore the software and hardware being used, the feasibility of the project, functional and non-functional requirements to mark the success, and the development approach to be used. To elaborate on the system plan, different diagrams will be proposed to showcase the proposed solution.
- **Chapter 4 (System Implementation)** - In-depth discussion of the development process at each stage, along with some most important concepts explained, and diagrams providing insights into the flow of the system.
- **Chapter 5 (Testing & Evaluation)** - The evaluation of the system based on the functional and non-functional requirements, and the evaluation of the robotic arm movement based on the results obtained.
- **Chapter 6 (Statement of Ethics)** - Reflection on different types of ethical, social, legal and professional issues addressed in the project development.
- **Chapter 7 (Conclusion)** - Closing statement which summarizes the success of the project, contributions, and possible future improvements or additions which can be made.

Chapter 2

Literature Review

This section will diverge into two different major analysis. Firstly, the review of different concepts and research done on manipulator control. Secondly, a review of existing frameworks and libraries will be carried to identify useful resources and tools which can be used for the creation of the software.

2.1 Manipulator Control

Over the years, there have been various implementations of robotic arm manipulation techniques. An analysis will be carried out to explore the different techniques that are currently being used, and how they are applied to different real-world applications. The main focus is to dive into the functionality of those implementations and explore their usability, and the algorithms that are highly adaptable to diverse tasks, accurate, fast and cheap to implement.

2.1.1 FK and IK

There are two major concepts that robotic manipulation works with; FK and IK. According to [12], the former is a mathematical technique which uses the robot's joint angles, to determine the position and the orientation of the end-effector in the 3D Cartesian coordinate system. Inversely, the latter is a mathematical technique that aims to find the joint angles of the robot based on the location and orientation of the end-effector. Both are valid methods for moving the arm, and getting the exact position or the joint angles. The conversion between the two is summarized in Figure 2.3. Even though the calculation is non-trivial, as there are an infinite number of

possible join solutions, computers can do it very fast, but still might lag behind in real-time applications. This is especially the case with robotic arms that have many DOF (more movable joints), as a unique solution might be more difficult to be found, as algebraic approaches might not guarantee a closed form solution [13]. Furthermore, the method might not find some joints if the given task requires some complex, and dexterous, motions.

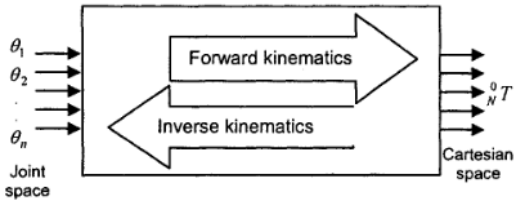


Figure 2.1: IK and FK Schematic [1]

2.1.2 IK with ANNs

ANNs are often used to find complex underlying patters and relationships in the dataset, with the help of biologically inspired sets of neurons, which are used to transform the data mathematically. There is a large research focus into how neural networks can be used in order to improve upon the existing IK solvers. A common identified issues of IK (especially by recent research conducted) have been that as the DoF increase, the non-linearity of kinematics equations spikes, making the tasks extremely difficult, and the speed used for computation is not optimal [14, 15]. For that reason, different supervised learning approaches have been used to learn the mappings between the joint angles and the position of the end-effector [14, 15, 16, 17, 2]. The method proposed by [17] has used joint angles to predict the end-effector position for the complete trajectory, by evaluating a vector-based mean absolute error. The conclusions drawn from the results are that the ANN predicts the joint angles quicker than other methods, and converges better. Furthermore, the most accurate predictions in terms of positional error have been by using Long Short Term Memory (LSTM) network. The main issue with this research is that despite the higher speed, the networks that have been trained do not perform as well. A possible improvement would be to explore more types of network architectures, or doing a more in-depth hyperparameter optimization. Despite those shortcomings, the approach is still helpful to understand if a framework was to be built for improving the issues associated with IK. Similarly, [16] explored learning the direct IK on position level. The main difference of this research was

that instead of finding direct approximation of the IK function, the joint probability distribution was found for the end-effector and the joints. The results obtained showed a better performance when doing task-space tracking. One important improvement of this research would be the speed, to have a solution similar to [14], which could be achieved by performing more efficient gradient search to make the predictions faster.

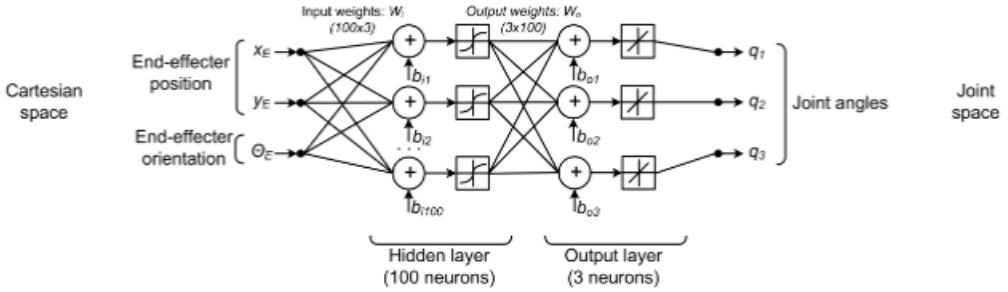


Figure 2.2: Example of joint prediction Neural Network [2]

A common issue with the above-mentioned methods is that they usually either focus on positional or joint-error. Both are important as small changes in joints closer to the base can result in big changes in the final position of the manipulator. Similarly, aiming for smaller positional error might not scale very well with more joints, and furthermore, might not teach the model to generalize well. A possible exploration area would be to find the most optimal architecture with a Multi-Objective NAS optimization [18], and finding the Pareto Front. The search space for a more simple network like this should not be is large as that of Deep Neural networks, and if the mapping is used as proposed in the paper, then the inputs are not going to be large and demanding to train, so a lot of iterations can be run. Furthermore, there are different loss functions that have been used, such as Mean Squared Error and Mean Absolute Error. Huber loss [19] is possible alternative to find the error, as it provides robustness when training - being influenced less by outliers rather than by inliers - and it combines the best properties of the absolute and quadratic loss.

2.1.3 Direct Movement

Before there were massive strides taken in the field of ML, in this case CV and DL, different methods had to be utilized in order to control robotic arms. The robotic arm can be controlled by directly mapping the movement of the human arm or the signals to the robotic arm through

a hardware interface, such as accelerometers [20, 21] or controllers [22]. These methods can utilize electromagnetic sensing to determine how much force was produced and the orientation in space.

In [20], the authors have developer a cheap and lightweight way to control the arm. The users wear 2 acceloremeters on a glove which capture the velocity and position of the arm. Furthermore, the glove relays the information about whether the hand is closed or open, which allows for control of the gripper. With the inputs from the glove, the algorithm determines where the robotic arm should be placed. The perks of using an external hardware device is that the orientation of the wrist as well as the speeds measured have high accuracy (as it was shown in [21]). Furthermore, in comparison to more modern techniques that use cameras, they are not susceptible to lighting conditions or the camera quality. However, using an external hardware device, which in this case is a fairly large glove with wires connected to the arm, is not a very portable or usable outside of the lab. This makes it not easily adaptable. If the connection was wireless, it would simplify it slightly. Using hardware devices also might require custom design or calibration for different robotic arms. In this case, a very basic arm is used, but it might fail to perform well with manipulators that have a higher degree of freedom. Finally, even though the hardware can be obtained for a relative cheap price, it is still an extra cost that has to be considered. It might not be the most viable solution for commercial use.

A useful technique, prominent in these approaches (which is harder to replicate with CV) is velocity-based control of the manipulators [23]. A recent research explored imitation-based mapping which was used for teleoperation of a robotic arm [24]. The velocity-based motion mapping to control the manipulator movements results in highly accurate and smooth mimicking of the movements. In comparison to calculating the joint angles of the target position, Jacobian IK (explained in 2.1.1) techniques is used to move the arm in real-time.

[21] and [22] use similar techniques, with different algorithms under the hood (such as [21] using ANN to predict the positions). Even so, they all suffer from similar benefits and flaws as it was discussed in regards to [20]. Therefore, the following section will focus on DL methods, which try to overcome some of the flaws and improve on the techniques.

2.1.4 Control with ML

With regards to controlling the manipulators with ML approaches, there have been many different approaches that utilize different algorithms. One of them is, visual servoing [25], which involves using CV techniques to control the robotic arm based on visual feedback (such as tracking the position of an object in the workspace or detecting key points on the object). This topic is helpful for this project, as the location of the manipulator has to be predicted and the arm has to be moved to that location. In 2015, Iscimen et al. [26] used identifying objects with CV to move the robotic arm to the desired location. The angles of the joints were found by implementing a coordinate dictionary for the desired location of the robotic arm. This study only works with tableware and the implementation is not very robust. Similarly, in 2020, Intisar et al. [27] designed a user-based application, where they can filter different objects based on their size, shape, and color. With that, they can identify the desired object's location and move the arm to that location with the use of inverse kinematics. The strength of this project is the minimal human interaction, in this case, selection in the GUI. The downside is that the arm goes through a set of preset positions to grab the object. This makes the method less robust and adaptable to other tasks and the method does not allow for full control, just moving to the desired location. A different approach, as presented by Zuo et al. [28], removes the need for using sensors to move the manipulator. The aim of the research was for the robot to use CV to predict the status of the joints with an external camera. This gives the arm more flexibility and they propose the possibility of the system following the behavior of the human operators with greater ease. Furthermore, they propose a solution that is a semi-supervised approach that doesn't require a lot of annotation as it generates the data synthetically.

The above-mentioned techniques used ML and CV to detect where the arm has to move. Some general issues include non-generalizable systems, feature errors converging quickly and trying to move out of the joint limits [29]. There have been recent advancements in this domain with the use of DL, but that research is outside of the scope of this project.

On the contrary to visual servoying, manipulators can be moved by following a demonstrator's hand, allowing for an adaptable and robust system. There is a big challenge in mimicking human movements, and that is mainly due to different kinematic and dynamic properties between the two. Likewise, the stability of the manipulator's posture has to be intact whilst reproducing the motion of the arm as accurately as possible [30]. For instance, Jan Graßhoff et al. [31] use

tracking of the arm and gesture recognition to move a 7 Degree of Freedom arm. They managed to achieve good results with multiple cameras, which allow them to flexibly get high point densities. More advanced techniques have been done by teaching the robot how to follow the arm rather than using CV to follow the estimated keypoints, such as it was presented by Sivakumar et al. [3]. In their system human user control the robotic arm and hand, by demonstrating the movements with their arm. The robot attempts to mimic human movements in real time. The system is trained to understand human motion by training a reinforcement learning algorithm on a large dataset collected from the internet. The system allows for using the arm for highly dexterous tasks with high accuracy. The technique to predict the arm position in this task was by estimating human hand poses and then a network was used to map the movement to the manipulator. The main drawback of this study is the required thousands of hours of raw footage collected from the internet to train the model. Furthermore, there is no ground-truth labels for this training, and thus, the manipulator could potentially learn incorrectly.

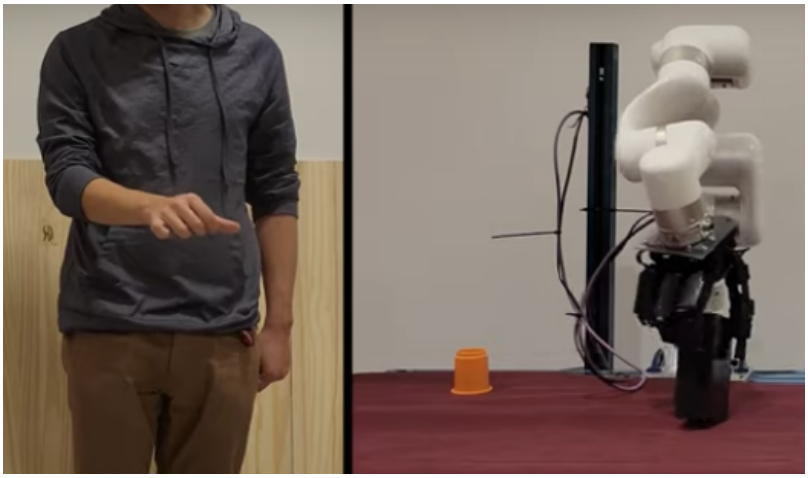


Figure 2.3: Control Example [3]

In general, having a manipulator that understands how to move correctly and with high dexterity like a human demonstrator, can lead to further research in the future which creates a hybrid of diverse techniques. For instance, applying it to different approaches, such as controlling robots through language [32] or through brain signals [33]. Similarly, other applications of such system were presented by [34], where they gathered some data to train an imitation learning algorithm to perform tasks such as picking up objects. This type of training allows for using only a small amount of data from a human demonstrator and then learn how to apply it to diverse tasks. Similarly, these approaches don't have to be solely applied to robotic arms. So far, in movies

and video games, motion capture control was used. However, developing a good system for mimicking human movements can also be applicable to CV character animations.

2.1.5 Evaluation

An important aspect of this project is the performance of the robotic arm manipulation. There have been several papers that discuss the performance measures which can be used. Russo & Mateo [35], delve into how to define the performance of the robot's behaviors and ways in which the behaviors can be quantified. First and foremost, the *accuracy* of a robotic arm is the measure of how capable it is to reach the desired pose, and how close it gets. There are various implementations of accuracy that can be used, but two most common ones are joint and positional error, which tell how far each joint is from the desired one, and how far the overall position is, respectively. Furthermore, most of the research depicted in Section 2.1.2 uses similar evaluation strategies for accuracy. The main one is using a helical function to map a trajectory to predicted values. Similarly, the orientation of the end-effector joint (and all the other joints) is compared by taking sampling the angles from the full range of motion.

Secondly, *resolution* is a metric that shows what the tiniest incremental motions can be done and measure by the manipulator, which defines how precise the manipulator can be. Thirdly, the workspace of the arm is the region that that manipulator can reach. Most of the research from Section 2.1.2 focuses on measuring the accuracy, but they fail to show the range of motion that the trained networks can reach. The Workspace index is a useful metric to determine the reachable percentage of the manipualtor's space [35]. Workspace closely coincides with the *dexterity* of the manipulation, and the Dexterity Index [36] and Dexterity workspace propsed by Patel et al. [37] - which define the set of points that can be achieved at extreme points and the directions it can reach different points at. However, dexterity could not be measured in this way, as it is measured by taking the number of IK solutions for a specific location, whereas an ANN would only produce one.

Finally, another downfall of the research from Section 2.1.2 is that despite the authors getting good results, most did not indicate how the speed compares to that of IK, which is one of the main drivers for this research. The main exception was the project by Vaishnavi et al. [17] which reported the computation time for each of their networks (in seconds), which seems to be the best way to report the speed and compare it to IK. There are many more metrics that the

papers identify, but the metrics which have been used are the ones that have been chosen to be most useful to identify the performance and the success of the mapped motions.

2.2 Libraries & Tools

2.2.1 Keypoints and Hand Estimation

Keypoint detection is a task which aims to localize people in an image and find their corresponding keypoints. A keypoint is a point of interest. For instance, joints such as shoulder, elbow or wrist. There has been a lot of research done on keypoint estimation, and there are various different libraries which implement models for easy integration. This topic is relevant as it is the best way to capture how different joints move. As the robotic arm has multiple joints as well (such as a gripper), it can help to map the location of both.

MediaPipe [4] is a framework which has a model used both for Body Pose and Hand estimation for a single person in the frame. The models are openly available and have been trained on BlazePose, whereas other models such as YoloV7 Pose are only trained on COCO (which is a subset of BlazePose) [38]. Analysis was done to compare the two frameworks [39].

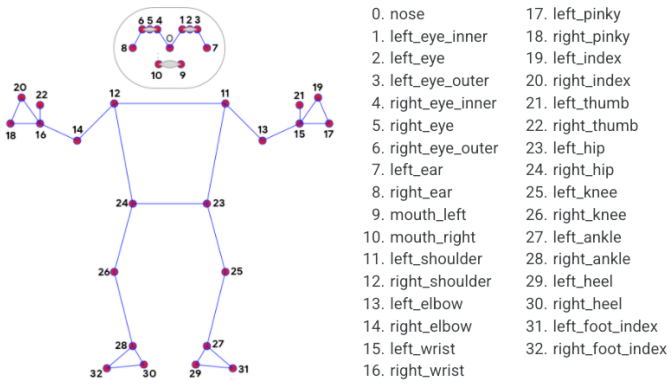


Figure 2.4: MediaPipe Keypoints [4]

It has been shown that MediaPipe can produce good results on lower resolution images, in comparison to YOLOv7. This can be beneficial for this project, as the cameras used to capture the movements might vary depending on the users - the system will be applicable to phone cameras, in-built laptop cameras etc. YOLOv7 was also shown to be slower when running inference on a CPU, in comparison to MediaPipe. In order to meet the project objectives of making a smooth motion, having a smaller bottleneck when predicting keypoints is important.

On the other hand, YOLOv7 has been shown to perform better when there are occlusions present. However, in the case of this system, occlusions should not be a problem, as a person recording the motions is expected to be standing right in front of the camera. Whilst YOLOv7 has the advantage of detecting multiple people in the image/video, once again that is not a trait that this system needs. In fact, focusing on searching only for one person can remove some unnecessary background noise in case more than one person are being detected, which could cause the model to get confused. The main upside of YOLOv7 is that it had been tested to estimate fast movements better, but the trade-off is that a high resolution images are required. Even though fast movements could be very useful for some applications that require fast movements (for instance catching an object falling). However, trading-off other benefits of MediaPipe, and requiring a higher resolution input, would make the system less applicable to different applications and audience. Furthermore, MediaPipe has been shown in different applications to yield good results for different tasks; for example for Yoga pose classification [40]. To conclude, MediaPipe looks like a better option for this project, and it should be easy to integrate with Python. If the it turns out that the MediaPipe causes problems with identifying the keypoints, YOLOv7 can be an alternative option.

2.2.2 ROS

After researching into various ways to make a robotic system, the outcome was that the most common and widely used one was with the help of ROS. ROS is an open source environment which can be used to develop and simulate robotic systems before they are deployed onto hardware [41]. Furthermore, ROS is the standard for research and for application creation. It yields a large number of benefits that can be used in this project. The main feature of ROS is in the way that it runs and how different packages communicate. A new, complex, systems can be designed by stitching, multiple existing solutions, and adding new ones. As the system works based on communication between different nodes, it is easy to do so. Therefore, using existing libraries can largely aid the development of this project, as otherwise, a whole pipeline for the robot would have be constructed from scratch, for instance for moving the manipulator (which would all take considerable amount of time and would make experimenting extremely difficult). For the same reason, making a ROS package that can be integrate with different ones, can be a great way to be incorporated in the further research, the same way this project will integrate already available packages.

The hardware is further discussed in Chapter 3, but in short, OpenManipulator-X is going to be used. Reviewing the documentation [7], there are a large number of relevant libraries that can be used with it:

- **open_manipulator_gazebo** - library used to simulate the manipulator in Gazebo (simulation software)
- **open_manipulator_controller** - used for controlling the manipulator (both physical and in simulation)
- **open_manipulator_control_gui** - GUI designed to control the manipulator with the use of either joints or positions

All of these packages will make experimenting and designing the proposed pipeline possible, without needing to re-create everything. The support for the robotic arm used is very high, and the hope is that this project will contribute by adding additional helpful library. Further researching into different openly available packages for this manipulator, there was not a single one that allowed control through recording of a human demonstrator. The only examples that were found, were in some of the videos attached to the research papers (for instance [3]), but not much of it is shown except for an occasional control with gestures which simplifies the control. On the contrary, there were some libraries which used CV in simulation to perform tasks such as picking and placing of different objects with manipulators [42], but not necessarily giving control of the arm to the user. However, using it will still be beneficial to understand how to construct a useful and functional library. It is important to understand how to design a ROS package, which will be fully-integrated and works correctly with the framework, as otherwise it won't be usable by others. The review of documentation and best practices, has shown that the package structure should be designed as shown in Figure 2.5.

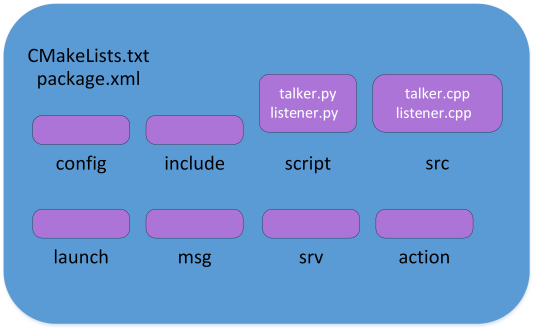


Figure 2.5: ROS Package Structure [5]

To summarize, following components are crucial to develop a ROS package. To build the software packages within the package, *CMakeLists.txt* is required, and *package.xml* is included to specify the dependencies that the package has. Out of the directories, most notably *launch* directory includes launch files that will run the *scripts* and *src* files within the package. They will also be running nodes from other packages (which were listed above).

2.2.3 Applications

To quickly summarize, there is a variety of applications that the robotic arms can be used, with different controlling techniques available, depending on which task it is being used for. Their applications range from medical purposes to helping in industry. Controlling the arm can have a very positive and large impact on people with disabilities, as it gives them the means to control an arm. For instance, non-invasive brain-computer interfaces have been used to control the manipulators [43]. Similarly, implementing picking up and placing of objects with a robotic arm, in a warehouse, has a large share of benefits too as explained by Sobhan & Shaikat [44]. Developing strong control systems allow for further improved automation in those industries and aid people in many ways.

2.3 Conclusion

To summarize, there is a lot of research available for robotic manipulation. The degree to which the manipulation can be taken varies. For the purpose of this project, the direct conversion from positions to joints seems the most suitable, but if the performance is not as expected it can be taken further by applying deep and reinforcement learning techniques. There is also a lack of readily available libraries that can be used for this type of research. The basic tools to build one however, are openly available to be used, such as OpenManipulator libraries and MediaPipe for keypoints. This research covered all the foundations needed to implement and engineer this solution, with potential to use it for research purposes.

Chapter 3

Problem Analysis & System Design

This chapter will explain and plan for developing a successful system based on the objectives outlined in Section 1.2, which was derived from the research in Chapter 2.

3.1 Problem Summary & Project Outline

The aim of this project is to engineer a system which will control a robotic arm accurately, based on the camera input of a human demonstrator. The following insights are conclusions based on Chapter 2 and were discussed more in-depth there. The main idea of the project is to move the arm without the need of any controllers (such as accelerometers), i.e. being done solely based on the visual information provided to the system. For that purpose, MediaPipe keypoint estimation was determined as the best solution for this. The framework is aimed to be easy to use, transferable to anybody using it and easy to use for future research (thus including visualizations and adaptable to different approaches). Furthermore, the research has found different techniques being used for the movement of the manipulator. The most prominent solution is usually with the use of IK. The project could use a package that solves the IK automatically based on the position of the manipulator. However, there have been numerous different ML techniques identified which can increase the speed of the system (which is a crucial part, further discussed in 3.3), smoother transition between the points in space, and convergence, as oftentimes the IK might fail to find a solution in a given time, or at all. Designing a good Machine Learning model would especially make systems with more DoF work better.Therefore, IK can serve as a baseline and a starting point for training a ML model, and the accuracy of the

model will be aimed at (more explanation on the training plan in Section 3.4.2). There are more advanced techniques being used in current research, namely reinforcement and unsupervised learning, which are options to be considered in the future if the accuracy of the model is not performing well. However, the lack of openly available systems to conduct this type of research, brought an incentive to create a good pipeline which will include the control and training. According to the findings in the research, ROS looks like the best choice to develop the software needed. The overview of the proposed framework can be seen in Figure 3.1.

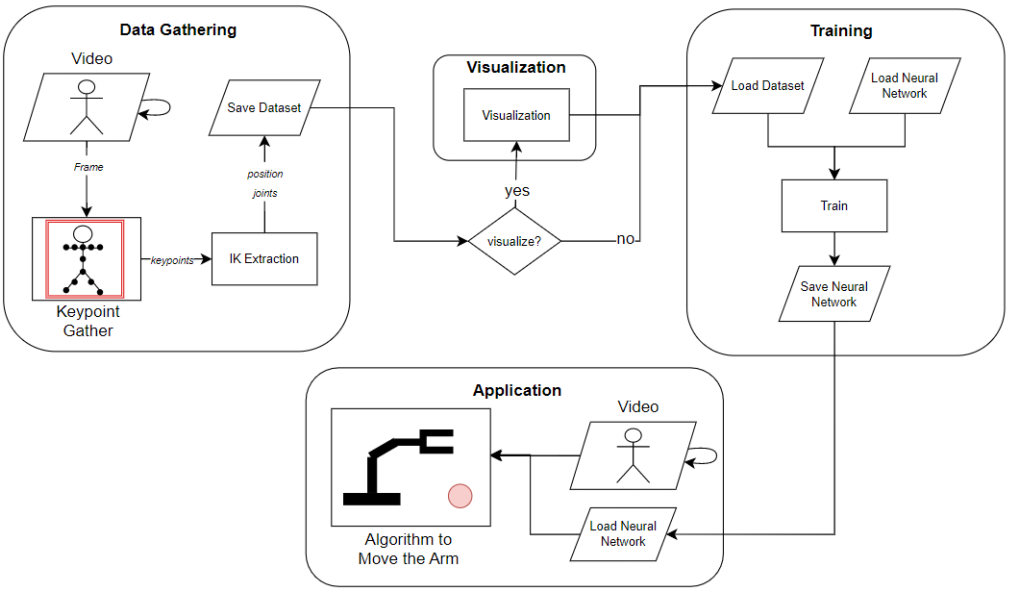


Figure 3.1: System Overview

3.2 Software and Hardware Specifications

This project predominantly revolves around the software side, but there are also some hardware requirements.

3.2.1 Software Specifications

This section will focus on describing what software requirements are there for creation and execution of this project. After combining the knowledge from the conducted research in Chapter 2, with the project plan in previous section (see Section 3.1), software specifications can be

defined.

The following is the outline of main software dependencies:

- Operating System - Ubuntu 20.04 (Focal)
- Environment - ROS Noetic
- Languages - Python 3.8 and C++
- Keypoint Estimation - MediaPipe
- Main Dependencies - OpenCV, PyTorch

ROS and Ubuntu ROS will be used to design the system (for the reasons outlined in Chapter 2). To summarize, the main reasons for using ROS are the learge number of available packages for OpenManipulator-X, which will save a lot of time, and secondly, it is a standard framework used across the world. The manipulator has libraries available to move the manipulator, simulate it with Gazebo, use RVIZ for visualization, and compute FK and IK. The version used will be ROS Noetic because of the Operating System available on the development laptop, and the version provides a lot of support for the libraries mentioned above.

When using ROS, two main languages are Python and C++. Either one can be used to communicate with the other subsystems (nodes) via publishing and subscribing to messages. All the subsystem are aware of each other through a master node (core) as visible in Figure 3.2.

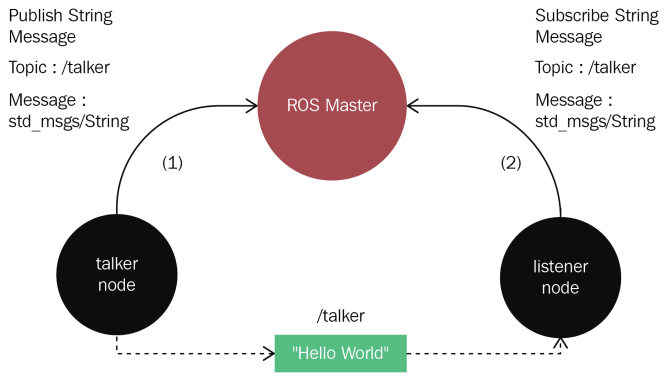


Figure 3.2: ROS Framework overview [6]

Python and C++ As mentioned in the above section, Python and C++ are the two main languages used in ROS. Out of personal preferences, the majority of work will be done in Python. This decision is further reinforced by availability of all the necessary libraries, such as PyTorch and MediaPipe, which will be used to develop and train the ML model. The reason for using PyTorch comes to the ability to dynamically compute graphs, and therefore allows for a lot of flexibility when designing models. When it comes to communicating with the rest of ROS packages, messages can be sent from any one language, so a Python script can easily communicate with a C++ script by creating specific messages (see Section 3.2), therefore choosing Python as default does not mean further down the line C++ won't be used. Another option would be to use Python bindings, but due to ease of communication between the nodes, writing a new C++ script would be easier if required.

Arm Keypoints As the research has shown (see Chapter 2) there are different frameworks to capture human pose and keypoints. The option that was selected was MediaPipe, as it supposedly works better than different alternatives, such as YOLOv7. MediaPipe is compatible with OpenCV, which will be used to capture the input from the camera. MediaPipe offers plenty of models specialized for different tasks openly available, but the two that are crucial for this system are: Human Pose Detection and Tracking, and Hand Tracking. The former is especially important, as it detects the person in the frame and predicts main body keypoints. The keypoints contain the three major joints on the arms, so with this framework it will be easy to obtain the wrist, elbow and shoulder keypoints. The latter is also useful, but more of an addition (will be further discussed in Section 3.3) as from the combination of hand and arm keypoints, the wrist angle can be inferred, as well as whether the hand is closed or open (which directly translates to whether the gripper is open or shut).

3.2.2 Hardware Specifications

This section will give a quick overview of some hardware requirements which are needed for creating and executing of the project.

The following is the outline major hardware used:

- Unix Laptop or Virtual Machine
- Camera

- OpenManipulator-X
- Dynamixel Starter Kit (for OpenManipulator-X)

Host Machine In order to use ROS, an Ubuntu system has to be used (see Section 3.2.1). This can be either in a Virtual Machine environment, or a Linux machine. For simulating the OpenManipulator-X in Gazebo, there are some minimum requirements [45]. When running on a Linux Machine, those are:

- Processor - Intel i5 (4 cores)
- Graphics Card (GPU) - Dedicated GPU with minimum of 1GB memory
- Memory - 4GB
- Disk Space - 500MB

These are the absolute minimum requirements that the user's machine needs, and to run the system without a simulation, the GPU is not required. The most important hardware component is the disk space is important to store the model, the dataset and the packages. Furthermore, a camera is required (either a webcam or external USB connected camera). This is necessary when capturing the dataset or when trying to move the manipulator. The better the quality of the image, the better the keypoint predictions will be with MediaPipe. As the accuracy mainly will depend on how well the keypoints are predicted, better cameras are preferable.

OpenManipulator-X The robotic manipulator which will be used for this project is ROBOTIS' OpenManipulator-X. The manipulator is composed of 4 joints and a gripper, and has 5 DOFs. It is not too big, having a reach of around 380 mm (dimensions and joints displayed in Figure 3.3). However, the project is designed to be applicable to any one manipulator, so if it was used in an application it is not specifically constrained to this one arm.

The main reason for using this manipulator is that the university has a couple of these available at CVSSP (at the University of Surrey). Furthermore, there are a lot of different libraries developed in ROS for this manipulator. Those include controllers, teleoperation, simulation files (used to simulate in Gazebo) and more. Out of ease, the majority of training and testing will be performed in the simulation, as otherwise all of the testing would have to be done in the

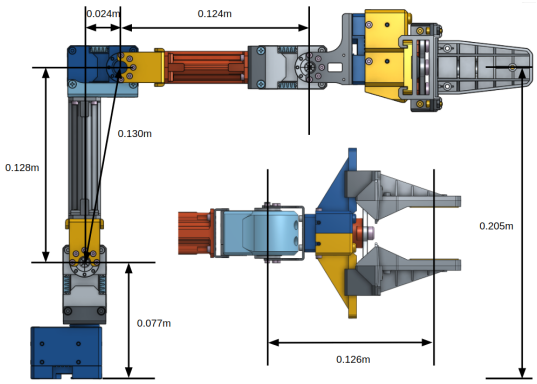


Figure 3.3: ROBOTIS OpenManipulator-X Size Chart [7]

university lab. Gazebo makes it easy to simulate the movement of the arm, and then it can be easily transferred to the actual arm. The DYNAMIXEL Starter Kit is a kit which contains smart actuators and provides a method to connect the manipulator to a computer.

3.3 System Requirements

There are two types of system requirements that will be described: functional and non-functional. Functional requirements (Figure 3.1) focus on the aims of the system and what is its function. Whereas non-functional requirements (Figure 3.2) are specifications that are used to test the operations of the system in order to work on its functionality. The following subsections will provide the tables summarizing the most important ones, both key and extended. The requirements have been inferred from the previous sections in this chapter to achieve the envisioned system with the aims and project scope described in Chapter 1.

3.3.1 Functional Requirements

Requirement ID	**Req. Statement** ("The system should...")	**Key / Extended**	**Dependency**
FR01	feed the input from the camera into the system	Key	-
FR02	recognize a person and detect arm keypoints (shoulder, wrist, elbow)	Key	FR01
FR03	detect the hand keypoints	Extended	FR01
FR04	track the movement of the arms	Key	FR02
FR05	calculate the position of the manipulator (in simulation) from the corresponding arm keypoints	Key	FR02, FR04
FR06	solve for the joint angles when the manipulator reaches the destination	Key	FR04, FR06
FR07	save the data	Key	FR06
FR08	visualize the moving points in RVIZ	Extended	FR07
FR09	use gesture recognition for triggering the data gathering	Extended	FR03
FR10	move the end-effector angle accordingly based on the hand position	Extended	FR02, FR03
FR11	load the dataset and save an ANN model after training the dataset	Key	FR06
FR12	move the manipulator based on the user's movements	Key	FR01, FR02, FR04, FR11
FR13	open and close the gripper when the user gestures an open/closed hand	Extended	FR03, FR10, FR12
FR14	display the video input and keypoints to the user	Extended	FR01, FR02
FR15	log all the information, errors, and warnings to the user in the command line	Extended	-

Table 3.1: Functional System Requirements

3.3.2 Non-Functional Requirements

Field	ID	Req. Statement	Comments	Key / Extended
Performance	NFR01	Collection Throughput - process 1 sec of video in under 3 sec	If keypoint processing is too slow, the movement will lag, affecting the smoothness of the control.	Key
	NFR02	Calculation Throughput - converting positions into joints should take under 0.2 sec/point	Longer processing will increase the training times considerably, especially the longer the video is.	Key
	NFR03	Accuracy - 90% of predictions should be within 1% cm and 95% within 1.5% cm (of the total reach) from the correct position.	See Section 3.4.3 for more details.	Key
	NFR04	Range of Motion - the extreme points should reach within 0.1 cm of min. and max. extreme points	See Section 3.4.3 for more details. See Appendix A for joint extremes.	Key
Robustness	NFR05	Environment - environmental conditions should not affect the performance of the model	Lighting, tolerances and parameters can be adjusted.	Extended
	NFR06	Users - Accuracies between different users should not vary (for instance, due to different arm lengths)	Possible adjustments include: calibration to normalize distances or different keypoint models	Key
User Experience	NFR07	Data Protection - saved locally for replay	Saved for visualization purposes; should not be saved anywhere online to not expose any confidential information	Key
	NFR08	Easy System Running - running the packages with one simple command and with all the environmental variables set with it	Simplify running it if there are multiple nodes involved and a lot of configuration variables (possibly could use a .ini file)	Key
	NFR09	GUI - Control from NFR08 with GUI	Makes it easier to run and visualize processes if it can be done with GUI rather than command line	Extended
Costs	NFR10	Adaptability - the system and control should be adaptable to any hardware arm	OpenManipulator-X can be costly, but there are cheaper alternatives or 3D printed ones	Extended
	NFR11	Cost of Running - running the simulation, training and pose estimation should not cost anything	Everyone should be able to run the system	Key

Table 3.2: Non-Functional System Requirements

3.4 System Design & Plan

Based on Figure 3.1 and the requirements defined in Section 3.3, a more thorough design plan was created to aid with the implementation of the softwares.

3.4.1 System Diagrams

The diagrams will showcase the interaction of different actors with the system and its components and how the information will move between the systems. Having diagrams for the implementation phase will immensely simplify the creation process, as a lot of planning on both function- and application-level will be done and ambiguity will be removed.

Use Case Diagrams The diagrams identify how the users and different actors are planned to interact between high-level functions of the system, as they were depicted in Figure 3.1, which in turn, will help derive lower level functions (as depicted in Section 3.4.1).

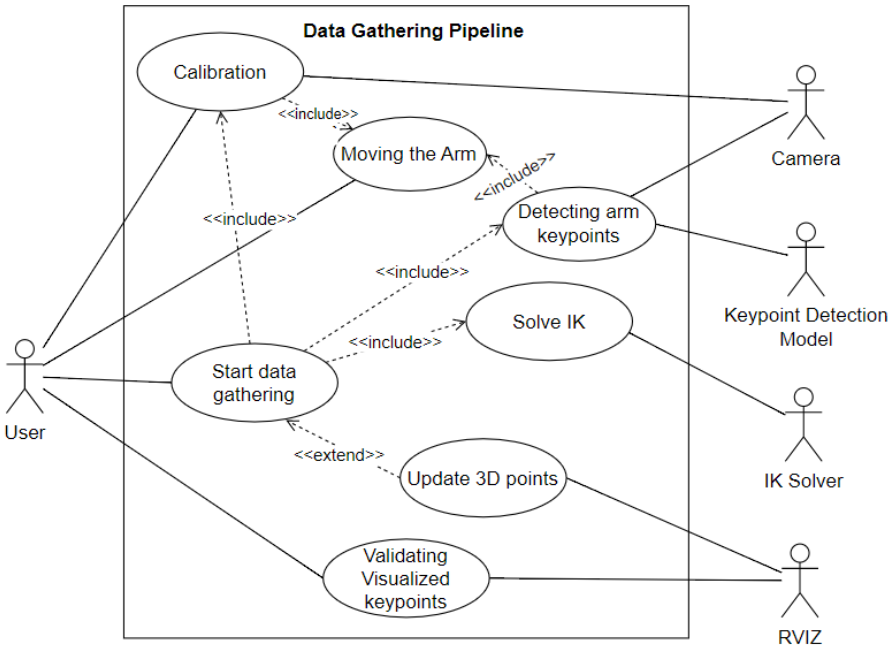


Figure 3.4: Use Case Diagram of the Data Gathering System

Figure 3.4 summarizes the different actors and they respective functionality in the data gathering pipeline. The user guides the process by starting the application which then includes the other

components. The main functions are calibrating the arm, which depends on moving the arm and keypoint detection, detecting keypoints, solving IK and visualizing the data. In order to achieve these functions, other actors are utilized, such as a camera (for recording the user), keypoint detection model, IK solver and RVIZ (for 3D visualization).

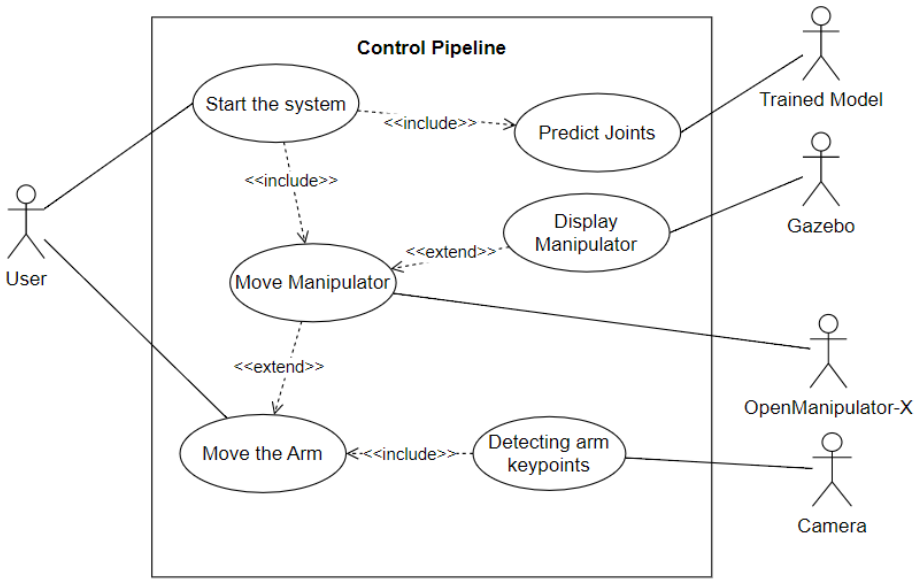


Figure 3.5: Use Case Diagram of the Control System

On the other hand, Figure 3.5 summarizes the control pipeline, where the user moves and controls the manipulator. Once the users start the system, they can move their arm around, while in the frame of the camera, and the model will estimate where the manipulator should move to. The manipulator can either be the physical arm or a simulated one in Gazebo.

Activity Diagram The following Figure 3.6 displays the flow of the system, and what functionality should be implemented in the data gathering pipeline. This system is used by a user to start gathering keypoints and later converting those keypoints into joint positions via IK solver.

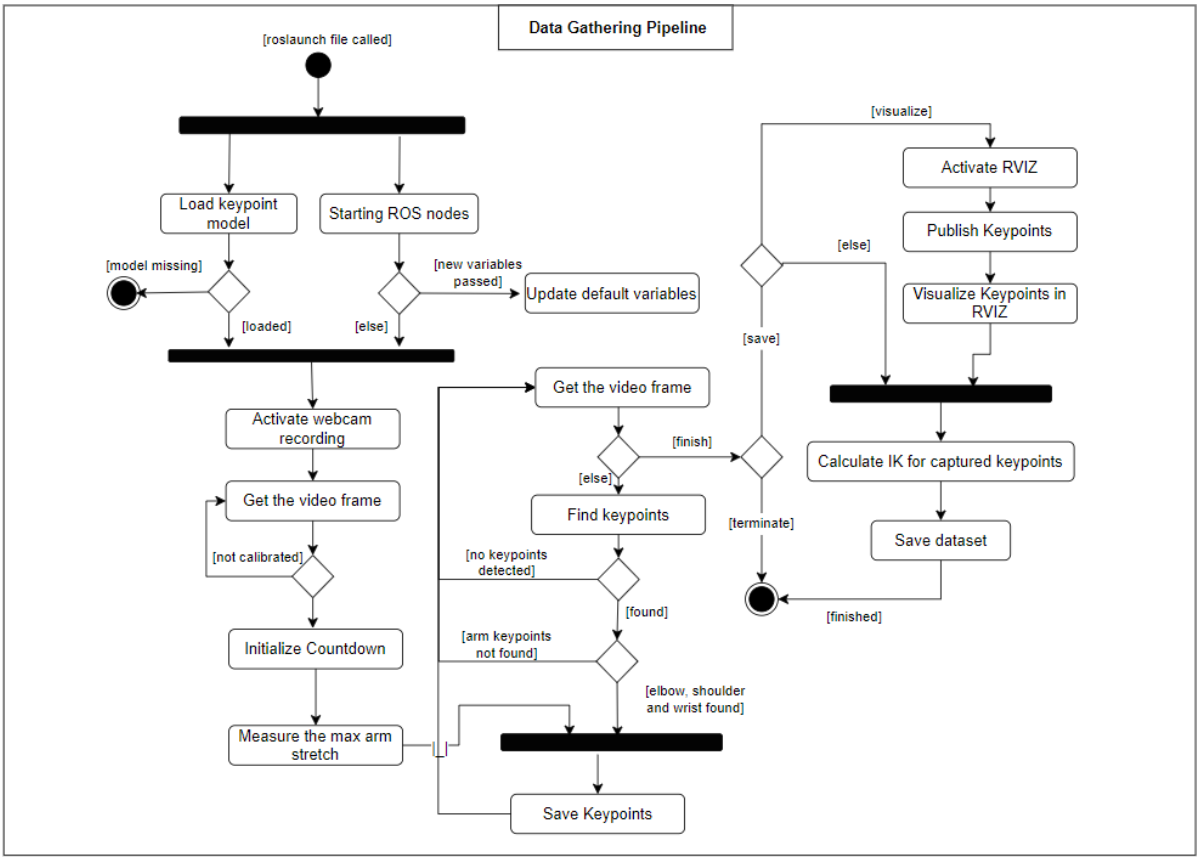


Figure 3.6: Activity Diagram of the Data Gathering System

The plan for this system is it make it into a ROS node, so that it can subscribe and publish to other nodes. To make it easier for the users, all the external nodes used, for instance RVIZ or IK solver, should be run with the node. This alleviates the need to run all of the packages separately, which can be tedious, time-consuming and cause errors if not run properly. Calibration is an added step which is meant to deal with the varied arm lengths and different distances from the camera that the users could be at. This can be accomplished by a simple button press in the terminal, or alternatively, a gesture recognition to make it easy for the user to manipulate when the data gathering starts and stops. Finally, before the IK are solved, there is an option to visualize the keypoints in 3D space, to understand the dataset, and see if there are any problems with it. Likewise, Gazebo could be used to verify if the keypoints are valid, and whether the arm can move there, however, this would slow down the system considerably as there would be path planning and moving involved.

Extensions and User Experience There are some possible extensions that can be added later on into the framework as defined in Section 3.3. As this is an Agile approach, there is flexibility to later add those requirements, and furthermore, all of the extended requirements can mainly be done after the key requirements are done. Ergo, they were removed from the diagrams in previous sections, and if there is enough time they can be added.

Having implemented MediaPipe in the framework, adding hand keypoint detection requires only adding additional model and processing of the frame with that model, and then it can be used in the pipeline. The main uses for this are, firstly, to get a correct angle of the wrist, and secondly, it opens up a possibility for gesture detection. Gesture detection can be used in order to control the manipulator (such as closing and opening the end-effector) or to control the calibration process without the need for the keyboard. That will improve the user experience as then everything is controlled with CV. If for instance, the person was recording the keypoints, and wanted to turn it off, he would have to get close to the keyboard which might cause collection of junk data.

Visualization with RVIZ was added as it is important for the users to view the data. To add further validation of the data collected and whether the calibration works as expected, simulation in Gazebo can be used. This would be slightly time consuming to run, but it can be run in parallel with RVIZ and the user can then verify how the points map from 3D to the manipulator. Likewise, there should be an option for the user to review the video and the keypoints collected as it can help with debugging of why the model is not training as well as expected. Finally, the user should be able to view all of the errors and warning when gathering the data, as that can save countless hours of debugging. To put it all together, GUI can be used to make it very easy to view everything and debug, instead of using a console. However, researching into different algorithms and neural networks is of higher priority in this project than user experience of the system, so it most likely won't be in the final solution.

3.4.2 Training Framework

As mentioned in Section 3.1, the plan for training the model is use a supervised training approach. The system will open up an opportunity to gather data from a human demonstrator. One way to do it would be by considering the velocity space of the joints. However, with the use of MediaPipe's keypoints, the location of each of the joints can be extraction in 3D space, and as the action space of the manipulator is pre-defined, the locations can be applied onto it. There

might be a need to calibrate the inputs, to make them fit the points within the manipulator's range (see Appendix A). With the data, an IK solver can be used to convert the joint angles, into positions of the manipulator. That would be sufficient enough to control the robotic arm, but as the research identified, that might not be the best approach. Therefore, those predictions can be used as ground truth in the supervised training, i.e. the numbers which the network would predict based on the positional data from the camera.

An ANN would be used to learn the relationship between the mapping of input positions and orientation of the end-effector to the actual joint angles. For that, back-propagation would be used with a loss function to train it. Some loss functions were identified during the literature review, but Huber Loss is one that seems like a good experimental option, which might improve upon more common techniques such as mean squared error and Mean absolute error. With the predictions, labels and the cost function, the network can be optimized to minimize the loss. Positional loss and joint angle loss can be considered when training, but this experiment will attempt to optimize for both with the use of NAS. This will further help with finding an optimal neural network and parameters, rather than experimenting with different one. The following Figure 3.7 summarizes the training network in a diagram.

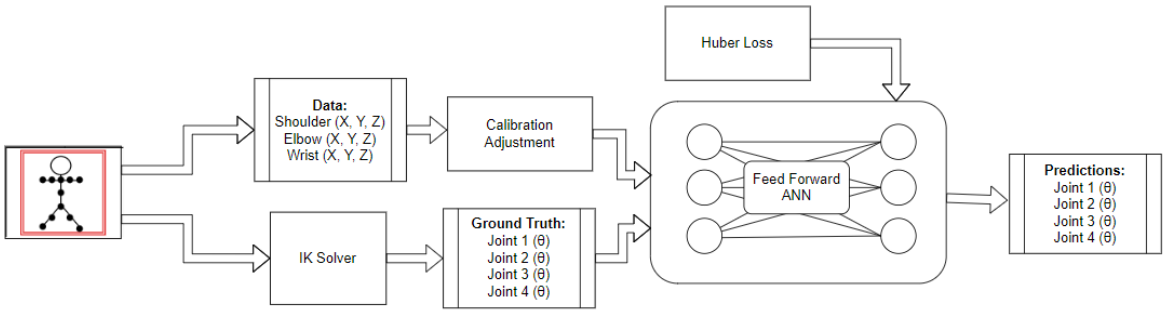


Figure 3.7: Training Framework

Depending on the results obtained from the above-mentioned solution (based on metrics identified in Section 3.4.3) the model could be improved upon based on other techniques identified in the literature review. Different networks, such as LSTMs, have been used to make the motion accurate and smooth. With the data gathering system, training one should be fairly easy,

and with the control pipeline adaptable to any model, testing it would be simple. Similarly, more advanced techniques could include reinforcement learning to teach the model to move the arm like a human with a reward function. However, such approach would probably require an extensive amount of data for the model to observe and learn from.

3.4.3 Evaluation

The evaluation of the model is extremely important as in order for the model to be useful, it has to be accurate, responsive and dexterous. Section 2.1.5 contains the literature findings of these proposed evaluation metrics.

Speed The processing of the manipulator arm should be very quick, in order to follow the movements done by the human, otherwise, the manipulator will start lagging behind the human. Tasks which require dexterity and quick response time (such as catching an object) would perform badly. To test the speed, an experiment can be devise which has a person move an arm from location A to location B and the time is measured. Then the same can be applied for a robot, and a simple difference between the two can be computed to find out whether the manipulator is much slower.

Accuracy The accuracy of the predicted joints is crucial, as without good accuracy, the manipulator will not follow the demonstrator accurately. By using IK solver, this would most likely not be a big issue, and thus it is a good baseline to compare to (especially in lower DoF manipulators). To measure the accuracy, the Huber loss function can be used, both for positional error and the joint angle error. Alternatively, mean squared error can be used for comparison between the two, but Huber loss deals better with outliers, which in this case can happen if the keypoint prediction was not entirely correct. A second experiment can be devised, to check how small changes in the positions can be made, in order to test how precise the model is.

A slightly different approach of testing the accuracy of the position of the manipulator in comparison to the arm, is by aligning point patterns by using Kabsch-Umeyama algorithm [8]. This algorithms is used to compare and find similarity between two different sets of points. It does that by computing the optimal scaling, rotation and translation (as it can be seen in Figure 3.8), by calculating root mean squared error deviation on those points. This is an option to calculate

how different the predicted points are from input points. The algorithm requires mapping between the two sets of points, however, the mapping will be created as one of the approaches to map the arm keypoints to the manipulator keypoints.

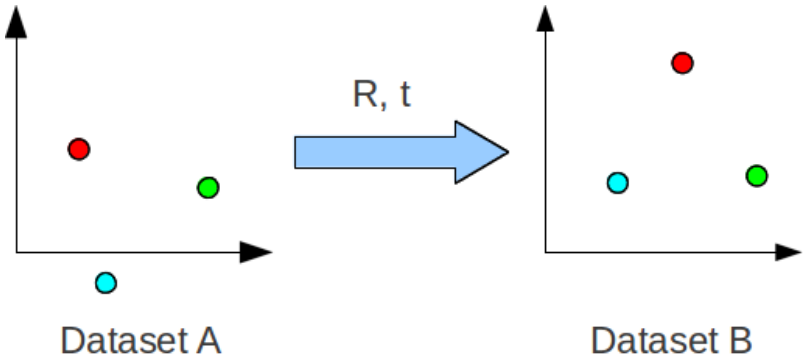


Figure 3.8: Search for Optimal Rotation and Translation (Kabsch-Umeyama algorithm) [8]

Range For the range (commonly referred to as the workspace), an experiment can be devised with the trained model. As the ranges of the joints are pre-defined due, to hardware limitations (see Appendix A), the expected range can be entailed. The range of the predictions should be considered for the minimum and maximum possible values. A subset of points on the sphere of movement can be taken, and verified how many of them are within the reach of the model, by passing the positions through and checking if the outputted joints are valid joint configurations.

3.5 Software Development Methodology

The approach to the development of the system has to be identified to choose one that best fits the proposed project. There are many different methodologies used in the industry, but the two major ones are Software Development Lifecycle (SDLC) and Agile development. The merits of both will be evaluated, based on the scope of this project.

3.5.1 SDLC

SDLC divides the whole project lifecycle into different stages: planning, defining the problem, designing the system, building and implementing it, then testing it and finally deploying it. This process assumes finishing one of the stages before moving onto the next one. There are several

popular SDLC models: Waterfall, Iterative, Spiral, V-Model and more.

If planning is done properly, and if there aren't many problems encountered throughout the project, this framework helps reduce costs and time. However, if problems arise, deadlines can be missed and the plan can quickly have to be disregarded. Furthermore, making changes later on in development might be difficult because of the sequential nature of the project.

3.5.2 Agile Development

In comparison to SDLC, Agile development focuses on iterative building of a project. It follows similar stages of the development, but it gives a lot of flexibility to development. Usually, the project is broken down into smaller pieces and each one is tackled one by one. Some Agile models include: Scrum, Kanban, Extreme Programming (XP).

Given the nature of the project, Agile development is more of a right choice. More specifically, this project will follow the Scrum process. Scrum's flow focuses on sprints, which are iterations in the development of the system (see Figure 3.9). Utilizing this methodology will allow for iterative and flexible development of the software, when aiming for the Product Goal.

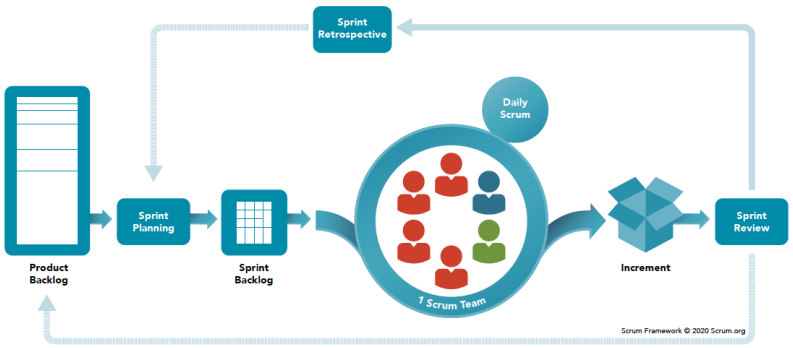


Figure 3.9: Stages of Scrum [9]

The flexibility will ensure that whenever new ideas or tasks are recorded in the backlog, they can be included in the upcoming sprints. The way that it would work for this project, is that every couple of week or two the list of tasks will be evaluated and prioritized for the upcoming week. Additionally, consistent and regular meetings with the supervisor will also help evaluate the current progress and steer the project towards the goal. It is important to revisit all the

tasks, to not lose sight of the goal and get stuck on an individual one. Finally, as mentioned before, I don't have a lot of experience in ROS development, and due to constant testing and possibility for improvements, this framework will work very well for this small scale project.

3.6 Feasibility & Risks

3.6.1 Feasibility

Finally, before implementing the project, a feasibility study has to be conducted to verify if the project is viable.

Technical Feasibility In general, the system is not largely demanding for the user from hardware perspective (hardware required was outlined in Section 3.2, and hence it is plausible for both using and developing it. The only requirements are access to a Linux system, a camera and a small amount of memory on a GPU in case of a simulation, otherwise a CPU is sufficient for keypoint estimation and the ANN training.

Availability to Linux highly plausible as most Robotics research is done in ROS, and hence, most users will be using Linux for it. Furthermore, it does not affect the development as there are available Linux machines at the university. The only issue might be admin privileges but an alternative free solution is using a Linux virtual machine for the development and testing. Furthermore, most laptops have in-built webcams so using a camera should not be a problem. Otherwise, due to COVID-19 and remote work, most people now have USB cameras available at home. The quality is not as important, as MediaPipe is expected to work well with bad resolution. Finally, in order to simulate the manipulator in Gazebo, GPU is required. However, it is not a crucial part of the pipeline, and hence, it should be an option to remove it to users. With regards to development, the laptop available has a GPUs, so developing and testing should not be an issue.

The more specific hardware, is the OpenManipulator-X. For development and testing, the access to the arm is not required, as it can be simulated. Therefore, the accessibility of the hardware does not affect the system in any way. In order to test the results and deploy the models on the arm, the time will have to be scheduled in the CVSSP labs and permission granted. However, the access to the arm does not necessarily affect the final result and hence it is still feasible to

complete the system without it.

From the software perspective, there are a couple considerations. ROS and MediaPipe are the major ones to ensure the success of this project. The software does not require a lot of computational power or memory to work. Furthermore, both of these are openly available, so they can be easily downloaded and used on any machine. The major computational load on the machine will be during the training of models. The GPUs are accessible on my local laptop and on HERON machines in the university labs, which will suffice. If more computational power is required, for instance to make the training faster with more GPU memory, online resources such as Google Colab or Amazon's SageMaker Studio Lab can be used to run it in the cloud. For the users, that will not be the case as they will use already trained models, and thus, the computational requirements for training ANN are only crucial in development.

Legal Feasibility Chapter 6 discusses the legal and ethical issues in a lot of depth, but in short, the development of the project is highly feasible, as all the libraries being used are open source, and thus should not infringe on any legal problems. Furthermore, there is no personal data stored anywhere which could be accessed by malicious users, and thus, the project does not cause any legal concerns.

Economical Feasibility Development cost for this project is an important factor to consider. The main external expenditure for this project is the hardware. There a number of OpenManipulator-X available at the university, and ergo, they themselves do not incur any additional costs for the development of the project. The camera being used can be any laptop or USB camera, and they are also available at the university to be used for the duration of the project. The only additional cost for the project is the ROBOTIS' starter kit which is required to connect the manipulator to a machine and a Raspberry Pi 4. The former has to be bought at £60. If the department does not have sufficient funding, the alternative is to complete the whole project in simulation, which would still lead to a successful project, as with Gazebo it is easy to deploy the code developed for simulation. A Raspberry Pi can be borrowed from the university for the duration of the project or a new one can be purchased. To conclude, those two external pieces are not essential, and thus, the project is feasible from economical perspective.

Schedule Feasibility The project is a mix of developing a system and experimenting into manipulator control. The second one largely depends on the first one. In order to stay on track, a Gantt Chart (see Figure 3.10) was devised indicating different milestones and tasks which need to be completed in order to stay on track. There is an expectation that the system will take a while to complete as I am self-learning ROS, so there is a high likelihood of getting stuck and having issues with it. However, with weekly (and sometimes bi-weekly) meetings with my supervisor, he can make sure that everything is on track and we can sort through any encountered issues. There has been a lot of time allocated, for these possible step backs, especially around the winter and Easter holidays. If the plan is followed, the project is feasible to be finished in time. Furthermore, completing the system is the main objective of the project (see Chapter 1) and performing some experimentation and research with the use of it. Thus, even if not much research is done, but the system is functional, the project will still be considered a success and it will open up a possibility for further research.

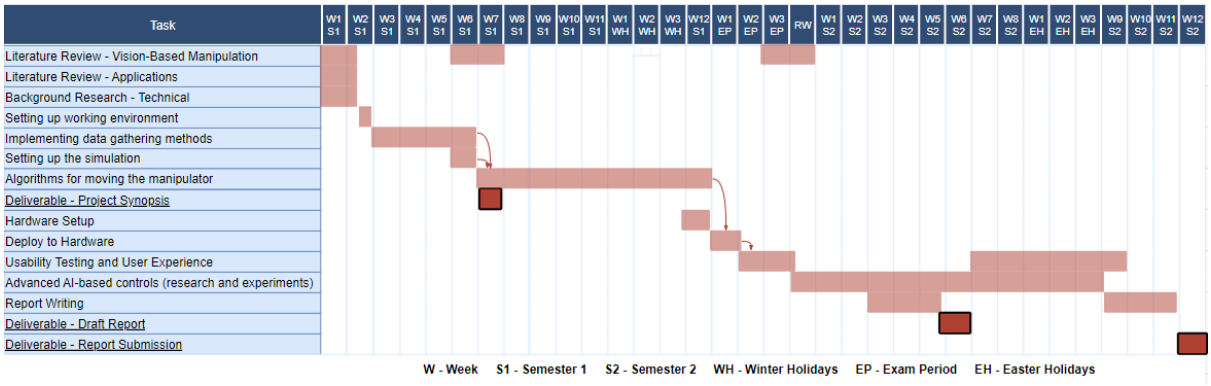


Figure 3.10: Schedule Plan

Operational Feasibility The system is built as a ROS package, and hence, to use it, users will download it into their ROS environment from a GitHub repository. Therefore the package has to include a *cmake* and *packages.xml* file, and all the required source files to run the system. Therefore, the only maintenance after the deployment, would have to be updating the system is if there are any bugs in the code or possible adjustments to be made. That does not require additional manpower or will not incur extra costs. It would only require extra time to do. In that regard, there the project is operationally feasible.

3.6.2 Risk Management

There are not many risks associated with this project, and not many that are in any way have dire effects on the completion of the project. Figure 3.11 gives the summary of the risks, how to minimize the possible risk and contingency plans.

What are the risks?	Who might be at risk and how?	Risk	What can be done to minimize the risk?	Contingency Plan
Faulty software	If the base software was used for an application, depending on where the arm would be used, malfunction could cause danger to the user.	Likelihood **High** Consequence **High** Risk **High**	* Accuracy and user testing * Safety features in case connection to the arm is lost (depending on what it is being used for) * Protect code from unforeseen errors	* Preventing the risk is the highest priority, but in case something happens protect the code as much as possible and test all the featuers
Hardware damage (ex. Robotic Arm, Manipulator Starter Kit, Raspberry Pi, Laptop)	Directly * Nobody Indirectly: * The student would not be able to continue working on some aspects of the project * The cost of the equipment is high	Likelihood **Low** Consequence **High** Risk **Medium**	* Take good care of the equipment when using it * Put everything back as it was and don't leave it lying around * Don't force the arm to go into extreme movements	* Laptop - can use a PC at the university * Robotic Arm - the project can be done in simulation and if someone needs to use the arm there is more available * Raspberry Pi - there are more available at the university
LIPO battery explosion	Will be used for powering the robotic arm. If the battery is overcharged or overdischarged it can burst open and cause smoke and fire.	Likelihood **Medium** Consequence **Medium** Risk **Medium**	* Cool batteryl before charging * Away from water * Don't leave charging overnight or unattended * Work in a safe environment * Don't leave around	* If working in a robotics lab, there are fire alarms and fire extinguishers
Privacy and Security - Data Exposure	Camera will be recording users person using the software (especially when trying to integrate into an application or during testing)	Likelihood **Low** Consequence **High** Risk **Low**	* Do not save the footage from the camera * Check if no footage is being saved after program running	* If footage is being saved somehow, delete the files
Physical Damage from Moving the arm	People around the arm when it is being moved could potentially be hit by accident (the motors shouldn't generate enough force for serious injury)	Likelihood: **Low** Consequence **Low** Risk **Low**	* Clear the area from around the arm when being in use	* Working in robotics lab has a first aid kid

Figure 3.11: Project Risk Analysis [Screenshot used from the Project Plan]

Chapter 4

System Implementation

System implementation is going to go through the step-by-step process of developing the system, and the experiments that have been done to design the manipulator control. The implementation follows the core design which has been described in Chapter 3, and it aims to achieve all the key planned goals from Chapter 1.

The system has been developed as an iterative approach following an Agile methodology. The way that sprints have been structured, were on 2 week basis. What that meant is that a TODO list would be set up for the upcoming two weeks, where new tasks are worked on or improvements are done. Throughout the sprint, different new tasks were added to the list, if some of them are pressing, they might be switched out for a currently allocated task. Usually, the end of the sprint will be marked by a supervisor meeting, in which progress was evaluated and new ideas were discussed.

4.1 Environmental Setup

Before commencing the creation of the system, the environment has to be setup with all the necessary libraries. As described in Section 3.2, this project will be using ROS Noetic, on Ubuntu 20.02. As there is a lot of packaged required to be installed, personal laptop was used with an Ubuntu 20.02 dual-boot, rather than the university computers, due to lack of admin privileges. Alternative plan was to use a Virtual Machine on Windows, but due to the incompatibility issues of the laptop's camera with the Virtual Machine and slower Gazebo simulation, the dual-boot Ubuntu was a preferred choice.

In order to install all the requirements for ROS, a step-by-step guide has been followed on the official OpenManipulator-X documentation by ROBOTIS [7]. With ROS installed, there were numerous libraries that had to be installed, as well as different repositories cloned. This is to access all the available files for OpenManipulator-X, which provide many openly available libraries, for things such as path-planning, simulation and control. The following are the main repositories used for this project:

- open_manipulator [46]
- open_manipulator_msgs [47]
- open_manipulator_simulations [48]
- open_manipulator_dependencies [49]

With the libraries setup, all the code is placed in the *catkin_ws/src* following the ROS directory structure (as shown in Figure 2.5, and *catkin_make* is run to build the directory.

4.2 Data Processing & Gathering System

The first objective of the project is to create a pipeline for data processing and gathering. As Figure 3.6 has depicted when designing the data gathering system, the input is the video frames from the camera. The aim of the system is to retrieve joint locations in 3D space, solely from the arm. The creation of the software was split into 3 main parts (which served as the sprints), allowing for iterative development and testing of the created software.

4.2.1 Keypoint Collection

The first requirement of this project, and one on which the whole pipeline depends on (see FR01 and FR02 in Section 3.3.2), is the keypoint collection. Without it, the data for training cannot be gathered and the manipulator can't be moved via CV. As indicated in Section 3.2.1, MediaPipe was identified as the most suitable keypoint detection library for this project.

The first step is to get a camera working and capturing the movement. The package chosen for this was OpenCV, which is a Python-compatible library which provides functions for CV. Most

importantly for this project, it provides the image from the webcam as a NumPy array, which can be directly fed into MediaPipe for processing of the keypoints.

As identified in the literature review (see Chapter 2). Mediapipe library has a pre-trained Pose model that can be used to return Pose Landmarks with the location in 3D space and visibility of the joint (as sometimes the location is predicted on other visible joints). The model suits this use-case, as the input to the model will be sequential frames (i.e. video), and the model takes into consideration previous locations of the joints, making it more accurate. Figure 4.1 displays how the joints are mapped in a 3D space. Depth is a crucial component to map onto the manipulator and get a full 3D range of motion. Furthermore, the model being used was trained with exercise in mind (using references such as yoga and dance), hence the model should be well adapted to precise and dexterous movements of the arm.

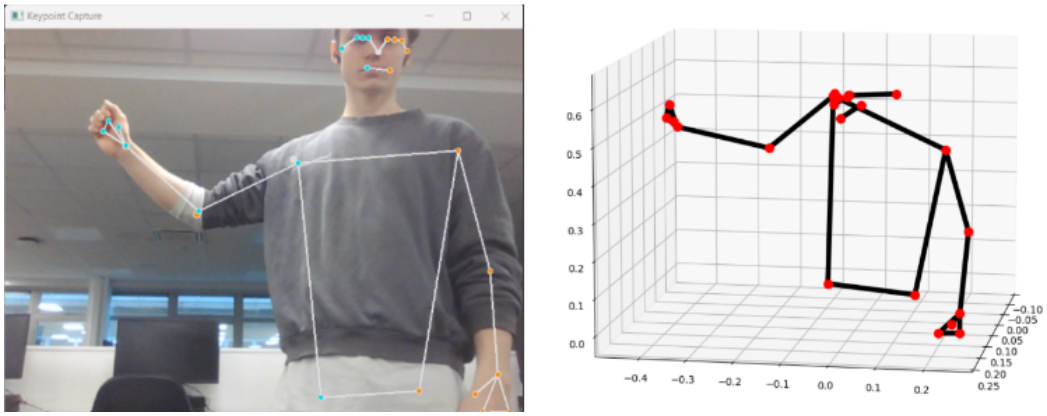


Figure 4.1: Example of MediaPipe Pose 3D coordinates mapping

However, for this project, there is no need to process all of the keypoints, other than the shoulder, wrist and elbow (with their respective MediaPipe IDs of 12, 14, and 16). Only one arm (right one) was considered, and everything else was filtered out (as there are 33 keypoints in total), to remove the noise. Even though the hips are not shown or used in any calculations, they should be visible in the video as all the keypoints are respective to their midpoint. Once the keypoints are filtered out, the video can be showcased to the user, so they can visually verify whether the keypoint capture system is working correctly, and whether it is making accurate predictions. This subsystem saves all the captured keypoints (if the current capture is not used to control the manipulator in real-time) in form of a pickle file, with all the XYZ-locations and visibility of the arm joints. That way, they can be mapped on the manipulator and converted into joint positions with IK. The output of this system with filtered out joints can be seen in Figure 4.2:

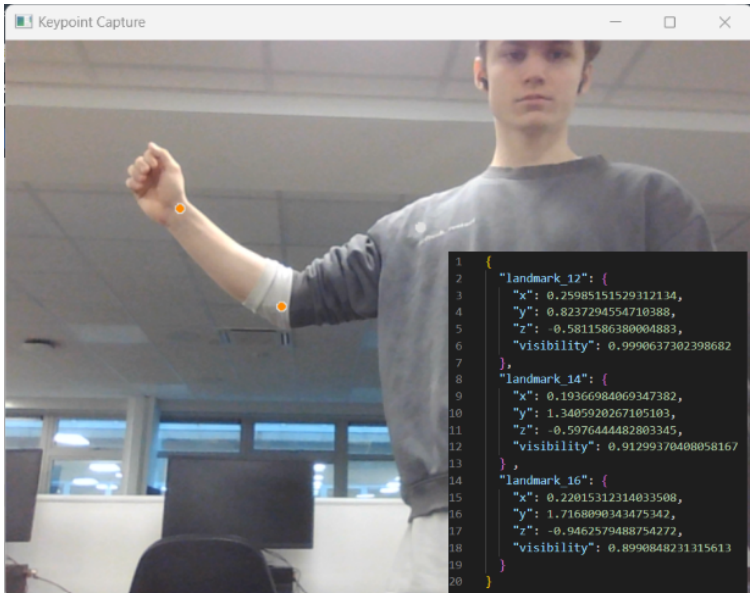


Figure 4.2: Saved Keypoint Example

Finally, to summarize the design of this subsystem, the following sequence diagram was used:

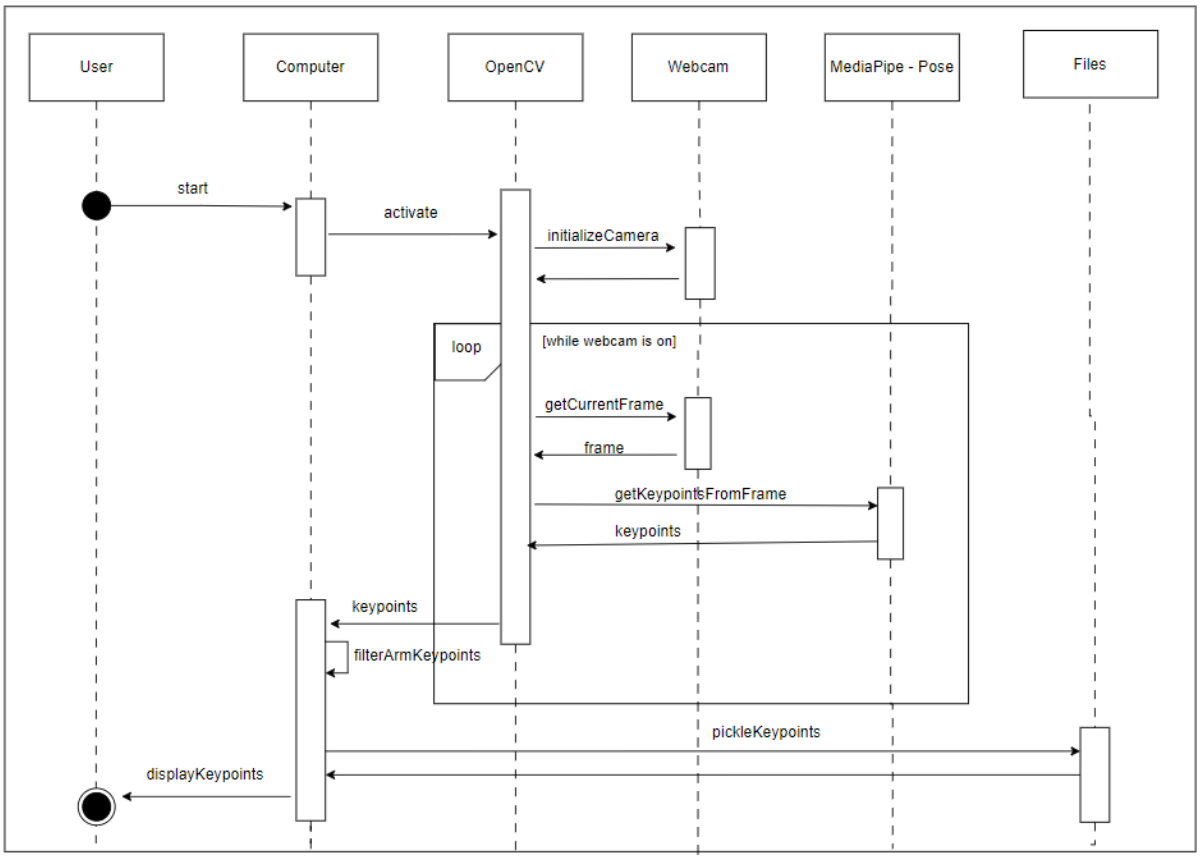


Figure 4.3: Sequence Diagram of Keypoint Gathering

4.2.2 Manipulator Conversion

The next step in the process is to use the keypoints obtained in Section 4.2.1 and apply them to the manipulator. Referring back to the plans from Figure 3.6, in order to calculate the IK for the end-effector, the joints from the human arm have to match the manipulator's (see FR05 in Section 3.3.2), as the MediaPipe 3D space won't directly match to the manipulator's action space. There are various approaches which can be used to implement this, but the one which suits the project the best was by using Transforms and 4x4 homogeneous transformation matrices. The reason for that is that it removes ambiguity, in contrast to manual computation, making it easier to manipulate the end-effector, whilst also giving additional resources for debugging the system. The initial plan was to map them as the user records the video, but that would slow down the system, and hence the keypoints were pickled and can be loaded in by another script.

Transforms ROS contains a package for transformations (named *tf*), which provides functions to find relations between different coordinate frames in a tree structure. Thus, the difference between the wrist's and end-effector's coordinate frame can be discovered. In general, the tree structure can be imagined as if the shoulder is moved (parent node), all of the joints connected to it (child nodes) will be moved accordingly. Same principle can be applied to the end-effector, if the wrist moves slightly, the end-effector should move with the same motion, i.e. the motion of the manipulator is mimicking that of the human. Furthermore, transformation chains between the arm and the manipulator can be visualized in RVIZ (a software for visualization in 3D space) for debugging - see FR07 in Section 3.3.2.

In order to create the transform tree structure between all the points, they have to be published in ROS. Publishing and Subscribing will be discussed more in-depth later in Section 4.2.3. The points which need to be published are shoulder, elbow and wrist, which are all respective to the shoulder (i.e. the shoulder is the origin). The shoulder should match the manipulator's base (as it is the reference point and the base around which the motion happens). Following formula summarizes it mathematically:

$$\bar{V}_t = \bar{V} - \bar{S}, \tag{4.1}$$

which a simple difference of vectors, where V_t is translated position V and S is the shoulder position. To publish the points, a correct message type has to be used. For this particular

case, a *sensor_ msgs/PointCloud* can be used, as it can be visualized by RVIZ. The *PointCloud* message is composed of various fields which are summarized below:

```
std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
geometry_msgs/Point32[] points
    float32 x
    float32 y
    float32 z
sensor_msgs/ChannelFloat32[] channels

```

Listing 4.1: PointCloud Message Type

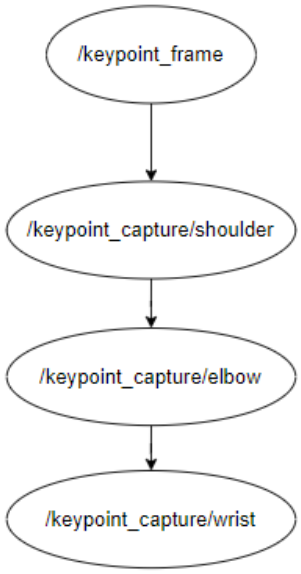


Figure 4.4: Arm Transform Tree

However, only the points and the header are of interest. The points will be displayed in the 3D space in RVIZ, but for them all to be displayed correctly and in the same frame, the frame ID has to be defined and the publishing time.

With published keypoints, the transform tree can now be composed. A *TransformBroadcaster* can be used to achieve this, sending the transforms and calculating the transformation and rotation between the reference frames. Thus, there is a link created between all of the joints, all of them having relative transformation corresponding to each other. A transform tree can be then composed for the arm as shown in Figure 4.4.

Manipulator Transforms The manipulator will be on the same vector as the wrist to track the motion. Scaling will have to be applied to match manipulator length. Otherwise, different arm lengths would result in different end-effector locations, therefore it has to be normalized. For that, calibration was implemented (see Section 4.4). From the previous section, a relative pose between the map and the shoulder (the base) was discovered, and a relative pose between the joints in the chain (shoulder to elbow, elbow to wrist). In order to get the relative end-effector point (i.e. the base) a 4x4 homogeneous transformation matrix can be used (see Equation 4.3).

Simply put, a transformation matrix ($\bar{T}$) applied onto a vector $\bar{A}$, transforms the points of the vector ($\bar{A}^t$):

$$\bar{A}^t = \bar{T} \cdot \bar{A}. \tag{4.2}$$

A homogeneous transformation matrix includes both a rotation and transformation [50], and it can be defined as follows for a 3D space:

$$\bar{T} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x_1 \\ r_{21} & r_{22} & r_{23} & x_2 \\ r_{31} & r_{32} & r_{33} & x_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.3}$$

where r's represent the Rotation Matrix and x's the vector which has the location of the old origin. With that, the transform between the wrist and the end-effector position can be compiled. And thus, the wrist position, as obtained from the keypoints (and scaled), can be then translated into the position of the manipulator. So when the wrist moves, the manipulator will move accordingly.

These are all the parts required to compose the full transform tree. All of the points can be visualized in RVIZ, as it can be seen in Figure 4.5.

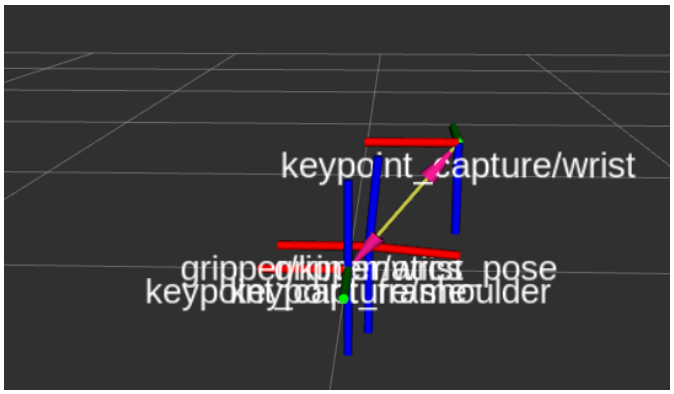


Figure 4.5: Transformations in RVIZ

4.2.3 IK Solver and Train Dataset

For now, the scaling which is used to get the manipulator position is an arbitrary, pre-defined, number which has been found to work for the specific recording (Section 4.4 will automate that scaling). In order to get the joints from the manipulator position, two approaches have been considered. The first one looked into simulating the path of the manipulator to get the final joints and second one was directly calling on the IK solver to calculate the joints for the desired position.

Gazebo The previously installed packages (Section 4.1) contain functionality which can be used to move the manipulator to the desired position (Figure 4.7), as well as calculate the joints of different positions. It can be simulated with *open_manipulator_gazebo* and moved with *open_manipulator_controller*. Using a service in ROS is similar to using a message, the main difference is that the service gets a response back. An useful example, is the */goal_task_space_position_only*, which sets the kinematic pose of the manipulator. Response in this case is a boolean of whether the arm was moved or not. In order to run this, the Gazebo simulation has to be running and the *pause* flag has to be turned off.

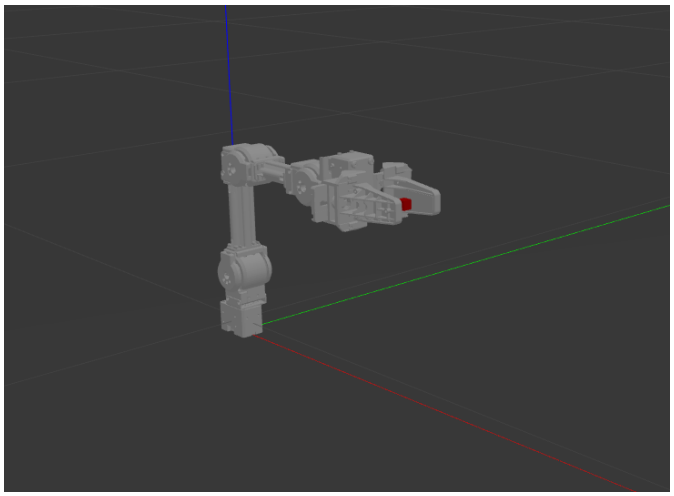


Figure 4.6: OpenManipulator-X in Gazebo

As shown in Figure 4.7, this serves as a good visualization of how the arm would be moving. However, there are some problems with it. Firstly, it is very slow. As the joints are not returned, but instead the path is planned and executed, it takes a couple of seconds for every single video frame. In order to convert the points into the training data (i.e. get joints) it is a very slow

process (which does not satisfy NFR02). Secondly, the path can not always be found. Under the hood, this service calculates the IK and tries to move the arm to the desired location. Sometimes, even though there is a reachable point, the function will fail and not move the manipulator. This causes problems as it limits the software to the current location that the arm is in. If it was in a different configuration it would be able to move there. Once again, for data collection, that would cause a lot of inaccuracies, because points in range would not be considered in training. During testing, over 50% of the points have been lost due to this. An alternative approach which helped with this was to make iterative movements if the planning fails. This entails trying to move the arm to the desired location, and if it fails, trying to move it half of the way, and then repeating the process for a couple of times, until it fails a couple of times or reaches the destination. This helped to increase the reachable points to about 75%, but making it considerably slower.

To conclude, this approach has a merit of visualizing the manipulator, but it is slow and it relies on path planning a lot for accuracy.

IK Solver The previous approach from Section 4.2.3 can be adjusted to overcome its flaws. Instead of executing the path, the function to only plan the path can be used. The main challenge of this is that there is no service to directly talk to the IK solver to plan the path. As the code for it is in a separate repository, changing the code is not an option. In order to get the flexibility of adjusting parameters and getting the joints, C++ code can be used as an intermediary between the Python script (*training_ data.py*) and the IK solver (see Figure 4.7).

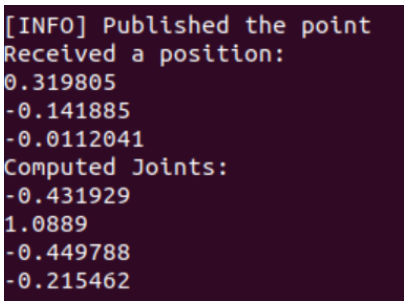


Figure 4.7: IK Solver Example

The C++ script has to therefore be both a subscriber and a publisher, same with the Python

script. On top of it, a custom message has to be defined in order to send the joint positions back. For sending the messages to the C++ script, a pre-defined *KinematicsPose* will be used. Both are summarized below:

```
### KinematicPose.msg ###
geometry_msgs/Pose pose
float64
   max_accelerations_scaling_factor
float64
   max_velocity_scaling_factor
float64 tolerance

### JointPositions.msg ###
JointAngle[] jointPositions
open_manipulator_msgs/
   KinematicsPose manipulatorPose

bool success

```

Listing 4.2: Joint and Position Messages

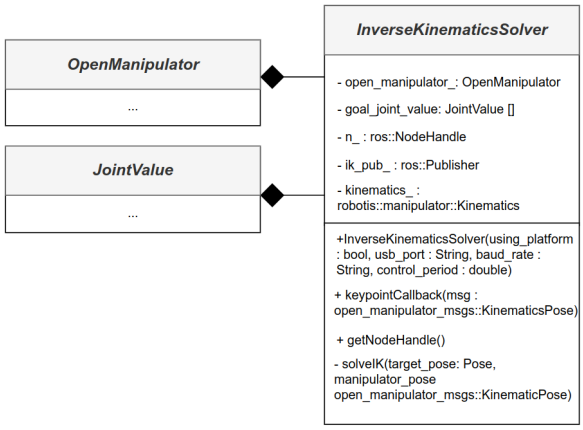


Figure 4.8: UML Diagram of Inverse Kinematics Solver

The above UML diagram (Figure 4.8) showcases the C++ that is being used for receiving and processing the IK. In general, it is a summary of the header file. There is only one class being used along with a main class, and it has a couple of different data types being used, but does not inherit any functions.

With the setup complete, the final step is to get back the joint values of the goal position. The *open_manipulator_controller* has a function to solve IK with the target position. This simplifies the process of calculating the IK, which are fairly hard to compute, and the C++ can return the computed joints and the target location which was used to th Python script via a message, and the data can be saved for training a ML model. Overall, each point takes less than a second to calculate, which is a significant improvement over the previous method, and the number of successfully calculated points is much higher (as it does not rely on starting position to reach the goal).

4.2.4 Conclusion

To conclude, with this part of the system the training data can be collected and this system can be tested. The following Figure 4.9 summarizes the system directory. The *data* file contains the necessary model for extracting keypoints, and the pickle files of the collected data. The *include* and *src* directories contain the C++ files. The *scripts* directory has the required Python scripts for collecting the data with the keypoints model, and the second one to convert the data with an IK solver. Launch files will be discussed further in Section 4.4. Finally, the custom msg files are located in *msgs*, which contain the body definiton of the messages being used.

Figure 4.9: Full Data Gathering Directory Structure

Moreover, the functionality of the data gathering and processing was summarized in Figure 4.3. Below the second part of the system is summarized in Figure 4.10.

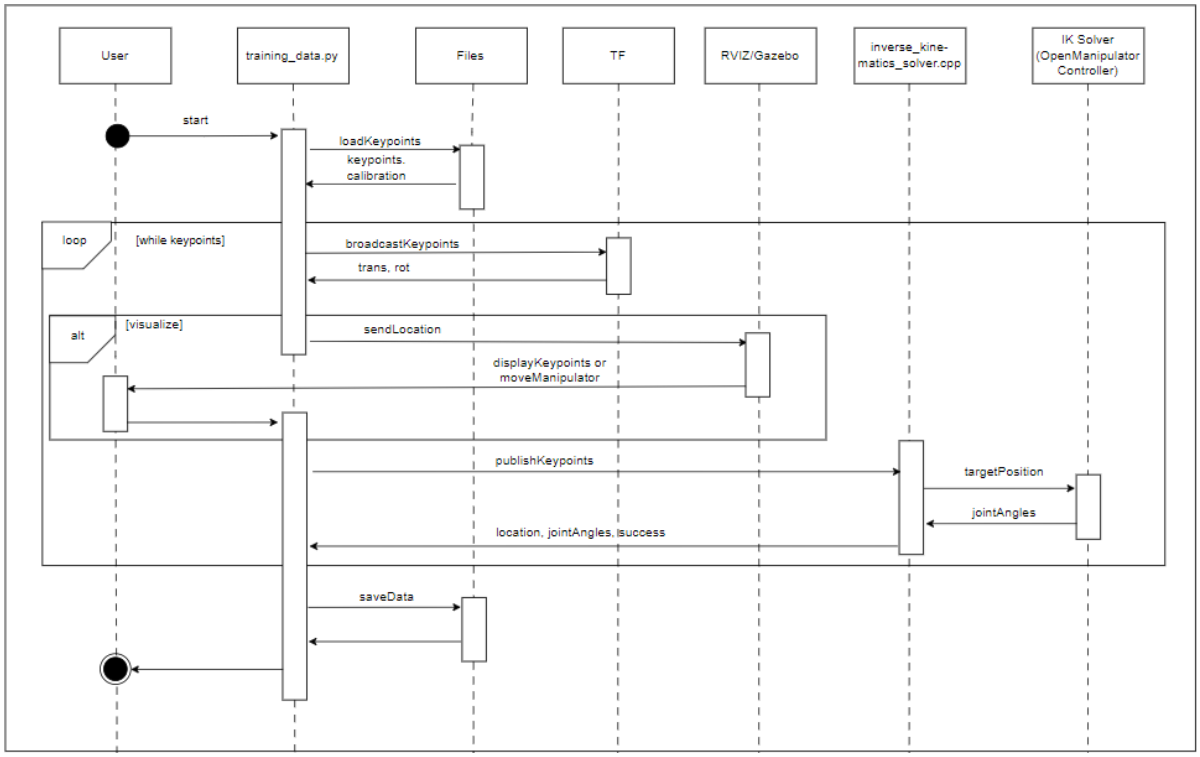


Figure 4.10: Sequence Diagram of Converting Keypoints into Valid Manipulator Joints

4.3 Control System & Training

This subsystem's main functional requirement goal is the FR12 (from Section 3.3.2), which relates to controlling the manipulator with arm movements. This system is the culmination of the findings and software created for the previous subsystem (see Section 4.2), without which this part would not be doable. The two main components of this pipeline are the control of the manipulator, and the training of the ANN model for control.

4.3.1 Control Pipeline

With regards to controlling the end-effector, most of the pieces required were already implemented. A new script could be created for control, however, the data collection subsystem has most of the components, and thus the sequencing as proposed in Figure 4.3 can be altered to meet the requirements.Most important components include capturing of the keypoints and calibration (see Section 4.4. This will allow for a more modularized code, and will only require an addition of some flags. The only two additional requirements, are a boolean flag indicating

whether control will be run, and secondly, the model which is going to be used to predict IK.

With the flag indicating whether control is being run, a new state can be added to the possible states. It will be similar to the collection, but without saving the keypoints, and once the control is stopped, the program will be terminated. With the keypoints being predicted, either directly with the IK solver (default baseline) or with a trained ANN (see Section 4.3.2), the joint angles are fed into the *goal_joint_space_path*. This service is similar to the one described in Section 4.2.3, but instead, it computes a trajectory which moves the manipulator to the desired joint configuration, bypassing the problems caused by moving with the end-effector location and IK. An example can be seen in Figure 4.11.

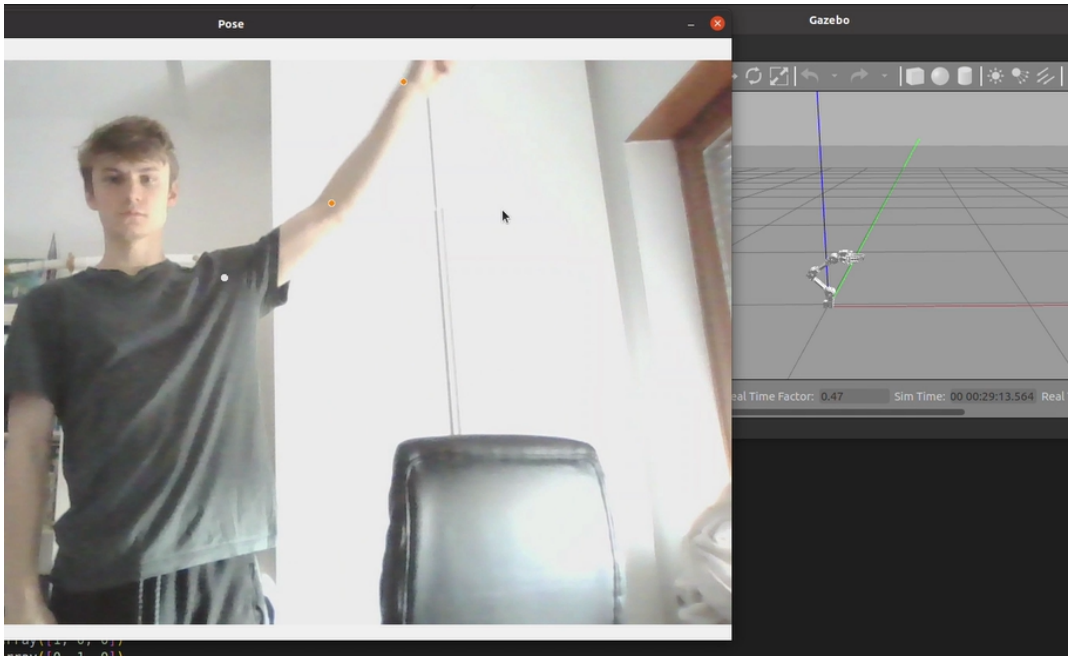


Figure 4.11: Manipulator Control in Simulation

The extended sequence diagram in Figure 4.12 summarizes the process described in this section. The data gathering portion was omitted in this Figure, as the functionality and data flow stayed the same.

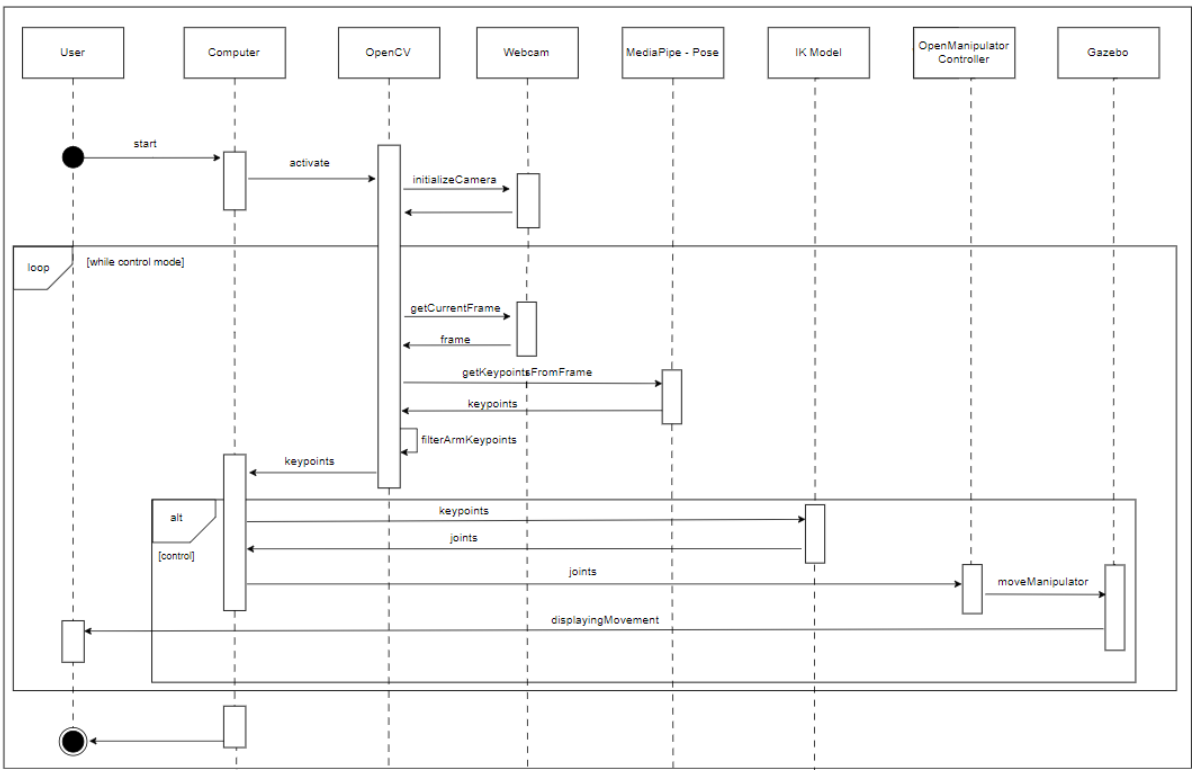


Figure 4.12: Sequence Diagram of Control Pipeline

4.3.2 ANN Training

To control the motion of the end-effector with the help of ML, the method proposed in Chapters 2 and 3 was used. ANN network is trained in a supervised manner, where the data gathered in Section 4.2, i.e. the locations of the mapped 3D locations to the manipulator, is used as the inputs. To train the the supervised model, joints (which were predicted with an IK solver) serve as the ground-truth during training.

The training pipeline is similar to the proposed design in Figure 3.7. Firstly, the data is being prepared for training, where the XYZ-positions are added to the input, along with the end-effector rotation (discussed further in Section 4.4). Firslty, a Euler angle is used as the end-effector only has one DoF, and thus two values of the quaternion would always be zero. Therefore, there are 4 inputs the network. For more DoF a better solution will be to incorporate quaternions. Secondly, the XYZ-positions are the positions obtained from the keypoints, as otherwise the transformation would have to be computed every time. The transformations described in Section 4.2.2 are only used to get the corresponding joint pair from the IK solver,

but other than that they are not used.

Two additional features were implemented to improve on the training. Duplicate removal was implemented to remove repeating data, just in case during the data gathering phase same data points were added, as it could skew neighboring points towards that point. Secondly, some synthetic data was created. For the actual positions that were collected, some of the data had a random rotation added to the angle of the wrist, to teach the network that twisting the wrist without any movement should only correspond to the end-effector rotation. A good range of data, with a lot of points has to be collected in order to make the training as accurate as possible. The processed, data is then placed into a tensor, ready to be inputted into the ANN designed in PyTorch.

The model which is being can be defined as a function g for the purpose of other equations, and the predictions ($\hat{y}$) from the network can be defined as follows:

$$\hat{y} = \begin{bmatrix} \gamma_1 \\ \vdots \\ \gamma_4 \end{bmatrix} = g(\begin{bmatrix} l_x \\ l_y \\ l_z \\ \theta \end{bmatrix}), \tag{4.4}$$

where γ is an angle of the available joints, l is the location of the arm, and θ the angle of the wrist.

NAS Initially, a randomly chosen, ambiguous, ANN architecture was used for training. However, unlike with Deep Learning tasks (such as vision and language), there are no SOTA pre-trained and pre-defined architectures. Additionally, this problem is not directly comparable to the research conducted, as the inputs to the network are not locations in the space that the joints are predicted to, but instead they are in the MediaPipe's action space. Therefore, the choosing of the network in this project was treated as an optimization problem. In order to find the best suited network, a NAS was implemented, which tests various different architectures with different learning rates. The experiment was setup to create a network which has:

1. 0 to 2 hidden layers
2. Each layer can consist of [8, 16, 24, 32, 64, 100, 128, 256, 512] neurons
3. Different dropout rates varying from 0% to 50%

4. Activation functions chosen from the following set: {Sigmoid, Tan(h), ReLU, Linear}.

These parameters were pre-defined, to decrease the search space, and make convergence faster. One main objective of this training is to detect whether using a shallow network is sufficient for the gathered dataset, and how wide it should be. A sufficiently wide ANN should be able to approximate well enough with sufficient amount of data. However, it might not generalize as well to previously unseen data (in this case, the unseen data would be the points in-between the points provided for training, as it would be impossible to gather all the possible points in the manipulator's action space and train a network on it). Multiple layers in the ANN help to learn features at different levels of abstraction.

On the other hand, a higher dropout rate can help with overfitting of the dataset. Once again, the ANN should be able to predict previously unseen data, and the best way to determine the dropout is through experimentation. The last parameter is the activation function being used. The activation functions serve as a transfer between the layers, and using a non-linear one can help to adapt better to the data and differentiate between the outputs. Similarly, using non-linear and linear functions in the last layer, will have a significant impact on the outputs that the network can predict. An example of two vastly different generated architectures can be seen below:

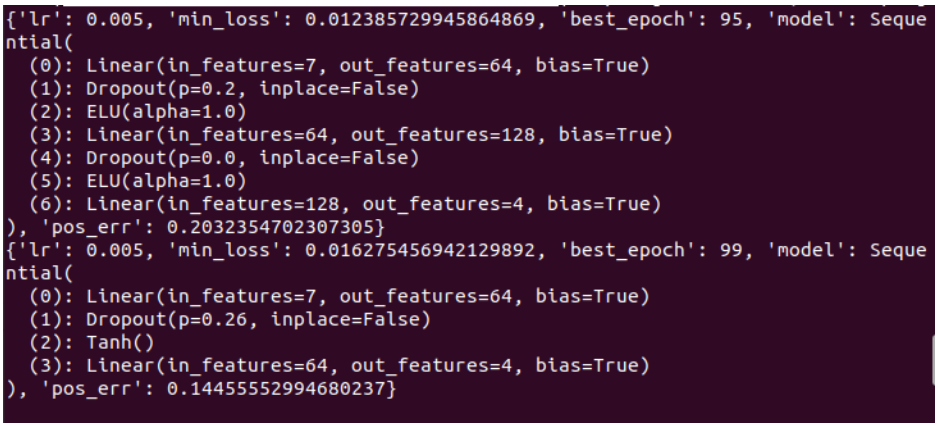


Figure 4.13: Example of Output Architectures Produced by NAS

All of the created network architectures will be tested on various learning rates. The learning rate is another value that usually has to be determined experimentally, as having a learning rate that is too big might make it hard to decrease the error of the cost function and thus improve the network. In contrast, having too small of a learning rate might cause the training to get stuck

in a local minima (rather than reaching a global one), and also making it slower to converge.

The evaluation metric for accuracy will be discussed later in Section 4.3.3. In short, the best model will be chosen via a multi-objective optimization, where the positional and joint loss will are the objective functions (f), which are being minimized. Pareto front will be computed to find the best solutions, which is defined as a set of points that are not dominated by any other points. A solution $x_1 \in X$ (set of all the solutions), is considered as non-dominated by other solutions $x_i \in X$, and therefore part of the Pareto front, if:

$$\{\{\{\forall j \in F, f_j(x_1) \leq f_j(x_i)\} \wedge \{\exists j \in F < f_j(x_i)\}\} : x_1 \neq x_i\}, \tag{4.5}$$

where i specifies the number of the objective functions, and $F = \{1, 2\}$, which signifies the 2 objective functions being optimized for.

The solution with the best trade-off between the two will be picked, manually by reviewing the graphs.

Loss Function In order to predict the numerical values, a cost function to minimize the difference between the predicted joints and provided ones has to be minimized. Figure 4.14 shows different options for a loss function.

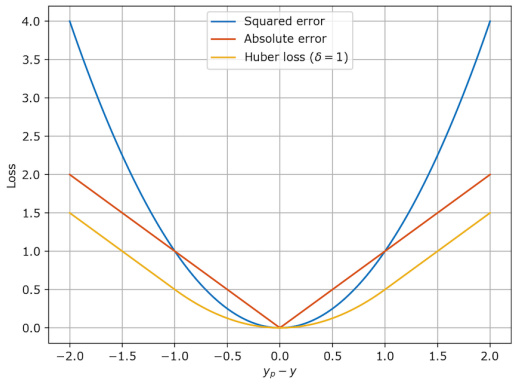


Figure 4.14: Loss Functions Graph [10]

In order to estimate the error of the predictions, the observed values (y) is compared to the predicted value from the network $\hat{y}$ (as defined in Equation 4.4) for a number of data points (n). Mean Squared Error is a first option, with the following definition:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{4.6}$$

However, there is an alternative option (which was discovered in Chapter 2) called the Huber loss, which can be used for robust regression and which is less sensitive to outliers in data. The data is expected to be noisy, as keypoint capture will not always be perfect due to many factors affecting the performance of the model, hence outliers can often skew the model away from the right solution. Huber loss, which in turn computes the Joint Error (JE) can be calculates shown below:

$$JE_\delta = \begin{cases} \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(y_i - \hat{y}_i)^2, & |y - \hat{y}| \le \delta, \quad (4.7a) \\ \frac{1}{n}\sum_{i=1}^{n}\delta(|y_i - \hat{y}_i| - \frac{1}{2}\delta), & \text{otherwise}, \quad (4.7b) \end{cases}$$

where δ is used to define a point where a transformation from quadratic to linear function occurs.

Furthermore, the difference between Adam and SGD has to be experimented, as the former converges faster, but SGD usually converges to a more optimal solution, but is at a disadvantage if there is a high variance in the data.

Data & Training Setup Based on the previous sections, the experimental setup can be created to train the most optimal network for solving IK. The dataset will be created by taking a couple of users, ideally with different arm lengths and in different lighting conditions to add variability to the data, and asking them to record keypoints by doing various full-range of movement motions. Full-range of movement in this context is defined as moving the arm in all the 3 dimensions around the body, whilst also trying to cover as much ground between the maximum and minimum span of motion.

The computed data will be split with an 80:20 train-test split, to have previously unseen data for the model, and to verify that the model can learn to interpolate between different data points in space as the whole action space won't be included in the training. Adam will be used for NAS training, as it usually converges faster, and thus, needs less epochs to run. The NAS training will be run for 15 iterations (creating 15 different architectures, each re-run with different learning rates). The best solution will be saved, and it will be run once again, but this time on SGD and with a learning rate scheduler. SGD will be used to verify if it has performed as well. The

learning rate scheduler will be used to decrease the learning rate over time, when the training loss reaches a plateau. Without it, the values will sometimes pre-maturely converge and get stuck in a local minima. There are various alternatives that can be explored, but this one is easy to incorporate, and easy to verify whether the training gets stuck.

Finally, the best chosen model will be used in the control pipeline to predict the joint configurations from the locations of the arm. The performance will be evaluated and compared to the IK solver in terms of the non-functional requirements (defined in Section 3.3). The evaluation metrics which will be used are going to be described in the upcoming Section 4.3.3.

4.3.3 ANN Evaluations

Accuracy Section 4.3.2 identified the different error functions which can be used to evaluate the model, including Huber's loss. Those cost functions will look at averages between predictions and actual values. Therefore, as the predictions of the network are, joints, it is the overall joint loss, which is used to optimize the network. Therefore, even though the best architecture will be used based on the trade-off between positional and joint loss, the network will be optimized based on joint loss.

A positional loss on the other hand, would be calculated by feeding the joint predictions into a FK solver (FK), to get the corresponding positions. Luckily, similar to the system developed in Section 4.2.3, the FK function can be called. The process is exactly the same, except a different function is called and a different message is sent to the C++ file. With that, the positional error (PE) can be computed:

$$PE = \frac{1}{n}\sum_{i=1}^{n}(FK(\hat{y_i}) - FK(y_i))^2, \tag{4.8}$$

where n is the number of samples, $\hat{y}$ is the model prediction as shown in Equation 4.4, and y is the label. The position has to be computed from both the ground truth joints and the predicted one, as the inputs to the model are positions in a different reference frame to the predicted joints.

To further test the accuracy and explain better the differences between the positional and joint loss, per joint average can be computed, using the same formula from Equation 4.9, where instead of positions, grouped joints are substituted. Alternatively, the absolute positional error

(APE) can be computed to understand the distribution of predictions better, with the following formula for each point (x_i) in a particular joint:

$$APE = (x_i - f(x_i)) \tag{4.9}$$

Per joint error can provide insight into which joints have been predicted well, and which ones hasn't. It is important to understand, as differences in the joints closer to the base will have a larger overall impact on the final location of the end-effector in comparison to joints closer to the end-effector itself. Research in the Chapter 2 has also depicted testing accuracy by mapping a trajectory and predicting what the outputs for the network. Mapping the trajectory shows the method's tracking capabilities on previously unseen data. It can be done for both the orientation and the position. A helical function can be used to present a possible circular trajectory within the workspace of the manipulator:

$$\begin{aligned} x_h &= a \cdot sin(t) + d_x, \\ y_h &= b \cdot cos(t) + d_y, \\ z_h &= c \cdot t + d_z, \\ t &= [0 : 2\pi]. \end{aligned} \tag{4.10}$$

In Equation 4.10, t represents the timestep within the full circular range. x_h, y_h and z_h represent the parametric components of the function. a, b and c, are used as scalars to put the function within range (i.e. define the radius and the height). Those constant values were determined experimentally, in order to fit the function within the workspace, but also cover a large portion of it. As the center of the function is defined around the origin, the function has to be shifted by d_x, d_y and d_z to be placed within the manipulator's workspace.

Similar trajectory can be composed for the orientation of the end-effector, to test how accurate it is. The trajectory (o) of the manipulator can be defined as follows:

$$\begin{aligned} o(t) &= sin(t), \\ t &= [-\pi : \pi], \end{aligned} \tag{4.11}$$

where t is the range of motion of the end-effector projected with a sine function for better visualization of the angles.

Speed To test the speed of the system and the reaction speed of the manipulator, Equation 4.12 can be used. The speed difference (Δs) can be computed by finding the difference between speed of the manipulator (s_m) and that of the arm (s_a). The speeds are calculated by dividing distance (d) over time (t).

$$\Delta s = s_m - s_a = \frac{d}{t_m} - \frac{d}{t_a}. \tag{4.12}$$

Depending on whether the output is positive or negative indicates whether the manipulator was faster or slower and by how much, respectively.

Range Finally, the range or workspace of the predictions can be tested by considering which manipulator-reachable points are reachable by the predictions made with the ANN. The range of motion of the manipulator is know (as shown in Appendix A), and ergo, a subset of points of the sphere range of the manipulator can be taken. Discretization [51] can be used to divide the continuous space to which the manipulator can reach and pass it through the model to test how closely can it get to a particular point. To simplify the calculations however, each point can be verified as either within the range of motion or outside the range of motion, and thus accuracy can be computed. This should be done for both minima and maxima. This was depicted as the Workspace Index (WSI) in Chapter 2, and can be computed by dividing the number of correct predictions (n_p) by the number of points being evaluated (N):

$$WSI = \frac{n_p}{N}. \tag{4.13}$$

The closer the Equation 4.13 gets to 1 the better the desired operational volume of the trained model. However, firstly the workspace sphere has to be computed and points from it sourced via a point cloud. The parametric function to describe the sphere is depicted below:

$$\begin{aligned} x_s &= r \cdot cos(u) \cdot sin(v), \\ y_s &= r \cdot sin(u) \cdot sin(v), \\ z_s &= r \cdot cos(v). \end{aligned} \tag{4.14}$$

In Equation 4.14, the parametric function defines a 3D sphere, u and v are the ranges of the sphere, and r is a constant radius of the sphere, defined as the maximum reach of the manipulator

(same as the one used for calibration in Section 4.4). As the arm only operates within half the sphere in the y-axis (see Appendix A for the joint limits), the following ranges summarize the constants:

$$\begin{aligned} u, v &= [0 : \pi], \\ v &= [0 : \pi], \\ r &= 0.38. \end{aligned} \tag{4.15}$$

4.4 Additional Features

When the system was completed and working, there were some additions that were added on top of it. Firstly, calibration was added in order to automate the scaling of the manipulator (NFR06, FR09). Secondly, wrist angle was added (FR10). Finally, the files were organized into a ROS launch file, logging and documentation was included, and visualization (NFR10, FR14, FR15).

Calibration Initially, calibration was considered as an extended requirement, however, it proved to be necessary for computations described in Section 4.2. The calibration in this context is how maximal reach of the user's arm corresponds to the maximum distance of the manipulator. The most simple way to get the calibrated distance is to calculate the euclidean distance from the shoulder point to the wrist point. The formula for distance (d) between two vectors $\bar{X}$ and $\bar{Y}$ is

$$d(\bar{X}, \bar{Y}) = \sqrt{\sum_{i=1}^{n} |\bar{X}_i - \bar{Y}_i|^2}. \tag{4.16}$$

Furthermore, when testing in Gazebo, the maximum reach of the manipulator can be calculated by placing it as far as possible parallel to the ground. It roughly comes to around 38 cm. Then to calculate the scaling the two can be divided to get a scalar which is applied on the wrist vector. In the first software iteration, the calibration was initialized by pressing a keyboard key. However, it makes it difficult for the user to keep a straight arm and press the keyboard. Likewise, it makes it difficult to stop keypoint collection if the user has to move to get to the keyboard. Therefore, gesture recognition from MediaPipe (specifically by using MediaPipe's

GestureRecongizer) was implemented into the system to make the control easier. To control the system the following commands can be used:

1. Thumbs Up (Calibration): during the initial phase the keypoints won't be recorded until a Thumbs Up is recorded
2. Thumbs Up (Collection/Control): once the keypoint collection starts if Thumbs Up is detected the program will save the keypoints or the control
3. Thumbs Down (Termination): if Thumbs Down is detected, the program will terminate and not save the keypoints

Final adjustment made to calibration was a timer. Once the Thumbs up is detected, a countdown is started for the user to adjust the arm. Without it, often-times there were no keypoints found in the frame when the calibration starts which would terminate the program. When the countdown elapses, the maximum length of the arm is taken. It is suggested to the user to extend the arm fully and move it slightly around to find the highest reachable distance.

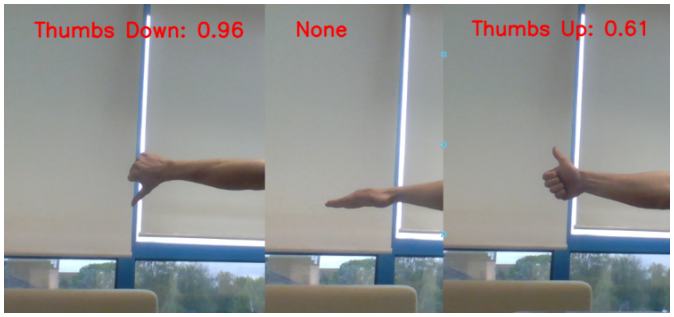


Figure 4.15: Gesture Recognition with MediaPipe

Wrist Angle Along with the manipulator joint movement, there is the gripper orientation that needs to get calculated (fourth joint, γ_4). The rest of the joints are computed from the location of the end-effector, but the wrist angle is required for computing the last one. The rest of the pipeline is not affected, as it only needs to be appended onto the locations. The most important thing is putting it in the correct range from the wrist input (see Appendix A).

Hence, the only change that has to be done is the actual calculation of the wrist angle. The current implementation for it is to calculate the vector from the elbow to the wrist and align the gripper with that vector. Figure 4.17 shows that it can be done by shifting the elbow to the

origin (along with the wrist), and calculating the angle between the vector and the x-axis.

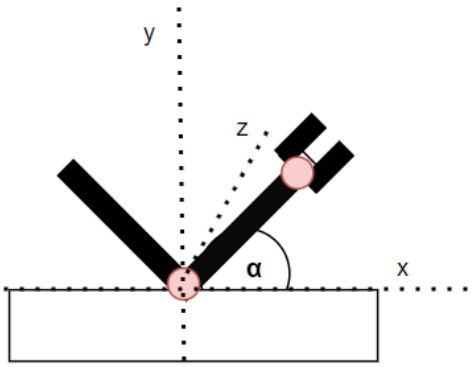


Figure 4.16: Sketch of Gripper Angle Calculation

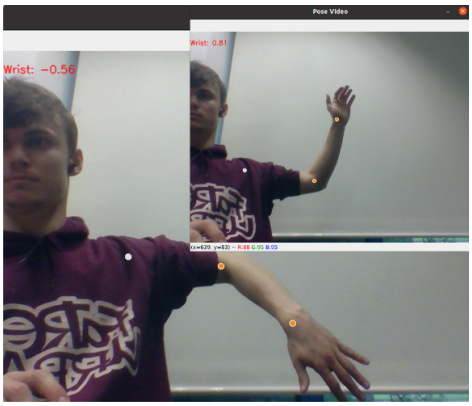


Figure 4.17: Wrist Angle Debug Visualization

The formula for the calculation of the angle is the following:

$$\theta = cos^{-1}(v_x \cdot \frac{V}{\|\|V\|\|_2}), \tag{4.17}$$

where the angle θ (see Section 4.3.2) is computed by taking the arccos of the dot product. The two vectors are a unit vector along the x-axis (v_x) and the unit vector representing the aforementioned vector between the elbow and the wrist V. Thus the output gives the angle presented in Figure 4.17. The approach is not ideal, as the movement of the hand is not considered, and thus, makes the accurate movement difficult without changing the position of the manipulator. Possible adjustment in the future is to use the hand keypoints to calculate the normal of the hand and compare it to the calculated vector from the shoulder. That way the angle between the two can be established to get a more accurate gripper orientation.

ROS Node Setup Finally, during the development user experience was highly considered. As during the creation of the control pipeline, I had to personally use the system, there was a lot of things that made it easier. Firstly, all of the subsystems were divided between different ROS launch files. With the ROS launch file, the system can be easily run with just a single command. On top of it, the external libraries that are being used, including Gazebo, RVIZ and the controller library, they can all be started inside of the ROS launch file. This simplifies the process massively, as all of them do not need to be run separately any longer, i.e. it is all done with a single command. This follows the typical package creation within ROS, and it makes it

usable by other users.

With the launch files, there is an option to add environmental variables. All of the flags and options that a user can change (such as file paths, training parameters and more) have been added. Therefore, all the subsystems can be simply run with one command, and within that command all of the environment variables can be set. Each one has a documentation and a default value associated to it, as shown below:

```
Optional Arguments:
  NAS (default "false"): Run Neural Architecture Search {boolean}
  NAS_iter (default "10"): Number of variations to run NAS [10] {int}
  dataset (default "keypoint_dataset.pickle"): The name of the dataset saved in data/ {s
tring}
  epochs (default "100"): Number of epochs the model will be trained for {integer}
  evaluate (default "true"): Run evaluation {boolean}
  initial_lr (default "1e-1"): Initial learning rate for training the model {float}
  loss (default "huber"): [huber (default), mse] {string}
  model (default "joint_predict.pt"): The name of the model to be saved in data/models/
{string}
  plot (default "false"): Plot training loss {boolean}
  scheduler (default "None"): [None (default)] {string}
  validate (default "false"): Run validation with positional loss {boolean}
```

Figure 4.18: Documentation Summary of Environmental Variables for the Training and Evaluation

The flags that were added into the environmental variables, now allow the users to run debug options, which mainly includes displaying the video, graphing things in Gazebo and RVIZ. All of the user processes have been summarized in the *README* file, and within the command line the logs are printed to guide the user.

4.5 Challenges

Throughout this system development there were various challenges encountered. Some of them slowed down the process for a relatively long time, which slightly delayed the initial scheduled plan (shown in Figure 3.10).

The initial struggle when creating the project has come in form of learning ROS. This project required an in-depth knowledge of using ROS and its libraries. This took a long time, and has affected the creation of the system throughout the implementation, as the best system possible was sought.

Secondly, the lack of the in-depth knowledge in the early stages coupled with the problem of mapping the keypoints to the manipulator has caused further issues. Different implementations were attempted to map the keypoints. The initial approach was using a more direct approach than the one presented in this Chapter (transformation matrices). The approach used the simulation and a control library, but it was very slow and most of the keypoints would not be predicted as the arm would fail to reach a plausible solution. Without this section the entire project was on hold as the rest of the pipeline dependent on it. Designing the new solution was not easy either, as it required communication between a C++, the C++ had to communicate with the IK solver library, and then send a message back to the Python script. Without prior experience in using C++ in ROS, implementing the algorithm that works with the libraries caused a lot of unexplainable errors that were difficult to debug.

Nextly, even though an IK solver implementation was being used, there was a lot of problems that came with it. There are a couple available for the OpenManipulator-X (that do not require remaking them), and they all had different issues. One would not convert any of the positions, and would simply return errors. The other one, would not classify over half of the dataset. The best one was an IK solver which used the chain rule and the positional Jacobian. However, it still has a problem when the y-axis moves away from zero or when the maximum reach is approached, even though it is still a valid position. A workaround was added for the first problem (as it was crucial to move it in all the spacial directions). As only one joint for this manipulator can rotate around the y-axis, the computation can be done manually (following Equation 4.17 with different unit vector). This of course wouldn't work for different manipulators, but different ones would be using different implementations of IK solvers, which means that the rest of the project can be completed, without losing sight of the goal.

In addition to all the problems encountered with the development of the system, from organizational and personal perspective, there was a lot of other workload with other modules. And thus, the work was split, so that this project was consistently worked on for a period of time, but then it would not be worked on for a while. This was mainly due to personal struggles with constant context switching. The schedule and plan proposed in Chapter 3 did not fully account for this, and thus, it affected the amount of work produced. However, diving the work into sprints and having constant list of tasks to implement has helped to keep consistent with the work and stay on top of it.

Finally, despite this project is predominantly a combination of software engineering and ML, there are still relevant concepts from Robotics that had to be explored. When reviewing the literature and research associated to the robotic movement, many of them were highly technical, which made them hard to follow and fully appreciate. This meant that some of the information that could have been useful for this project might not have been considered. Chapter 6 considers the Professional issues associated to this.

Chapter 5

Testing & Evaluation

5.1 Introduction

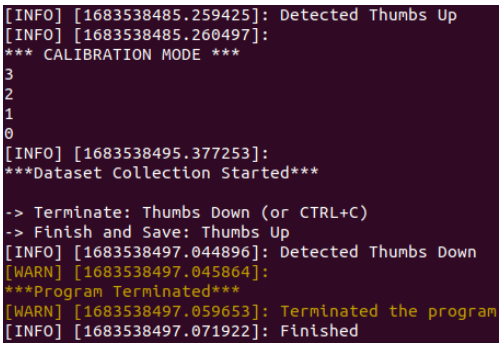
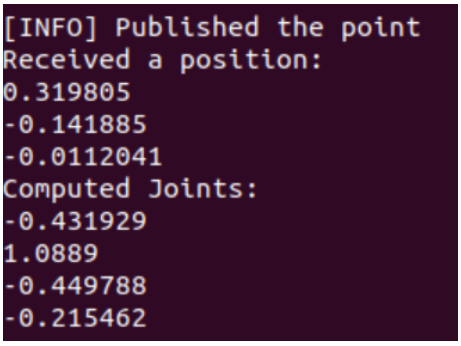
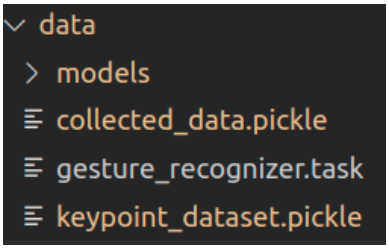
With the system completed, system testing against functional and non-functional requirements (from Section 3.3) has to be performed to verify the success and usability of it. Secondly, the model performance has to be evaluated against criteria defined in Section 3.4.3. Consequently, the IK solver and the trained model can be compared in performance and usability in the control pipeline.

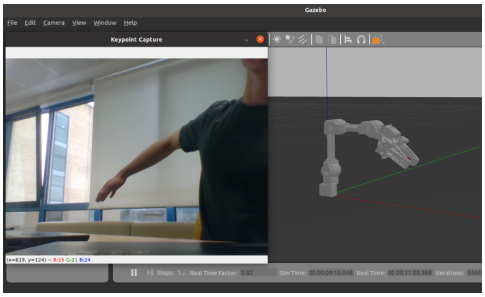
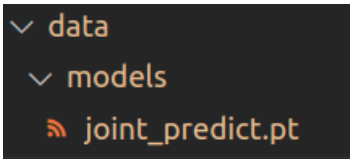
5.2 System & Requirement Testing

There were a handful of key and extended requirements that were identified for this project. The success of the developed software can be analyzed in-depth with it.

5.2.1 Functional Requirements

The functional requirements have been depicted in Table 3.1.

ID	Expected Outcome	Actual Outcome	Pass?
FR01	Getting the camera feed information into the system.	Figure 4.11 shows that the camera input is processed by the pipeline. Without it the rest of the system would not function.	✓
FR02, FR04	Recognize the person in the frame and detect the relevant arm keypoints (shoulder, wrist, elbow) which are used for tracking the motion.	The keypoints can be seen in the displayed frame in Figure 4.2, and the keypoints being processed can be seen in the snippet of the data being saved as a pickle file. The keypoints saved have 3D coordinates associated to them which can be used to detect the motion.	✓
FR03, FR09	Detect hand gestures and use them to trigger data collection.	 The logs show modes being changed after detecting a gesture for thumbs up and down (whole data collection and control are automated with gesture recognition). Similarly, Figure 4.15 showed the gesture detection.	✓
FR05	The keypoints are used to mathematically compute the corresponding location of the manipulator.	Figure 4.5 shows the location of the manipulator that was converted from the keypoints. Equation 4.3 shows the transformation applied.	✓
FR06	Solving for the joint angles, based on the manipulator location, with an IK solver.		✓
FR07	Saving the collected dataset as a pickle file so that it can be loaded for training.		✓

FR08	The moving parts are visualized in RVIZ for debugging, where the arm keypoints and the projected manipulator can be seen.	Figure 4.5 graphs the transformations between the arm and the manipulator, having a useful graphic to check whether the motions from the arm correspond to the motions that the manipulator did, and helps to explain why.	✓
FR10	Moving the end-effector angle according to the wrist orientation.	 The actual output does not entirely match the predicted one. As it was described in Section 4.4, the algorithm does not use the hand orientation, but instead it uses the vector from elbow to the wrist. If the motion was included in the above Figure, the control of the wrist would get very difficult when keeping the location.	✓ / ×
FR11	Saving the ANN model after the training.		✓
FR12	Moving the manipulator based on the user's movements.	Figure 4.11 showcases the manipulator being moved via the arm movements. The control itself is further tested in Section 5.3.	✓
FR13	Opening and closing the gripper when the user gestures an open/closed hand.	This was not attempted due to the lack of time, and the key features were a higher priority to complete. The software contains the gesture recognition which was used for FR09, and could be easily extended in the future to recognize open and closed gesture.	×

FR14	Displaying the video input and keypoints to the user.	`roslaunch --ros-args pose_estimation training_data.launch`
		`Optional Arguments:`
		`  input_keypoint_file (default "collected_data.pickle"): Name of the input file with keypoints (from data_collection)`
		`  show_video (default "false"): Show the input video`
		`  simulate (default "false"): Simulate the motion in Gazebo` The Figure above shows the flag to activate the command (along with documentation for each of the variables). See Figure 4.11 for the outcome.
			✓

FR15	Log all the information, errors and warnings to the user in the command line.	Example of information being shown to the user can be seen when evaluating FR03. An example of the error can be seen below:
		`[INFO] [1683546673.696973, 0.000000]: Detected Thumbs Up`
		`[INFO] [1683546673.698208, 0.000000]:`
		`*** CALIBRATION MODE ***`
		`3`
		`[ERROR] [1683546676.426164, 0.000000]: Wrist has to be to the right of the shoulder`
		`[ERROR] [1683546676.545005, 0.000000]: Wrist has to be to the right of the shoulder`
			✓

Table 5.1: Functional Requirements Evaluation

5.2.2 Non-Functional Requirements

ID	Expected Outcome	Test	Actual Outcome	Pass?
NFR01	Process 1 sec of video in under 3 sec.	Python's time command can be used to measure how long it takes to process n number of frames, depending on the FPS rate.	At 30 FPS: **2.8 sec** for keypoint detection, **3.6 sec** for keypoints and gestures.	✓ / ×
NFR02	Converting positions into joints should take under 0.2 sec/point.	Again, time function can be used on a batch of points, and an average can be computed.	Original method (see Section 4.5): 2 sec/point. Proposed Method: **0.014 sec/point**.	✓
-	The data collection software should accurately convert the positions to joints.	Percentage of correctly converted points.	Original method (see Section 4.5): 40%. Proposed Method: **92.3%**. Synthetic data generation, further adds over double the points.	✓
NFR05	Environmental conditions should not affect the performance of the model.	Test whether the model can be controlled smoothly in different environments.	In general, the observation was that the manipulator arm can be controlled just as smoothly in different locations.	✓
NFR06	Accuracies between different users should not vary.	Take a sample of users that have different arm sizes, place them in the same room and measure the calibrated values.	The calibration was tested to be on average below **1%** difference between users, which can be considered negligible (as perhaps for some users a total maximum length was not found).	✓
NFR07	Data saved locally for replay.	N/A	For the saved data see Table 5.1 - FR07; the data is not saved externally.	✓
NFR08	Running the packages with one simple command and with all the environmental variables set with it	When running a script, check if anything has to be run in addition, or any variables needed cannot be set.	There are 3 ROS launch files, each one corresponding to each of the three subsystems (collection/control, keypoint processing and training/evaluation).	✓
NFR09	Control from NFR08 with GUI.	N/A	There is no GUI.	×
NFR10	The system and control should be adaptable to any hardware manipulator.	Plug in a different manipulator to collect the data and test the model	It was not explored due to time constraints. It would require a lot of work to adapt the code to a different arm with different libraries (further discussed in Chapter 7).	×
NFR11	There is no cost when running the system.	Note down the costs during the development.	The system can be run in simulation and on the hardware, thus users without robotic arms can still run the system and experiment with it. The minimum requirement for simulating it is a GPU.	✓

Table 5.2: Non-Functional Requirements Evaluation

5.2.3 Conclusion

To summarize, the system which was developed is a really good ROS package which is user-friendly and provides all the key functionality that was planned for. Most of the extended ones have been completed, but some more work would have to be put into them. Specifically the gripper controls could be improved by adding opening and closing, as well as including a better algorithm for control of the angle. To elevate it further, a GUI could be added, similar to that of the open_manipulator_control_gui.

The requirements NFR03 and NFR04, both are going to be evaluated in Section 5.3, as they directly refer to the performance of the model and the control of the manipulator. In general, the key requirements have been met, except for NFR01. The inital plan did not consider gesture recognition into the calculations. The frame drop does affect the experience where it is very hard to be used [1]. Alternatively, YOLOV7 could be used as it provides inference on a GPU, and the performance should be similar (see Section 2.2). Some extended requirements have not been explored due to the lack of time, and thus, no testing could have been done on them. However, an additional test was done to compare the original method with the implemented method on how many points have been converted correctly, showing the value of the implemented method.

5.3 Model Evaluation

An ANN was implemented as a regression model, to substitute IK which is slow and often fails to converge. This framework was designed with higher DoF in mind, where the IK can be a real problem. An ANN is meant to instead converge to a single point, by estimating a mapping/function between the inputs given and the joint angles predicted.

This was not a direct research, but instead existing research on the topic (see Chapter 2) was used to design engineer a solution, along with some new ideas such as NAS. The reason why this work cannot be directly compared is because it aims to engineer a solution that fits the inputs from the keypoint estimation, where all the similar research on the topic focuses on direct substitution of IK. However, having the system in mind first, transforming the points each time

[1] A fix was added for the demonstration which runs inference on gesture recognition only every half a second. It increases the frame rate significantly as it can be seen in the demo. Further tests and improvements can be added.

between the reference frames would create an unnecessary overhead for control. Additionally, the literature that was explored, contain predictions with different test sets, and most importantly with different numbers of outputs and inputs, due to different DoF being used.

5.3.1 NAS

Firstly, in order to select an optimal ANN for this problem, NAS search was used, and optimized for positional and joint loss (as shown in Equations 4.9 and 4.7). The results are depicted below in Figure 5.1, where the red points show the non-dominated models.

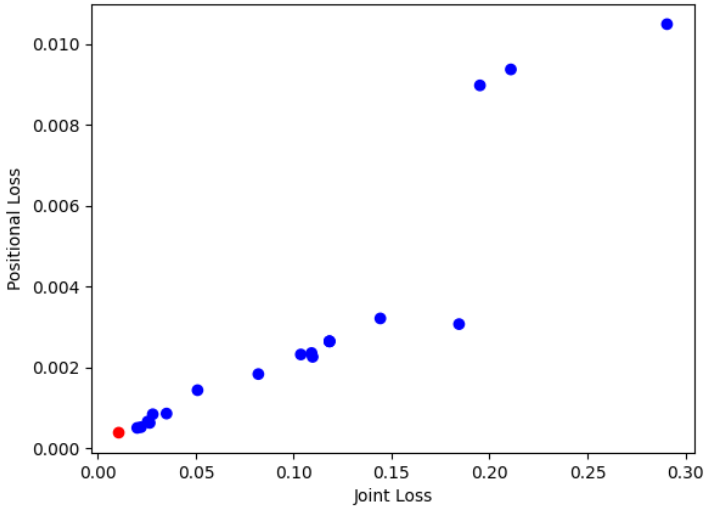


Figure 5.1: Pareto Front Results

Positional & Joint Loss The first and the most important thing that can be noticed, was that despite very different architectures being used (shallow or deep; wide or narrow; with different activation functions), there is a very strong correlation between the positional loss and the joint loss. The correlation is that the lower the joint loss, the lower the positional loss. The anticipation was that despite this, there would still be some networks that do not learn the spacial mapping very well and end-up focusing solely on the joints. There are a couple of exceptions where they are slightly skewed, most likely in the cases where the slight change in the joints has resulted in different configuration of the joints that by chance happened to reach closer to the desired destination.

Some of the literature discussed the differences between using the two types of losses, and how

they can differently affect the performance of the model, but it seems that with the given dataset, the better the predictions are in the joint space, the better they will be in positional space. Therefore, using the Multi-Objective optimization in this case did not yield any benefits, nor does using a Pareto front to find a best solution (as the best solution will be a minimum of either of the two error functions), other than providing insight into how the joint and positional error will be affected.

Finally, some architectures that were generated on one of the runs by NAS, can be summarized in the table below (where the obtained joint loss was used and multiplied by 100; on 50 epochs, step learning rate scheduler [steps of 10], and with an initial learning rate of $1e^{-2}$):

No.	**Architecture Layers**	**Activation Functions**	**Final Activation**	**Optimizer**	JE **(rad)** $\downarrow$
1	Hidden Layers: 100	ReLU	None	Adam	$1.6e^{-2}$
2	Hidden Layers: 24, 16	ReLU	None	Adam	$5.1e^{-2}$
3	Hidden Layers: 100	Sigmoid	None	Adam	$1.1e^{-1}$
4	Hidden Layers: 100	tanh	None	Adam	$7.9e^{-2}$
5	Hidden Layers: 128, 64	tanh	None	Adam	$4.1e^{-2}$
6	Hidden Layers: 128, 64	ReLU	None	Adam	$4.1e^{-2}$
6	Hidden Layers: 100	ReLU	None	SGD	$6.8e^{-2}$
7	Hidden Layers: 100	ReLU	ReLU	Adam	8
8	Hidden Layers: 128, 64	ReLU	None	Adam	$5.4e^{-2}$
9	Hidden Layers: 128, 256, 128	tanh	None	Adam	3.6^{*}

Table 5.3: Non-Functional Requirements Evaluation

Some of the findings were summarized below, including number 9 (*), which converged to a wrong position.

Activation Functions On the contrary, performing NAS itself has yielded helpful results on what the best networks are for this problem. Firstly, different activation functions have been considered, which affected the outputs of the model the most. The reasoning behind it, is that the range that the output is placed in, highly depends on which ones are used in the architecture. For instance, one difference could be seen between using a ReLU function and a hyperbolic tangent. In the literature that was analyzed (see Chapter 2, hyperbolic tangent is the usual activation function that is used for this problem. However, through the NAS analysis, it was discovered that this is not the case for the dataset being used. Below in Figure 5.8, the comparison between the predictions of joint γ_4 (i.e. the end-effector angle) can be seen for the two activation functions that were used with the same architecture (the graphs will be discussed

in the further sections in more depth). Afterwards, Figure 5.4 has the trajectory planning in 3D between the two.

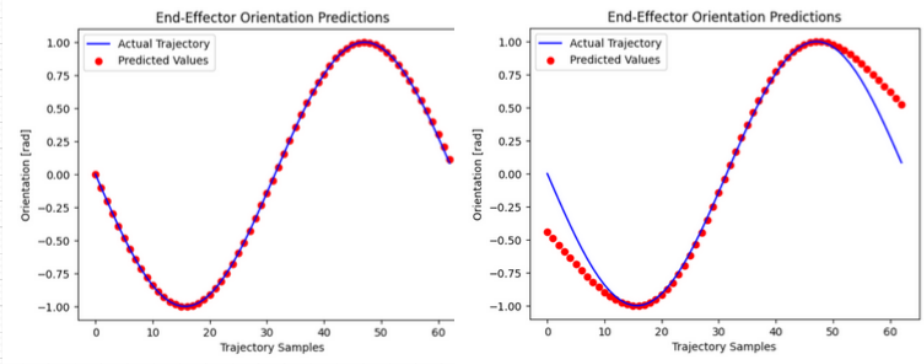


Figure 5.2: ReLU [left] vs. hyperbolic tangent [right] joint 4 predictions

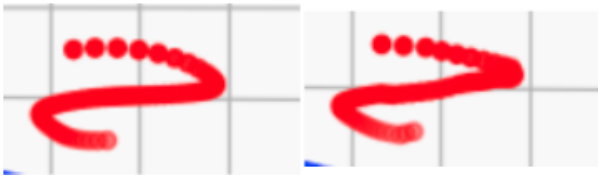


Figure 5.3: hyperbolic tangent [left] vs. ReLU [right] 3D trajectory track

The differences might not be massive, but if the goal to reach the model as precise as possible, ReLU certainly performs better (especially after looking at Table 5.3). Similarly, using no activation function can not predict the values of the joints correctly. The problem of predictions is not a linear regression problem, and thus, non-linearity has to be added to the predictions through the use of those activation functions. However, the final layer seems to require no activation function as then the values that get predicted are capped within a specific range, causing inaccurate predictions as shown below:

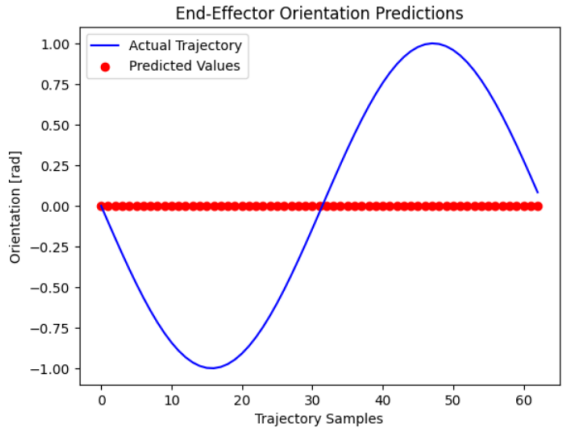


Figure 5.4: ReLU in the final layer

The main difference is that the use case for the neural network in this project is not to directly convert the manipulator positions to joints, but instead joints from the camera and keypoint estimation. Therefore, the architectures being used in the research can have different affects, due to different ranges of values used. To conclude, ReLU was the best activation function to use between the hidden layers, and no activation function in the last layer.

Architecture Complexity & Data Similarly, the depth and width of the network affect the model performance. Before diving deeper into the evaluation, it is crucial to mention that an issue was stumbled upon when running this. Some networks would have reached a low loss during training, however, it was because a single point in space (or a small subset) was found that minimized the mean between all of the points, and thus the network predicted all the points as that one (set). The explanation for that could be that the model simply decides not to do anything, as it learnt that doing nothing, might get a lower error than making a random guess. This is extremely hard to identify when looking just at the loss, as usually it looks as valid as the actual predictions. Implementing different evaluation metrics (see Section 5.3.2) has helped to identify those networks. This phenomenon was especially prominent in the more complex architectures, and thus, a conclusion was made that the task requires a less complex architecture. This Figure the convergence, where the data should be filling the whole sphere:

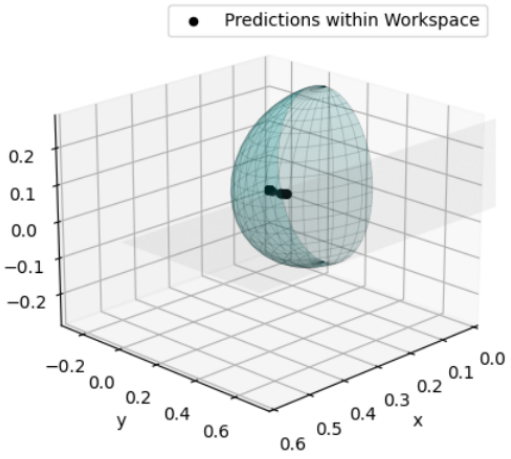


Figure 5.5: Converged Workspace of Complex Network (See No. 9 in Table 5.3)

Oftentimes, having a lot of bias and not a lot of variance in the dataset suggests the need for a more complex network. Ergo, it seems that the dataset is not too biased and is fairly well varied. Various techniques have been considered to create a good dataset. Sampling of the points from the full range of motion and with varied motions across all of the planes was done during collection. The data was normalized between the range of [-1,1] to help with the stabilization of the gradient descent steps. When splitting the data, a care was taken to shuffle it to get the best variety of the data, and so that the training set represents the test set fairly well. All the duplicates were removed to avoid skweing of the training. Finally, synthetic data was created to add variety to the dataset, especially with regards to the wrist angles (see Section 4.3.2). In general, creation of the dataset can be viewed as successful.

The best architecture that was found during the search based on the findings described above is shown below, and it very closely resembles one from Figure 2.2, except for different inputs and a different activation function.

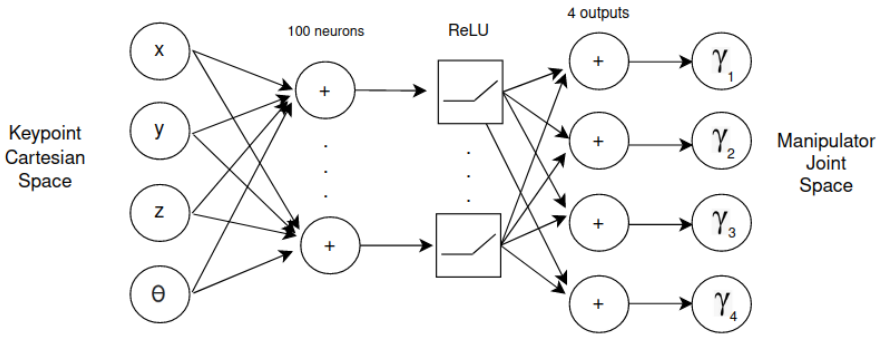


Figure 5.6: Sketch of the best generated architecture

Learning rate seemed to have an impact, but overall decreasing the learning rate looks like it makes the network converge more. Based on that, the new network was trained by decreasing the learning rate every 10 epochs with the step learning rate scheduler. Thus, if a plateau is reached within those epochs (which is very likely due to the small values that are being used for training), the drop in learning rate can aid it in getting past it and converge closer to the minimum. Finally, initially SGD was expected to perform better, but Adam seemed to converge to better values (see Table 5.3).

5.3.2 Model Performance

The model performance was evaluated based on the metrics discussed in Section 4.3.2. Table 5.4 showcases examples of the hold-out test set which includes the keypoint locations and the wrist angles as the inputs, and the angles of each joint in the manipulator as the output. It can be noticed that θ matches γ_4 directly, however it was important to incorporate it as different end-effectors can have varying DoF, and thus, it should be included in the research as for other manipulator's this would be less trivial.

No	x	y	z	θ	γ_1	γ_2	γ_3	γ_4
01	0.24	0.17	0.05	1.57	-0.63	0.62	0.34	1.57
02	0.16	-0.27	0.08	0.0	1.05	0.67	0.61	0.00

Table 5.4: Example of the Test Inputs and Expected Outputs

Accuracy Firstly, as per NFR03 the majority of predictions, should be *accurate* within should be within 1% of the maximum reachable distance of the manipulator. That means that the

desired positional error is lower than 0.38 cm. The summary of graphs in Figure 5.7 shows the actual and predicted values of some of the data points in the hold-out set for the three joints. The predictions are very closely matched, especially in the base joint, which is responsible for rotation around z-axis (yaw). The other two have some errors, but for the most points, the small change in the predicted value, should not have that much of an impact on the final position.

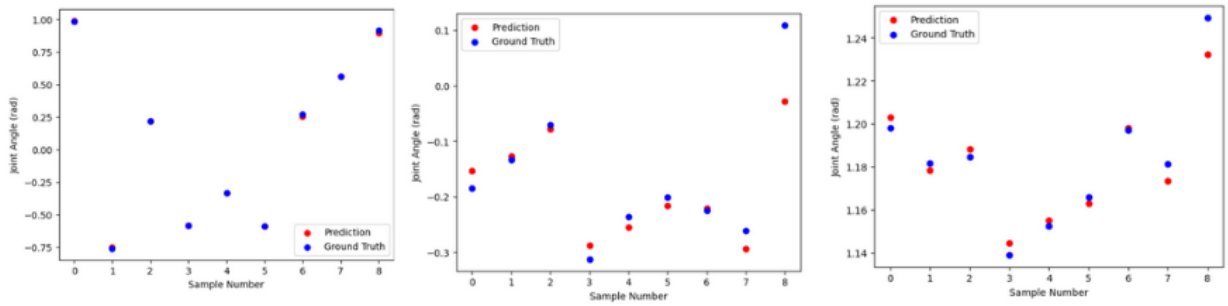


Figure 5.7: Sample of joint predictions in test set for γ_1 [left], γ_2 [middle] and γ_3 [right]

The overall training can be summarized in the following graphs, which compare the difference between the MSE and Huber loss, and shows that they both converge around the same epoch:

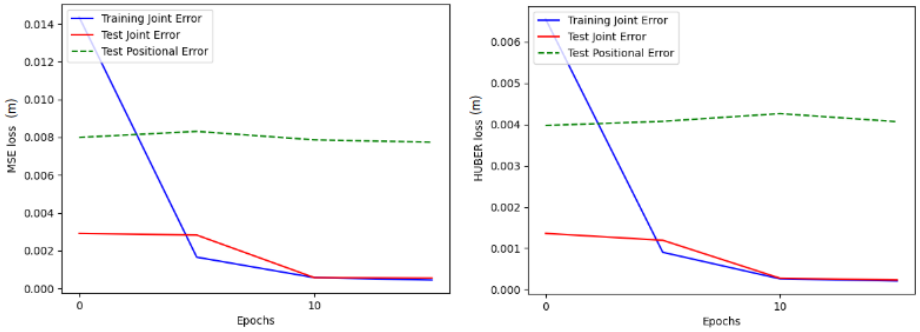


Figure 5.8: MSE [left] and Huber error [right] across the epochs

From the above-shown figure, it can be entailed that the training between the two looks very similar, but Huber loss must deal better with outliers, and thus the error goes down further. Considering the positional loss, on average the error is around the border for a pre-defined successful accuracy (about 0.38 cm). Overall, with better predictions of the joints, the positional loss does not seem to change much at all from the first epoch. This could suggest that rather than the training, there could be some bottleneck with regards to the data provided. Therefore,

further analysis has been conducted to evaluate it.

A more in-depth analysis for each of the joints (see Figure 5.9, suggests that the mean for all of them is somewhere around zero. The first joint seems to have performed relatively well, but there is a relatively large spread between the data, where the predictions can be a large way off. That would result in the locations being rotated away from the desired output. The following two joints have a much smaller spread and more centered around the mean, however, the mean seems to be slightly further away from zero, which is not a good thing. Especially joint 3, where the model is confident in the predictions, but it is almost always off. That results in the x-axis reach being affected.

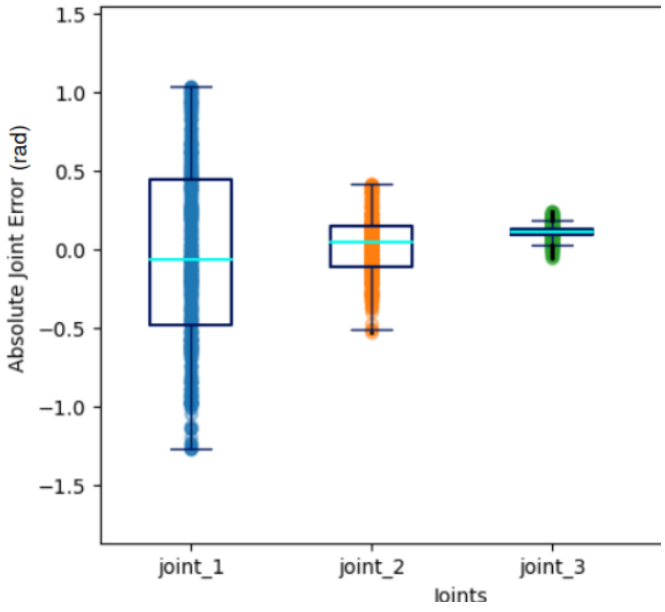


Figure 5.9: Box of Absolute Joint Error for all joints

Taking the end-effector joint through the full range of motion, without taking into account the locations (i.e. keeping them constant), the predictions very accurately map the desired trajectory (see Figure 5.11). Likewise, the locations were evaluated based on the helic function from Equation 4.10. The metric came from the Literature review, but it does not entirely fit the need. The helic function has to be described in the "Keypoint Cartesian Space" (see Figure 5.6), but the the joints are predicted in the manipulator action space. Therefore, when an FK solver is used to convert the joints to the positions, they are in the manipulator action space. Therefore the two are not directly comparable, as it can be seen in Figure 5.10, where they are in separate reference frames. The most important finding is that the trajectory is roughly

followed.

Unfortunately, for this use case, this might not be the best indicator, and neither is the one from Figure 5.12. Both struggle with a similar downfall, which is that the sampled points might not be directly usable when converting between the two reference frames. The best way would be to map a trajectory with an arm, collect those points and then predict them. That would be directly applicable to this project, and yield more benefit. However, even though the graphs cannot be directly compared like those from Figure 5.11, the shape is still roughly followed, and it seems that small changes in the input still result in the trajectory, indicating that the *precision* of the predictions is good.

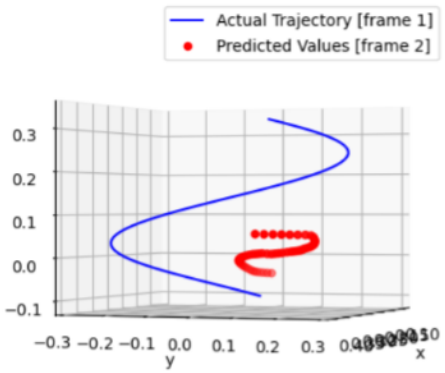


Figure 5.10: Trajectory of Predicted Locations

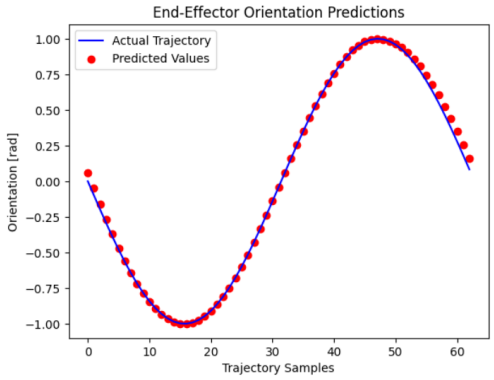


Figure 5.11: Trajectory of the end-effector joint

Range of Motion Similar, but more prominent, problem was discovered when trying to plot the workspace of the manipulator, by sampling the extreme points from the full sphere. As Figure 5.12 shows, the predictions once again do not match with the sphere due to different frames of reference. The predicted points somewhat shape out the half sphere, but not accurately. There is also another explanation for the poor performance on this metric. The extreme points, have often been not considered by the IK solver. So, even if the point is technically reachable by all the joint limits, the IK solver might not find a solution for it. This results in a not full dataset, and thus, the points would tend to converge closer to the center, where more data points have been sampled. This could be tackled by synthetically creating more data on the extremes, but that would be more difficult than the angle rotations, especially to be done accurately. Alternatively, as it will be evaluated in Chapter 7, an IK solver could be implemented to better fit the need here. And similarly to the trajectory, it would be better to do usability testing for verification.

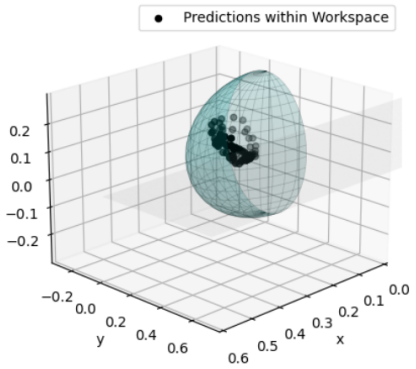


Figure 5.12: Workspace predictions

The WSI was meant to be computed, but in this case it will be equal to 1, as all of the points are within the range of the manipulator. That entails that all the predictions should be within the defined joint limits. Overall, the NFR04 was not satisfied as the predictions do not reach the desired range of motion.

Speed & Control The model performance was mainly evaluated through the aforementioned methods. They were initially preferred over usability testing, as they can show empirical results, and are easier to evaluate. Nonetheless, some of the evaluations that tried to go deeper than just the error evaluation have proven to not as fruitful as anticipated. The problem arose with the difference in reference frames, which is a crucial part of the system. Therefore, not much usability testing was done. The main one was the speed at which the manipulator can be controlled at.

Before implementing the ANN, the IK solver was used to control the arm. The speed at which the predictions were made were roughly every 2 seconds, as the solution has to be computed among the infinite joint configurations. This problem scales as the DoF increase, and thus, for different manipulators that will be even slower. Moreover, the transformation from the keypoints to the manipulator has to be computed on the fly to get the joints. The whole system does not work very well, or efficiently and there is a lot of errors. The major problem however, is the lag, where it influences how well the arm can be controlled, updating every 2 seconds. There is no precision in the movement, and it can not be claimed that it imitating a human motion. Updating the joints directly takes up significantly less time and the manipulator moves smoother. However, as FR01 was already elaborated, the speed at which the manipulator is moved is largely affected by the big lag in the system, which should be taken into consideration

in the future improvements.

Unfortunately, even though the evaluation has shown promises, deploying it did not work within the timeframe of the project. The wrist moves correctly, and y-axis somewhat works. However, the motion in the x-axis does not work. Due to a time constraint it could not be evaluated in detail, but the two main possible culprits are either the data that is being used, or the regression model. In the future, usability testing should be done earlier and possibly unit testing to better verify that all functions work as expected.

Chapter 6

Statement of Ethics - Legal, Social, Ethical and Professional Issues

During the development of this project legal, social, ethical and professional issues have been considered, and the appropriate codes of conduct followed to complete this project. The SAGE form has been completed, and the considerations analyzed with the supervisor, to assess the ethical issues associated with the project.

6.0.1 Ethical Issues

There are several ethical considerations that have to be taken into account when designing a software system. The main area of concern in software systems is with regards to data, and more specifically, personal data. The ethical concerns that were taken into consideration during the development include:

- **Do No Harm** - the research being done should not cause any harm to anyone. Given that the project is done entirely in simulation, entails that there is no physical way that the development of this project can hurt anyone. If the system was deployed onto hardware, as per initial plan, that could be unethical if the robotic arm was tested in an unsafe environment, as it could have unpredictable movements. Furthermore, using the final model in an application (for instance in medical or manufacturing setting), could lead to some big harmful impacts if the system malfunctions (further discussed in Section 6.0.2). Finally, The Computer Misuse Act (1990) does not specifically apply to this project, as

the system does not provide any ability for an unauthorized access to impair a computer's operation.

- **Informed Consent & Voluntary Participation** - the research only used a handful of participants to test the control of the system. The participants agreed to it and have been briefed on the fact that the camera will not record them. Furthermore, they did not have to provide any of the details, as their personal information does not matter in the context of this project.

- **Confidentiality & Anonymity of Data** - the system has an option to record the movements and save them for later replay and visualization of the movement. However, that is a feature that can be turned off, and the recordings are not saved anywhere externally or uploaded to the cloud, but they are saved locally. The system only uses the minimal amount of data from the users that it has to, which in this case is the joint locations in the 3D space, which are not considered personal data, as they cannot identify a user, and furthermore, that data is not saved anywhere other than the local hard-drive (and the data can be very easily deleted). Therefore, the data processing follows The Data Protection Act (DPA) 1998 [52], where only relevant data, without any excessive information is stored securely, and which can be deleted when no longer used for training.

6.0.2 Social Issues

When the system was being created, the social impact of it has to be considered. As the project plan outlined, the goal of this system is to be a widely usable library which can allow control of robotic manipulators. This project can have a positive impact on the society, as this research is applicable to a variety of different domains (including medical or industrial). The possible applications are further discussed in Section 7.

However, as mentioned earlier in Section 6.0.1 there are some social considerations that have to be also regarded when using such a system. Currently, the system follows the British Computer Security (BCS) Code of Conduct [53], where the project has to regard the health and well-being of the public and the environment. If the system was deployed in the future, it could cause physical harm to people (and the environment around) if there was some malfunction of control. Therefore, the best way to ensure these malfunctions don't happen, was through the testing and evaluation. Any harmful problems arising, would be unlikely, but as the plan outlined, if the

next step of deploying it would be met, it would have been done in a lab, which is considered the safest environment for testing it. Also, with the deployment of the model onto GitHub, all the results should be accurately shown and should not include any deceptions, for the users who will be using the model. In the extreme cases, the project could be used to cause harm (for instance in the military), however, that is highly unlikely with the current iteration of the project, and there is a lot of other research on the topic which would not yield extremely new large benefits for such cases.

Finally, according to the BSC Code of Conduct, the project should promote equal access to everyone in the society, for which this project attempted to do by accounting for different technical constraints that users might have, such as lack of access to robotic arms, lack of a GPU, lack of a good camera (see Chapter 5).

6.0.3 Legal Issues

When discussing Ethical Issues in Section 6.0.1, some legal issues have been touched upon with regards to the data protection and usage. The main legal concerns that had to be considered with regards to this project are with regards to the code being used for the development of this project. According to the Copyright, Designs and Patents Act 1988 [54], the creators (in this case of software) has the control of how their work is being used. In that regard, Copyrighted works restrict the copying and reproduction of work. With regards to the the data, there is no external dataset being used, and thus there is no infringement on IP in that case. There are various software tools and libraries being used by this project, that were not created by me. However, all of the libraries and software used are openly available libraries, and all of them have been cited accordingly, to not be passed off as part of the created code for this project. To further solidify it, the libraries have been removed from the GitHub, to not give off assumption that the code has been created there, instead they are recorded as requirements needed to run the project. Therefore, there has been no plagiarism or copyright infringement committed.

Furthermore, according to the IP Code, as part of the University Code of Practice, no confidential information will be disclosed when publishing the work, as there is none. The code being free, and publicly available, on GitHub, there is not much IP regarding this system. The library and the model trained will be available to anyone to be used within a ROS pipeline. Different people can use it and attempt to make improvements to it, to drive the its development forward,

however, if they were using it for their own projects, they would have to cite it and acknowledge the creator, so that this work is not plagiarised.

6.0.4 Professional Issues

In order to identify whether the project has had any Professional Issues, the *Duty of the Profession* and the *Duty of Relevant Authority* in the BSC Code of Conduct [53] can be used. Some of the following pointers have been considered:

- **Value different alternatives suggested, as well as, seeking and accepting honest evaluation of the work done** - it was important to act with integrity and respect towards the people that wanted to help me with my work. The feedback on the work, especially from Dr. Mendez Maldonado, has been extremely helpful and sought out, even if sometimes it resulted in having to recreate a lot of the system. But it was important to take criticism and accept the flaws of the system, which lead to more professionalism, and in turn, better project.
- **Upholding the reputation of the profession** - when developing the system, all the knowledge gained across the undergraduate degree with regards to Software engineering and Machine Learning has been used for this project, in order to ensure the project is up to the best standard. Furthermore, the most relevant and up-to-date research was considered, to ensure that all the decisions made were justified.
- **Improvement of professional standards and helping fellow members with their development** - the aim of this project was to create a project which can aid in future research in the topic, and throughout the development and testing, the goal was followed.
- **Undertaking work within the limits of professional competence** - the topic chosen was a challenging endeavour as it combined different things that have been explored throughout the undergraduate degree, but also have been combined with a different topic (robotics) for which I needed to learn a lot. Therefore, this project is still predominantly a software engineering and machine learning project, which uses a robotic arm for evaluation. Not being an expert in the field of robotics, would make a research project solely on that topic highly unfeasible. However, the project has allowed me to significantly grow and learn, as well as discover various research area on this topic.

Chapter 7

Conclusion

The general outline of the project is that; a system was developed which allows for a control of manipulator arm in simulation through a ROS package. Initially, an IK solver was used, but due to some limitation outlined before, an ANN was considered as a good substitute. Therefore, along with the system, a framework for creating the model for control was developed.

Evaluating Project Aims The main objective of the project was to develop a pipeline that can be used for collection of data, training a neural network related to the conversion of the joint locations to the angles of the manipulator arm, and finally, to control the arm. The following list summarizes the evaluation of the objectives:

- The system which was developed is **user-friendly**, as it incorporates a large number of utilities provided by ROS. Those include the visualization systems such as Gazebo or RVIZ to visualize to the user and easier debug the collected keypoints and their effect on the manipulator. The whole system is wrapped and accessible within a ROS package, and can be activate via the command line and roslaunch files where only one command is needed, and all of the variables that are editable by the user are contained within it. Similarly, the system is **low-cost** as there is no need to have a robotic arm in form of hardware, but it is enough to run everything in simulation.
- The system was created to **gather the data** based on the human motion. It incorporates a MediaPipe library to performs very well with collection of the keypoints and predicting their location in 3D space. The data collection system incorporates useful features, such as calibration (for more accurate comaprison of the collected keypoints to the manipulator

arm) and gesture recognition (for aided user control). Both of which were initially just considered as additions.

- The collected keypoints have been successfully **converted into the manipulator space**, which paved the way for the research into controlling the arm with a neural network. At this stage the manipulator arm could already have been moved with IK, but as pointed out in the Problem Analysis, better performance was sought.

- The culmination of the project was to use the system as a use case for **implementing an ANN** from researched literature, and engineer a **control of the manipulator** without needing the IK solver. An IK solver was first implemented and tested to serve as a baseline. The model was then trained and evaluated, as per the aims outlines, and during evaluation it provided good results across the board of different evaluation metrics.

  Unfortunately, when it was deployed in simulation, something seems to have gone wrong. Some debugging was done, but due to time constraints it could not be explored further. First hypothesis is that the data collected from the IK solver might not be entirely correct for the two joints associated to the x-axis. This would cause the model to perform poorly, and give a false sense of good predictions. Up to that point, all the points were visualized, and the whole range of motion should be included in the dataset. The best hypothesis is that there is discrepancy between the reference frames, where a wrong one is fed somewhere, which could cause the whole system to break.

  The other possibility is that the network architecture does not suit the problem well. Some possible future improvements were discussed in the following sections to possibly improve this. More usability testing should also be done to understand what is happening. In summary, the system provides a control, that worked fine with an IK solver, but the network could not be deployed and contrasted with the IK solver. The only thing that could be compared was the speed, which did improve and make the motions smoother, but first the control with the ANN has to be fixed for it to be usable. Therefore, this success criteria was not met, even though a control is still possible with the code implementation (but not through a ML method).

- One aspect which was on the border of being in the scope from the start (but was still considered as one of the less primary objectives) was deploying the software on the hardware. This feature was not attempted due to the interest of time. There was a possibility

to do so with an IK solver and test the control on it. That would most likely lead to better usability testing of the system. However, a decision was made to implement a ML model.

To summarize, the first major aim of developing a strong pipeline and a control system through CV was achieved in this project. The only major thing which was missing for this aim was better usability testing. However, a deep evaluation of the results was performed outside of the control framework, which was still useful. On the other hand, developing a ML algorithm to map the inputs to the manipulator was not achieved fully. A model was trained, but does not work in deployment. More time has to be spent on that to make it work. Furthermore, the task of applying this research to higher DoF would have to wait until the better model is developed. In Chapter 7, future improvements will be considered, specifically talking about how the research can be carried on, and improvements that can be made.

Contributions One main problem with this project is that initially it was supposed to be a research project. Throughout the design it can be seen that both system development and research into possible applications is being considered. For that reason, sometimes there is a discrepancy between what should be prioritized. Nevertheless, there was always one major goal in mind, and that was to control the manipulator arm with CV.

Despite the control with the manipulator not working anything like one from the research by Sivakumar et al. [3], there is still a major positive outcome of this project, and that is the system built for it. With the pursuit of trying to do research, one thing constantly in the way was the lack of good solution to do it. That drove the whole system forward, and made it good for debugging, training, and controlling. The control only requires a network that will return joints, so any model can be plugged in. Overall, this project has contributed by using the tools and research available to create something that can be used in the future for more on this topic, which was not available before. It also provides a lot of additional features, and makes it easy to build on top of it and work with it, just as any good ROS package should.

Future Work & Improvements There have been various possible improvements suggested throughout the evaluation of the project. Improvements that could be added to the system are better control of the end-effector through the angle joint (it can be easily added now that gesture recognition has been implemented). Similarly, a new gesture could be added to control the opening and closing of the grippper, to make it applicable into different use cases.

Secondly, the speed has been a big part of this project. In order to improve the processing speed of the control pipeline, the bottleneck is the gesture and pose recognition inference running at every frame. Alternatively, a GPU compatible ones could be run to speed the process up (such as using YOLOv7 identified in Section 2.2). The bottleneck could also be decreased, if the gesture recognition does not run on every frame, but instead every couple of frames. That could decrease the overload by quite a bit, and would not affect the usability greatly. Similarly, when controlling the arm (not collecting the keypoints), where a lag could be impact, the keypoints could be sampled every couple of frames, as the setting of the joints cannot follow every frame, which could only slightly affect the precision.

The data could have also been better, with the IK solver not being ideal when it comes to converting the points. The transform could have possibly also been prone to error, due to some expected errors when computing the transforms and calibration, and thus affecting the predictions. The best approach in the future would be to use a better IK solver, ideally one built from scratch. That would make debugging easier, increase the usability of it, and make it more flexible to use with other DoF. Likewise, the system could be adopted to different robotic arms, but as it stands, it only works with the OpenManipulator-X arm.

Finally, as with any research, this project has many possible future directions it could be taken with (from the training of the model perspective). Even with a good ANN implemented, there is still a problem that a human is needed for control and that every time a point is being reached, the same configuration of joints will be predicted. Therefore, the most important research topic would be to explore more advanced options for manipulation. With the system ready, it would be easier to develop and test them now. The possible use cases, include the methods mentioned in the Chapter 2. One that would work very well with the given pipeline would be to use imitation learning to teach the model to perform human-like actions. There is already a system to collect data points, ergo, a motion of performing a task could be added. Afterwards, the training pipeline could be used to train it. That could be used for automating tasks and making the conrol more robust, where the model learns to move, rather than learns exact mathematical formulations. Alternatively, deeper, network could be used to further make the motion accurate, such as using LSTMs. Alternatively, the control pipeline could be used to develop a real-world application that could be applied in medical or manufacturing robots, where mimicking human movements is crucial.

Closing Statements Overall, the running of the project went well. Unfortunately, the outcome was not satisfactory from the sense of control, and one of the main goals was not met. With IK, the arm can still be controlled with CV, even if it is slow, but the control with ANN is not far away either, just more time would be needed.

The project has allowed me to explore a new area of interest, and learn a lot on the way. The main issues came with the novelty of the whole topic, which in the end affected the planned schedule by quite a bit. Thus, the evaluation and testing has slightly suffered in the end, due to a cut time. But as with any research that is being done, it is an ongoing process, and many future directions were identified. The main objective was achieved to an extent, and with the system being accessible on GitHub, it can hopefully be helpful to anybody who needs it, and wants to follow this line of research.

Appendix A

Joint Limits

The following Figure A.1 contains the diagram of all four joints of the OpenManipulator-X arm.

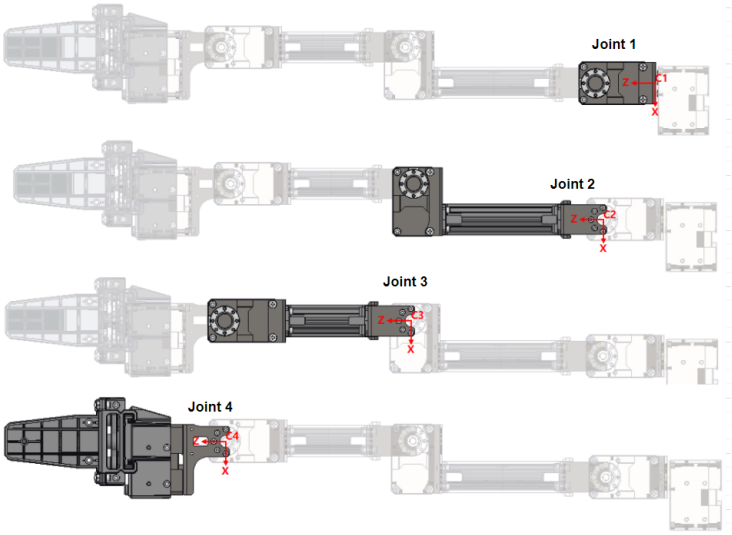


Figure A.1: Manipulator Joints Diagram [7]

The following summarize the angle limits (in radians) of all the joints:

- **Joint 1** - [-π, π] over z-axis
- **Joint 2** - [-2.05, 1.67] over y-axis
- **Joint 3** - [-1.67, 1.53] over y-axis
- **Joint 4** - [-1.8, 2.0] over y-axis

Bibliography

[1] Z. Bingul, H. M. Ertunc, and C. Oysu, "Applying neural network to inverse kinematic problem for 6r robot manipulator with offset wrist," in *Adaptive and Natural Computing Algorithms*, B. Ribeiro, R. F. Albrecht, A. Dobnikar, D. W. Pearson, and N. C. Steele, Eds. Vienna: Springer Vienna, 2005, pp. 112–115.

[2] A.-V. Duka, "Neural network based inverse kinematics solution for trajectory tracking of a robotic arm," *Procedia Technology*, vol. 12, pp. 20–27, 2014.

[3] A. Sivakumar, K. Shaw, and D. Pathak, "Robotic telekinesis: learning a robotic hand imitator by watching humans on youtube," *arXiv preprint arXiv:2202.10448*, 2022.

[4] google, "mediapipe," https://github.com/google/mediapipe, 2019.

[5] J. C. Lentin Joseph, *Mastering ROS for Robotics Programming - Second Edition.* Packt Publishing, 2018, p. 7.

[6] L. J. Ramkumar Gandhinathan, "Ros robotics projects," p. 21, 2019.

[7] ROBOTIS-GIT, "emanual (openmanipulator_x)," https://github.com/ROBOTIS-GIT/emanual/blob/master/docs/en/platform/openmanipulator_x/overview.md, 2019.

[8] W. Kabsch, "A solution for the best rotation to relate two sets of vectors," *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, vol. 32, no. 5, pp. 922–923, 1976.

[9] Scrum.org, [Accessed on 19.04.2023]. [Online]. Available: https://www.scrum.org/resources/what-scrum-module

[10] E. Innovations, "Blog: Machine learning: Loss functions," [Accessed on 19.04.2023]. [Online]. Available: https://www.evergreeninnovations.co/blog-machine-learning-loss-functions/

[11] M. E. Moran, "Evolution of robotic arms," *Journal of robotic surgery*, vol. 1, no. 2, pp. 103–111, 2007.

[12] Craig, *Introduction to robotics : mechanics & control / John J. Craig.* Reading, Mass.: Addison-Wesley Pub. Co.,, 1986, includes bibliographies and index.

[13] S. Tejomurtula and S. Kak, "Inverse kinematics in robotics using neural networks," *Information sciences*, vol. 116, no. 2-4, pp. 147–164, 1999.

[14] X. Wang, X. Liu, L. Chen, and H. Hu, "Deep-learning damped least squares method for inverse kinematics of redundant robots," *Measurement*, vol. 171, p. 108821, 2021.

[15] J. Lu, T. Zou, and X. Jiang, "A neural network based approach to inverse kinematics problem for general six-axis robots," *Sensors*, vol. 22, no. 22, p. 8909, 2022.

[16] B. Bócsi, D. Nguyen-Tuong, L. Csató, B. Schoelkopf, and J. Peters, "Learning inverse kinematics with structured prediction," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE, 2011, pp. 698–703.

[17] J. Vaishnavi, B. Singh, A. Vijayvargiya, and R. Kumar, "Deep learning framework for inverse kinematics mapping for a 5 dof robotic manipulator," in *2022 IEEE International Conference on Power Electronics, Drives and Energy Systems (PEDES).* IEEE, 2022, pp. 1–6.

[18] H. Shi, R. Pi, H. Xu, Z. Li, J. T. Kwok, and T. Zhang, "Multi-objective neural architecture search via predictive network performance optimization," 2019.

[19] K. Gokcesu and H. Gokcesu, "Generalized huber loss for robust learning and its efficient minimization for a robust statistics," *arXiv preprint arXiv:2108.12627*, 2021.

[20] J. Jiang, A. McCoy, E. Lee, and L. Tan, "Development of a motion controlled robotic arm," in *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON).* IEEE, 2017, pp. 101–105.

[21] P. Neto, J. N. Pires, and A. P. Moreira, "Accelerometer-based control of an industrial robotic arm," in *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication*, 2009, pp. 1192–1197.

[22] A. Syed, Z. T. H. Agasbal, T. Melligeri, and B. Gudur, "Flex sensor based robotic arm controller using micro controller," 2012.

[23] J. Moreno and R. Kelly, "Velocity control of robot manipulators: analysis and experiments," *International Journal of Control*, vol. 76, no. 14, pp. 1420–1427, 2003.

[24] Y.-P. Su, X.-Q. Chen, T. Zhou, C. Pretty, and G. Chase, "Mixed-reality-enhanced human–robot interaction with an imitation-based mapping approach for intuitive teleoperation of a robotic arm-hand system," *Applied Sciences*, vol. 12, no. 9, p. 4740, 2022.

[25] F. Chaumette, S. Hutchinson, and P. Corke, "Visual servoing," *Springer handbook of robotics*, pp. 841–866, 2016.

[26] B. Iscimen, H. Atasoy, Y. Kutlu, S. Yildirim, and E. Yildirim, "Smart robot arm motion using computer vision," *Elektronika ir Elektrotechnika*, vol. 21, no. 6, pp. 3–7, 2015.

[27] M. Intisar, M. M. Khan, M. R. Karim, and M. Masud, "Computer vision based robotic arm controlled using interactive gui," in *Intelligent Automation & Soft Computing*. Tech Science Press, 2021, vol. 27, no. 2, pp. 533–550.

[28] Y. Zuo, W. Qiu, L. Xie, F. Zhong, Y. Wang, and A. L. Yuille, "Craves: Controlling robotic arm with a vision-based economic system," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4214–4223.

[29] Y. Zhang and S. Li, "A neural controller for image-based visual servoing of manipulators with physical constraints," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 11, pp. 5419–5429, 2018.

[30] D. Kulić, *Human Motion Imitation*. Dordrecht: Springer Netherlands, 2019, pp. 1657–1677. [Online]. Available: https://doi.org/10.1007/978-94-007-6046-2_34

[31] J. Grasshoff, L. Hansen, I. Kuhlemann, and K. Ehlers, "7dof hand and arm tracking for teleoperation of anthropomorphic robots," in *Proceedings of ISR 2016: 47st International Symposium on Robotics*. VDE, 2016, pp. 1–8.

[32] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar *et al.*, "Inner monologue: Embodied reasoning through planning with language models," *arXiv preprint arXiv:2207.05608*, 2022.

[33] Y. Mishchenko, M. Kaya, E. Ozbay, and H. Yanar, "Developing a three- to six-state eeg-based brain–computer interface for a virtual robotic manipulator control," *IEEE Transactions on Biomedical Engineering*, vol. 66, no. 4, pp. 977–987, 2019.

[34] Y. Zhu, Z. Wang, J. Merel, A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas *et al.*, "Reinforcement and imitation learning for diverse visuomotor skills," *arXiv preprint arXiv:1802.09564*, 2018.

[35] M. Russo, "Measuring performance: Metrics for manipulator design, control, and optimization," *Robotics*, vol. 12, no. 1, p. 4, 2022.

[36] A. Kumar and K. Waldron, "The workspaces of a mechanical manipulator," 1981.

[37] S. Patel and T. Sobh, "Manipulator performance measures-a comprehensive literature survey," *Journal of Intelligent & Robotic Systems*, vol. 77, pp. 547–570, 2015.

[38] WongKinYiu, "yolov7," https://github.com/WongKinYiu/yolov7/releases, 2022.

[39] Kukil and Gupta, "Yolov7 pose vs mediapipe in human pose estimation," 2022, [Accessed on 18.04.2023]. [Online]. Available: https://learnopencv.com/yolov7-pose-vs-mediapipe-in-human-pose-estimation/

[40] S. Garg, A. Saxena, and R. Gupta, "Yoga pose classification: a cnn and mediapipe inspired deep learning approach for real-world application," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–12, 2022.

[41] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[42] martinandrovich, "rb-rovi," https://github.com/martinandrovich/rb-rovi, 2020.

[43] J.-H. Jeong, K.-H. Shim, D.-J. Kim, and S.-W. Lee, "Brain-controlled robotic arm system based on multi-directional cnn-bilstm network using eeg signals," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 28, no. 5, pp. 1226–1238, 2020.

[44] N. Sobhan and A. S. Shaikat, "Implementation of pick place robotic arm for warehouse products management," in *2021 IEEE 7th International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA)*, 2021, pp. 156–161.

[45] MathWorks, "Gazebo simulation for robotics system toolbox - matlab amp; simulink," [Accessed on 19.04.2023]. [Online]. Available: https://www.mathworks.com/help/robotics/ug/gazebo-simulation-for-robotics-system.html

[46] ROBOTIS-GIT, "open_manipulator," https://github.com/ROBOTIS-GIT/open_manipulator, 2019.

[47] ——, "open_manipulator_msgs," https://github.com/ROBOTIS-GIT/open_manipulator_msgs, 2021.

[48] ——, "open_manipulator_simulations," https://github.com/ROBOTIS-GIT/open_manipulator_simulations, 2021.

[49] ——, "open_manipulator_dependencies," https://github.com/ROBOTIS-GIT/open_manipulator_dependencies, 2020.

[50] F. P. Kevin Lyncgh, *Modern Robotics*. Cambridge University Press, 2017, p. 89.

[51] G. Castelli and E. Ottaviano, "Manipulators workspace analysis as based on a numerical approach: Theory and applications," *Manipulators: Theory, Types and Applications, Nova Science Publishers*, pp. 59–76, 2011.

[52] legislation.gov.uk, "Data protection act," 1998, [Accessed on 04.05.2023]. [Online]. Available: https://www.legislation.gov.uk/ukpga/1998/29

[53] BCS, "Bcs code of conduct," 2022, [Accessed on 04.05.2023]. [Online]. Available: https://www.bcs.org/membership-and-registrations/become-a-member/bcs-code-of-conduct/

[54] legislation.gov.uk, "Copyright, designs and patents act," 1988, [Accessed on 04.05.2023]. [Online]. Available: https://www.legislation.gov.uk/ukpga/1988/48/contents