NTNU
Kunnskap for en bedre verden

Department of Engineering Cybernetics

Hand Tracking for Robotic Control through Deep Convolutional Neural Networks

Simon Julian Nagelsaker Lexau

Project Thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
December, 2021

Preface

This thesis presents my work in the course TTK4550 during the autumn of 2021 at NTNU, which lays the foundation for my MSc dissertation in the spring of 2022. I began brainstorming the project in April of 2021, but the only thing that was clear to me was that I wanted to be creative and have fun developing it.

The project ended up being an exciting journey of problems, solutions, backtracking, and sidesteps. To my surprise, the most challenging part of the project was perhaps to complete the loop, stop developing and feel satisfied with the results.

NTNU supplied me with the equipment necessary to perform my research. The robotic manipulator I worked on was the *OpenManipulator-X*, and the stereoscopic depth camera used to capture sensory information of the operator's hand was the *Intel RealSense Depth Camera D435*. The human-machine interface was developed on a desktop computer, with an *Intel i7-8700* CPU, *32 GB* of RAM, no GPU, and running *64-bit Ubuntu 20.04*. For specific information regarding the software packages, the reader is referred to Section 4 of this thesis.

I would like to express my gratitude to my supervisor Anastasios Lekkas for accepting my project proposal. Lekkas made the project possible by lending me both the manipulator and depth camera. He also supplied me with valuable inspiration, motivation, and precise guidance throughout the semester. Sindre Remman, PhD candidate at the *Department of Engineering Cybernetics*, helped me get started with ROS, explainable artificial intelligence, and provided me with a second depth camera.

Omega Verksted supplied me with equipment, guidance, and a workshop to construct the camera stand, holding the depth sensor, and deserves special thanks for their hospitality. During the semester, my fellow students, who shared an office with me, served as user candidates for the human-machine interface and provided valuable user experience feedback. The feedback most definitely raised the system's overall quality, and for that, they deserve my gratitude.

Abstract

Peer-to-peer cooperation between humans and robots is the holy grail of human-robotic interaction. Such a high level of cooperation demands more intuitive methods of communicating with robots. Vague voice commands would be the best way of fulfilling such a goal, but the technology is still years away. As a step in that direction, this thesis will investigate whether state-of-the-art artificial intelligence-powered hand tracking methods can be applied to construct an intuitive human-machine interface for robotic control. The goal is to allow users to efficiently and intuitively operate a robotic manipulator and solve unforeseen problems in environments inaccessible to humans.

An alphabet of intuitive hand gestures is designed, which serves as input signals to a finite state machine, overseeing the control of a robotic manipulator. To improve robustness and trust in the system, explainable artificial intelligence methods are employed in a graphical user interface, yielding the user an online interpretation of the hand tracking commands.

To validate the system, users are presented with a lever manipulation challenge. The operator uses the interface to control a robotic arm to the lever's position before successfully pulling it. The promising results show how such a system can solve unforeseen challenges in environments where only unmanned vehicles operate. Such a system might come in handy for Lunar and Martian astronauts in the near future, who can employ it for solving problems outside the habitat without venturing outside themselves.

Contents

Acronyms

AI Artificial Intelligence. 2–4, 7, 33

ANN Artificial Neural Network. 7

API Application Programming Interface. 8, 32

CNN Convolutional Neural Network. 8, 10, 11, 32

DH Denavit-Hartenberg. 5, 15, 22

DNN Deep Neural Network. 7

DOF Degrees of Freedom. 2, 5, 6, 11, 28

DT Decision Tree. 12, 37

FOV Field of View. 23

FSM Finite State Machine. 2, 3, 12, 13, 15, 17

GUI Graphical User Interface. 11, 17, 18, 31, 33

HMI Human Machine Interface. 2–4, 10, 11, 21–23, 25, 27, 28, 32, 33

HRI Human Robot Interaction. 3, 9, 33

IMU Inertial Measurement Unit. 32

IR Infrared. 23

ML Machine Learning. 7, 8, 14

ROS Robot Operating System. 6, 11, 12, 16, 21–23, 28

SDK Software Development Kit. 23

STEM Science, Technology, Engineering, and Mathematics. 28

STL Standard Triangle Language. 21

UAV Unmanned Aerial Vehicle. 33

XAI Explainable Artificial Intelligence. 2–4, 31, 33

1 Introduction

1.1 Background and Motivation

Robotic manipulators are being used for a wide variety of tasks, from automating repetitive jobs on assembly lines to drilling in rocks on other planets and disarming bombs. Yet these use-cases can be divided into two categories: Boring and physically repetitive tasks and tasks that are too dangerous for humans. Recent developments in robotics made it possible to automate repetitive chores that are slightly more complicated than the assembly line problems, such as vacuuming [1] and lawn mowing [2].

With the development in AI (Artificial Intelligence), a not-so-distant future where robots can manage complex, unforeseen tasks while dealing with real-world uncertainty seems like a possibility. A steep increase in the number of AI related patents for the last few years was presented in [3], depicting the technological advances we have witnessed in the previous decade. Furthermore, AI has proven a practical, powerful tool for analyzing speech, but also body language [4].

Buttons, levers, and joysticks are among the most widespread HMIs (Human-Machine Interfaces) but require extensive training and expertise to operate correctly. A more natural way of communicating would be to mimic human-to-human interactions, which often boils down to body language and speech. Thus the machine would have to interpret spoken commands or gestures from the operator, as in [5] where a robotic arm controlled by continuous human voice commands is proposed. A semi-autonomous robotic arm controlled by a hybrid gaze-brain interface was presented in [6], while [7] describes a gesture-controlled HMI for safe bomb disarming. In the future, one can imagine a scenario where a power line worker inspects power cables in cooperation with a robot and simply tells the machine what actions to perform via spoken commands or hand gestures.

The challenge of hand tracking was previously solved through complex algorithms specifically designed to look for contours, colors, and other features in the image to find objects resembling hands. These methods proved to be computationally demanding and challenging to implement, as they require hand-designed filters. The previously mentioned advancements in AI have, however, also contributed to the field of hand tracking [8]. In [9], the author writes a program that opens and closes the fingers of a 3D printed robotic arm with human features, designed by the french designer Gaël Langevin, by using Google's AI-powered hand tracking software. The results are inspiring. However, the manipulator's position remains fixed, and the project becomes more of a neat gimmick than a functional robotic control system.

Mimicking human movements is a complicated challenge. The authors of [10] present a model of the human hand with 27 DOF (Degrees of Freedom), explaining it as follows: "4 in each finger, 3 for extension and flexion and one for abduction and adduction; the thumb is more complicated and has 5 DOF, leaving 6 DOF for the rotation and translation of the wrist". Excitingly, the Shadow Robot Company [11] has developed a human-like robot hand, the Dexterous Hand, with 20 DOF, not far away from the ideal 27 DOF presented in [10]. Meanwhile, typical robotic manipulators only have six DOF. These advancements contribute heavily to the increasing level of cooperation between humans and robots.

One of the latest fields of AI, XAI (Explainable Artificial Intelligence), is dealing with the problem of building trust in AI-powered systems. Such applications are often considered black-box models, portraying limited insight into their reasonings. Methods such as [12] and [13] are proven methods within this field, yet little has been done for robotic applications. One example, however, is [14], which focuses on explaining the black-box model used in an automated ship docking application for different end-users. The captain, for instance, needs to trust that the system is performing the correct actions in real-time. While the developers designing the system are interested in more complex detailed explanations, which cannot be delivered in real-time due to computational requirements.

In this thesis, an HMI for robotic control is proposed. The system combines XAI methods with cutting-edge hand tracking technologies in an FSM (Finite State Machine) structured system. The goal is to achieve an intuitive, easy-to-use control system that can rival traditional interfaces such

as joysticks and keyboards. Although it will not solve generic, semi-automatic tasks at this point, the interface can be used to control such functionality later on, if desired.

1.2 Objectives and Research Question

We will adopt technologies such as deep neural networks, robotic control, and XAI to answer the following research questions:

- Can a hand tracking software powered by CNNs, such as the MediaPipe Hands framework, be applied to construct an intuitive HMI system based solely on hand movements and gestures?
- Can the HMI be used to perform any meaningful work with a robotic manipulator?
- How can XAI methods be used online to help users understand the system?

Hand tracking is not a new field of study and was first developed in 1969 [15], yet the number of use cases remains limited. In the last decade, we have seen advancements in object recognition powered by deep neural networks. It is the author's belief that the classical hand tracking problem can benefit from these advancements.

A significant objective is to construct an intuitive, easy to learn, control system. As is further elaborated in the Hand Tracking section, it is crucial to validate the system by testing how easily other people can adopt it. If the system is difficult to use, traditional approaches such as joysticks will remain preferred.

Another critical objective in this thesis is to lay the groundwork for a robust and intuitive control system that will be the subject of the author's MSc thesis next semester. Should the goals of this thesis be fulfilled, the next step would be to compensate for disturbances present on underwater, aerial, and ground vehicle- manipulator systems. Such an application will enable the operator to perform tasks on external platforms such as underwater equipment since the robotic arm dynamically stabilizes its position.

1.3 Main Contributions

The thesis contributes to the fields of robotic control systems, HMI, and XAI, by integrating existing technologies. A summary is given below, followed by more thorough explanations.

- An intuitive and responsive HMI system for improved HRI (Human Robot Interaction)
- A system-specific online XAI method presenting the outputs from a hand tracking AI understandably, for multiple levels of end-users

The control system is designed as an FSM (Finite State Machine) and utilizes hand tracking in combination with an alphabet of different gestures and poses to switch between states. The result is a set of commands controlling an affordable open-source robotic manipulator. Further, a low-cost stereoscopic depth sensor provides the input to the hand tracking module. The choice of components demonstrates that cheap, off-the-shelf items can contribute valuable information to a cutting-edge robotic control problem.

The Shadow Robot Company [11] provides haptic gloves, which allow users to take control of up to two human-like Dexterous Hands from a remote location. The equipment truly is cutting-edge HMI technology but comes at a steep price. The equipment used in this thesis is approximately 200 times cheaper than the high-end gear. A similar system that does not need gloves or specially constructed human-like robotic hands could benefit a larger group of people due to the dramatic price difference.

Google's MediaPipe framework will be used to achieve high accuracy in hand tracking. The tracking software is powered by AI, making it more reliable than traditional methods but simultaneously introducing the need for explanations. Adequately explained hand tracking information is utilized to separate seven hand gestures, enabling the operator to control all the manipulator's degrees of freedom.

Furthermore, explanations will be performed by methods from the field of XAI to demonstrate how different end-users can gain insight into how the robotic control system interprets their commands in real-time. Black-box models need interpretation, and several methods have been presented for explaining AIs dealing with both classification and regression problems, see [12], and [13]. However, these methods are limited by demanding computations, making them unsuitable for online explanations.

To the author's best knowledge, there does not exist another HMI that utilizes visual hand tracking, hand gestures, and depth information to control a robotic manipulator intuitively. However, a few similar projects were found. In [7] a robotic manipulator is controlled with hand gestures, and a YouTube search revealed a similar project also using the MediaPipe framework [9].

1.4 Thesis Structure

The thesis consists of five chapters, excluding the introduction, briefly reviewed here. The relevant theory of robotic manipulators, machine learning, explainable artificial intelligence, and hand tracking are presented in Section 2. Next, the problem is formulated, followed by a thorough presentation of the system design and implementation. Following, in Section 4, the experimental setup, including the depth sensor and robotic manipulator, are introduced. Further, the results are presented in Section 5, and the research questions are answered in the Conclusion.

2 Theory

2.1 Robotic Manipulators

Robotic manipulators are mechanical devices designed to perform a wide range of automated tasks. Most manipulators consist of a series of rigid links connected by joints with one adjustable parameter, commonly referred to as one DOF (Degrees of Freedom). [16] explains degrees of freedom as the number of scalar variables that are necessary and sufficient to describe the locations of all the components in a mechanical system. Thus, a robotic manipulator with four movable joints will have four DOF. In some cases, the manipulator might be restricted, however. For instance, a pendulum attached to a rigid body can only move in a spherical space and has only two DOF instead of three. Its degrees of freedom is the difference between its DOF in an unrestricted system and the number of constraints. In this case, the pendulum has originally three DOF, but loses one since the body it is attached to is rigid. Furthermore, typical robotic manipulators have six DOF, which means the end effector of the arm is free to move in three translational axes, X, Y, and Z, but also its orientation through rotations.

There exist many different types of joints, but the two most common are prismatic and revolute joints.

Prismatic joint
Often also labeled linear joints, the movements from these joints are translational along a single axis, where the axes of the connected links remain parallel.

Revolute joint
The relative movement between the connected links is rotational, perpendicular, or parallel to the input link.

2.1.1 Forward Kinematics

Forward kinematics is the process of computing the end-effector's position by using the geometry of the robotic manipulator together with the varying link orientations. These varying angles are commonly known as generalized coordinates. In Figure 10 a total of four coordinate frames represented by vectors of the type $\vec{X}_i, \vec{Y}_i, \vec{Z}_i$ has been drawn. The $i = 0$ frame has been drawn in the first joint and is static. The subsequent three frames move in relation to each other and are drawn to represent the movement from all joints in a straightforward manner. Lastly, the fourth frame has been drawn suitably on the end-effector. The frames have all been assigned according to the DH (Denavit-Hartenberg) convention.

$$\boldsymbol{H_n^0} = \begin{bmatrix} \boldsymbol{R}_n^0 & \boldsymbol{o}_n^0 \\ \mathbf{0} & 1 \end{bmatrix} \in SE(3) \tag{1}$$

By using the representation depicted in fig. 10, we can introduce homogeneous transformations between the frames. These transformation matrices can be multiplied to produce a single transform from zero frame to end-effector frame, as presented in eq. (1), where $\boldsymbol{R}_n^0$ is the rotation from 0-frame to end-effector frame, $\boldsymbol{o}_n^0$ is the position of the end-effector in 0-frame, and n is the number of movable joints. Thus we can acquire the end-effector coordinates in the static zero frame if we know the DH-parameters. In short, forward kinematics involves calculating the $\boldsymbol{H_n^0}$ matrix, such that we can find the position of the end-effector in the 0-frame, often called world coordinates.

2.1.2 Inverse Kinematics

While forward kinematics can be used to calculate the end-effector position, given joint and link parameters, inverse kinematics is regarded as the opposite operation. That is, calculating the variable joint parameters needed to move the end-effector to a pre-determined position.

$$\boldsymbol{H}_n^0(q_1, q_2, \cdots, q_n) = \boldsymbol{A}_1(q_1)\boldsymbol{A}_2(q_2)\cdots\boldsymbol{A}_n(q_n) \tag{2}$$

We start with the solution to the forward kinematics problem; the transformation matrix given by eq. (1). Equation (2) states that the transformation matrix is made up of the product of the transformation matrices from one frame to another. That is, $\boldsymbol{A}_i$ is the transformation from frame $i-1$ to frame i, and q_i is the variable parameter of joint i. We often have more than six DOF, which is needed to achieve all possible orientations and positions. Because of this, it is not trivial to solve eq. (2) since multiple solutions can exist. Without considering this, one can experience sudden jumps in the variable joint parameters as the controller switches between solutions.

The robotic manipulator used in this thesis, however, has only five DOFs, and the inverse kinematics computations are calculated by a controller supplied by Robotis, the designers of the manipulator.

2.1.3 Obstacle Representation

It is vital to avoid damaging the equipment and causing injuries when dealing with trajectory planning. To achieve this, we choose a topological approach, where the subspaces workspace, configuration space, and obstacle space are defined. According to [16], trajectories are either "a set of functions describing time-evolution of the generalized coordinates, or a set of functions describing time-evolution of the end-effector." In our case, the pre-built ROS controller for the OpenManipulator, see Section 3.5, takes care of the inverse kinematics. The trajectories in this thesis will be the second definition, that is, time-evolution of the end-effector.

The trajectories are not large or complex since the system will be taking many contiguous commands from a user. One such command, the system state *MOVE FORWARD*, is the equivalent of a time-evolution trajectory lasting $0.6s$ with a straight path of $8.33cm$ in the X^0Y^0 plane from **??**. Even though the trajectories are not complicated, there are still obstacles to avoid in the manipulator's workspace. For a comprehensive list of all the system states and their actions, see Section 3.6.

The robot arm cannot be allowed to crash into itself. As a result, we can define the configuration space as a set containing all valid values of the generalized coordinates, q, given n links. See Equation (3). However, the distributed ROS controller handles self-crashing avoidance, and it will not need further work here.

$$\boldsymbol{Q} = \{\boldsymbol{q}\} \tag{3}$$

$$\boldsymbol{W} \subset \mathbb{R}^3 \tag{4}$$

Further, the workspace can be defined according to Equation (4), where W is a subset of 3D space and represents where the robot moves. The workspace does, however, contain a set of obstacles, Equation (5).

$$\boldsymbol{O}_i \subset \mathbb{R}^3, i \in \mathbb{R}_{\geq 0} \tag{5}$$

$$\boldsymbol{A}(\boldsymbol{q}) \subset \boldsymbol{W} \tag{6}$$

$\boldsymbol{A}(\boldsymbol{q})$ from Equation (6), is another subspace of the workspace occupied by the robotic manipulator. The obstacle space can then be defined as shown in Equation (7).

$$\boldsymbol{QO} = \boldsymbol{q} \subset \boldsymbol{Q} : \boldsymbol{A}(\boldsymbol{q}) \cap \boldsymbol{O}_i \neq \varnothing, \forall i \tag{7}$$

When $\boldsymbol{q} \subset \boldsymbol{QO}$, the robot is in collision with an obstacle. Thus we can define the collision-free subspace as Equation (8)

$$\boldsymbol{Q}_{free} = \boldsymbol{Q} \setminus \boldsymbol{QO} \tag{8}$$

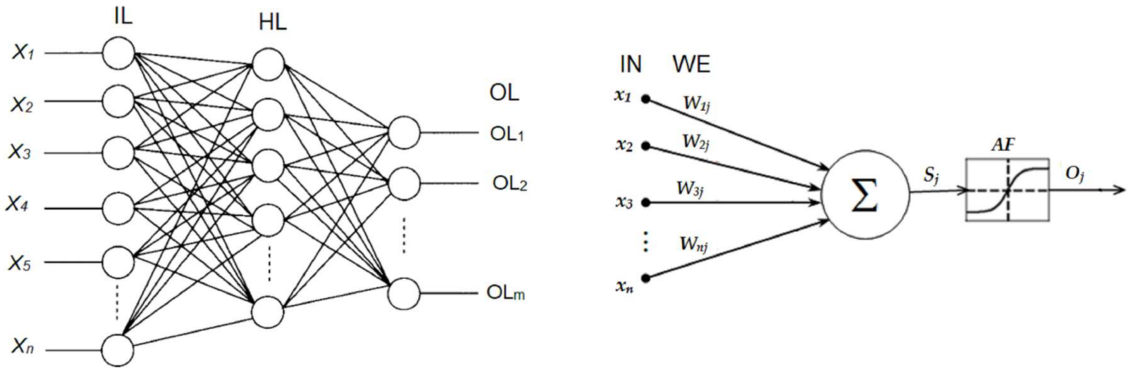
2.2 Machine Learning

ML (Machine Learning) is the study of computer algorithms that can improve automatically through experience and gathered data [17]. It is a large part of artificial intelligence (AI). Artificial neural networks (ANNs) are function approximators used extensively in AI where the goal is to create computers that exhibit intelligent behavior [18]. Modern techniques utilize DNNs (Deep Neural Networks), which can be seen as an extension of the typical ANN, containing multiple hidden layers. The applications for AI methods are countless, yet a common theme is often problems with vast amounts of data, where some structure is to be found. Examples include driving a car autonomously, recognizing objects in images, or even outperforming humans at strategic games such as Go [19]. Generally, an ML algorithm creates a model based on training data gathered and processed in advance. The topic is, however, often divided into three subcategories:

Supervised learning
The machine receives input data and compares the calculated output with the corresponding label. For example, a set of images of cars that are masked or labeled makes the computer learn what features to expect in a new unlabeled image of a car.

Unsupervised learning
The machine does not receive labeled data but learns of any structure in the data by itself. Examples of use cases include creative tasks, such as writing a screenplay, composing music, or generating videos.

Reinforcement learning
Reinforcement Learning is often used in cases where rules are easy to define, resulting in measurable bad or good behavior. Videogames are good examples, with readily defined world spaces and reward functions, but the algorithms are also useful for real-world robotic applications [20].



(a) (b)

Figure 1: (a): A neural network. (b): A close up of a single neuron in a fully connected network Image borrowed from [21]

2.2.1 Artificial Neural Networks

ANNs are computational models that intend to simulate biological neurons. A typical ANN, shown in figure Figure 1, consists of an input layer, n hidden layers, and an output layer. The network in the figure has exactly one hidden layer but can have many more. Looking at figure (b), we can see

that for all the neurons in a layer, all the inputs from the previous layer are multiplied by weights, summed, and passed through an activation function. A bias, not shown in the figure, is normally multiplied by a weight and summed together with the inputs before the activation function. The activation function mimics whether the neuron is activated or not, just like in a natural brain. The ReLU activation function, for instance, makes all negative inputs zero while returning the positive ones. See Equation (9)

$$f(x) = max(x, 0) \tag{9}$$

Another commonly used activation function is the Sigmoid, which maps all real values into the range from 0 to 1. See Equation (10)

$$f(x) = \frac{1}{1 + e^{-x}} \tag{10}$$

While calculating the output from some input is called forwards propagating, the distinctive learning algorithm is called backpropagation. Assuming we have a set of images of either cats or dogs which are labeled, we perform a forward propagation on a single image and compare the output with the known label. The output is a vector with two decimals, each having a value between 0 and 1, where 1 means that the network is certain that a cat or a dog is present. Thus we can calculate the error between prediction and label with a loss function. The weights are updated according to how much they contributed to the error metric in question. This thesis will not delve into further details concerning the backpropagation algorithm.

A class of neural networks commonly referred to as convolutional neural networks (CNN) is popularly utilized to analyze visual imagery. [22]. These CNNs consist of hidden convolutional layers, where the term convolutional comes from the convolving process. A filter, or kernel, is initialized with random numbers for each convolutional layer and slides across the input, often an image. The dot product between the input image and the kernel is thus outputted from the convolutional layer. These filters are good at pattern recognition. By having multiple such layers consecutively, one layer can detect corners, circles, or squares while the next can detect more complex shapes such as hands, eyes, or hair. Further, each filter is trained, removing the need for designing each one by hand, which is a substantial advantage over traditional filtering methods.

MediaPipe, a framework by Google, offers cross-platform, customizable ML solutions for live and streaming media [23]. It consists of packages for a range of problems, including *Face Detection*, *Hands*, *Pose*, *Hair Segmentation* and much more. The *Hands* API, an ML solution consisting of two CNNs will be used to solve the hand tracking problem in this thesis. For more information on these networks, see Section 3.4.1

2.3 Explainable Artificial Intelligence Methods

Neural networks are trained on large amounts of data to create a model function that makes predictions based on the input information. While impressive, even the designer of the network does not know how the model makes its predictions. Neural networks are usually just regarded as black boxes. This phenomenon comes with some difficulties. For example, in 2017, The Guardian wrote an article about teachers in Houston who had their performance assessed by an AI [24]. The agent made predictions based on their students' test results compared to the average score in Texas. The teachers who received a good performance were awarded bonuses, while those with low scores risked getting fired. The company responsible for the AI refused to reveal how it made predictions, calling it a trade secret. Thus the teachers were not able to tell if the predictions were fair or faulty. Later, a federal judge ruled that the AI program could be violating their civil rights. The school district stopped using it and paid the teachers' fees.

In the article, The Guardian wrote about similar cases where an AI was judging humans. It is thus a solid ethical argument to improve on these systems by making the AI's thought process understandable for humans. Explainable Artificial Intelligence, or XAI for short, is a field within

AI with precisely that goal in mind. The authors of [12] have developed LIME, "an algorithm that can explain the predictions of any classifier or regressor in a faithful way, by approximating it locally with an interpretable model." As an illustrative example from the paper, Figure 2 shows how an AI makes a wrong prediction on a husky. It is clear that the network has been looking for the wrong kinds of features under training and may have instead noticed that snow was present in most of the wolf images. Thus when the input is an image of a husky in the snow, it completely ignores the essential features we are interested in, instead wrongly classifying it as a wolf due to the presence of snow.

(a) Husky classified as wolf (b) Explanation

Figure 2: Image borrowed from [12]

$$\xi(x) = \underset{g \in G}{argmin} \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{11}$$

The LIME algorithm obtains its explanation by minimizing Equation (11), where f is the black-box function, and g is a simple model which works in the neighborhood of the input x. Π_x is a proximity measure between samples z and x. The samples are drawn from the vicinity of x. According to the authors, the running time is around ten minutes when explaining a single prediction for image classification. This means we cannot run the algorithm online, and we will not be able to answer the third research question.

2.4 Hand Tracking

Ten years ago, hand tracking was no new field of study, yet the applications remained limited. Due to advancements in artificial intelligence, the field of hand tracking has seen many improvements since [25] was written in 2011. The paper looks into a range of different applications for hand tracking. It discusses the limitations that must be solved before we can see hand tracking gaining commercial success and widespread use. The authors divide the use-cases into four different classes: namely *medical systems and assistive technologies*, *crisis management and disaster relief*, *entertainment*, and *human-robot interaction*, HRI for short, which is where the focus of this thesis lies.

These limitations addressed by [25] could be seen as guidelines for proper hand tracking software and are described by the six concepts:

- Intuitiveness
- Comfort
- Come as you are
- Reconfigurability

- Interaction space
- Gesture spotting and immersion syndrome

While *Comfort* is trivially explained, the other concepts deserve an introduction.

Intuitiveness is a somewhat loosely defined term, but Cambridge Dictionary provides the following definition: "Able to know or understand something because of feelings rather than facts or proof" [26]

Come as you are is an HMI design approach, where the user should not be required to wear any extra utility, such as markers, gloves, or having a fixed background, to interact with the computer. Most of the HMIs have been utilizing extra equipment or other assumptions and limitations. In [27] the students built an exoskeleton mounted to the shoulder and down the entire arm of the user. The exoskeleton controlled a robotic manipulator based on signals from potentiometers properly placed in relation to the operator's joints. Another example is [28], where a glove is used to maneuver a robotic arm. The glove is equipped with a gyroscope, an Arduino, and two flex sensors. Other similar examples include [29] and [30].

Reconfigurability can be explained by the *one size fits all* terminology, which means that the system should yield the same behavior when used by different users. Some papers use color segmentation, movement, or shape for hand detection, which can be argued to be lacking in reconfigurability. Examples of such implementations are [31] and [32]. The latter makes use of both color and shape to detect human hands. The correct color is decided by running face detection on an image of the user and then finding the largest segmented image region that is not the face. This method limits the use to only focus on one hand at a time and could cause challenges for people with different skin tones on their hands than in their faces. Further, the only hand gesture that the method can detect is an open hand with all fingers spread out, limiting the applications. Yet, the researchers utilized this information to control a robotic manipulator by moving a hand in different positions in the image.

Interaction space is the space in which the system assumes the user is standing. Ten years ago, it was far more common for systems to demand that the users were standing in a fixed location with their hands extended and head visible, according to [25]. Even though some methods today require such limitations, as [32], with the use of deep learning techniques, one can easily detect hands without a static background, visible head, or a specific skin tone. Even under low light conditions, [33] shows that it is feasible to detect hands with thermal cameras using deep neural networks.

Gesture spotting and immersion syndrome is the art of distinguishing useful hand gestures from unintentional movement. This is a challenge, especially with temporal gestures, with both start and endpoints in time and space. [34] shows how this is feasible, but such a solution will not be attempted in this thesis.

The terms briefly discussed here were necessary for analyzing the status quo for hand tracking applications in 2011 and why they were not in everyday use then. Thus the same terminology will be used to validate the quality of the HMI implemented in this thesis.

As we shall see in Section 3.4.1, modern hand tracking methods powered by CNNs do a great job at addressing the limitations discussed here.

3 Problem Formulation and System Design

3.1 Problem Formulation

The problem at hand is the challenge of controlling a five DOF robotic manipulator with the help of a custom-made gesture-recognizing HMI. The level of control should allow the operator to take full advantage of all the manipulator's degrees of freedom. Further, the accuracy of the HMI should be high enough to allow the operator to solve a lever manipulation challenge. Additionally, it should be explained to the operator how the system interprets their given commands in a continual fashion.

3.2 System Overview

Figure 3 depicts the information flow and overall workings of the control system. The camera sensor is the red block in the upper right corner and records the operator workspace, outputting both RGB- and depth images. The black box CNN is the MediaPipe Hands machine learning model, which receives the RGB image on its input, and yields estimated locations for 21 3D points where the third dimension is a synthetic height. The points are used to produce 2D and 3D skeleton hands, visualized in the operator panel. Meanwhile, the points are fed into a finite state machine, which also receives the depth image and tunable parameters from the operator panel. The information is scrutinized to determine the system state, which the controller uses to activate the correct proportional velocity controller. The controller communicates with the manipulator through the ROS 2 framework. Further, a feedback loop is incorporated into the system by the presence of a human operator. The user sees the movement of the robotic manipulator and the interpretations of their hand gesture in the operator panel. Based on this information, the user provides the system with new gestures to achieve the intended manipulator movement.

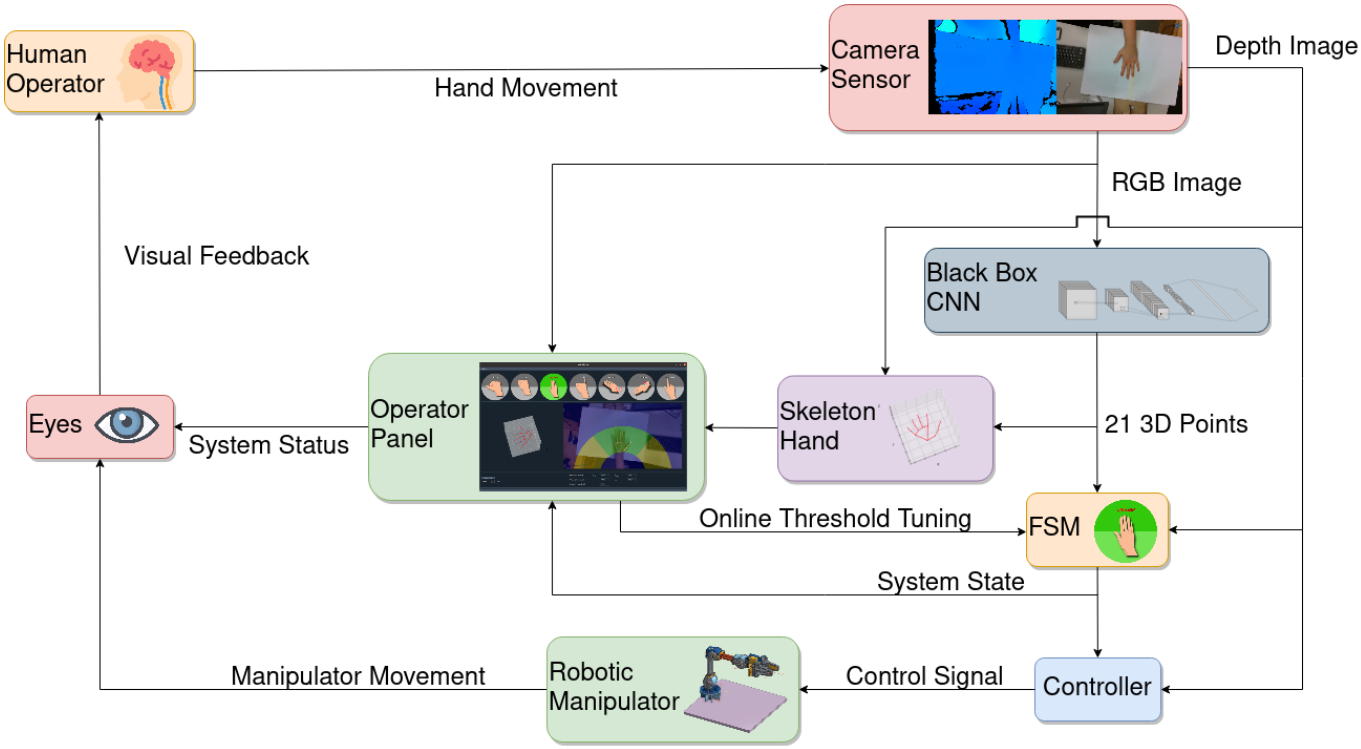


Figure 3: Information flow, and system overview

The class diagram Figure 20, in Appendix C, shows the most important classes of the system, excluding the GUI application. The graph does not show inputs to the class methods to reduce complexity, but the return types and class variables are appropriately defined. The FSM has

exactly one Controller object, one HandTracking object, and one HandModel object. There is no limit for how many Obstacle objects it can have, but the implementation discussed in this thesis has five obstacles. The Controller communicates with the robotic manipulator through ROS 2 with two subscriber objects and four client objects. The HandTracking module, which produces the 21 3D landmarks, has one CameraStream object, which communicates with the depth sensor. The HandModel module estimates current gestures based on the landmarks from the HandTracking module, which is received through a call to the addMeasurement() method.

3.3 Finite State Machine

An FSM (Finite State Machine) is a mathematical model of computations that can be in exactly one state out of a limited number of states. The FSM can change between states in response to some inputs, in what is called a transition [35].

The states of the FSM implemented in this thesis are visualized in Figure 4. The diagram is a simplified state diagram, where transitions between the major states are not drawn for readability. The removed transitions are, however, easily described by an extra transition within *STOP*. The text along the arrows represents the input necessary to trigger the transition. Should these signals no longer be active, the state will transition to the *STOP* state. The signals are generated in the HandModel class from Figure 20 and accessed through the methods getCurrentGesture() and getWorkspaceLocation(). The different signals are presented in Table 1.

When it became apparent that the state machine would have thirteen states, research was done on implementing the control system as a DT (Decision Tree) instead. The idea was dropped, but the analogous DT is added in Appendix B, as it does an excellent job at representing the system logic. The FSM implementation proved to be a more readable and maintainable code.

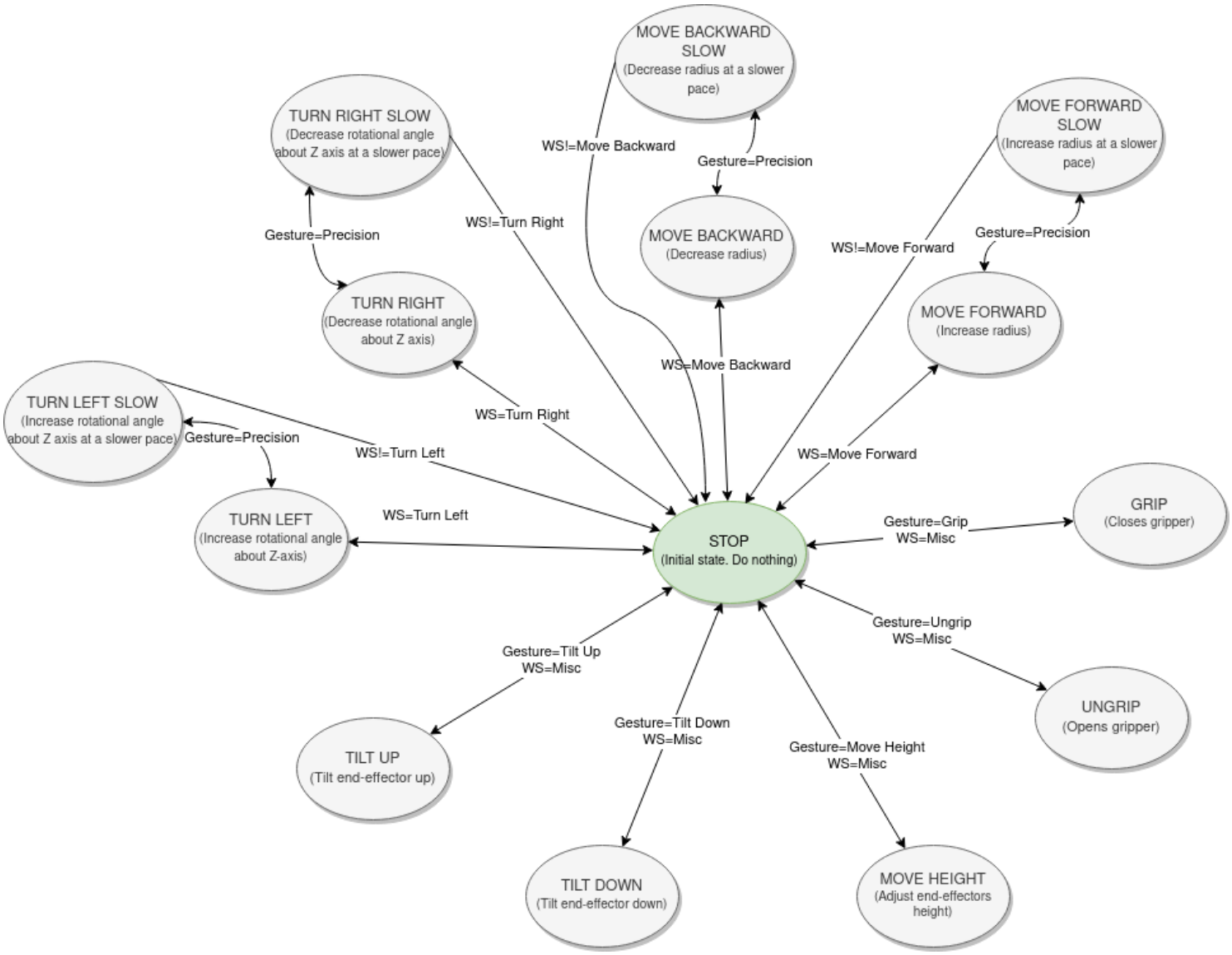


Figure 4: FSM diagram depicting all system states and transitions.

Hand gestures	**Workspace locations**
Stop	Turn Left
Grip	Turn Right
Ungrip	Move Forward
Precision	Move Backward
Tilt Up	Misc
Tilt Down	Hand Not Present
Move Height	

Table 1: The input signals determining the state of the FSM

3.4 Hand Tracking

3.4.1 MediaPipe

MediaPipe is an open source ML library provided by Google, where the Hand Tracking API is the part we are interested in. Researchers at Google have trained a neural network to recognize human hands, and estimate the position of joints from a single image frame. The software detects initial hand locations, with the ability to recognize occluded hands in various sizes and environments. Next, a hand landmark model localizes 21 3D coordinates inside the detected hand via regression. According to the MediaPipe team, the model "learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions."

$$\boldsymbol{X}_k^w(t) = \begin{bmatrix} x_k^w(t) \\ y_k^w(t) \\ z_k^w(t) \end{bmatrix}, \; k \in [0,\, 20] \text{ and } t > 0 \tag{12}$$

The joint positions are outputted as 21 3D landmarks on the form given by eq. (12). The notation for the points is as follows: A hand point in coordinate system of joint $(i-1)$, with an index $k \in [0, 20]$ is written as $\boldsymbol{X}_k^{(i-1)}$. Further, x_k^w and y_k^w are normalized pixel coordinates. z_k^w is a representation of the depth of the landmarks, where $z_0^w = 0$ at the location of the wrist. Note that the information given by z_k^w only describes the estimated depth of the joints in relation to each other. Further, since the depth information is artificially constructed by a machine learning model, it is referred to as synthetic information. Figure 5 illustrates how the landmarks are structured.

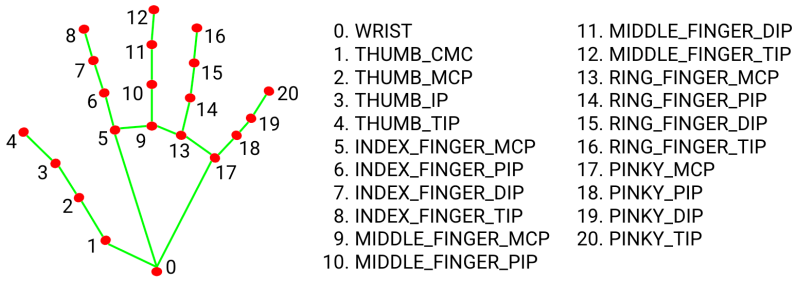


Figure 5: How the hand landmarks from MediaPipe are structured.
Image borrowed from [23]

According to [36], the machine learning pipeline is a two step Convolutional Neural Network consisting of a single-shot detector, followed by a regression model. The input can either be a single image frame, or a video stream, while the outputs are: "21 3-dimensional screen landmarks", "A float scalar represents the handedness probability of the predicted hand", "21 3-dimensional metric scale world landmarks." Note that for both sets of predicted 3D points, the z-screen value and z coordinate, are provided by synthetic data based on the GHUM hand model ([37]). Due to the synthetic nature of the depth information outputted from the model, we will be comparing the synthetic depth information with measurements from a stereoscopic depth sensor.

The detector model detects the palm location(s) in the input data. A crop of the input data, containing the hand, is then used as input to the regression model, that outputs a hand skeleton as the 3D points discussed above.

The authors have done extensive evaluations on the method, where the hand tracking algorithm was tested on 14 different groups of peoples from around the world, divided into groups based on the United Nations geoscheme. The results from this test yielded no error pattern with respect to regions, but showed that the error metric was smaller at the base of each finger, with larger values closer to the finger tips. Another test did not confirm any error pattern with respect to skin tone or gender.

3.4.2 Hand Gestures

Based on the hand tracking information, seven different gestures and six hand locations are defined in the operator workspace. These are the input signals listed in Table 1, which are used to transition between states in the FSM. To separate the gestures from one another, a coordinate system for the hand representation was designed and implemented in the HandModel module from Figure 20. A human hand consists of revolute joints, where the joints between fingers and palm are more complicated, rotating around two axes. Transformation matrices for each finger were designed, inspired by the convenient DH-convention. The landmarks from the hand tracker are given in workspace coordinates, Equation (12). We are, however, interested in the relative angles between the finger links.

For every finger, each transformation is represented by an angle $\delta_i^{(i-1)}$, from joint $(i-1)$ to i in the $X_i^{(i-1)}Y_i^{(i-1)}$-plane, an angle $\gamma_i^{(i-1)}$ in the $X_i^{(i-1)}Z_i^{(i-1)}$-plane, and a translation $t_i^{(i-1)}$ from joint $(i-1)$ to i.

$$\boldsymbol{H_0^w} = \begin{bmatrix} \boldsymbol{R}(\delta_0^w) & \boldsymbol{t_0^w} \\ \boldsymbol{0} & 1 \end{bmatrix}, \boldsymbol{t_0^w} = \begin{bmatrix} x_0^w \\ y_0^w \\ 0 \end{bmatrix} \tag{13}$$

The first transformation matrix, from workspace coordinates w to point 0 coordinates (from Figure 5) is given by Equation (13), and is the same for all fingers. The general transformation matrices from joint $(i-1)$ to i, are on the form Equation (14).

$$\boldsymbol{H_i^{(i-1)}} = \begin{bmatrix} \boldsymbol{R}(\delta_i^{(i-1)})\boldsymbol{R}(\gamma_i^{(i-1)}) & \boldsymbol{t_i^{(i-1)}} \\ \boldsymbol{0} & 1 \end{bmatrix}, \boldsymbol{t_i^{(i-1)}} = \begin{bmatrix} x_k^{(i-1)} \\ y_k^{(i-1)} \\ z_k^{(i-1)} \end{bmatrix} \tag{14}$$

$$\begin{bmatrix} x_8^3 \\ y_8^3 \\ z_8^3 \end{bmatrix} = \boldsymbol{H_7^w X_8^w} = \boldsymbol{H_7^6 H_6^5 H_5^0 H_0^w} \begin{bmatrix} x_8^w \\ y_8^w \\ z_8^w \end{bmatrix} \tag{15}$$

As an example, Equation (15) shows how the coordinates for the fingertip of the index finger, represented in the second to last joint, can be acquired. The angles are the essential results from this process, however, and are found iteratively by Equation (16) and Equation (17). To determine whether a finger is extended, we are interested in $\delta_i^{(i-1)}$ and $\gamma_i^{(i-1)}$ of the two outer joints for each finger.

$$\delta_i^{(i-1)} = arctan\left(\frac{y_i^{(i-1)}}{x_i^{(i-1)}}\right) \tag{16}$$

$$\gamma_i^{(i-1)} = -arctan\left(\frac{z_i^{(i-1)}}{\sqrt{(x_i^{(i-1)})^2 + (y_i^{(i-1)})^2}}\right) \tag{17}$$

By trial and error, threshold values were found to separate between an open and a closed finger. The threshold values were the same for all fingers and angular variables, except the thumb, which responded better to other thresholds. To check if the thumb is extended, $\delta_i^{(i-1)}$ for both joints must be greater than $-15°$. Meanwhile, for the remaining fingers, if either of $\delta_i^{(i-1)}$ and $\gamma_i^{(i-1)}$ for any of the outer two joints are larger than $25°$, the finger is marked as closed.

Further, the mean of landmarks $\boldsymbol{X}_0^w$, $\boldsymbol{X}_1^w$, $\boldsymbol{X}_5^w$, $\boldsymbol{X}_9^w$, $\boldsymbol{X}_{13}^w$ and $\boldsymbol{X}_{17}^w$ is used when deciding on where in the operator workspace the hand is located.

3.5 ROS - Robot Operating System

The Robot Operating System is a general open source framework for writing robotic applications. It contains conventions, libraries and tools that are helpful when working with a robotic manipulator. In this project, ROS will be used due to its tools for communicating with the robotic manipulator, which has custom built packages specifically for use with ROS. The framework is available in multiple distributions, but here *ROS 2 Foxy* will be used due to its compatibility with both Python and OpenManipulator-X.

To communicate with the manipulator, one should have a general understanding of the following modules in ROS: *Nodes, Topics, Services* and *Actions*.

Nodes
A node is responsible for a single modular operation, which could be to control the wheels in a car, or in our case to control a robotic manipulator. Another node could handle user input, and would probably want to transmit this information to the controller node. Which is what *Nodes, Topics* and *Services* are used for.

Topics
Nodes use topics to either publish data, or subscribe to data being published by other nodes. Topics can either be one-to-one, one-to-many, many-to-one or many-to-many communication and are meant for continuous streams of data. There are however no handshakes between the nodes, and one node does not know how many nodes has successfully received the data being pushed on a topic.

Services
Services are a more direct way of communicating than topics, and are based on a call-and-response model. Each Service has exactly one server node that waits on requests from one or more client nodes, and transmits a response to the client that made a request. A client can for example ask for the joint positions of a robotic arm with one dedicated service.

Actions
Actions combine topics and services into a third communication type, meant for longer running tasks. Every action consists of three parts, a goal, a feedback and a result. The goal and result are services while the feedback is a topic. An action client first sends a goal to the server, which responds and acknowledges the goal. Next, the client asks for a result, and while the action is being performed, the server streams data though the feedback topic. Once the goal has been achieved, or is cancelled, a result is transmitted to the client.

3.6 Controller

This section describes the Controller class from Figure 20. As discussed in Section 4, the Robotis OpenManipulator-X comes equipped with controllers dealing with motor control and inverse kinematics. The results are positional controllers for the end-effector, abstracted into ROS topics. Thus, the controllers presented in this section use these topics to control the manipulator to velocity references. The Controller object consists of four proportional velocity controllers, each with a default reference velocity of $0m/s$. The controllers set the velocity of the manipulator by requesting a new pose and a path time, which is set at $0.6s$.

One controller sets the reference velocity for the horizontal radius of the manipulator, and is activated in the *MOVE FORWARD*, *MOVE BACKWARD*, *MOVE FORWARD SLOW*, and *MOVE BACKWARD SLOW* states. For the former two, the speed is set at $8.33cm/s$, while the latter controllers operate at $1.67cm/s$. Another controller sets the reference velocity for the horizontal turning angle β, which is used by the states *TURN LEFT*, *TURN RIGHT*, *TURN LEFT SLOW*, and *TURN RIGHT SLOW*. As for the previous controller, the reference speed is set at $19.10°/s$ for the former states, and $3.82°/s$ for the latter two. A third controller is used for the second to last revolute joint, effectively adjusting the tilt angle of the end-effector. When active, the end-effector will tilt by $14.32°/s$. The last controller, responsible for setting the end-effector's velocity along

the Z^0-axis, calculates the reference velocity based on the positional height of the operator's hand, using information from the depth sensor. The numerical values described here, were found through tuning of the proportional constants, $K_{p,\beta}, K_{p,r}, K_{p,Z_e^0}, K_{p,\theta}$.

Meanwhile the end-effector, also known as gripper, is controlled in a binary fashion, and is either closed or opened. Unlike the previously discussed controllers, it has a positional reference, the distance, $d_{gripper}$. Possible values are of $-1cm$ to close the gripper, and $1cm$ to fully open it.

Table 2 sums up the relationship between controllers and their velocity references to the states of the FSM.

State	**Controlled variable**	**Reference**
MOVE FORWARD	r	$8.33cm/s$
MOVE BACKWARD	r	$-8.33cm/s$
MOVE FORWARD SLOW	r	$1.67cm/s$
MOVE BACKWARD SLOW	r	$-1.67cm/s$
TURN LEFT	β	$19.10°/s$
TURN RIGHT	β	$-19.10°/s$
TURN LEFT SLOW	β	$3.82°/s$
TURN RIGHT SLOW	β	$-3.82°/s$
TILT UP	θ	$14.32°/s$
TILT DOWN	θ	$-14.32°/s$
MOVE HEIGHT	Z_e^0	Dynamic
GRIP	$d_{gripper}$	$-1.0cm$
UNGRIP	$d_{gripper}$	$1.0cm$

Table 2: The states of the FSM, the variables that are controlled when they are active, and the respective references.

3.7 Explainable Artificial Intelligence Implementation

The approach to XAI implemented in this section is not a typical method like SHAP [13] or LIME [12], but follows a more general concept of explaining model outputs through visual representation. While SHAP and LIME do not allow for real-time explanations, this approach does, at the cost of being problem-specific and yielding shallow explanations. As seen in Section 2.3, LIME can give more in-depth explanations, telling us which pixels of the image are responsible for a specific classification. SHAP yields information on how much each input feature contributes to the output value. These thorough explanations come at the cost of computational time. Thus they will not be able to help an operator of the control system understand how their hand gestures are being interpreted. That is the reasoning behind choosing a GUI (Graphical User Interface) as an explanation method.

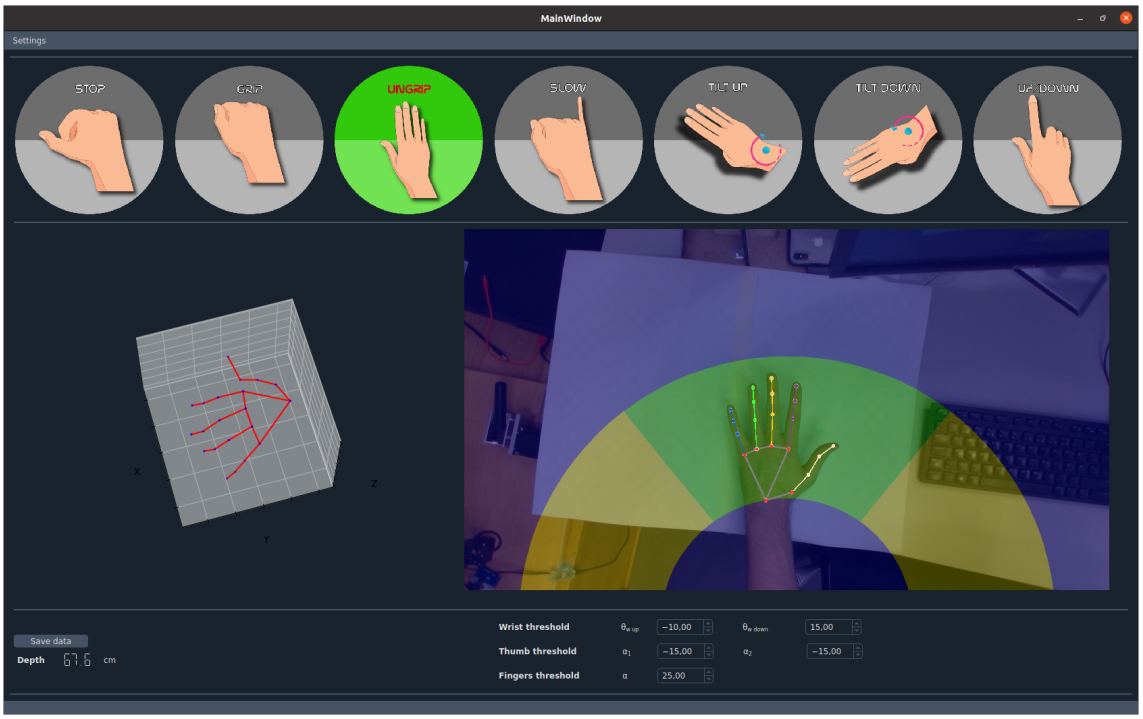


Figure 6: Operator panel

The GUI was designed using the Qt 5 Designer and implemented with PyQt 5, a set of Python bindings for the Qt Company's Qt application framework [38]. The result can be seen in Figure 6. The GUI, or operator panel, consists of a video stream, known as the operator workspace, where a depth sensor is recording the operator's hand. Colored, curved boxes are drawn in the video frame to provide the user with a set of different commands by placing their hand inside these boxes. Further, a 2D hand skeleton is drawn on top of the detected hand, visualizing where the machine learning model estimates the position of the hand's joints.

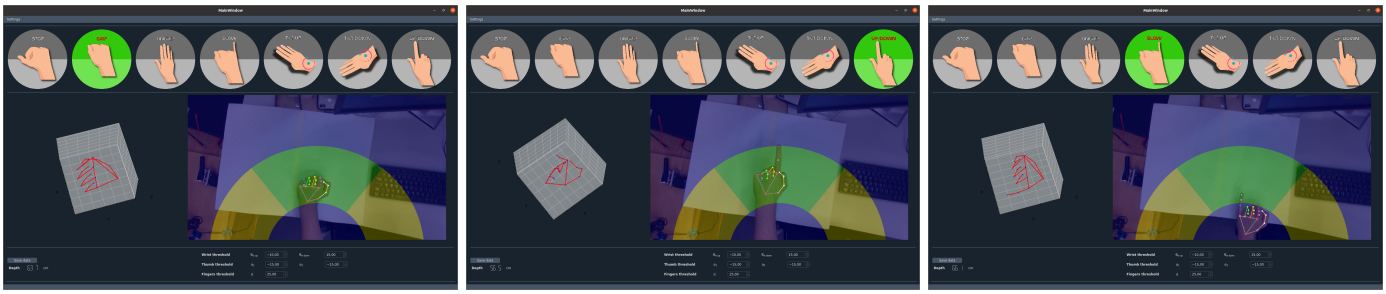


Figure 7: More examples of the operator panel in action

Above the video stream, seven hand gestures are displayed on grey backgrounds. When the system selects a given hand gesture based on the information provided by the machine learning model, the gesture's background switches from grey to green, notifying the user how their hand gestures are being interpreted. This information is not needed to control the manipulator but certainly elevates the user's trust in the hand tracking model and overall system. The control panel also works as a user interface for new users unfamiliar with the different commands. For more experienced users and developers, a 3D hand skeleton and a metric of the measured distance to the hand palm from the stereoscopic sensor are provided. The additional information is easily toggled in the settings file.

Just below the operator workspace, three tunable parameters are accessible. The wrist thresholds

are used to differentiate between *Tilt Up* and *Tilt Down*, where the angle between the wrist and fingertips must surpass the thresholds for the gestures to be detected. Similarly, the thumb thresholds are angles that the two outer thumb joints must surpass before the thumb is registered as extended. The finger threshold functions similarly to the thumb threshold, but it was experimentally determined that the same threshold worked for both outer joints for all remaining fingers.

With the exception of *Slow* and *Stop* gestures, all commands using hand gestures are only in effect while the palm is located within the green section of the operator workspace. The green and yellow sections are shaped like half-circles, allowing the operator to keep his shoulder and elbow more or less stationary when performing rotations in a natural movement. The yellow areas control the manipulator's polar coordinate β, turning the robot arm left or right. Further, the blue sections control the manipulator's radius, r, where the upper blue area increases it, and the smaller region reduces it. Figure 7 displays a few user commands in action.

4 Experimental Setup

4.1 Overview

Figure 8: The Robotis OpenManipulator-X and a custom made lever

Figure 9: The plywood camera stand for the stereoscopic depth sensor

The manipulator was mounted on a wooden platform, together with a lever constructed by the author of [20]. A motor is connected to the lever, but it is not employed in this thesis. The lever is approximately $20cm$ in front of the manipulator, along its X^0-axis. Figure 8 depicts both the lever and the manipulator. Further, Figure 9 exhibits a stand for the stereoscopic depth sensor, made out of plywood. Three grooves can be seen in the vertical arm, at approximately $58cm$, $72cm$, and $85cm$. Experiments were carried out to determine at what distance the depth sensor worked optimally, but in the end, the middle notch yielded the most reliable hand tracking.

Ubuntu 20.04.3 LTS was selected as the operating system due to dependencies by the provided control system for the OpenManipulator-X and ROS 2. The decision of using ROS 2 instead of ROS was made, as ROS 2 supports Python 3.5, while ROS only has support for Python 2 [39], which the author is most familiar with. Table 3 lists all the dependencies necessary to run the HMI code from this project. The HMI, named Tyr, is accessible from the author's GitHub at [40]

Name	**Version**
Ubuntu	20.04.03 LTS
ROS 2	FOXY
DynamixelSDK	3.7.40
Dynamixel Workbench	2.2.1
OpenManipulator	2.0.1
OpenManipulator Msgs	1.0.1
OpenManipulator Dependencies	-
Robotis Manipulator	1.1.1
Python	3.5
PyQt	5.9.2
MediaPipe	0.8.9.1
OpenCV	4.5.3.5
Numpy	1.20.3
Scipy	1.7.1
Matplotlib	3.4.3
QDarkStyle	3.0.2
Intel RealSense SDK 2.0	2.49.0

Table 3: The packages used to develop the HMI. A guide for installing the ROS 2 packages can be found in the OpenManipulator-X e-Manual [41]

4.2 Robotic Manipulator

The robotic manipulator OpenManipulator-X was used in the experimental setup. The manipulator is cost-effective, consisting of both open-source software and hardware. The software is based on ROS, while most of the manipulator parts are available as STL CAD models, allowing for 3D printing. The manipulator is mounted on a wooden platform, as seen in Figure 8.

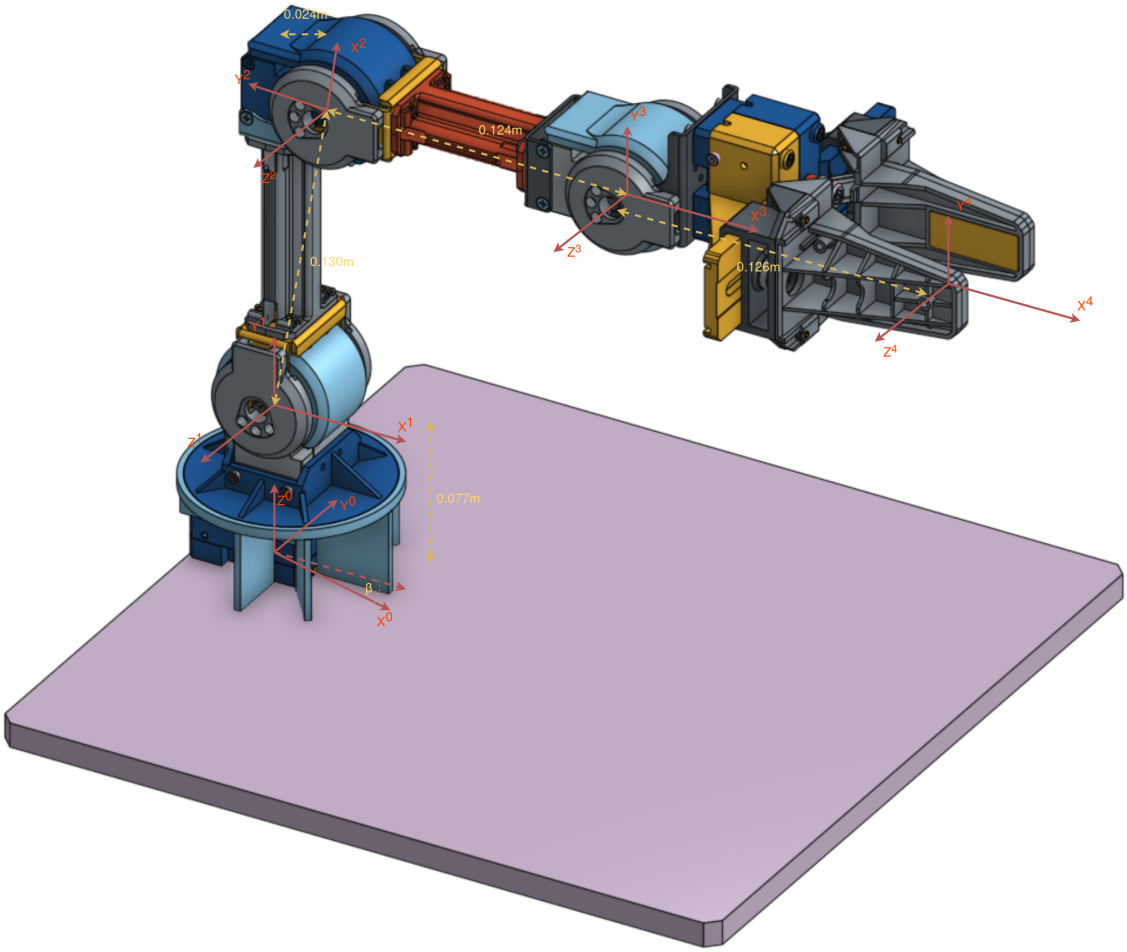


Figure 10: A model of the robotic manipulator used in this project

Figure 10 displays a model of the manipulator, with DH frames drawn for all joints. Note that both the inverse- and forward kinematics are handled by the included ROS packages, and the actual frames are unknown to the author. From tests, it became clear that both the end-effector frame represented by X^4, Y^4, Z^4, and the 0 frame X^0, Y^0, Z^0 are correctly drawn in the model. The author has drawn the frames in between according to the DH-convention. Table 4 lists the parameters in the figure. All numerical values are extracted from the OpenManipulator-X e-Manual [41]

Link	θ_i	d_i	a_i	α_i
1	$\theta_1 = \beta$	0.077m	0	$\frac{\pi}{2}$
2	$\frac{\pi}{2} - arcsin(\frac{0.024}{0.130}) + \theta_2$	0	0.130m	0
3	$\theta_3 - \frac{\pi}{2}$	0	0.124m	0
4	θ_4	0	0.126m	-$\frac{\pi}{2}$

Table 4: Caption

Further, the manipulator is connected to the computer running the HMI through a USB communication converter, U2D2. Table 5 lists the OpenManipulator ROS topics, which are used by the Controller module, discussed in Section 3.6.

Topic	Type
/open_manipulator/gripper/kinematics_pose	open_manipulator_msgs/KinematicsPose
/joint_states	sensor_msgs/JointState
Service	**Type**
/open_manipulator/goal_joint_space_path_to_kinematics_position	open_manipulator_msgs/SetKinematicsPose
/open_manipulator/goal_joint_space_path_to_kinematics_orientation	open_manipulator_msgs/SetKinematicsPose
/open_manipulator/goal_tool_control	open_manipulator_msgs/SetJointPosition
/open_manipulator/goal_joint_space_path	open_manipulator_msgs/SetJointPosition

Table 5: The ROS topics and services for the OpenManipulator-X, supplied by Robotis, which were used by the HMI to communicate with the manipulator.

4.3 Depth Sensor

A depth sensor is used together with the hand tracking software to utilize depth information from the hand. Seen in Figure 11 is the stereoscopic sensor, Intel RealSense Depth Camera D435. It is powered by a USB 3.0 cable and consists of a pair of depth sensors, one plain RGB camera, and an infrared projector. Further, the camera has a wide depth field of view of 87° x 58° and a range of up to 10 meters. But since we are relying on the RGB sensor information when performing hand tracking, we are constrained by the RGB sensor's FOV, which is FOV of 69° x 42°. The IR projector is handy for improving depth accuracy in environments with lower texture gradients.

Figure 11: The stereoscopic depth camera

Because the depth and RGB sensor are not perfectly overlapping with the depth sensors, we cannot directly extract depth information from pixels in the RGB image. Luckily, Intel has supplied the depth camera with an SDK: Intel RealSense SDK 2.0. Using the lines of Python code in Appendix A Listing 1, we can properly align the depth information to fit onto the RGB frame.

The dept sensor supports ROS as well, but the framegrabbing module was readily implemented in Python with the SDK. One can argue that it would have been advantageous to extract data from the sensor with ROS, as the code would become more modular, allowing for easier maintenance and support for other depth sensors in the future. However, these points were not considered essential for this project, and a more straightforward approach was implemented.

5 Results

5.1 Control Systems

Controlling the robotic manipulator with a simple velocity P-controller and human hand movement was surprisingly easy. Reasons for this may be because the controller was only setting the velocity, which is a more straightforward task than setting an exact position. The more complicated task of determining the correct position was left to the human operator. The human operator becomes a natural part of the control system by choosing this approach, as is visualized in Figure 3.

A naive control system with the intent of making the manipulator control feel more fluid, natural, and fast is presented below. Then a redesigned approach addressing the drawbacks of the first system is discussed before the final improved method is reviewed.

First control system
In the first approach, the manipulator followed the operator's hand along the Z^0-axis using measured hand depth. A range of 30 to 50 cm away from the camera was directly mapped to the manipulator's operational height range by a simple proportional position controller. Simultaneously, a proportional velocity controller mapped the hand's position along the X^W-axis of the camera image to a rotational speed in the robot's X^0Y^0-plane. Another P-controller mapped the hand position from the Y^W-axis in the image plane to a rate value, changing the radius of the manipulator. In theory, such a control system should allow fast and accurate manipulator control. It was, however, difficult to control the manipulator as the user intended. The author noticed that the system performed well when controlling either height or (X^0, Y^0) position separately but not simultaneously. By maintaining the hand at a constant depth distance from the camera, the user achieved control in the X^0Y^0-plane at a satisfactory level. Keeping the hand position along the X^W- and Y^W-axes fixed while controlling the manipulator's height, were also deemed adequate.

Second control system
The lessons learned from the first control system were utilized when designing the second iteration of the control system. The system was similar to the first but used finger angles to detect when the user had their thumb open and the remaining four fingers closed. The thumbs-up gesture was used to deactivate the height controller and activate the velocity controllers for the polar coordinates. It was a significant improvement and the manipulator became easier to control to intended locations. Yet, some issues were still apparent. Firstly, the manipulator could not tilt its end-effector, nor could it open or close the gripper with this control system. Thus work was done on adding a set of different hand gestures to swap between system states. It soon became clear that the hand gestures could not be too similar, as the system had trouble differentiating between them. Further, to rotate the manipulator, the operator was required to move his hand somewhat unnaturally across the image plane, resulting in a stiff shoulder for the author. Figure 12 depicts how the operator workspace was designed for the second generation control system.

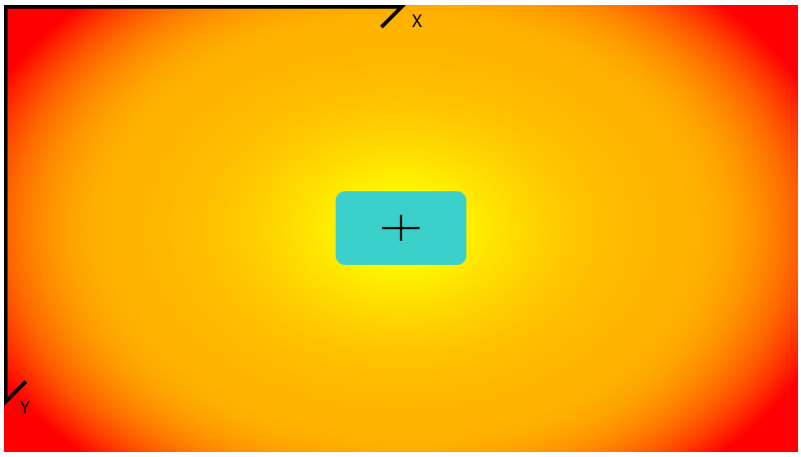


Figure 12: Workspace design for second generation control system. The blue rectangle represents a desired speed of 0 m/s, while any deviations along x or y axis would adjust the desired velocity accordingly

Third control system
To solve the issues with the second control system, an effort was made to design more comfortable hand gestures that remained dissimilar to avoid instabilities and unintended manipulator movement. The operator panel resulting from the third iteration of the control system was thoroughly discussed in Section 3.7. The green and yellow sections were shaped like half-circles, which allowed the operator to keep his shoulder and elbow more or less stationary when performing rotations, greatly increasing the comfort.

5.2 Lever Manipulation Challenge

5.2.1 Results

To verify the HMI has acceptable performance, a task was designed where the robotic manipulator would pull a stationary lever. The resulting trajectory can be seen in the polar plot of Figure 13. The lever is mounted approximately at 0° and 27 cm out from the robot. Further, the path is colored according to the time spent performing the task. Figure 14 depicts the manipulator's pose, as well as the states of the FSM that were used during run-time. The first row is the height of the end-effector, given in meters. The following two rows are the polar coordinates of the end-effector in the X^0Y^0-plane The angle β is the angular position of the first joint of the manipulator. The next row contains the orientation of the end-effector, represented as euler angles on the Tait–Bryan from: yaw (ψ), pitch (θ), and roll (ϕ). The last row depicts the state of the system at time t. The datapoints are all synchronized according to the time steps on the X-axes.

The reader is advised to watch the demonstration video of the HMI in action, see [42].

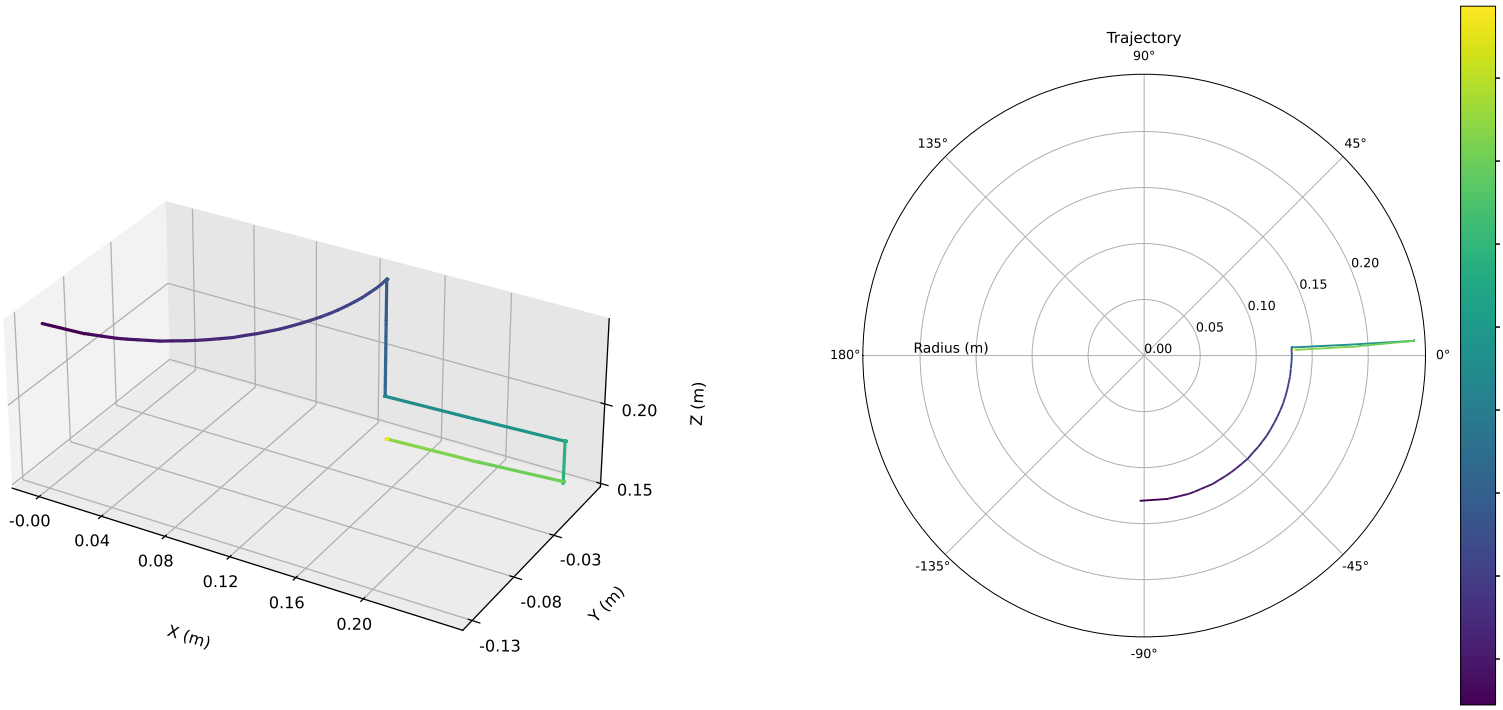


Figure 13: Manipulator trajectory plotted in a 3D cartesian environment, and in a 2D polar coordinate system. The color is dependent of time.

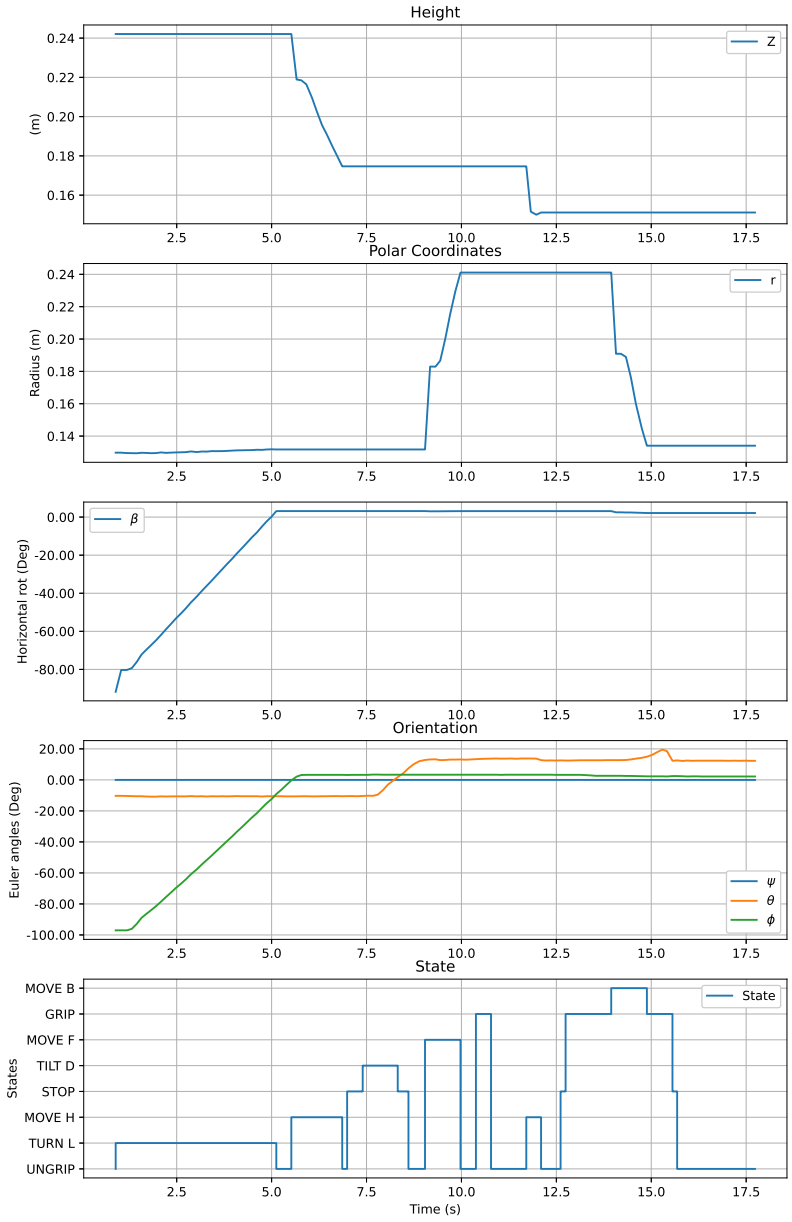


Figure 14: Manipulator pose and states plotted synchronously. Some state names have been abbreviated, and only states visited are shown.

5.2.2 Discussion

The plots depict the responsiveness of the HMI, and one can readily verify that when the user activates the *TURN LEFT* state, the manipulator turns left. As expected, β increases during the length of the state, five seconds. However, ϕ is also similarly affected as β. The manipulator is not equipped with a gyro and does not measure its yaw. Further, the only actuator capable of

adjusting the yaw of the manipulator is the first joint. Thus ϕ should be identical to β. However, this is not the case and a slight time delay between the angles can be seen in the plot. The angles are calculated from different values, which are likely updated at different times. The author has not delved deep into how the ROS topics work, and the reader should consider it an educated guess. Further, ψ remains at 0 due to OpenManipulator-X only having five DOF, lacking a sixth controllable variable.

Next, the manipulator responds to the *MOVE HEIGHT* state. The translational dynamics can be explained by the user dynamically adjusting the depth of his hand. The user then tilts the end-effector downwards, which can be seen in θ at approximately $t = 7.5s$. Following, the radius is increased when the *MOVE FORWARD* state is activated. The radius increases rapidly at first, but then at a lesser pace when the user extends his pinky finger. At this point, $t = 10s$, the manipulator is situated just above the lever, as the user activates the *GRIP* state. The manipulator is, however, not gripping the lever as the user intends, and the user transitions to the *UNGRIP* state before lowering the end-effector a few centimeters with the *MOVE HEIGHT* state. At approximately $t = 12.5s$ the user again grips the lever, before pulling it backwards at around $t = 14s$.

The forces applied on the manipulator by the lever from the dynamics of its circularly constrained movement is not modeled. Thus when the robot moves backward while gripping the lever, it is being pulled downwards, following the motion of the lever. This effect is noticed when the *UNGRIP* state is activated, as θ experiences a small change, which is soon corrected. A slight disturbance can also be seen in β when the manipulator is pulling on the lever.

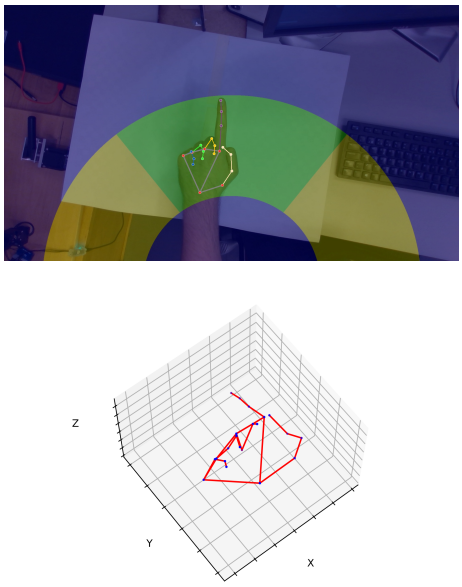
Being experienced in the system, the author managed to move the manipulator to the lever and pull it down from an upright position in under 15 seconds. The results prove the system certainly has a quick reaction time in the hands of an experienced operator. Further, it illustrates how operators can use the system to perform tasks, such as pulling levers and pushing buttons in inaccessible locations, using a manipulator mounted on some remotely operated vehicle.

5.3 Explainable Artificial Intelligence Methods

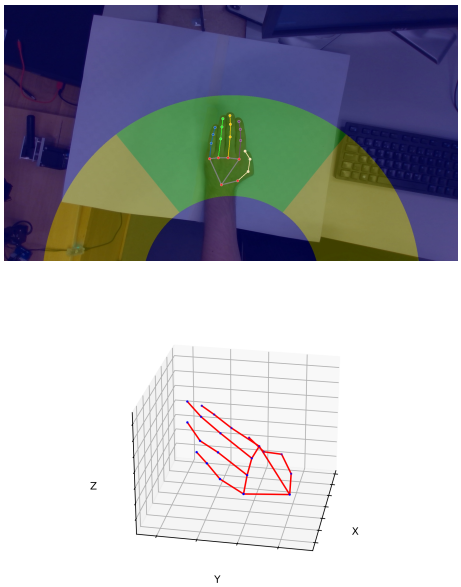
5.3.1 Results

Presented in Figure 15, are images of all seven hand gestures in the operator workspace, with the corresponding 3D hand skeletons. The skeletons are easily distinguishable.

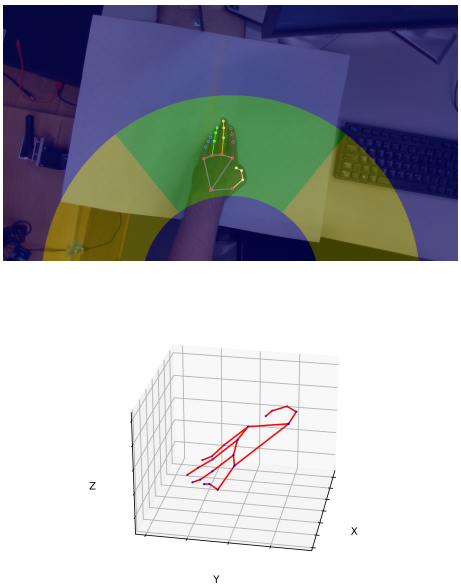
The HMI system was tested on five candidates including the author. The four other candidates had no prior experience with control interfaces for robotic manipulators but had backgrounds from STEM fields. The operators quickly learned the different hand gestures and managed to control the manipulator to the intended locations. Minor problems arose when performing more demanding tasks, such as the lever manipulation challenge. Most of the candidates performed the job within a few minutes, but one candidate failed to pull the lever within a reasonable amount of time. The one who failed became confused and kept moving their hand out of the green section while tilting or adjusting the height.



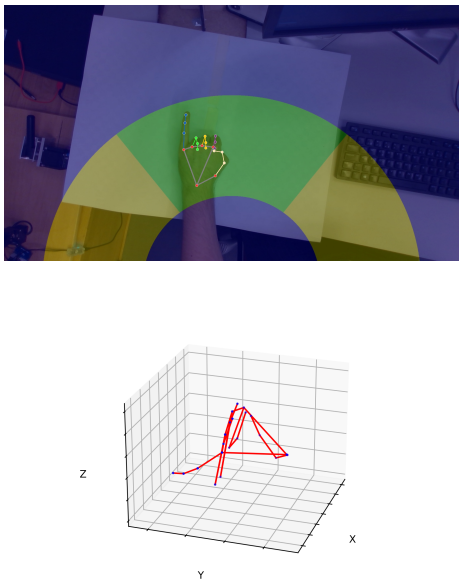
(a): Up / Down



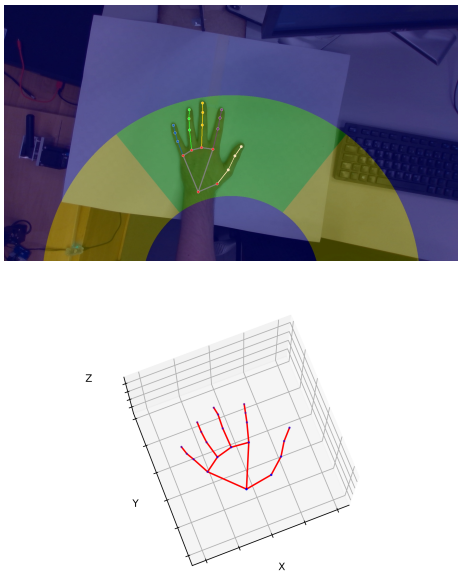
(b): Tilt Up



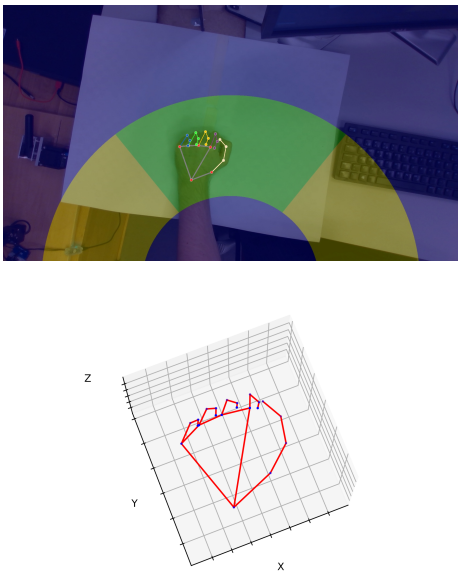
(c): Tilt Down



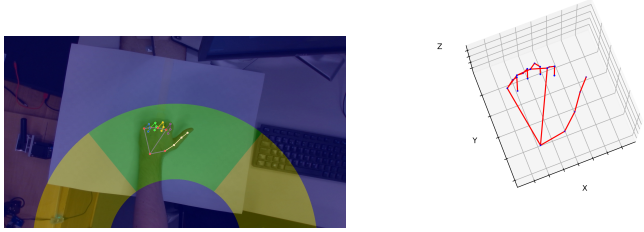
(d): Slow



(e): Ungrip



(f): Grip



(g): Stop

Figure 15: All possible hand gestures visualized in the operator workspace, with corresponding 3D skeletons

It was noticed that the stereoscopic depth sensor sometimes faulted and produced depth images high in noise. Though easily corrected by replugging the camera, this error was quickly recognized by interpreting the 3D hand skeleton, which became unrecognizable. The 2D hand skeleton drawn in the operator workspace did not reveal this vital information. See Figure 16. Further, the validity of the synthetic depth information produced by the MediaPipe hand tracking AI was tested. To the author's surprise, switching from measured depth to synthetic depth had minor effects on the control system. Most notably, the threshold angles used for tilt control had to be retuned, and the measured depth information was still necessary for height control. However, nothing had to be changed to calculate finger angles and determine the remaining hand gestures.

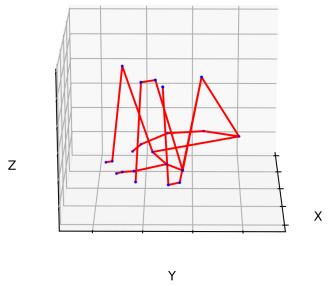


Figure 16: 3D Skeleton with unreliable depth information

The 2D skeleton drawn in the operator workspace did not overlap perfectly with the user's hand on some occasions, causing the measured depth to be that of the table underneath the hand instead. The synthetic depth proved a more stable source of information in these cases. On further inspections of the 3D skeleton, the author noticed that the model sometimes could not decide whether the fingertips were located above or below the palm in Z^w-direction.

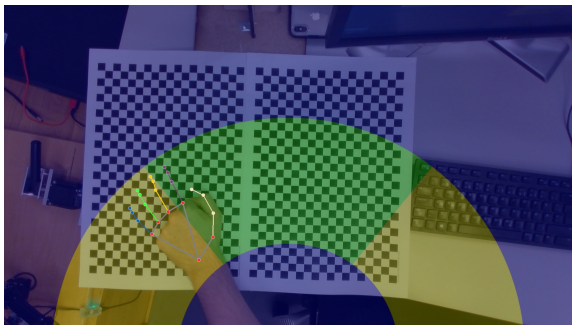
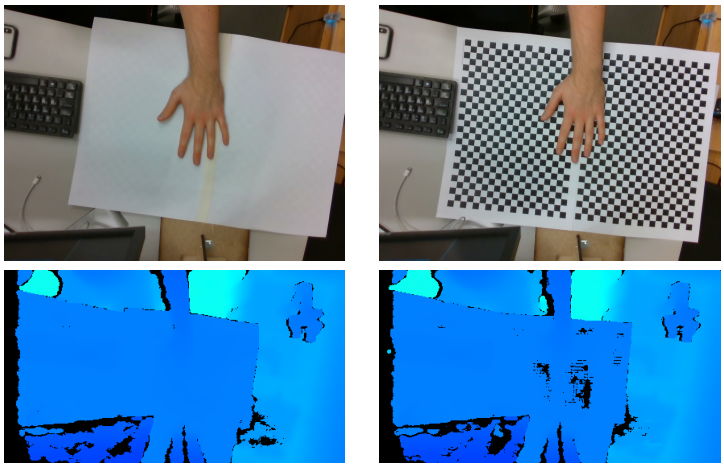


Figure 17: Operator workspace with a checker background, showing a misdetection

Different backgrounds were tested because the stereoscopic depth sensor relies on features visible in two cameras overlooking the same scene. At first, the wooden stand holding the sensor and the table served as background. A checkerboard pattern was printed and placed underneath the sensor to add more features to the scene. The results were not promising, and a higher frequency of misdetections was noted, see Figure 17. Further, the black pixels in the depth image show that the sensor could not be trusted either, as it erroneously gave parts of the background a depth of 0 cm, as seen in Figure 18 (b). A white background with few detectable features gave the most reliable results. In this case, the machine learning model yielded more stable outputs, but also the depth information was more trustworthy, as depicted in Figure 18 (a).



(a) Featureless background (b) Checkerboard background

Figure 18: The effects of a checkerboard background in the operator workspace. RGB images on top and depth images under. Black pixels represents a distant of 0 cm

5.3.2 Discussion

The hand skeletons serve multiple purposes. Firstly, it explains fast and accurately where the MediaPipe machine learning model has estimated the locations of the hand joints by drawing them on top of the operator's hand. The user can thus visually verify that the system behaves as expected. However, the skeleton drawn in the operator workspace does not explain if the system interprets depth correctly. Depth information is crucial since it is utilized when controlling the manipulator's height, tilting angle, and determining each finger's angles. That is the reasoning behind implementing a 3D hand skeleton, as seen in Figure 15. It is used extensively in this section to discuss the results and illustrate how different levels of XAI are helpful for various end-users.

Inspection of the 3D hand skeleton revealed that the machine learning model correctly predicted the relative depths of the hand joints, albeit the synthetic numbers had no physical meaning in the same sense as the measured values. The constructed information was still good enough to differentiate between gestures. Further, even when the model failed at precisely estimating the hand position, the finger angles would not change as drastically as when they relied on measured depth. On the other hand, the tilting maneuver became less stable when using synthetic information, as the model could sometimes rapidly switch between tilting up and down. The effect was readily visible in the 3D skeleton. Thus, we can conclude that relying on synthetic or measured depths worked well in practice, with some occurrences of unstable estimates for both methods.

A reason for the failure of the checkerboard background could be that the overload of features caused the machine learning model to predict extended fingers wrongfully. Intuitively, the machine learning model could better estimate hand locations when not distracted by the background features. The datasheets for the depth sensor [43] revealed that the stereoscopic camera uses an infrared sensor to project a grid pattern onto the background, which supplies the cameras with detectable features even when the scene is featureless, which explains why the depth sensor works well on a blank background.

The results show how different levels of end-users can benefit from XAI methods. The developer and author of this thesis appreciated the additional intuition from the hand skeletons, which helped at debugging the system and evaluating the quality of the data. Further, the non-developers of the system quickly learned the hand gestures to control the manipulator, by observing the GUI.

6 Conclusion

The research questions from Section 1.2 will be revisited and answered in this section. Finally, the chapter encompasses a discussion of further work, including suggestions to improve the HMI and work that will be visited in the author's MSc thesis next semester.

6.1 Answering the Research Questions

Question one
The first research question is readily answered. With modern hand tracking software powered by CNNs, it is possible to construct an HMI system based solely on hand movements and gestures. It is, however, important to debate the quality of the system. Therefore we shall provide arguments based on the six concepts from Section 2.4, including intuitiveness. For each concept, the HMI will be given a score of either *Low*, *Medium*, or *High*

Intuitiveness
Score: Medium
As the candidates spent a lot more time completing the lever manipulation challenge than the author, there is certainly still work to be done on making the HMI more intuitive. However, since the candidates were not trained and only given a couple of attempts each, the level of intuitiveness can be considered adequate, as four out of five successfully pulled the lever.

Comfort
Score: High
The system allows the user to rest their hand on the table while controlling the manipulator and limits stress on the shoulder and arm. Further, the hand gestures feel natural and do not cause any noticeable strains on the hand joints. Thus one can argue that the level of comfort is high.

Come as you are
Score: High
An operator would need a cost-effective depth camera, a computer, and a monitor to successfully control the robotic manipulator with the HMI software. Though the depth camera reduces the level of *come as you are*, it poses no additional limitations such as gloves, markers, or IMU's the user would need to wear. Considering the other attempts at creating hand gesture-based HMIs, the method proposed in this thesis have a high level of *come as you are*.

Reconfigurability
Score: High
The system has a high level of reconfigurability due to measures being taken by the developers of the MediaPipe Hand API, ensuring the hands of people from a multitude of ethnicities, ages, and genders can be tracked.

Interaction space
Score: Medium
To estimate the hand gestures, no limitations are posed on where the operator needs to place their hand. However, the operator workspace is static with pre-designed regions. Though such an implementation allows the system to take advantage of both the user's hand gesture and palm location, it is a downside as some users got confused by the non-intuitiveness of the workspace regions.

Gesture spotting and immersion syndrome
Score: High
The system readily separates between hand gestures. It does not consider temporal gestures but decides the gesture from which fingers are extended, allowing for reliable estimates.

Question two
The second research question can be answered more straightforwardly, referring to the results of Section 5.2. An experienced user managed to solve the lever manipulation challenge in approx-

imately 15 seconds, proving that practical tasks may be solved with the use of the HMI. Such an application could prove helpful in remote control of robotic manipulators in unknown environments. Mounted on a UAV it could be used to clear power lines by having the operator pick up branches. Further, astronauts on future Lunar and Martian outposts could employ the technology to perform trivial but unforeseen tasks outside the habitat without venturing out into the harsh environments.

Question three
The third research question can be answered by using the results from Section 5.3. The online XAI scheme, suggested in this thesis, is an insightful representation of the input- and output data in relation to each other, in a manner that is helpful for both the developer and operator. The hand predictions of the AI are drawn as skeletons on top of the inputted image, which is displayed to the operator through a GUI application. Further, the developer benefitted from a 3D skeleton, created from the prediction, and depth measurements, to detect anomalies and logical bugs. The XAI method is problem-specific. Still, it proves that it is possible to gain an understanding of a black-box model by a representation of input- and output data alone.

6.2 Further Work

The system's intuitiveness could be improved by designing a better operator workspace and hand gestures. Another approach could be to make the system more robust to operator-generated errors, such as an operator misplacing their hand in the workspace, resulting in unintended movement.

In the author's MSc thesis next semester, the system will be extended to address some of the problems regarding intuitiveness. An interesting approach would be to implement the results from [20] into a semi-automatic control system, where the manipulator would solve the lever manipulator challenge on its own when ordered by a simple hand gesture. Such a high level of HRI would have many potential use cases.

Further, in a real-world scenario, the object to be manipulated would not be mounted on the same platform as the robotic arm and would experience environmental forces different from the robot. Thus to increase the number of use-cases for such a technology, an exciting solution could be to compensate for said forces by the use of reinforcement learning.

Bibliography

[1] Iwan Ulrich, Francesco Mondada and J-D Nicoud. 'Autonomous vacuum cleaner'. In: *Robotics and autonomous systems* 19.3-4 (1997), pp. 233–245.

[2] Haydar Sahin and Levent Guvenc. 'Household robotics: autonomous devices for vacuuming and lawn mowing [applications of control]'. In: *IEEE Control Systems Magazine* 27.2 (2007), pp. 20–96.

[3] Vincent Van Roy, Daniel Vertesy and Giacomo Damioli. 'AI and robotics innovation'. In: *Handbook of Labor, Human Resources and Population Economics* (2020), pp. 1–35.

[4] Peter Vamplew. 'Recognition of sign language gestures using neural networks'. In: *European Conference on Disabilities, Virtual Reality and Associated Technologies* (1996).

[5] Brandi House, Jonathan Malkin and Jeff Bilmes. 'The VoiceBot: a voice controlled robot arm'. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2009, pp. 183–192.

[6] Hong Zeng et al. 'Semi-autonomous robotic arm reaching with hybrid gaze-brain machine interface'. In: *Frontiers in neurorobotics* 13 (2020), p. 111.

[7] Siddharth Narayanan and C Ramesh Reddy. 'Bomb defusing robotic arm using gesture control'. In: *International Journal of Engineering Research and Technology* 4.02 (2015).

[8] Fan Zhang et al. 'MediaPipe Hands: On-device Real-time Hand Tracking'. In: *Google Research* (2020).

[9] Murtaza's Workshop - Robotics and AI. *AI ROBOT ARM using Python Arduino OpenCV CVZone — Computer Vision*. https://www.youtube.com/watch?v=7KV5489rL3c. Published: 2021-05-11.

[10] Karan Singh George ElKoura. 'Handrix: Animating the Human Hand'. In: *Eurographics/SIGGRAPH Symposium on Computer Animation* (2003).

[11] Shadow Robot Company. *Shadow Robot Company*. URL: https://www.shadowrobot.com/ (visited on 9th Nov. 2021).

[12] Marco Tulio Ribeiro, Sameer Singh and Carlos Guestrin. '"Why Should I Trust You?": Explaining the Predictions of Any Classifier'. In: (2016).

[13] Scott M Lundberg and Su-In Lee. *A Unified Approach to Interpreting Model Predictions*. Curran Associates, Inc., 2017, pp. 4765–4774. URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[14] Vilde B. Gjærum et al. 'Explaining a Deep Reinforcement Learning Docking Agent Using Linear Model Trees with User Adapted Visualization'. In: *Journal of Marine Science and Engineering* (2021).

[15] Wikipedia. *Finger tracking*. URL: https://en.wikipedia.org/wiki/Finger_tracking (visited on 9th Nov. 2021).

[16] Leonid B. Freidovich. *MODELLING IN ROBOTICS and CONTROL METHODS FOR ROBOTIC APPLICATIONS*. Umeå University, 2021.

[17] Tom Mitchell. McGraw Hill. *Machine Learning*. 1997. URL: http://www.cs.cmu.edu/~tom/mlbook.html.

[18] Wikipedia. *Artificial Neural Networks*. URL: https://en.wikipedia.org/wiki/Artificial_neural_network (visited on 10th Nov. 2021).

[19] David Silver et al. 'Mastering the game of Go with deep neural networks and tree search'. In: *Nature* 529 (2016), pp. 484–489.

[20] Sindre B. Remman. 'Robotic manipulation using Deep Reinforcement Learning'. In: (2020).

[21] Svajone Bekesiene, Rasa Smaliukiene and Ramute Vaicaitiene. 'Using Artificial Neural Networks in Predicting the Level of Stress among Military Conscripts'. In: *Mathematics* 9.6 (2021). ISSN: 2227-7390. URL: https://www.mdpi.com/2227-7390/9/6/626.

[22] M.V. Valueva et al. 'Application of the residue number system to reduce hardware costs of the convolutional neural network implementation'. In: *Mathematics and Computers in Simulation* 177 (2020), pp. 232–243.

[23] Google. *MediaPipe GitHub*. URL: https://google.github.io/mediapipe/ (visited on 16th Dec. 2021).

[24] Ian Sample. *Computer says no: why making AIs fair, accountable and transparent is crucial*. URL: https://www.theguardian.com/science/2017/nov/05/computer-says-no-why-making-ais-fair-accountable-and-transparent-is-crucial (visited on 10th Nov. 2021).

[25] Juan Pablo Wachs et al. 'Vision-based hand-gesture applications'. In: *Communications of the ACM* 54 (2011), pp. 60–71.

[26] Cambridge Dictionary. *Definition of 'intuitive'*. URL: https://dictionary.cambridge.org/dictionary/english/intuitive (visited on 14th Dec. 2021).

[27] Miko Nore and Caspar Westerberg. 'Robotic Arm controlled by Arm Movements'. In: *Bachelor's Thesis at ITM* (2019).

[28] Poltak Sihombing, Muhammad Rifky B and Elviwani Herriyance. *Robotic Arm Controlling Based on Fingers and Hand Gesture*. Medan, Sumatera Utara, Indonesia, 2020.

[29] Shamsheer Verma. 'Hand Gestures Remote Controlled Robotic Arm'. In: *Advance in Electronic and Electric Engineering (AEEE)* 3 (2013), pp. 601–606.

[30] Love Aggarwal, Varnika Gaur and Puneet Verma. 'Design and Implementation of a Wireless Gesture Controlled Robotic Arm with Vision'. In: *International Journal of Computer Applications* 79 (2013).

[31] Ganesh Choudhary B. and Chethan Ram B. V. 'Real Time Robotic Arm Control Using Hand Gestures'. In: *2014 International Conference on High Performance Computing and Applications (ICHPCA)* (2014).

[32] Shinichi Ikegami et al. 'A Study on Mobile Robot Control by Hand gesture Detection'. In: *2018 3rd International Conference on Information Technology Research (ICITR)* (2018).

[33] Daniel Skomedal Breland. 'Hand Gestures Recognition using Thermal Images'. In: *Master's thesis in Information- and communication technology (IKT590) at University of Agder* (2021).

[34] Hairong Jiang and Juan P. Wachs. 'Integrated vision-based system for efficient, semi-automated control of a robotic manipulator'. In: *International Journal of Intelligent Computing and Cybernetics* 7.3 (2014).

[35] Jiacun Wang. *Formal Methods in Computer Science*. CRC Press, 2019.

[36] Google. *Model Card: MediaPipe Hands*. URL: https://drive.google.com/file/d/1-rmIgTfuCbBPW_IFHkh3f0-U_lnGrWpg/preview (visited on 4th Nov. 2021).

[37] Hongyi Xu et al. 'GHUM & GHUML: Generative 3D Human Shape and Articulated Pose Models'. In: *Google Research* (2020).

[38] The Qt Company. *Qt application framework*. URL: https://www.qt.io/ (visited on 11th Dec. 2021).

[39] Dirk Thomas. *Changes between ROS 1 and ROS 2*. URL: http://design.ros2.org/articles/changes.html (visited on 14th Dec. 2021).

[40] Simon J. N. Lexau. *Tyr - An HMI for Robotic Control Through DCNN Powered Hand Tracking*. URL: https://github.com/Nagelsaker/Tyr (visited on 14th Dec. 2021).

[41] Robotis. *OpenManipulator-X e-Manual*. URL: https://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/ (visited on 14th Dec. 2021).

[42] Simon J. N. Lexau. *Tyr video demonstration*. URL: https://www.youtube.com/watch?v=2q_15t-pAeA (visited on 19th Dec. 2021).

[43] Intel. *Intel Realsense D400 Series Datasheet*. URL: https://www.intelrealsense.com/wp-content/uploads/2020/06/Intel-RealSense-D400-Series-Datasheet-June-2020.pdf (visited on 11th Dec. 2021).

Appendix

A Code Examples

Listing 1: Python code for extracting and aligning depth information with RGB images from the Intel RealSense Depth Camera

```
import pyrealsense2 as rs
import numpy as np

def main():
    pipeline = rs.pipeline()
    # Configure streams
    config = rs.config()
    config.enable_stream(rs.stream.depth, 1280, 720, rs.format.z16, 30)
    config.enable_stream(rs.stream.color, 1920, 1080, rs.format.bgr8, 30)

    profile = pipeline.start(config)

    depthSensor = profile.get_device().first_depth_sensor()
    depthScale = depthSensor.get_depth_scale()

    alignTo = rs.stream.color
    align = rs.align(alignTo)

    # Get frameset of color and depth
    frames = pipeline.wait_for_frames()

    # Align the depth frame to color frame
    alignedFrames = align.process(frames)

    # Get aligned frames
    alignedDepthFrame = alignedFrames.get_depth_frame()
    colorFrame = alignedFrames.get_color_frame()

    pipeline.stop()
```

B Decision Tree

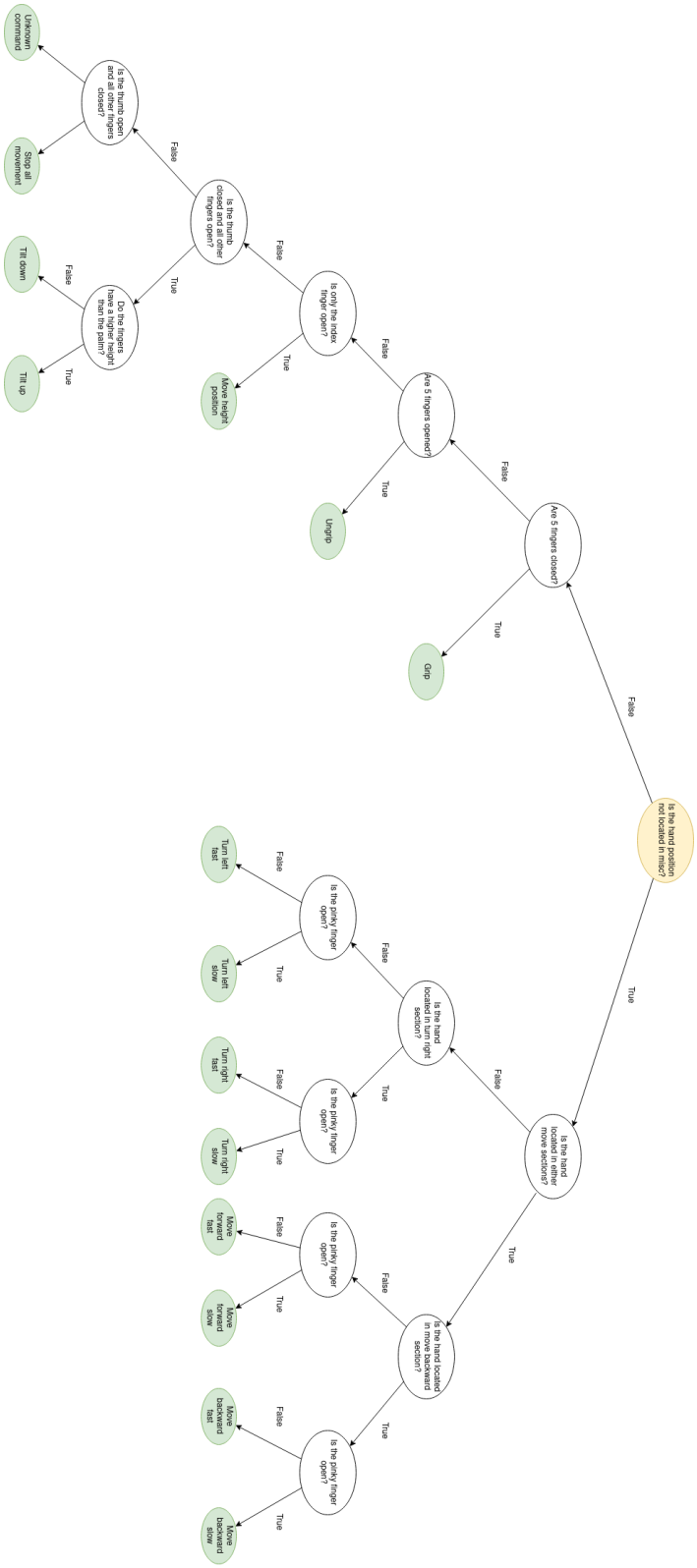


Figure 19: The DT analogy to the FSM Diagram.

C Class Diagram

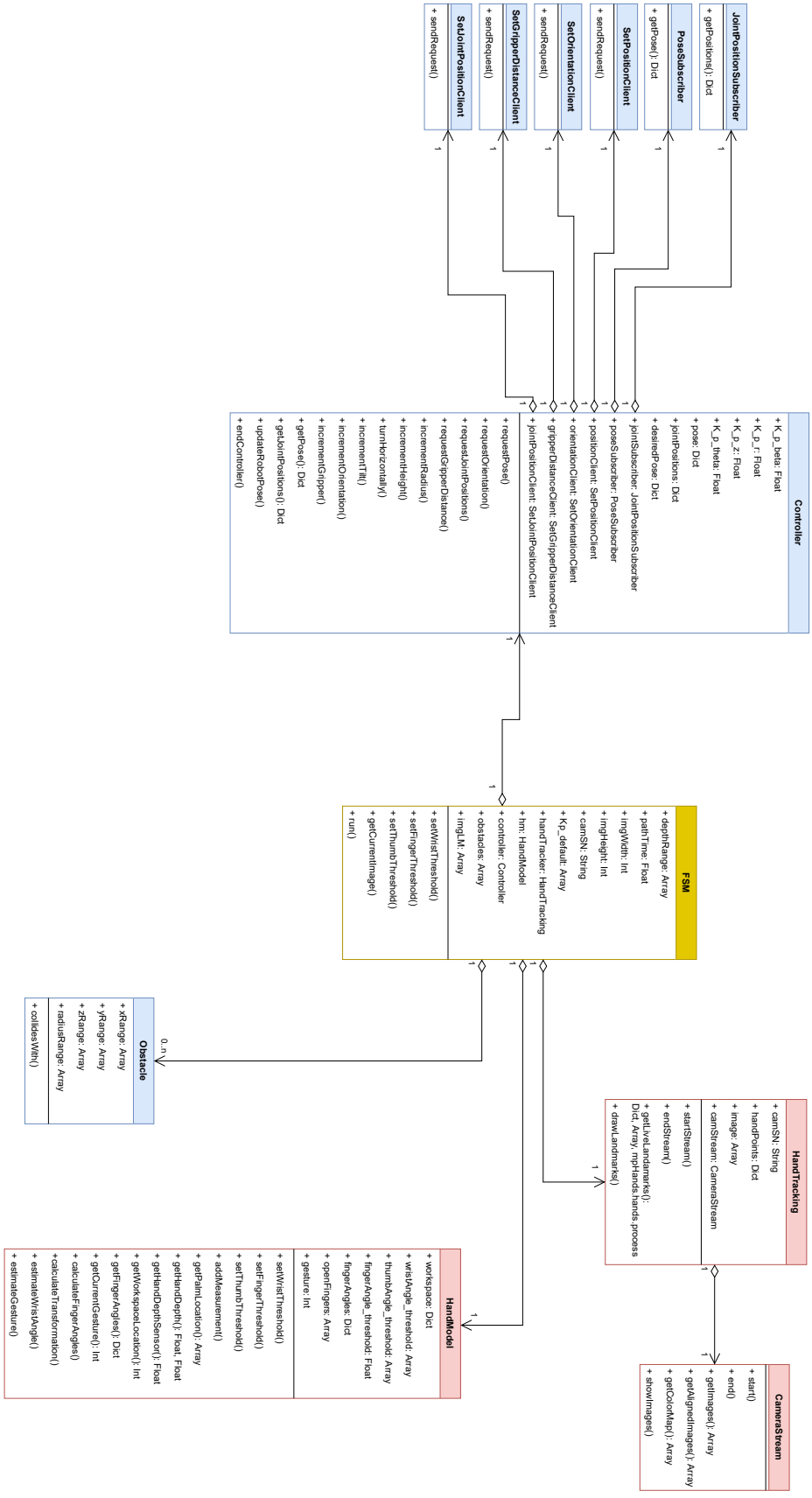


Figure 20: Class diagram depicting the most important parts of the system.