Zurich University of Applied Sciences

Department School of Engineering

Centre for Artificial Intelligence

PROJECT THESIS

# Building a Vision-Based AI Demonstrator with Unitree A1 Quadruped Robot

*Authors:*
Tenzin Samdrup LANGDUN
Martin OSWALD

*Supervisors:*
Prof. Dr. Thilo STADELMANN
Pascal SAGER

Submitted on
December 23, 2022

Study program:
Computer Science

# *Declaration of Authorship*

We, Tenzin Samdrup LANGDUN, Martin OSWALD, declare that this thesis titled, "Building a Vision-Based AI Demonstrator with Unitree A1 Quadruped Robot" and the work presented in it are our own. We confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where we have consulted the published work of others, this is always clearly attributed.
- Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely our own work.
- We have acknowledged all main sources of help.
- Where the thesis is based on work done by ourselves jointly with others, we have made clear exactly what was done by others and what we have contributed ourselves.

Signed: [illegible] [illegible]

Date: 23.12.022

ZURICH UNIVERSITY OF APPLIED SCIENCES

# *Abstract*

Centre for Artificial Intelligence
School of Engineering

Bachelor of Science

**Building a Vision-Based AI Demonstrator with Unitree A1 Quadruped Robot**

by Tenzin Samdrup LANGDUN, Martin OSWALD

One of the challenges in demonstrating the capabilities of Artificial Intelligence (AI) is finding effective ways to showcase its abilities in a tangible manner. In this thesis, we present a vision-based demonstrator for AI using the Unitree A1 quadruped robot. This demonstrator is intended to be used by the Centre of Artificial Intelligence (CAI) at Zurich University of Applied Sciences to exhibit the capabilities of AI in real-life settings, such as expositions. To achieve this, we developed an application that allows the robot to respond to four specific hand gestures. The software receives a live image feed from the integrated camera of the robot and utilizes the MediaPipe framework for hand tracking and landmark points generation, which is displayed on a Remote-PC in real-time. A Logistical Regression model that we trained on 3768 hand gesture recordings then detects a hand gesture made by a user standing in front of the robot. This model communicates with the robot's system, allowing for control over its detection and posture through a user interface. In our laboratory tests, the robot demonstrated an average precision of 91% for each gesture. However, we found that in environments with strong or dim lighting, the robot's performance is less reliable and leads to an accuracy of 70%. To improve the performance of the robot in these conditions, we suggest implementing additional algorithms or finetuning the MediaPipe pipeline. Overall, our demonstrator provides a valuable tool to the CAI department for showcasing AI, as it allows the audience to use intuitive gestures to interact with the robot and experience AI firsthand by seeing the immediate reaction of the robot.

# *Acknowledgements*

We would like to thank Prof. Thilo Stadelmann and Pascal Sager for their valuable guidance and support throughout the project.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **PCA** | **P**rinciple **C**omponent **A**analisys |
| **HMM** | **H**idden **M**arkov **M**odel |
| **AI** | **A**rtificial **I**ntelligence |
| **ML** | **M**achine **L**earning |
| **CV** | **C**omputer **V**ision |
| **ROS** | **R**obot **O**perating **S**ystem |
| **SDK** | **S**oftware **D**evelopment **K**it |
| **ULSDK** | **U**nitree **L**egged **S**oftware **D**evelopment **K**it |
| **API** | **A**pplication **P**rogramming **I**nterface |
| **UI** | **U**ser **I**nterface |
| **LLC** | **L**ow **L**evel **C**ontrol |
| **HLC** | **H**igh **L**evel **C**ontrol |

# 1 Introduction

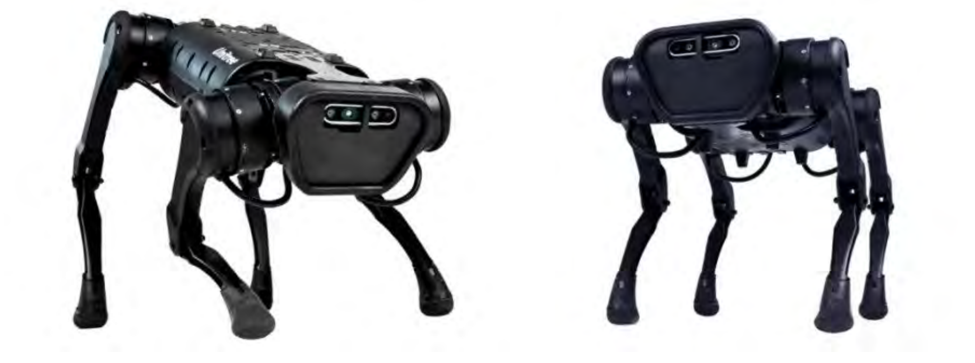


FIGURE 1.1: Unitree A1 quadruped robot [1].

## 1.1 Motivation

Artificial intelligence (AI) has become pervasive in our daily lives, with applications ranging from spam-free email to personalized product recommendations on online shopping sites. From healthcare to finance to transportation, AI is being used to automate tasks, improve efficiency, and provide better services to customers. In fact, it is not easy to think of an industry that AI has not impacted in some way. One of the driving forces behind the growing role of AI is the economic opportunities it presents. A study by PriceWaterhouseCoopers estimated that AI could increase global GDP by 14% or $15.7 trillion by 2030 [2]. Nevertheless, despite its widespread adoption, many people who are not familiar with this technology might not fully understand the capabilities of AI and struggle to identify its usage in their daily lives. In a study conducted by Pegasystems, out of 6000 consumers who were asked whether they have ever interacted with AI, 34% answered "no". However, 84% of the deniers actually had and were unaware of it [3]. One way to address this need for more understanding is to create tangible demonstrators that allow people to interact with AI visually and intuitively. By providing an interactive way for people to experience AI in action, tangible demonstrators can help to make the concept of AI more accessible and concrete for those who may not fully understand it. Seeing the technology in action and interacting with it firsthand could demystify AI and make it feel more familiar and approachable. Such demonstrators could inspire people to learn more about AI and its potential to shape the future in a fun and engaging way.

## 1.2 Problem Statement

This project aims to utilize the A1 robot to create a tangible demonstrator of AI that sparks curiosity and inspires people to learn more about how it functions. By encouraging exploration and questioning, this demonstrator should foster a greater understanding and appreciation for the capabilities and potential of AI. The Unitree A1 quadruped robot provided to us by the

Centre of Artificial Intelligence (CAI) department at the Zurich University of Applied Sciences (ZHAW) is an excellent platform for building such a demonstrator. It can perform dynamic, dog-like movements that can be programmed with an existing Sofware Developer Kit (SDK) provided by Unitree [1]. It is equipped with a High-Definition camera that can be used to construct a system that allows for human-machine interaction. Considering all the factors above, we decided to build a vision-based demonstrator that uses the robot's onboard camera to detect gestures and let the robot respond in real-time with predetermined movement sequences. In addition, the "vision" and hand detection of the robot should be visualized. By seeing the robot recognize and respond to their gestures, users should understand better how AI works and what it is capable of. We aim to provide a concrete example of how AI can be used to enable machines to detect and respond to human inputs.

## 1.3 Work Outline

The first section of the thesis provides an overview of the foundations and related research in this field. The second section details the methods and implementation of the gesture recognition and discusses the choice of technologies and why they were selected based on certain criteria. We also explain our experimental setup in detail, including the hardware and software used, the data collection and annotation process, the evaluation metrics and procedures employed. In the third section, we present the results with a demonstration of its capabilities and discuss any limitations or challenges encountered during the development process. The final section offers a discussion of the implications of the work and suggests potential avenues for future research.

# 2 Foundations and related works

This chapter examines research related to gesture recognition technologies and pose estimation systems. As the aim of the thesis is foremost building a demonstrator and not comparing a variety of models, we focus on the three pose estimation systems that are considered state-of-the-art [4], [5].

## 2.1 Gesture Recognition Technologies

Hand Gesture Recognition belongs to the field of Human-Machine-Interaction. Early approaches used statistical modelings, such as PCA and HMMs [6], as well as Particle Filter and Condensation algorithms [7]–[9]. Modern methods leverage ML, computer vision (CV), Data-Glove, and colored marker technologies [10].

*Data-Glove approach* (2.1a) utilizes gloves fitted with sensors. Researchers use various technologies to read the sensor data, such as magnetic, ultrasonic, optical, and inertial trackers [11]. The gloves provide high accuracy regarding finger/palm location and orientation. Although very precise, this method makes the interaction cumbersome, as special equipment is required [10].

*Colored markers* (2.1b) are special gloves that allow a system to track the hand and locate critical points. These gloves usually have different colors, which allows for image segmentation. Similar to the Data-Glove, this method limits the naturality of the interaction with the system, as it requires wearing gloves. This approach is favorable when absolute precision in orientation is not required [10], [12].

*CV methods* (2.1c) use the user's bare hand to interact with the system, making the interaction easier compared to the Data-Glove and marker approaches [12]. This approach usually utilizes RGB or RGB-D cameras to capture images. Compared to RGB, RGB-D also records depth data. It became viable when Microsoft released the Kinect camera, coincidentally causing a surge in research in gesture and pose recognition using depth images [13]. The primary problem hindering this system is that it needs to generalize to arbitrary situations. This approach can work exceptionally well within well-lit, high-contrast environments [14].

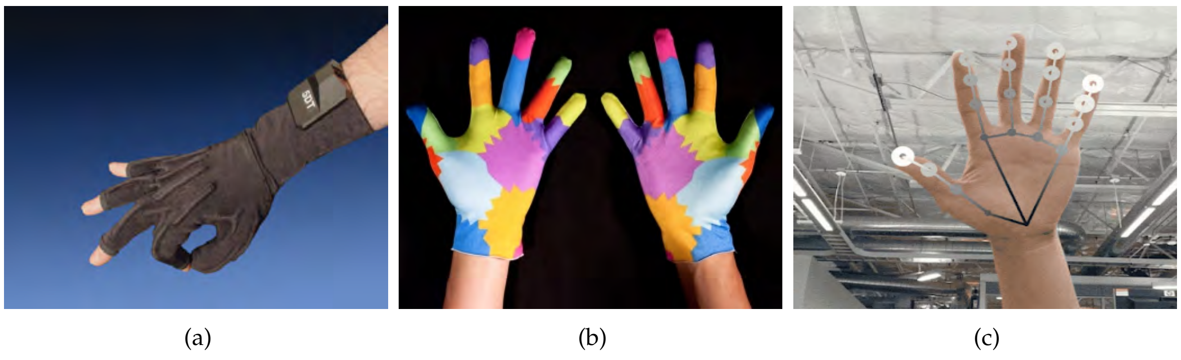

(a) (b) (c)

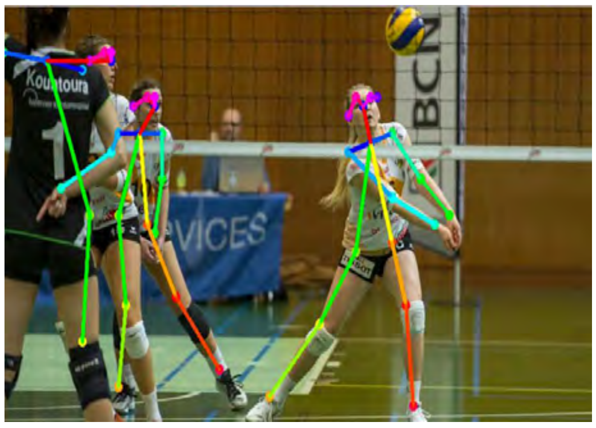
FIGURE 2.1: data capture devices: (a) Data-Glove from [10], (b) Marker Gloves from [12], (c) Hand image from [15]

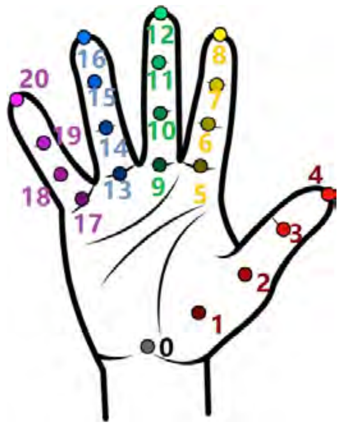
## 2.2 Pose Estimation Systems

The CV methods described in Section 2.1 allow for gesture recognition using only a camera or other visual sensor, eliminating the need for additional equipment like gloves. As this can make the demonstrator more user-friendly and accessible to a broader audience, we solely focus on systems that are built upon this technology. Pose estimation is a field of study in computer vision that aims to determine and track the spacial position of an object. In Human Pose Estimation, the goal is to identify and track key points, such as joints, eyes, head, and hands from images or videos, as shown in Figure 2.2a [16]. Twenty-one key points have been identified for hand pose estimation, as shown in Figure 2.2b. This is the de facto standard as is implemented in the major pose estimation systems [5], [15], [17], [18].

In practice, the top-down and bottom-up methods are employed to address the pose estimation problem: The top-down approach employs an object-detector and a pose-estimator. The object-detector is responsible for detecting human bounding boxes in the image, and the pose-estimator estimates the pose within each bounding box. This means that the pose-estimator is applied to each detected bounding box, and the number of times the pose-estimator is executed is proportional to the number of people in the image. When compared to the bottom-up approach, the top-down approach is often more efficient and accurate, especially when there is a clear view of the subject and the pose is relatively straightforward. This is due to the fact that the object-detector usually only takes a single pass over the image (e.g. YOLOv7 [19]) to extract the bounding boxes. However, it may be less accurate with occlusions and less effective with complex or unusual environmental poses or changes. This is because the object-detector is committed to the location and extent of the bounding box before the pose is estimated, which can lead to inaccuracies if the bounding box is not accurately placed or is too small or large [16], [17].

On the other hand, the bottom-up approach starts from the raw pixel data and works upward to identify and classify the different components within the image. This approach typically involves extracting local features from the image, such as edges, corners, or blobs, and grouping them into larger structures, such as lines, contours, or regions. These structures are then used to identify body parts, such as the head, shoulders, arms, and legs. The bottom-up approach is generally more robust to variations in appearance or changes in the environment, as it relies on the local information in the image rather than external models or prior knowledge. However, this approach may face challenges in grouping body parts when there is considerable overlap between people or when the poses are highly variable or unusual [16], [20], [15].


(a)



(b)

FIGURE 2.2: (a) Human pose estimation with key points and rigid parts from [16]. (b) Hand key points from [17]

In this thesis, we examine pre-trained, state-of-the-art [4], [5] pose estimation systems for our demonstrator instead of implementing a novel solution due to time constraints and resources available for training a pose estimation model from scratch. By leveraging pre-trained models, we can build a functional prototype within a shorter time frame than by implementing a completely novel model.

*AlphaPose* is a real-time human pose estimation system that uses Convolutional Neural Networks (CNNs) and Pose-Guided Attention to identify and estimate poses in images and videos. It is a top-down approach that can run in real-time and achieve high accuracy on benchmark datasets, including COCO [20] and PoseTrack [21]. AlphaPose is able to handle occlusions, making it suitable for various applications, such as action recognition, human-computer interaction, and sports analysis [16], [17].

*OpenPose* is a real-time human pose estimation system that uses CNNs and part affinity fields to identify and estimate poses in images and videos. This system utilizes a bottom-up method and can operate in real-time while maintaining high precision on benchmark datasets like COCO [20] and MPII [22]. The bottom-up approach makes OpenPose scale-invariant, allowing it to track multiple people simultaneously[16], [20].

*MediaPipe* Hands is a real-time hand-tracking system that uses CNNs to track and extract hand landmarks from images and videos. It utilizes a bottom-up approach that first detects palms and localizes hand landmarks. It is a highly portable and reliable framework for various settings, including mobile devices, web applications, and desktop systems[15], [23].

## 2.3 Robot Operating System (ROS)

This section focuses on the original ROS [24], not the newer ROS-2 version, as the Unitree A1 robot driver currently only supports ROS-1. ROS is a free and open-source software platform for robot development, allowing users to create robot applications quickly and easily. ROS provides a standard set of software libraries and tools that can be used across a wide range of robot hardware platforms, enabling users to create complex robot behavior without having to deal with the low-level details of the individual hardware components. It is widely used in academia and industry to develop a wide range of robot applications, including robots for manufacturing, inspection, and service tasks. One of the critical features of ROS is its modular design, which allows developers to create and reuse software components. This modularity allows developers to create custom solutions tailored to their specific needs and requirements and to easily add new features and functionality as needed [25].

This is achieved through the concept of nodes and topics [24]. Nodes are individual software components that can be combined and integrated into larger, more complex systems. Each node is designed to perform a specific task or function and can communicate with other nodes to exchange data and information. Nodes can publish messages to a topic, and other nodes can subscribe to that topic to receive the messages. More specifically, a topic is a named pipe that allows nodes to communicate with each other and with the rest of the system, even if they are not directly connected. This 'publish/subscribe' communication model decouples the production of information from its consumption, allowing for flexible and scalable communication within a robot system [26]. Every node is registered with a ROS Master responsible for managing the overall organization and communication within a ROS system [24].

ROS is built on a package system, where code is organized into self-contained units called packages [27]. The ROS community includes many developers who create and maintain packages for various applications and use cases [28]. Packages for the A1 robot are also available, such as the Unitree-ROS packages that enable virtual simulations of the robot [29].

Figure 2.3 illustrates how a ROS package for the A1 robot could be designed to record camera and processed images with generated landmarks. The camera node is responsible for getting raw images from its onboard camera and publishing the image data to an image topic. The image processing node (IPN), which is subscribed to the image topic, receives the raw data as a message. The IPN, in this case, is a pipeline that detects hands and generates superimposed landmark points. It publishes the processed image data to the processed image topic. The recorder node is subscribed to both the image and the processed image topics. As soon as it receives messages, it records them in a ROS bag [30] file. These nodes can be developed independently, combined, and integrated into a single, cohesive system.

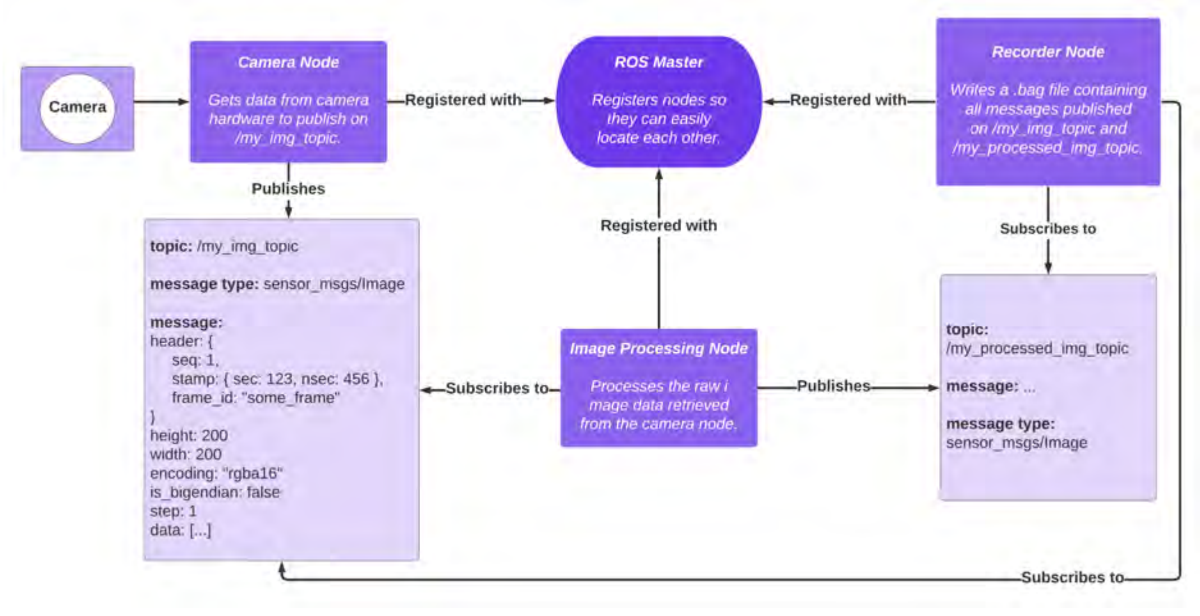


FIGURE 2.3: An example on how ROS nodes can be used in a real application [31].

## 2.4 Unitree Legged Software Development Kit

The Unitree Legged Software Development Kit (ULSDK) toolkit allows developers to create software for controlling a quadruped robot [32]. It offers two control modes: low-level and high-level [33].

Low-level control (LLC) allows the user to control the joints of the robot and apply force to them precisely. Figure 2.4 illustrates the axis and joints that can be manipulated. This mode is helpful for fine-tuned movements and tasks that require a high degree of control over the robot's movements. Projects that involve adjusting the locomotion of quadruped Unitree robots rely on this mode to fine-tune their movements [34].

In contrast, high-level control (HLC) allows the user to make the robot walk, change its position, and perform pre-configured movement sequences. This mode is useful for tasks that require the robot to move around and interact with its environment, such as navigating through a cluttered space or avoiding obstacles. While in low-level control, one has complete control over the robot's mobility, high-level control only allows one to execute a limited set of movements. However, One unique aspect of the ULSDK is that it communicates using 'structs' rather than function calls. A struct is a collection of parameters that can be assigned values, allowing the user to configure the robot's behavior easily. For example, setting the "cmd.mode" parameter to "2" will transition the robot into walking mode [33].

Overall, the ULSDK offers a flexible and powerful way to control a four-legged robot, with both low-level and high-level control modes to suit various tasks and scenarios. A detailed guide to using the ULSDK is added to Appendix A.

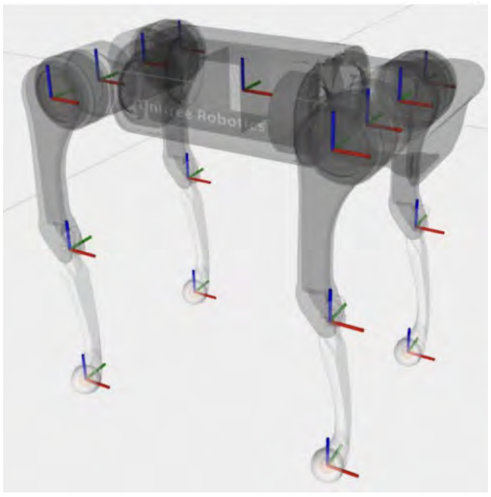


FIGURE 2.4: Joints that can be controlled in low-level mode [33].

# 3 Methods

In this chapter, we describe upon which criteria we built our demonstrator and present a method and an implementation for gesture recognition using pose estimation and machine learning techniques. Our system operates as follows: First, a frame is captured by the robot's camera. We then use pose estimation to extract hand key points from the frame. These key points are stored in a queue, enabling us to represent temporal information. Subsequently, the entire queue is processed by an ML classifier. If the classifier is confident that it has identified a gesture, it triggers the robot to execute the corresponding action.

## 3.1 Demonstrator Analysis

As this thesis aims to build a tangible demonstrator for AI, the first crucial step is to examine which attributes a demonstrator should have to arouse the audience's interest successfully. Out of our problem statement, we derived four main goals the demonstrator should fulfill:

- *Powered by AI*: The aim is to demonstrate AI; hence, the demonstrator should be powered or assisted by AI.
- *Captivating*: When used at an exposition, the demonstrator should attract visitors and encourage them to engage in a conversation with the exhibitors.
- *Usability*: To improve usability, operating the demonstrator should not require any knowledge of AI or programming. The demonstrator must be able to execute the demonstration within any confined, well-lit space not smaller than 3 m x 3 m.
- *Scalable*: The development framework should allow adding new features so that we can scale the capabilities in the future more conveniently.

After analyzing existing demonstrators for quadruped robots [34]–[38] we identified two different types of demonstrators:

- *Task-based*: An example for a task-based demonstrator is displaying locomotion skills [34], [37]. The quadruped robot walks around and imitates the gait patterns of real-world animals, such as dogs. Such demonstrators are developed to show the potential of Reinforcement Learning [34]. They are not dependent on human interactions but remotely activated by an operator and aim to complete a predetermined task.
- *Interaction-based*: Interaction-based Demonstrators require human input. Those inputs can be either physical (touching the robot), visual-based (gestures), or sound-based (voice commands) [38].

Considering the points above, we decided to implement a visual-based demonstrator that accepts human gesture inputs and reacts with a set of movement sequences. Such a visual-based demonstrator is more suited for fulfilling the criteria above than a sound or task-based demonstrator due to the following reasons:

- Robots that move based on human-machine interactions have been shown to increase attention, and attention span in children [39]. Reactions of the robot that involve movements have the potential to be perceived as entertaining to the audience [38]. Research shows that the movements of a robot, indicating that it is paying attention, can lead to a significantly higher willingness to engage with it [40]. Thus, in terms of keeping the audience engaged, an interactive demonstrator is superior to a task-based one.

- Task-based demonstrators that rely on pathfinding and locomotion require larger spaces than interactive demonstrators with shorter movement sequences [35]. Therefore, this could cause some difficulties for an exhibitor to create enough space among a crowd of visitors to demonstrate locomotion skills. The A1 robot does not have an onboard microphone; therefore, it cannot record voice inputs without equipping it with an external microphone. However, it is equipped with an onboard camera that enables it to receive a real-time video feed.

- Training the robot to walk like an animal is considered quite challenging [34]. Although there are existing platforms to train new gaits [41], adding new gaits does not add any functionality or capability to the robot. However, with a new platform that divides the functionality of an interactive demonstrator in three modules, namely User Interface (UI), input recognition (AI) and robot control, future contributors are able to update or swap out the input recognition with their recognition technology of choice and add new movements to the robot control. This would even allow the fusion of task-based actions and interaction-based inputs.

  In the following section, we explain how we select the best suited gesture recognition technology for our demonstrator.

## 3.2 Gesture Recognition

Incorporating gesture recognition technology into an AI demonstrator using a quadruped robot enhances the system's ability to interpret and respond to human input more interactively. This can facilitate the robot's interactions with its environment, rendering them more natural and intuitive for the human operator. In order to identify the best technology for this project, it is important to consider a range of criteria. We define the following benchmarks to determine the best fit for our specific requirements and goals:

- *Performance*: The technology should be able to accurately and reliably recognize gestures in real-time, with minimal latency or errors.

- *Compatibility and interoperability*: The technology should be compatible with the robot's hardware and software, and should be able to integrate seamlessly with other project components as defined in Subsection 3.3.2.

- *Ease of use*: The technology should be user-friendly and straightforward to implement, requiring minimal expertise or specialized knowledge because time constraints are an essential consideration.

- *Documentation and support*: The technology should have comprehensive documentation and support resources available, such as tutorials, user guides, and forums.

### 3.2.1 Pose estimation

In order to determine the best framework for our purposes of pose estimation, we conducted a comprehensive comparison of three leading frameworks: AlphaPose, OpenPose, and MediaPipe. Our evaluation was based on a set of benchmarks that were defined in Section 3.2. The results of our comparison are summarized in Table 3.1 and discussed in more detail below.

- *Performance*: To assess the framework's suitability for our project, we conduct a performance evaluation in a controlled, well-lit lab environment using a laptop with a built-in webcam. The laptop is equipped with an Intel I7-12700h CPU and 16 GB of RAM, but does not have a dedicated GPU. The laptop is positioned approximately 1 meter away from the subject and records a simple hand wave gesture. We measure each framework's average frames per second (fps) achieved while processing the gesture in real-time. The results indicate that MediaPipe has the highest average fps of 28, followed by AlphaPose with 25 fps, and OpenPose with 11 fps.

- *Compatibility and interoperability*: MediaPipe can be installed using the pip package manager for Python, while AlphaPose requires local compilation from the source code using a Python script. OpenPose requires installation from source code using the 'make' build tool. All the frameworks have a Python API and are compatible with major operating systems.

- *Ease of use*: MediaPipe is the easiest to use as it simply allows importing its packages in the installed environment. AlphaPose has the best customization options. It allows one to choose between many classifiers and on what training datasets the models are built. As OpenPose is primarily a C++ library, its Python API is not fully supported.

- *Documentation and support*: OpenPose has the most comprehensive documentation, with a dedicated documentation page that is well-structured and extensive. MediaPipe offers a single page containing a quick start guide for hand pose estimation, and AlphaPose provides only example scripts.

| **Framework** | **Performance** | **Compatibility and interoperability** | **Ease of use** | **Documentation and support** |
|---|---|---|---|---|
| AlphaPose | + | + | + | - |
| OpenPose | - | - | - | ++ |
| MediaPipe | ++ | ++ | ++ | + |

TABLE 3.1: Comparison of the pose estimation frameworks AlphaPose, OpenPose, and MediaPipe according to the criteria defined in Section 3.2

Table 3.1 illustrates the comparison between OpenPose, AlphaPose, and MediaPipe. It is important to note that the grades with "+" and "-" are relative to one another and reflect our assessment of each framework's relative strengths and weaknesses in each category. Specifically, we chose a framework that we believe is the best, neutral, and worst compared to the other options within each category.

After thoroughly evaluating the available pose estimation frameworks, we have determined that MediaPipe is the most suitable option for our project. One of the primary considerations in our decision was that MediaPipe is production-ready and has a well-documented API, which facilitates its implementation and integration into our project. Furthermore, MediaPipe exhibited superior performance in our experiments compared to the other frameworks we evaluated. MediaPipe's installation process, which utilizes the pip package manager for Python, is significantly more streamlined than the other frameworks, which require building from source code. Given these factors, MediaPipe represents the optimal choice of technology for our project.

### 3.2.2 Hand Gesture Recognition

A dataset of 3768 hand gestures was collected using the Unitree A1 robot in a well-lit laboratory environment, as depicted in Figure 3.1. The subject was positioned at various distances and angled to record each gesture, as shown in Figures 3.2, 3.3, 3.5, and 3.5d. The dataset includes 600 samples for each gesture and 1368 random move samples, which serve as a baseline to prevent the machine learning classifier from wrongly classifying every slight movement as a gesture.

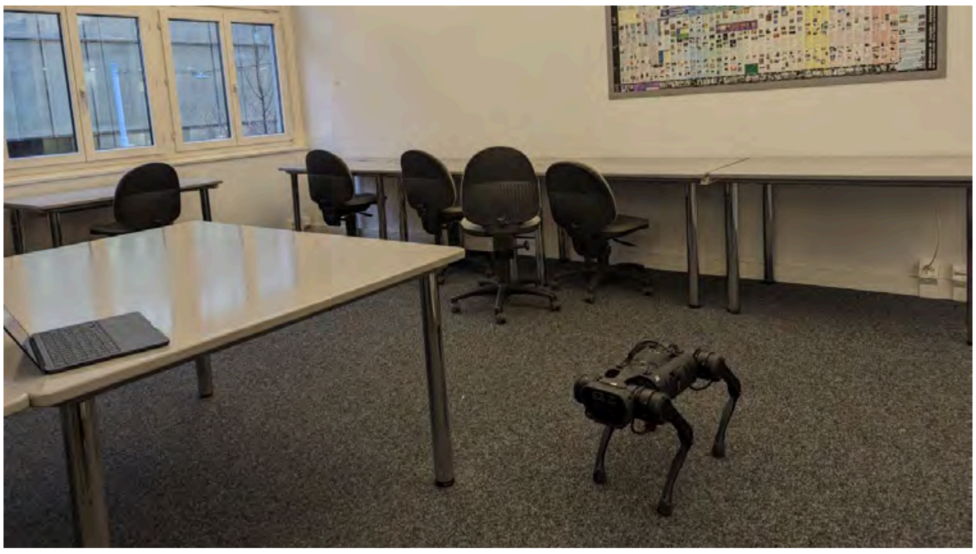


FIGURE 3.1: Lab environment setup within a well lit room

Multiple quantitative metrics were used to evaluate the system's performance, including accuracy, precision, recall, f-score, and classification time. Our primary focus is classification time, as the system must perform in real-time. We further prioritize accuracy over recall, as it is more critical that the system does not perform actions in response to random moves than that it detects every single gesture.

Table 3.2 presents the results of our comparison of various machine learning classifier algorithms on a gestural recognition task. In order to establish a baseline for comparison, the hyperparameters for all classifiers were left at their default values. We conducted 70,000 runs for each classifier to determine the average classification time. The Random Forest classifier exhibited the highest accuracy, achieving 97.6%. The Decision Tree classifier demonstrated the quickest performance, with an average classification time of $67\mu s$. As a result, we selected the logistic regression classifier, which achieved a good balance between these two factors. It attained an accuracy of 94.7% and an average classification time of $122\mu s$.

A concern with using the pose estimation framework for hand tracking is determining which hand to track when multiple people are present. To address this issue, we implemented a strategy of tracking the hand closest to the robot. This was achieved by calculating the distance

| Classifier | Accuracy | Precision | Recall | F-score | classification time |
|---|---|---|---|---|---|
| Logistic Regression | **0.9469** | 0.9532 | 0.9452 | 0.9487 | **$122\mu s \pm 7.16\mu s$** |
| SVM | 0.8395 | 0.8666 | 0.8137 | 0.7976 | $2.86ms \pm 34.8\mu s$ |
| Naive Bayes | 0.6659 | 0.6507 | 0.6540 | 0.6499 | $279\mu s \pm 5.72\mu s$ |
| Random Forest | **0.9761** | 0.9740 | 0.9665 | 0.9698 | $7.39ms \pm 222\mu s$ |
| Decision Tree | 0.8177 | 0.8040 | 0.8098 | 0.8044 | **$67\mu s \pm 4.03\mu s$** |
| K-nearest neighbour | 0.8828 | 0.8787 | 0.8767 | 0.8688 | $4.64ms \pm 680\mu s$ |
| AdaBoost | 0.6442 | 0.6649 | 0.6256 | 0.6293 | $1.39ms \pm 39.2\mu s$ |
| Gaussian Process | 0.9327 | 0.9275 | 0.9186 | 0.9213 | $324ms \pm 2.84mss$ |
| Neural Net | 0.9002 | 0.8939 | 0.9044 | 0.8921 | $41.6ms \pm 981\mu s$ |

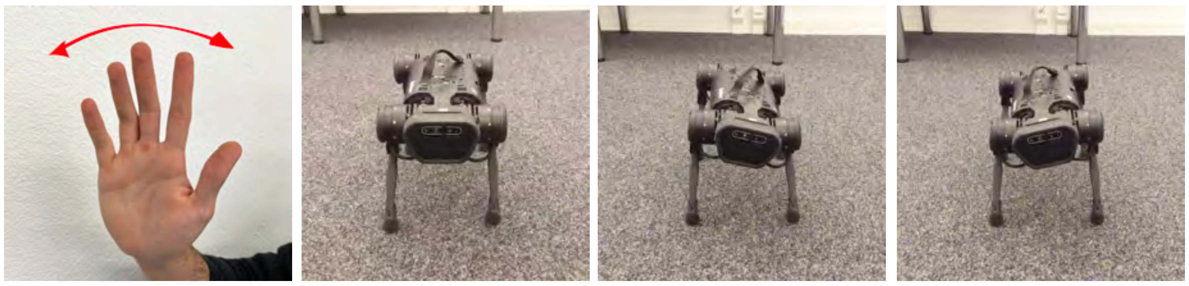
TABLE 3.2: Comparison of different ML approaches to classify the gestures defined for the project

between the key points 0 and 5 (see Figure 2.2b) and selecting the hand with the greatest distance. This approach effectively allows the application to distinguish between multiple hands and accurately track the desired one.

| Gesture | Hits | Errors | Accuracy |
|---|---|---|---|
| Wave | 25 | 0 | 1.00 |
| Up-Down | 18 | 7 | 0.72 |
| Spin | 24 | 1 | 0.96 |
| Walk | 24 | 1 | 0.96 |
| Average | | | 0.91 |

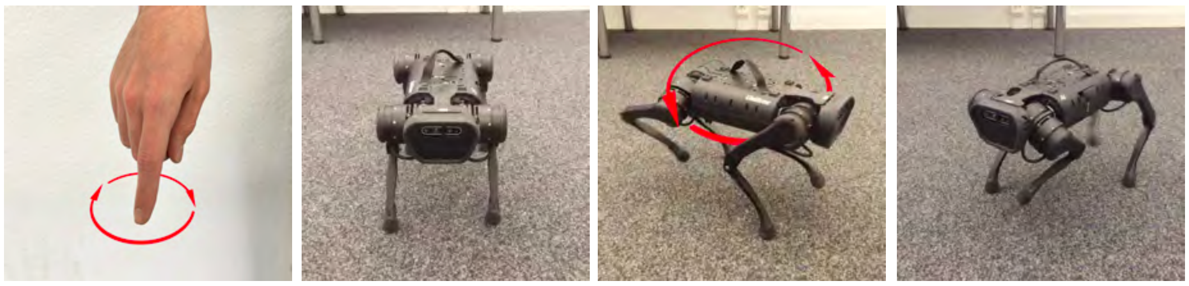
TABLE 3.3: Detection accuracy of the different gestures in the lab.

We conducted a lab experiment to evaluate the gesture recognition capabilities of our application under optimal conditions using the logistic regression classifier. The setup included good lighting conditions and a generic background as depicted in Figure 3.1 with a subject positioned about 1 meter in front of the robot. Each gesture was tested 25 times for a total of 100 runs. Table 3.3 shows that the overall accuracy of the gesture recognition system is 91%. However, the up-down gesture is a challenge for the pose-estimation framework, as it struggles to extract hand key points accurately. As a result, the accuracy for this gesture was 72%, while the other gestures have accuracy rates of 96% and 100%.



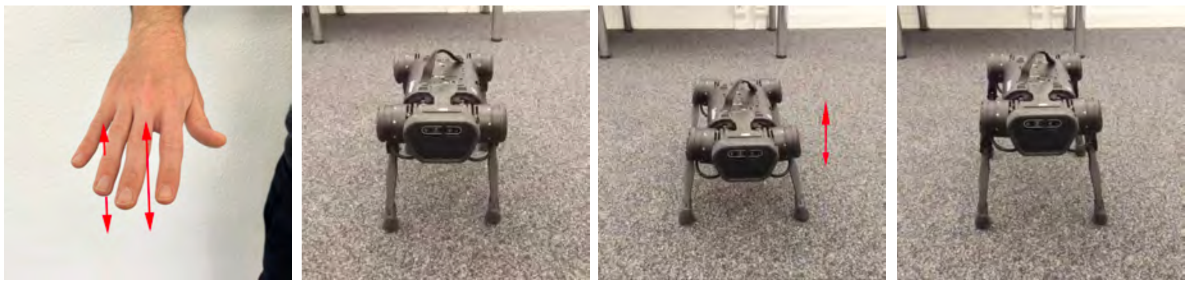
(a) (b) (c) (d)

FIGURE 3.2: Wave gesture: (a) hand wave gesture, (b), (c), (d) robot standing still and leaning laterally from side to side.



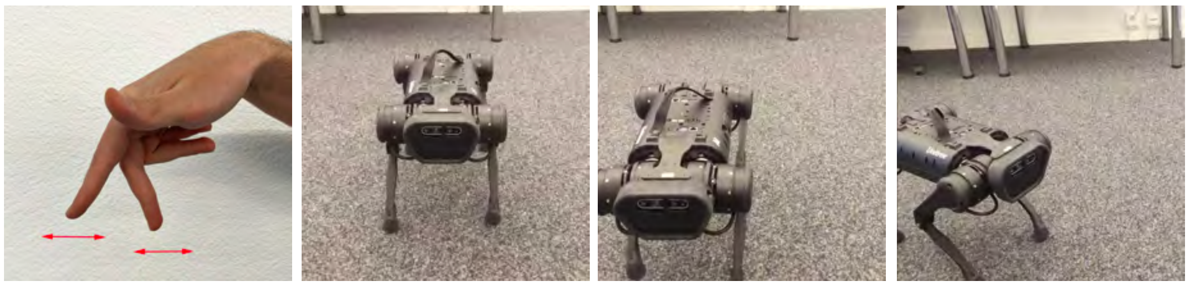
(a) (b) (c) (d)

FIGURE 3.3: Spin gesture: (a) hand spin gesture, (b), (c), (d) robot spinning around itself



(a) (b) (c) (d)

FIGURE 3.4: Up-down gesture: (a) hand up-down gesture, (b), (c), (d) robot atanding still and moving up and down



(a) (b) (c) (d)

FIGURE 3.5: Walk gesture: (a) hand walk gesture, (b), (c), (d) robot walking diagonally and turning

## 3.3 Implementation

In this section, we describe the software architecture design choices that were made for the project. Specifically, we outline the various components, their relationships, and the decisions and considerations that influenced the overall design. This information is crucial for understanding the functionality and capabilities of the software, as well as the trade-offs and limitations that were considered during the development process.

### 3.3.1 Robot Control

To communicate with the robot, the first step is to establish a connection. The A1 robot officially only offers two different types of connections:

1. Connection to remote PC via Ethernet

2. Connection to remote PC via Wireless hotspot

Although the robot has a wireless hotspot, it cannot connect to other wireless networks without additional hardware. Therefore, to directly connect to a router, it is necessary to use either an Ethernet-to-USB adapter or a Wi-Fi dongle. Out of these options, we chose to connect to the robot with Wi-Fi as it does not restrict the robot's movement compared to Ethernet.

Once a connection with the robot has been established, it can be controlled using the Unitree Legged SDK (ULSDK). We developed commands such as "walk" and "wiggle" using the SDK and wrote a bash script to run these commands. The bash script can be triggered by calling it via Secure Shell (SSH) from the remote computer. As illustrated in Figure 3.6, we can effectively control the robot's movements using the SDK and a bash script. Another option would be to

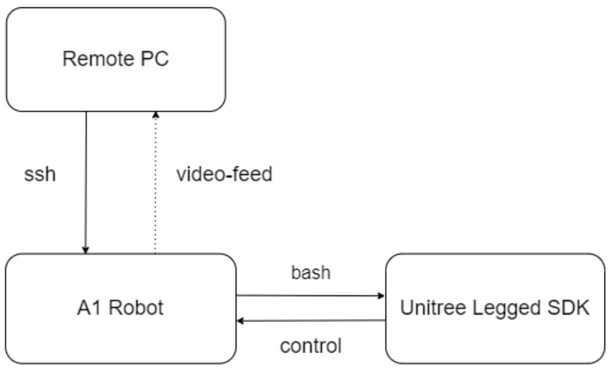


FIGURE 3.6: Communication with robot via Unitree Legged SDK.

develop a ROS package that contains nodes that are based on the ULSDK. Figure 3.7 demonstrates the control of the robot using a ROS package. The remote computer sends commands via Secure Shell (ssh) and runs the ROS package. The command node receives the commands, processes them, and publishes them to the command topic. The SDK node, which is based on the ULSDK, is subscribed to the command topic and receives the processed commands. The commands are then executed using the ULSDK, thus causing the robot to move as instructed. While ROS can be a valuable tool for developing complex and sophisticated software for controlling a robot, it is not always the best choice for every situation. In the case of simply moving the robot, using the ULSDK alone may be a better option, as developing a ROS package requires more time and ROS expertise. Additionally, one of the most important advantages ROS provides is the package library. However, for our use case, there were no fitting packages available that would add value to the demonstration. The downside to our approach of communicating with the robot via SSH and ULSDK is that passing custom commands directly with a Command-Line-Interface (CLI) is not possible with their provided control templates [32]. This is due to the ULSDK creating a UDP connection between the controller and the Nvidia board to pass commands, which creates an endless loop and thus an encapsulated process.

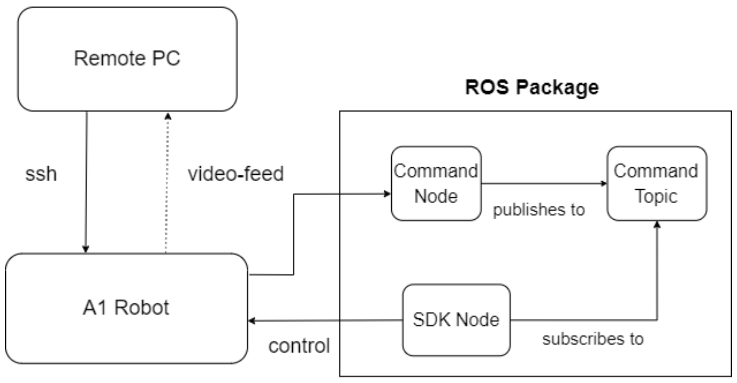


FIGURE 3.7: Communication with robot via ROS.

### 3.3.2 User Interface

We implemented a modular design with three separate processes that communicate with each other using Websockets. The architecture shown in Figure 3.10 was designed to ensure efficient and reliable communication between the remote PC and the robot. The incorporation of multi-processing prevents the User Interface (UI) and gesture recognition process from blocking each other. The UI allows the user to toggle the gesture recognition and sitting position of the robot (3.8).

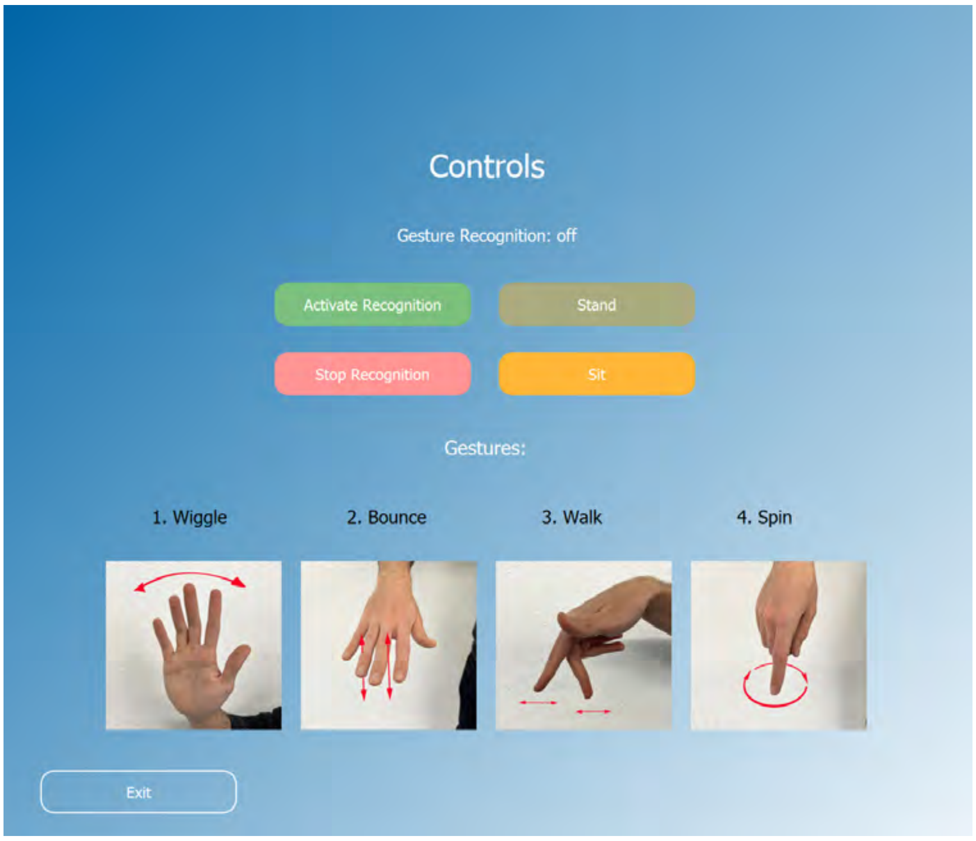


FIGURE 3.8: The UI allows the user to toggle sitting position and gesture recognition.

The gesture recognition module is controlled by the UI and captures images, performs pose estimation, and recognizes gestures. It also visualizes hand tracking and gesture recognition, which has the benefit of informing the user if their gesture is detected.

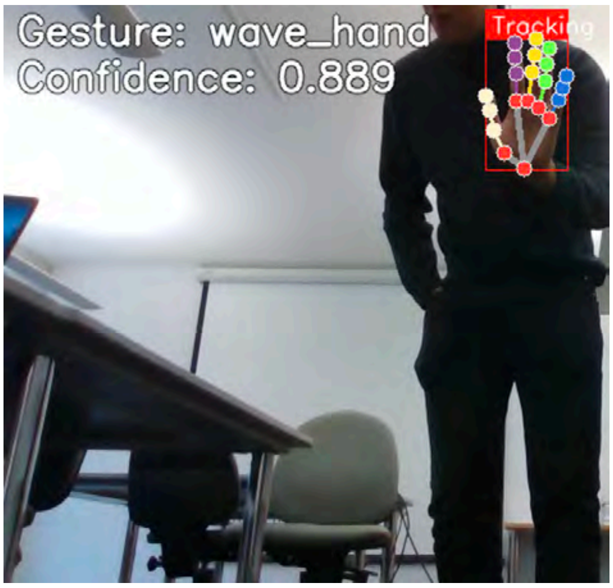


FIGURE 3.9: The gesture recognition is visualized on the remote PC. The top left text indicates the detected gesture, and below it states the confidence.

The command-executor module controls the actions of the robot based on the recognized gestures. This modular design allows us to easily toggle the gesture recognition feature and control the robot's actions while maintaining the independence and encapsulation of each process. Such a framework enables us to effortlessly swap out modules and add additional features in the future if needed.

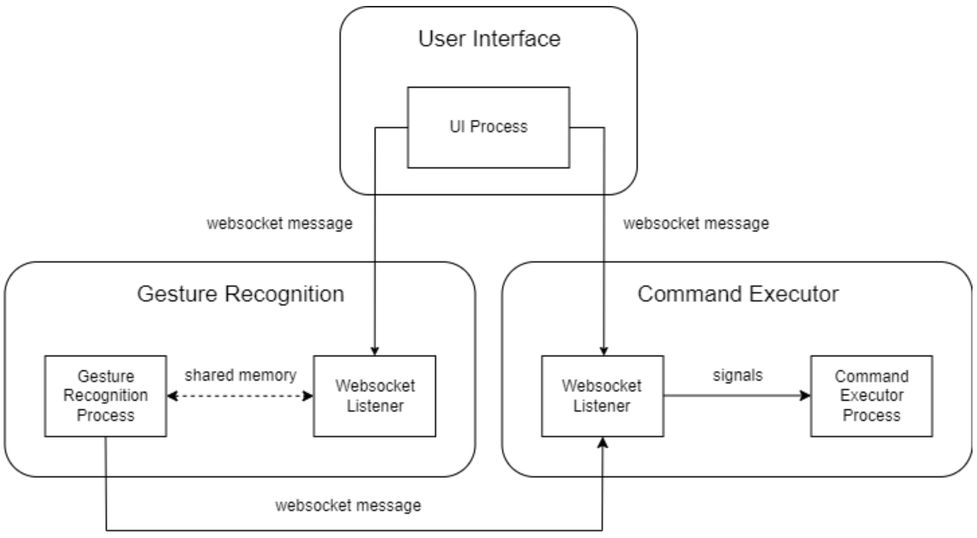


FIGURE 3.10: Three processes that run independently on multiprocessing: UI, gesture recognition and command executor.

### 3.3.3 Workflow

In this subsection, we describe the workflow of our demonstrator (3.11). The following steps are executed with the robot turned on:

1. Connect remote-PC to the robot via UI.
2. Turn on gesture recognition and make the robot sit via UI.
3. Perform hand gestures in front of the robot.
4. Robot records and creates hand landmarks for a sequence of images and queues them for classification.
5. Logistical regression model classifies gesture.
6. Robot receives the corresponding command and executes it.

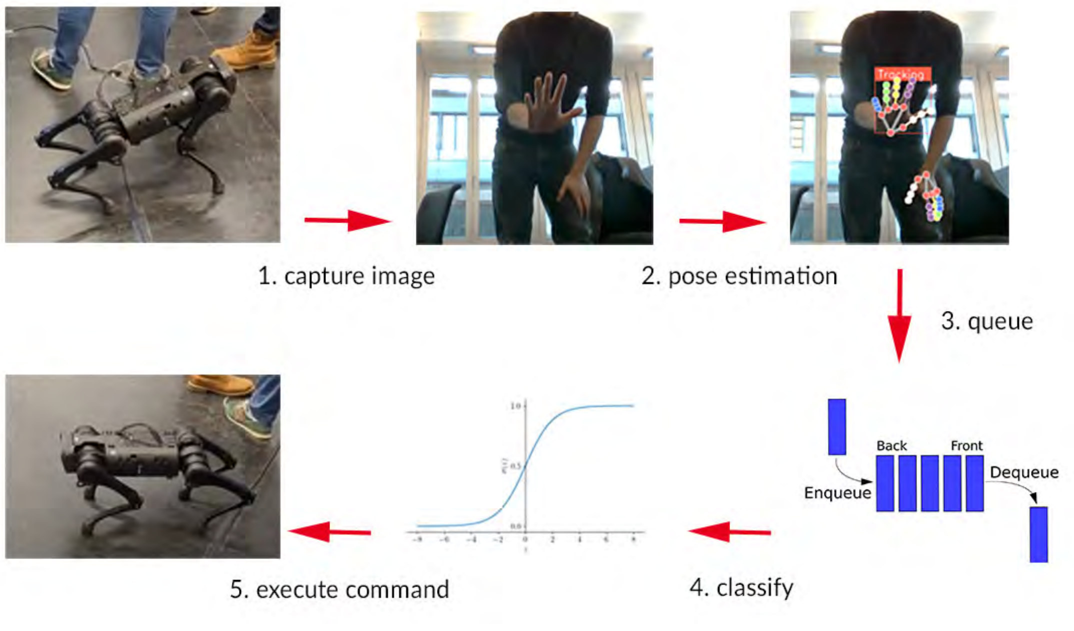


FIGURE 3.11: Demonstrator workflow: 1. collect an image, 2. extract key points, 3. queue key points, 4. classify queue, 5. perform action

# 4 Results

This thesis presents an AI demonstrator that utilizes a Unitree A1 quadruped robot to detect and classify hand gestures. We employed pose estimation through the MediaPipe framework and implemented a logistic regression classifier to classify hand gestures. In a laboratory test, the demonstrator achieved an average accuracy of 91% across four different gestures in detecting and classifying hand gestures. The subsequent sections of this study will discuss the methods employed to demonstrate the capabilities of AI in this system and the system's limitations.

## 4.1 Demonstrating AI with A1 Robot

We had the opportunity to showcase the demonstrator at an event where it garnered some attention. The attendees stood in front of the robot and performed gestures, as shown in Figure 4.1, where a participant waves at the robot, and the robot reacts by leaning laterally from side to side. The feedback towards the demonstrator was overall positive. To further demonstrate the capabilities of the demonstrator, we provide QR codes in Figure 4.2 that showcase videos from the event, where a participant performs various gestures in front of the robot. The robot was able to detect the gestures successfully. In some positions where the robot faced strong direct light, it only detected 7 out of 10 gestures correctly. Like in the lab, it struggled the most with recognizing the up-down motion, detecting only 5 out of 10 gestures correctly.

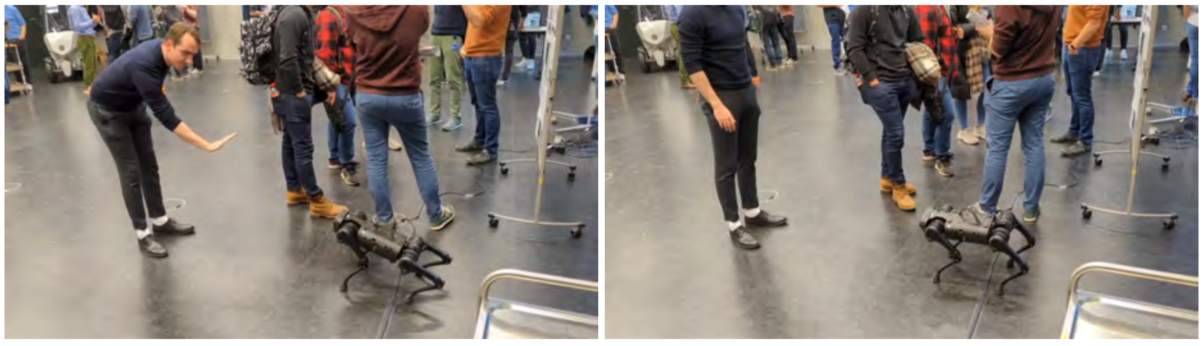

(a) (b)

FIGURE 4.1: (a) Participant performing a wave gesture towards the robot, (b) robot leaning from side to side as a response.

## 4.2 Limitations

Although the detection performs exceptionally well in evenly lit lab environments or outside in daylight, it struggles with certain undiffused lighting. We experienced this at the previous section's event, where strong lights directly shined upon the robot's camera. As the robot has a relatively shallow field of view, we programmed it to lower its hind legs to shift its view up in order to get a full view of the upper body of the person standing in front of it. Due to the viewing angle, it is susceptible to facing intense direct light from the ceiling, which may be a reason for false detection. In that case, Mediapipe is not always able to track hands accurately with wiggle up and down motion spin walking

FIGURE 4.2: Robot demonstration videos

the robot camera, which impacts the reliability of our gesture recognition system. It also cannot reliably detect the up-down gesture. Out of ten gestures, only five were correctly detected and classified.

The aforementioned stance modification is needed due to the robot having a relatively shallow field of view. In its standard standing position, the robot's vision is limited to the user's knees. As a result, the user would have to lower their posture to knee level in order to have their hand gestures detected. While our solution allows the user to give commands while standing, it also introduces an additional step where the robot must revert to its standard standing position in order to execute a command. This can lead to delays for the user, as they are forced to wait for the robot to finish executing a command before giving a consecutive command.

A drawback of a demonstrator solely based on ULSDK is, that passing custom commands directly with a CLI is not possible, as mentioned in Chapter 3. As described in Appendix A, the commands must first be defined in a file and then called via terminal. This prevents the user from issuing new movement patterns in real-time. Additionally, the movement sequences we developed are relatively simplistic. The ULSDK on the robot is not yet been updated by the supplier to its newest version yet and therefore lacks more complex features such as dancing and performing backflips. Furthermore, ensuring that the robot is unobstructed within a radius of 2 meters is crucial to prevent accidents, as our system currently lacks a collision avoidance system.

# 5 Discussion and Future Work

## 5.1 Improving Usability and Robustness

In terms of robustness of the demonstrator, there is still room for improvement. To enhance the detection, one could either try to tune the MediaPipe library [42] or experiment with a different detection framework. The MediaPipe Machine Learning pipeline includes a palm detection model that identifies the position and orientation of the hand, and a hand landmark model that accurately locates 3D keypoints on the hand by analyzing the cropped image region identified by the palm detector. Therefore, to reduce detection errors, one can either retrain the palm detector or optimize the keypoints locator algorithm. The former is the easier option, as it only requires you to retrain the detector on new data. The latter is considered more laborious, as the labeling of the data can pose a challenge [42]. The new data should be recorded in environments that MediaPipe struggles with and also contain the defined gestures.

To improve the experience for the user, the UI could be fully replaced by a system that recognizes voice commands and handles the detection mode and stance accordingly. By strapping a microphone and small loudspeakers to the robot, the system could confirm the commands verbally as well as inform the user on its capabilities. When connected to our Command-Executor module, it could even serve as an additional input source and be used to steer the robot. To implement such a system, state-of-the-art speech recognition technologies should be compared and evaluated [43]. Additionally, some thought should go into how to filter out background noises as well as overlapping voices. If these considerations are not addressed, the presence of multiple individuals speaking near the robot at expositions may cause the speech recognition feature to either mistakenly activate or fail to activate altogether [44].

## 5.2 Enhancing the Capabilities of the Robot

Implementing a collision detection system would not only prevent the robot from taking damage, but also protect bystanders from getting injured. The A1 robot is equipped with an integrated depth camera that can capture depth images. These images provide distance data that can be utilized to create a collision avoidance system for the robot [45], [46]. This system would automatically halt the robot's movement if it detects that it is approaching an object too closely. However, this system based on the depth camera can only detect objects in the front and does not prevent lateral or backward collisions.

A more holistic solution would be to attach a Light Detection and Ranging (LiDAR) device, which is officially supported by the A1 robot. LiDAR is often used in self-driving cars and robotics to enable Simultaneous Localization and Mapping (SLAM) [35]. SLAM is a process by which a robot or vehicle can determine its position and orientation within an unknown environment, while simultaneously creating a map of that environment. This is done by using sensor data, such as LiDAR, to detect features in the environment and to measure the distances between them. By combining this data with algorithms that track the movement of the robot or vehicle, it is possible to build a map of the environment and to determine the robot's or vehicle's position within that map [35]. In addition, this feature could be combined with a track and follow algorithm that allows the robot to walk behind or alongside a designated person [47]. LiDAR could also serve as a replacement to the integrated camera of the robot for gesture recognition. A recent paper by Chamorro et al. [48] demonstrates how LiDAR can detect gestures 360 degrees around the robot. Although it did not outperform systems based on stereo imagery regarding accuracy, it added robustness to lighting conditions and enabled omnidirectional usage [48]. The combination of LiDAR and the integrated camera offer a great foundation for developing robust industry solutions, such as automated surveillance of wind energy stations [36]. For example, the robot could be programmed to patrol a predetermined path and detect any anomalies that indicate infrastructure damage.

We recommend to use ROS as developing platform for all the suggestions in this section as Quadruped [49] as well as Unitree [29] offer ROS packages for Teleoperation, SLAM, navigation and simulations.

## 5.3 Conclusion

The demonstrator, we built and trained in the lab, performed well in real-world situations as well. It was able to attract an audience to the robot and enticed them to engage with it. We found that MediaPipe performs well in most situations, but is not reliable in detecting hands in strong light. However, due to our modular platform, new features and capabilities can be added swiftly and thus serves as a great starting point for future contributors wanting to work with the robot. In the future, a ROS-based demonstrator could further enhance the scalability of the software and certainly needs to be considered when developing new features.

# A Appendix

The code and files necessary to run the demonstrator can be found on our Github repository: https://github.com/oswald-martin/zhaw_pa_robodog

## A.1 Unitree A1 Instructions

Most of the instructions needed to operate the robot can be found on the Quadruped website: https://www.docs.quadruped.de/projects/a1/html/index.html.

### A.1.1 Connecting to Robot

A1 has its own native hot-spot to which users can connect to. The SSID of the Wifi network of A1's hot-spot begins with UnitreeRoboticsA1 and the default password is 00000000.

Once connected to the WiFi network, one can access A1's Nvidia IP address 192.168.131.12 as well as the Raspberry Pi's IP address 192.168.123.161. To achieve this enter the command terminal of the computer that has connected to the WiFi hot-spot of the robot and enter the following:

To connect with the Nvidia's on-board PC:

```
ssh -X unitree@192.168.123.12
123
```

To connect with the Raspberry Pi's on-board PC:

```
ssh -X unitree@192.168.123.161
123
```

Beware that contrary to the official documentation, the IP's of the Raspberry Pi and Nvidia board are switched on this model.

The Nvidia board is the one you should be using as it has all the drivers installed. The Raspberry Pi is supposed to be the main PC, however the ULSDK does not work on there. The Nvidia board is also much more powerful than the Raspberry Pi and is well suited for Machine Learning.

### A.1.2 Driver Installation

The robot has already been configured successfully. However, should there arise a need to reinstall the drivers the following instructions might prove useful. The driver can be downloaded from https://my.hidrive.com/share/0y.4q83h08#$/.

Make sure to delete everything in the catkin_ws folder on the Nvidia board first. Download the files and transfer them to the robot and run the installation script with the following commands via remote-pc terminal:

```
scp -r A1_3.2.0 unitree@192.168.123.12://home/unitree
ssh -X unitree@192.168.123.12
123
sudo chmod +x a1_installation_script.bash
./a1_installation_script.bash
```

Detailed instructions for installing the drivers are in the README file. For troubleshooting, the forum might be helpful:

https://forum.mybotshop.de/t/controlling-robot-via-vm/432/30

### A.1.3 Connecting Robot to Internet

The instructions in the official documentation did not work and required IP configurations. The better alternative is to use an Ethernet-to-USB adapter. If it does not work immediately, it might be necessary to create a new connection in the NetworkManager first. Note that although the A1 has a WiFi hotspot, it is not able to connect to another WiFI network.

### A.1.4 Accessing the Camera Feed

1. Connect PC to the robot with WiFi
2. Open a browser and enter: 192.168.123.12:8080

### A.1.5 Unitree Legged SDK

The SDK version that is currently installed is 3.2.0. A newer SDK will be available soon that unlocks many features such as dancing and jumping. The GitHub repository will be updated as soon as it is released by Quadruped.

#### Controls

The Unitree Legged SDK has a high level and low level mode. Only one mode can be used at a time. High level commands enable the robot to walk, tilt and perform tricks.

#### Running the files

The SDK and its control files are in the catkin_ws/utils/unitree_legged_sdk-3.2.0 folder. The C++ files we coded to steer the robot are all in the 'examples' folder. New control files need to be built first. First, add the new file to CMakelist.txt, then navigate to the build folder and enter the following commands:

```
cmake ../
make
```

If you rebuild an already built file, make sure to delete or clean the build files in the build directory.

Then run the file with:

```
sudo ./example_walk
```

**Building a new command file**

The files in catkin_ws/utils/unitree_legged_sdk-3.2.0_examples are example files that can serve as a template. We chose the example_walk file as a template for our commands. Keep in mind to not change the IP's in the UDP constructor. Here are the modes that one can configure in the current SDK version:

```
uint8_t mode;   // 0.idle, default stand | 1.force stand
                // 2. walking
```

A detailed list of the commands can be found in the catkin_ws/utils/unitree_legged/sdk-3.2.0/include/unitree_legged_sdk/comm.h. Don't forget to build every file you created and add them to CMakelist.txt.

### A.1.6 ROS

How to launch high level mode:

```
sudo su
source ~/catkin_ws/devel/setup.bash
roslaunch qre_ros high_level_mode.launch
```

How to launch low level mode:

```
sudo su
source ~/catkin_ws/devel/setup.bash
roslaunch qre_ros low_level_mode.launch
```

How to launch teleoperation (real-time control with keyboard):

```
sudo su
source ~/catkin_ws/devel/setup.bash
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

# Bibliography

[1] *Unitree A1 Quadruped Robot*, 2020. [Online]. Available: https://www.unitree.com/a1/.

[2] PriceWaterhouseCoopers, “Sizing the prize: What’s the real value of AI for your business and how can you capitalise?” PriceWaterhouseCoopers, Tech. Rep., 2017. [Online]. Available: https://www.pwc.com.au/government/pwc-ai-analysis-sizing-the-prize-report.pdf.

[3] Pegasystems, “What Consumers Really Think of AI: A Global Study,” Pegasystems, Tech. Rep., 2019. [Online]. Available: https://www.pega.com/ai-survey.

[4] N. Ienaga, S. Takahata, K. Terayama, *et al.*, “Development and Verification of Postural Control Assessment Using Deep-Learning-Based Pose Estimators: Towards Clinical Applications,” *Occupational Therapy International*, vol. 2022, pp. 1–9, Nov. 2022, ISSN: 1557-0703. DOI: 10.1155/2022/6952999.

[5] J. Docekal, J. Rozlivek, J. Matas, and M. Hoffmann, “Human keypoint detection for close proximity human-robot interaction,” Jul. 2022. DOI: 10.48550/arxiv.2207.07742. [Online]. Available: https://arxiv.org/abs/2207.07742v1.

[6] J. Yamato, J. Ohya, and K. Ishii, “Recognizing human action in time-sequential images using hidden Markov model,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1992-June, 1992. DOI: 10.1109/CVPR.1992.223161.

[7] M. Isard and A. Blake, “CONDENSATION - Conditional Density Propagation for Visual Tracking,” *International Journal of Computer Vision*, vol. 29, no. 1, 1998, ISSN: 09205691. DOI: 10.1023/A:1008078328650.

[8] C. Kwok, D. Fox, and M. Meilă, “Real-time particle filters,” in *Proceedings of the IEEE*, vol. 92, 2004. DOI: 10.1109/JPROC.2003.823144.

[9] S. Mitra and T. Acharya, “Gesture Recognition: A Survey,” *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 3, pp. 311–324, May 2007, ISSN: 1094-6977. DOI: 10.1109/TSMCC.2007.893280.

[10] N. A. Ibraheem and R. Khan, “Survey on various gesture recognition technologies and techniques,” *International journal of computer applications*, vol. 50, no. 7, pp. 38–44, 2012.

[11] L. Dipietro, A. M. Sabatini, and P. Dario, “A survey of glove-based systems and their applications,” *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 38, no. 4, 2008, ISSN: 10946977. DOI: 10.1109/TSMCC.2008.923862.

[12] M. M. Hasan and P. K. Mishra, “Hand Gesture Modeling and Recognition using Geometric Features : A Review,” *Canadian Journal on Image Processing and Computer Vision*, vol. 3, no. 1, 2012.

[13] D. Mohr and G. Zachmann, “A Survey of Vision-Based Markerless Hand Tracking Approaches,” *Computer Vision and Image Understanding*, 2013.

[14] G. R. S. Murthy and R. S. Jadon, “A Review of Vision Based Hand Gesture recognition,” *International Journal of Information Technology and Knowledge Management*, vol. 2, no. 2, 2009.

[15] F. Zhang, V. Bazarevsky, A. Vakunov, *et al.*, "MediaPipe Hands: On-device Real-time Hand Tracking," *CoRR*, vol. abs/2006.10214, Jun. 2020. [Online]. Available: `https://arxiv.org/abs/2006.10214`.

[16] T. L. Munea, Y. Z. Jembre, H. T. Weldegebriel, L. Chen, C. Huang, and C. Yang, "The Progress of Human Pose Estimation: A Survey and Taxonomy of Models Applied in 2D Human Pose Estimation," *IEEE Access*, vol. 8, 2020, ISSN: 21693536. DOI: `10.1109/ACCESS.2020.3010248`.

[17] H.-S. Fang, J. Li, H. Tang, *et al.*, "AlphaPose: Whole-Body Regional Multi-Person Pose Estimation and Tracking in Real-Time," Nov. 2022.

[18] T. Simon, H. Joo, I. Matthews, and Y. Sheikh, "Hand Keypoint Detection in Single Images using Multiview Bootstrapping," *CoRR*, vol. abs/1704.07809, Apr. 2017.

[19] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," Jul. 2022.

[20] Z. Cao, G. Hidalgo, T. Simon, S. E. Wei, and Y. Sheikh, "OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 1, 2021, ISSN: 19393539. DOI: `10.1109/TPAMI.2019.2929257`.

[21] U. Iqbal, A. Milan, and J. Gall, "PoseTrack: Joint Multi-Person Pose Estimation and Tracking," *CoRR*, vol. abs/2006.10214, Nov. 2016. [Online]. Available: `http://arxiv.org/abs/1611.07727`.

[22] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, "2D Human Pose Estimation: New Benchmark and State of the Art Analysis," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, Jun. 2014, pp. 3686–3693, ISBN: 978-1-4799-5118-5. DOI: `10.1109/CVPR.2014.471`.

[23] Google, *Mediapipe Hands*. [Online]. Available: `https://google.github.io/mediapipe/solutions/hands.html`.

[24] *ROS Official Wiki*, Sep. 2022. [Online]. Available: `http://wiki.ros.org/ROS/Concepts`.

[25] A. Koubâa *et al.*, *Robot Operating System (ROS)*. Springer, 2017, vol. 1.

[26] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: a practical introduction to the Robot Operating System*. O'Reilly Media, Inc., 2015, vol. 1.

[27] *ROS Packages*, 2022. [Online]. Available: `http://wiki.ros.org/Packages`.

[28] *ROS Community*, 2021. [Online]. Available: `https://www.ros.org/blog/community/`.

[29] Unitree Robotics, *Unitree ROS*, Nov. 2020. [Online]. Available: `https://github.com/unitreerobotics/unitree_ros`.

[30] *ROS bag*, Jun. 2020. [Online]. Available: `http://wiki.ros.org/rosbag`.

[31] E. Weon, *The Building Blocks of ROS 1*, May 2021. [Online]. Available: `https://foxglove.dev/blog/the-building-blocks-of-ros1`.

[32] Unitree Robotics, *Unitree Legged SDK*, 2020. [Online]. Available: `https://github.com/unitreerobotics/unitree_legged_sdk`.

[33] Unitree Robotics, *A1 Software Developer Guide*, 2020. [Online]. Available: `https://www.trossenrobotics.com/Shared/XSeries/A1SoftwareGuidev2.0-en.pdf`.

[34] X. B. Peng, E. Coumans, T. Zhang, T.-W. E. Lee, J. Tan, and S. Levine, "Learning Agile Robotic Locomotion Skills by Imitating Animals," 2020. [Online]. Available: `https://xbpeng.github.io/projects/Robotic_Imitation/2020_Robotic_Imitation.pdf`.

[35] J. Chen and F. Dellaert, "A1 SLAM: Quadruped SLAM using the A1's Onboard Sensors," Ph.D. dissertation, Nov. 2022. DOI: `10.48550/arxiv.2211.14432`. [Online]. Available: `https://arxiv.org/abs/2211.14432v1`.

[36] C. Gehring, P. Fankhauser, L. Isler, *et al.*, "ANYmal in the Field: Solving Industrial Inspection of an Offshore HVDC Platform with a Quadrupedal Robot," in 2021, pp. 247–260. DOI: `https://doi.org/10.3929/ethz-b-000360083`.

[37] N. T. Thinh, N. T. V. Tuyen, and D. T. Son, "Gait of quadruped robot and interaction based on gesture recognition," *Journal of Automation and Control Engineering*, vol. 4, no. 1, 2016.

[38] M. Fujita and H. Kitano, "Development of an Autonomous Quadruped Robot for Robot Entertainment," *Autonomous Robots*, vol. 5, no. 1, pp. 7–18, 1998, ISSN: 1573-7527. DOI: `10.1023/A:1008856824126`. [Online]. Available: `https://doi.org/10.1023/A:1008856824126`.

[39] Z. E. Warren, Z. Zheng, A. R. Swanson, *et al.*, "Can Robotic Interaction Improve Joint Attention Skills?" *Journal of Autism and Developmental Disorders*, vol. 45, no. 11, pp. 3726–3734, 2015, ISSN: 1573-3432. DOI: `10.1007/s10803-013-1918-4`. [Online]. Available: `https://doi.org/10.1007/s10803-013-1918-4`.

[40] A. Bruce, I. Nourbakhsh, and R. Simmons, "The role of expressiveness and attention in human-robot interaction," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 4, 2002, pp. 4138–4142. DOI: `10.1109/ROBOT.2002.1014396`.

[41] E. Coumans, *https://github.com/erwincoumans/motion_imitation*, 2021.

[42] A. Maltsev, *How to improve Mediapipe Skeletons Recognition*, May 2022. [Online]. Available: `https://medium.com/@zlodeibaal/how-to-improve-mediapipe-skeletons-recognition-7c3009774dd4`.

[43] J. Li, "Recent Advances in End-to-End Automatic Speech Recognition," Ph.D. dissertation, Nov. 2021. [Online]. Available: `https://arxiv.org/abs/2111.01690`.

[44] Q.-S. Zhu, J. Zhang, Z.-Q. Zhang, M.-H. Wu, X. Fang, and L.-R. Dai, "A Noise-Robust Self-Supervised Pre-Training Model Based Speech Representation Learning for Automatic Speech Recognition," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 3174–3178. DOI: `10.1109/ICASSP43922.2022.9747379`.

[45] B. Schmidt and L. Wang, "Depth camera based collision avoidance via active robot control," *Journal of Manufacturing Systems*, vol. 33, no. 4, pp. 711–718, Oct. 2014, ISSN: 0278-6125. DOI: `10.1016/J.JMSY.2014.04.004`.

[46] M. Fischer and D. Henrich, "3D collision detection for industrial robots and unknown obstacles using multiple depth images," *Advances in Robotics Research: Theory, Implementation, Application*, pp. 111–122, 2009. DOI: `10.1007/978-3-642-01213-6{\_}11/COVER`. [Online]. Available: `https://link.springer.com/chapter/10.1007/978-3-642-01213-6_11`.

[47] Z. Li, B. Li, Q. Liang, W. Liu, L. Hou, and X. Rong, "A quadruped robot obstacle avoidance and personnel following strategy based on ultra-wideband and three-dimensional laser radar," *International Journal of Advanced Robotic Systems*, vol. 19, no. 4, Jul. 2022, ISSN: 1729-8806. DOI: `10.1177/17298806221114705`.

[48] S. Chamorro, J. Collier, and F. Grondin, "Neural Network Based Lidar Gesture Recognition for Realtime Robot Teleoperation," in *2021 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2021, pp. 98–103. DOI: `10.1109/SSRR53300.2021.9597855`.

[49] Quadruped, *Quadruped A1 Documentation*, 2022. [Online]. Available: `https://docs.quadruped.de/projects/a1/html/index.html`.