Gesture-Enabled AI Robot for Adaptive Human Interaction

A PROJECT REPORT

Submitted to

Amrita Vishwa Vidyapeetham

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING (ARTIFICIAL INTELLIGENCE)

By

Avanindraa Nagarajan (Reg. No. CH.SC.U4AIE23004)

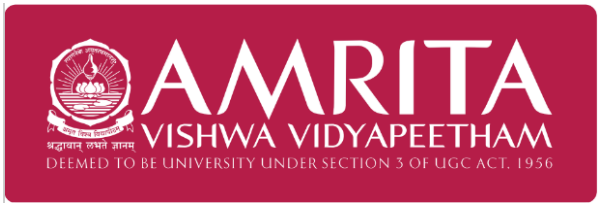
Endla Vaishnavi (Reg. No. CH.SC.U4AIE23013)

Faiz Rahaman (Reg. No. CH.SC.U4AIE23014)

Harinderan Thirumurugan (Reg. No. CH.SC.U4AIE23019)

Under the guidance of

Mr. G Deenadayalan



AMRITA VISHWA VIDYAPEETHAM
AMRITA SCHOOL OF COMPUTING
CHENNAI – 601103

October 2024

BONAFIDE CERTIFICATE

Certified that this project report **"Gesture-Enabled AI Robot for Adaptive Human Interaction"** is the bonafide work of **"Avanindraa N (Reg. No. CH.SC.U4AIE23004), E Vaishnavi (Reg. No. CH.SC.U4AIE23013), Faiz R (Reg. No. CH.SC.U4AIE23014), Harinderan T (Reg. No. CH.SC.U4AIE23019)"** who carried out the project work under my supervision.

SIGNATURE

Mr. G DEENADAYALAN
SUPERVISOR
Department of MEE
Amrita School of Computing
Chennai.

SIGNATURE

Dr. S. BAGHAVATHI PRIYA
PROGRAM CHAIR
Department of CSE - AI
Amrita School of Computing
Chennai.

INTERNAL EXAMINER

EXTERNAL EXAMINER

AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF UGC ACT, 1956

SCHOOL OF COMPUTING CHENNAI

DECLARATION BY THE CANDIDATE

I declare that the report entitled **"Gesture-Enabled AI Robot for Adaptive Human Interaction**" submitted by me for the degree of Bachelor of Engineering is the record of the project work carried out by me under the guidance of **"Mr. G Deenadayalan "** and this work has not formed the basis for the award of any degree, diploma, associateship, fellowship, titled in this or any other University or other similar institution of higher learning.

SIGNATURE

Avanindraa Nagarajan (Reg. No. CH.SC.U4AIE23004)

Endla Vaishnavi (Reg. No. CH.SC.U4AIE23013)

Faiz Rahaman (Reg. No. CH.SC.U4AIE23014)

Harinderan Thirumurugan (Reg. No. CH.SC.U4AIE23019)

ACKNOWLEDGEMENT

This project work would not have been possible without the contribution of many people. It gives me immense pleasure to express my profound gratitude to our honorable Chancellor **Sri Mata Amritanandamayi Devi,** for her blessings and for being a source of inspiration. I am indebted to extend my gratitude to our Director, **Mr. I B Manikantan** Amrita School of Computing and Engineering, for facilitating us all the facilities and extended support to gain valuable education and learning experience.

I register my special thanks to **Dr. V. Jayakumar,** Principal, Amrita School of Computing and Engineering for the support given to me in the successful conduct of this project. I wish to express my sincere gratitude to **Dr. S. Sountharrajan,** Chairperson Department of Computer Science, **Dr. S. Baghavathi Priya,** Program Chair, Department of Artificial Intelligence, **Mr. G Deenadayalan ,** Supervisor for their inspiring guidance, personal involvement and constant encouragement during the entire course of this work.

I am grateful to Project Coordinator, Review Panel Members and the entire faculty of the Department of Computer Science & Engineering, for their constructive criticisms and valuable suggestions which have been a rich source to improve the quality of this work.

Avanindraa Nagarajan (Reg. No. CH.SC.U4AIE23004)
Endla Vaishnavi (Reg. No. CH.SC.U4AIE23013)
Faiz Rahaman (Reg. No. CH.SC.U4AIE23014)
Harinderan Thirumurugan (Reg. No. CH.SC.U4AIE23019)

TABLE OF CONTENTS

LIST OF FIGURES AND FLOWCHARTS

LIST OF SYMBOLS AND ABBREVIATIONS

ML - Machine Learning

CNN - Convolutional Neural Network

RNN - Recurrent Neural Network

TTS - Text to Speech

HRI - Human Robot Interaction

SVM - Support Vector Machine

NLP - Natural Language Processing

STT - Speech to text

NLU - Natural Language Understanding

ABSTRACT:

Over the past few years, Human-Robot Interaction (HRI) has evolved greatly with the introduction of gesture-based control systems, which allow robots to sense and react to human movements in real time. These systems obviate the need for the use of conventional programming techniques, which are frequently time-consuming and domain specific. Rather, they utilize natural human gestures and decision-making based on artificial intelligence to facilitate adaptive and dynamic interaction. This thesis focuses on the development and simulation of a gesture-enabled AI robot for adaptive human interaction. The proposed system is implemented in Coppelia-Sim, where the robot's movement is governed by real-time gesture recognition, including directional navigation based on pointing gestures and the number of fingers displayed by the user. In addition to motion control, the robot integrates a conversational AI chatbot via API, enabling the system to respond to verbal user queries, thereby enhancing multimodal interaction. Furthermore, the robot is equipped with vision-based capabilities to detect human waving gestures and determine human proximity, classifying whether an individual is within close or far range, which allows the robot to dynamically adjust its behavior. Inverse kinematics and path optimization techniques are applied within the simulation environment to ensure smooth and accurate robot responses, independent of manipulator configurations or simulation constraints. The combination of gesture-based navigation, AI conversational capabilities, and adaptive proximity sensing highlights the potential of the proposed system for deployment in environments requiring intelligent, intuitive, and responsive robotic assistance. This research contributes to advancing HRI frameworks by integrating gesture recognition and AI to reduce the reliance on manual programming and improve autonomous robot decision-making.

Keywords:
Gesture Recognition, Human-Robot Interaction (HRI), Adaptive Robotics, CoppeliaSim Simulation, Inverse Kinematics, Proximity Detection, AI Chatbot Integration, Natural User Interfaces, Robot Path Optimization, Vision-Based Gesture Detection

1. INDRODUCTION

This chapter presents the motivation, justification for the problem statement and the re-search questions of this Master thesis. In addition, this chapter introduces the scope, the limitations and the objectives to consider during the implementation of this Master thesis work.

1.1 Motivation

Nowadays, robots are increasingly becoming a part of human life. From robotic home appliances such as vacuum cleaners to advanced robotic structures applied in the healthcare industry, industrial automation, and even space exploration, robots have a direct or indirect impact on human daily lives. The development of robotics, particularly in human life-related applications, is supported by increasing investments and research efforts, according to reports from the International Federation of Robotics (IFR) [1], [2].

Traditional industrial robots are typically programmed by offline or online techniques. Offline programming involves the creation of a virtual simulation world where the programmer simulates robot work cells, targets, and trajectories and validates collision-free paths. This contrasts with online programming, which requires physical close-range proximity to the robot, utilizing a teach pendant to perform manual guiding and storing of motions and the creation of executable robot code. While useful, these approaches require sophisticated robot programming skills and require lengthy setup and downtime, resulting in a higher cost and lower flexibility.

The increasing demand for natural and adaptive Human-Robot Interaction (HRI) has spurred research on various alternative methods such as voice-controlled control systems [3], body gesture tracking [4], and hand-gesture recognition [5]. These approaches attempt to provide more convenient control of the robot by allowing non-experts to control robots using natural, intuitive interfaces. However, providing accurate, dependable control as well as safe robot navigation with such intuitive interfaces is a primary research challenge. This thesis investigates the creation of an AI robot with gesture recognition capability to enable adaptive human-robot interaction. The framework involves vision-based gesture recognition, enabling the robot to understand directional pointing and finger-count gestures representing movement commands, all simulated within the Coppelia-Sim environment. The robot is also augmented with an AI chatbot through API integration, enabling it to respond to questions and facilitate conversational interaction. The system also involves wave gesture recognition and proximity sensing to identify human presence and adapt the robot's response accordingly.

The aim is to enable non-experts to command and interact with the robot via natural gestures and speech signals, with the help of minimal training. The robot control system is designed to be robot structure or robot manufacturer independent, with a focus on scalability and flexibility. Inverse kinematics and path-planning optimization techniques are employed in order to provide smooth and accurate responses. The contribution of the presented work is towards the development of intuitive HRI systems, with potential applications in industrial, service, and assistive robotics domains.

1.2 Justification

Over the last decade, there has been a lot of research aimed at creating natural human-robot interaction and gesture-control systems that are user-friendly. The intense pace of development and proliferation of new input modalities like depth cameras, hand-tracking technology, and computer vision-based gesture recognition algorithms — originally developed for gaming and virtual reality have revealed new horizons for extended applications in robotics and adaptive control systems. These systems enable users to interact with robots through natural body gestures and hand movements, avoiding or minimizing the use of complex hardware controllers or pre-defined interfaces. To this degree, the historical necessity for specialized programming tools and advanced devices, like tech pendants or code-based robot control interfaces, has been significantly minimized. Modern robotic systems can now read hand gestures, finger numbers, and proximity signals to execute movements and actions that are sensitive in a human-like manner. This transition not only improves the simplicity of robot control for individuals without technical knowledge but also supports real-time adaptive interaction, whereby robots can react to human proximity, waving motion, and speech signals. This, in turn, can make the use of professional robot programmers obsolete and replace it with specialists in specific task domains who can operate robots in an intuitive manner, thereby enhancing the usability and flexibility of robotic systems in various industries.

1.3 Problem Statements and Research Questions

As the technology of human-robot interaction continues to improve, adaptive and intuitive robot control becomes increasingly important. Existing paradigms for robot programming are domain-specific, time-consuming, and not available to non-programmers or occasional users. Static programming models and inflexible interfaces are not adequate for dynamic environments where interaction patterns and user intentions shift at high rates.

There is increasing need for systems that enable robots to react in real-time to human gestures, proximity, and spoken commands in conversation in order to enable them to function in social and industrial settings. The creation of gesture-recognition systems and conversational capabilities through AI is a viable path for adaptive human-robot interaction. Nevertheless, there are challenges in mapping human gestures onto robotic movements in a precise manner, hardware-independent programming, and efficient robot response and movement.
So, the following research questions are raised:

- How can non-programmers intuitively control and communicate with robots through natural gestures and conversational interfaces?
- What are the most appropriate sensors and devices for accurately recognizing human gestures, proximity, and waving actions?
- How can gesture-based programming and control be implemented in a way that is independent of the robot's structure and manufacturer?
- How can the robot's motion planning and task execution be optimized after initial gesture-based programming, ensuring efficiency and accuracy in human-interactive scenarios?

1.4 Objectives

The primary objective of this project is to develop an AI-powered robot control system that enables intuitive, gesture-based interaction for non-programming users, along with adaptive conversational capabilities. The robot will be controlled and programmed using natural human gestures and conversational inputs, eliminating the need for traditional programming interfaces.

The specific sub-objectives are as follows:

- To develop a gesture recognition system capable of interpreting hand gestures and the number of fingers shown, enabling the robot to move in specified directions based on user gestures.
- To integrate this gesture-based control with the Coppelia-Sim environment, simulating real-time robot movements in response to user commands.
- To implement a conversational chatbot (via an external API) that allows the robot to answer user queries, facilitating natural language interaction alongside gesture-based control.
- To enable the robot to detect a human waving gesture, signaling the start or end of interaction or acknowledgment commands.
- To develop proximity detection functionality that allows the robot to sense whether a human is close or far and adapt its behavior accordingly for safe and context-aware operation.
- To ensure that all control methods (gesture-based direction, proximity detection, waving detection, and chatbot response) function independently of the robot's structure or manufacturer, making the solution modular and scalable.
- To optimize robot motion using inverse kinematics calculations where applicable, ensuring smooth and precise joint-space control based on the detected gestures and user positioning.
- To create a flexible framework where non-technical users can intuitively control the robot, making it suitable for both industrial simulation environments and potential future real-world applications.

1.5 Limitations

The limitations that are present in developing the robot's system are listed here.

- The robot control and gesture detection are only implemented in simulation (Coppelia-Sim) and not yet tested on real hardware.
- The system supports controlling only one robot at a time.
- Gesture commands are limited to simple directional movements and finger-based actions.
- Chatbot and human detection features depend on API and simulated sensors, not real-world devices
- Limited Environmental Adaptability: The simulation does not account for varying real-world lighting conditions, occlusions, or unexpected obstacles that could affect gesture recognition and robot response.

- Restricted Gesture Vocabulary: The system currently recognizes only a predefined set of gestures, limiting its adaptability to user-defined or dynamically learned gestures.
- Latency in Gesture Recognition: The processing speed of gesture recognition and chatbot responses may experience delays due to reliance on external APIs and simulated sensors.

2. LITERATURE REVIEW

[1]This chapter reviews and discusses pertinent research studies on Human-Robot Interaction (HRI) for adaptive robot control and robot gesture programming. It also reviews research studies on other devices for intuitive robot control, including vision-based systems, hand tracking devices, and gesture recognition sensors. The chapter also reviews studies on integrating AI chatbots for robot-to-human communication and reviews current online robot programming techniques and simulation-based software Human-Robot Interaction (HRI) has been transformed in recent years, with gesture-based control being one of the most promising technologies for intuitive and adaptive human-robot communication.[2] Industrial robots in the past were based on sophisticated programming techniques requiring high-level coding skills, thus restricting their use to non-experts. The breakthrough of computer vision, artificial intelligence, and machine learning, however, transformed the human-robot interaction with robotic systems. Gesture recognition, in fact, has become popular as a natural and efficient robot control technique, minimizing manual programming. [3]This literature review covers the most important advances in gesture-based control, multimodal interaction, machine learning for adaptive recognition, multi-robot coordination, and real-world applications of human-robot interaction. [4]The chapter also reviews and discusses pertinent research studies on adaptive robot control and robot gesture programming. It also reviews research on other input devices for intuitive robot control, including vision-based systems, hand tracking devices, and gesture recognition sensors. [5]It also reviews the integration of AI chatbots for robot-to-human communication and current online robot programming techniques and simulation-based software.

[7]Gesture recognition is the central component of natural robotic control systems. In the past, robotic interfaces involved physical controllers, but vision-based gesture recognition is the area that has been prioritized in new research. Computational experiments have already shown that hand gesture classification significantly improves with the use of computer vision techniques like Convolutional Neural Networks (CNNs) and Support Vector Machines (SVMs). [6]Depth cameras, along with motion tracking algorithms, have also become more effective with real-time detection of gestures for robots to make accurate interpretations from user input. The hand gesture recognition research study based on the depth camera intends to develop a system to recognize static and dynamic gestures for robotic hand communication using sign language. The research strategy combines the Leap Motion Controller with high-end preprocessing, feature extraction, and machine learning classification. A custom dataset of static hand postures of American Sign Language (ASL) numbers and letters and dynamic gestures such as 'no,' 'good,' 'love,' 'grab,' and 'click' was generated. Preprocessing involved fault frame removal, smoothing of data with a moving average filter, and palm center and fingertip coordinates normalization. Classification used Multi-Class SVM for static gestures and ND-DTW for dynamic gestures with a recognition rate of 98.34% and 97%, respectively. System integration with robotic hands made it easy to facilitate sign language communication, and the success illustrates the potential of high-resolution depth sensing and AI-based classification algorithms in human-robot interaction.

Other advances in gesture recognition have explored the use of sensor fusion, in which input sources such as depth cameras, LiDAR, and inertial measurement units (IMUs) are combined to increase robustness. Gesture control of exoskeletons is explored to demonstrate that the fusion of myoelectric sensors and computer vision increases robotic assistance to mobility-impaired users. Inverse kinematics and path-planning algorithms employed in implementation provide accurate robotic motion based on gestures, increasing user flexibility and accuracy. Though gesture recognition presents a natural form of robot control, multimodal interaction has been a more unified means of HRI. Multimodal systems combine gesture recognition and speech processing to provide more efficient and accessible robotic communication. Natural language processing (NLP) and AI-based chatbot interfaces enable robots to respond to verbal commands, thereby making them more interactive and user-friendly.

[8]A machine learning-based voice recognition study demonstrates the viability of speech-based classification in smart homes. The study trains a home automation speech recognition system, enabling users to manage lighting, temperature, blinds, and appliances via voice commands. The system accurately classified user intent with Multinomial Naïve Bayes and Random Forest classifiers with an overall category recognition rate of 82.99% and action recognition rate of 90.28%. The application of real-time speech input via Google's speech recognition API also improved system responsiveness. The study demonstrates the potential of AI-based voice recognition for natural HRI and smart spaces. Emotion detection and sentiment analysis have also been applied to multimodal HRI, allowing robots to adjust responses based on user emotions. Facial detection robots can detect stress, exhaustion, or joy and change behavior accordingly. These capabilities have been integrated into care robots and AI assistants to improve human-robot interaction in caregiving environments.

Machine learning plays an important role in enhancing the flexibility and accuracy of gesture recognition systems. Predefined commands in conventional gesture recognition may limit the flexibility of the robot in dynamic environments. [9]Recent advances in deep learning have brought reinforcement learning and adaptive neural networks into the picture, where robots are able to recognize gestures more accurately. Several research studies have explored the use Long Short-Term Memory (LSTM) networks, Transformer-based techniques, and Generative Adversarial Networks (GANs) to improve gesture-based decision-making for robots. With the use of self-learning algorithms, robots can learn the unique users and develop their gesture recognition models as time goes by. One-shot learning and few-shot learning studies demonstrate that robots can be trained to learn new gestures with very few labeled data, hence becoming more adaptable for personalized user interaction.

The second significant field of robotics research includes multi-robot coordination, where several robots interact and autonomously cooperate on the basis of gesture and voice commands. As per studies, decision-making frameworks based on AI allow robots to coordinate activities with effectiveness in the fields of industrial automation, health care, and logistics. Recent advances in swarm robotics indicate the potential of gesture-controlled robot swarms to perform coordinated tasks, such as autonomous warehouse sorting and hospital medicine delivery. Blockchain technology has also been suggested for use in coordinating teams of robots, to enable secure, decentralized communication among the robots in multi-robot teams.

2.1 Depth Camera-Based Hand Gesture Recognition

The research Study here is to design an effective depth camera-based hand gesture recognition system capable of recognizing static and dynamic gestures to train a robotic hand for sign language communication. The methodology is a combination of the capabilities of the Leap Motion Controller, advanced preprocessing, detailed feature extraction, and machine learning-based classification. First, the system needed the creation of a specific dataset comprising static and dynamic gestures. [10]The static dataset included the ten numerical digits zero to nine, along with some hand positions corresponding to letters of American Sign Language (ASL). The data was collected by utilizing the Leap Motion Controller, which provided very accurate three-dimensional tracking data of the human hand. The dynamic dataset included ten specially designed gestures, which comprised rotational motions, such as 'no,' 'good,' and 'love,' and non-rotational motions, such as 'grab' and 'click.' Every sample was written down with utmost accuracy so that the kinematic constraints of the robotic hand were being considered.

Preprocessing the data that was captured was necessary to improve the quality of the input for classification. In the case of static hand postures, preprocessing consisted of the elimination of faulty frames, smoothing the data using a moving average filter on the data, and normalizing the palm center

and fingertip coordinates with respect to the palm. Dynamic gestures needed a more complex approach, since each gesture was captured as a sequence of frames. In this case, preprocessing consisted of having a sliding buffer of thirty frames, eliminating sequences with faulty data, and accepting only continuous and coherent sequences of gestures for training and testing.

[9]The feature extraction process was critical in transforming raw three-dimensional coordinates into meaningful parameters for classification. For static gestures, features were extracted through relative finger positions, each fingertip-to-palm-center distance, and finger angular orientations. This allowed the system to detect the spatial hand and finger posture, thus a full posture descriptor for every posture. For dynamic gestures, feature extraction involved analyzing time-series information regarding hand positions, palm orientation, and single-finger traces. This data was stored in float matrices and related to the dynamic hand motion evolution over time.
Static and dynamic gesture classification algorithms were selected on the basis of their ability to handle multidimensional data and how quickly they are able to make real-time predictions. For static hand postures, the following classifiers were compared: Adaptive Naïve Bayes Classifier (ANBC), k-Nearest Neighbour (k-NN), and multi-class Support Vector Machine (SVM). Multi-class SVM was selected ultimately on the basis of its better performance as it was able to provide higher accuracy rates along with less prediction time compared to other classifiers. For dynamic gestures, Discrete Hidden Markov Model (DHMM) and N-Dimensional Dynamic Time Warping (ND-DTW) were compared in the research. ND-DTW performed well in the detection of the temporal dynamics of gestures and gave faster processing times and was hence the better approach for the recognition of dynamic gestures.
[5]The architecture of the system comprised real-time prediction modules for static and dynamic gesture recognition. In the static module, the data was preprocessed and feature extracted beforehand and then input to the trained multi-class SVM, where the most likely gesture label was obtained. This label was then converted to movement commands for the robotic hand. Likewise, the dynamic module worked on continuous data streams and used ND-DTW for classifying real-time sequences of gestures. The identified gesture labels were forwarded to the robotic hand through the Robot Operating System (ROS) for smooth interaction.
The experimental evaluation results highlight the efficacy of the system. Under Human-Robot Interaction (HRI) mode static gesture recognition, the multi-class SVM registered an impressive recognition rate of 98.34%, surpassing k-NN and ANBC classifiers. The same classifier's recognition rate under Robot-Robot Interaction (RRI) mode was a staggering 99.94%, reflecting the system's precision and reliability. Confusion matrices under both modes verified minimal misclassification, reflecting the strength of the feature extraction and classification method. The multi-class SVM also recorded the lowest average prediction time, reflecting its high suitability for real-time applications.

Dynamic gesture recognition results also validated the success of the system. The ND-DTW classifier performed with a 97% recognition rate, much greater than the 91% recognition rate of the DHMM method. Moreover, ND-DTW had a much lower average prediction time, making it very suitable for dynamic, real-time systems where fast recognition is a priority. The dynamic gesture recognition confusion matrices proved that the system accurately recognized gestures with minimal error, thereby maintaining the use of the adopted methodology. The comparative evaluation of the classifiers emphasized the higher performance of multi-class SVM and ND-DTW in accuracy and speed. The incorporation of the classifiers in the robotic system facilitated precise and smooth interpretation of both static and dynamic gestures, thereby effectively enabling the robotic hand to communicate via sign language. The study concluded that the synergy of high-resolution depth sensing, efficient preprocessing, full feature extraction, and sophisticated classification algorithms produced a robust and efficient hand gesture recognition system appropriate for training robotic hands to communicate via sign language. The results demonstrated here determine the high potential for applying depth camera technology to hand gesture recognition to facilitate human-robot interaction, especially where sign language is a significant method of interaction. The methods and results outlined here not only

advance the field of human-computer interaction but provide an informative template for future advancements in assistive robotics.

2.2 Gesture Controlled Prosthetic Hand

[2] The method discussed herein focuses on designing and engineering a gesture-controlled prosthetic hand based on the inclusion of computer vision algorithms, actuation via servo motors, and communication via a microcontroller. The system revolves around the general use of real-time hand gesture recognition via camera input, using Python programming on open-source packages such as OpenCV, MediaPipe, and CVZone. The hardware takes the form of a 3D-printed prosthetic hand based on the InMoov design, activated by servo motors driven by detected gestures relayed via Bluetooth from the processing station.

The development process started with a comprehensive review of the literature and knowledge of human hand anatomy, i.e., movement achieved through muscles and tendons, offering a basis for the design of mechanical imitation. Commercial prosthetic hands such as Bebionic, i-Limb, and Vincent hands were analyzed to learn technical details and constraints. These observations formed the basis to design a cost-effective and versatile solution through 3D printing.

The core element of the methodology was recording real-time hand movements using a USB camera mounted on a laptop. Media Pipe allowed for the detection of 21 three-dimensional landmarks on the hand, thereby allowing the system to analyze hand movements and positions effectively. CVZone's Hand Tracking Module was implemented within a Python script to find the status of each finger, i.e., closed or open, by comparing the fingertip landmark coordinates with the respective proximal interphalangeal joint (PIP) landmarks' coordinates. A y-coordinate of a fingertip higher than that of the PIP joint indicated a closed finger; otherwise, if lower, the finger was open. The binary status of each finger was then sent via Bluetooth to an Arduino Uno, which translated these inputs into servo motor control signals.

[8] The hand was built from STL files downloaded from the InMoov open-source project and 3D printed. The hand consisted of five fingers, each with an independent servo motor attached to pulleys that were linked by braided fishing lines. The lines converted the rotational motion of the pulleys into linear motion to close and open the fingers. Tension springs and plastic safety tubes were utilized to control the tension and avoid friction-induced wear on the lines.

The electronic system consisted of an Arduino Uno microcontroller connected to a sensor shield and supplied by a 5.5V direct current power supply. Communication between the laptop and Arduino was accomplished using the HC-05 Bluetooth module. The control system architecture was such that it allowed the output of gesture recognition to be immediately translated into corresponding motion of the prosthetic fingers.

Prototype testing was done in three phases. The first was the measurement of response time by having the user's hand held at a constant distance from the camera and measuring the time difference between gesture recognition and movement of the prosthetic hand. The test was done ten times, and the average response time was 190 milliseconds, which is a very responsive system for real-time applications. The second test was for the accuracy of gesture recognition across different distances from the camera. The hand was set at distances between 40 cm and 280 cm, and ten different gestures were done at each interval. Surprisingly, the system was able to register a perfect accuracy rate of 100% across all distances, indicating its strength in gesture recognition irrespective of the user's proximity.

The third test investigated the performance of the system to recognize gestures of various individuals. Five participants of varying ages performed gestures before the camera, and the system was able to recognize the gestures of each participant's right hand. A major limitation was noticed when both hands were within the view of the camera; MediaPipe recognized the first hand it detected, which in some circumstances could result in unanticipated control actions.

The findings affirm that the prototype prosthetic hand can efficiently, and rapidly mimic hand movements learned from a live feed video. The system demonstrated stable performance regardless of varying distances and between different users. Moreover, the ease of employing open-source libraries combined with cheap hardware components and 3D printing technology makes this method both accessible and economical. The study concluded that improvement was possible to enhance the capacity of the prototype, e.g., variable control of the rotation angles of finger joints rather than binary open-close, correct hand selection to always control the appropriate hand when both hands are available, and inclusion of wrist rotation to provide additional functionality. This work in general offers a real-world and scalable solution to gesture-controlled prosthetic hand development that bridges the gap between assistive robotic and computer vision technologies.

2.3 Embodied Human-Robot Interaction with Natural Gestures

The work proposes a novel paradigm for human-robot collaboration in embodied visual navigation tasks via the fusion of natural human gestures. The paradigm is built upon the creation of a 3D photorealistic simulation environment called GesTHOR, an extension of AI2-THOR, with humanoid and robot agents in a shared world. Users are given virtual reality headsets and command input through natural body and hand gestures sensed by state-of-the-art sensing equipment, including Oculus Rift, Microsoft Kinect v2, and Leap Motion controllers. Human demonstration data constitute the Gesture ObjectNav Dataset (GOND), consisting of varied and context-dependent gestures acquired without labeling in advance, allowing the learning model to deduce semantics autonomously. The design process begins with the construction of the simulation world in which the agents perform navigation tasks. The user is the humanoid agent and the robot agent that identifies gestures and responds. The environment simulates home settings in real life and offers assistance for tracking body movement and VR-based interactive communication. Naturalistic gestures are of deictic (pointing) and metaphoric (symbolic) nature, reflecting human intuitive means of communication.

Gesture data are collected in fully immersive virtual environments, allowing subjects to provide navigation instructions in the form of gestures without the use of words. Motion capture refers to the process of collecting human movement and synchronizing it with environmental interaction. Collected gestures are saved in the GOND dataset and include metadata like spatial relationships and object categories, and they have high context for learning. The learning paradigm leverages deep reinforcement learning, prioritizing Proximal Policy Optimization (PPO). The multimodal perception paradigm is utilized within the reinforcement learning framework, in which visual input, object class, and gesture signals all collaborate towards decision-making. The robotic agent takes egocentric RGB and depth images, spatial signals, and continuous gesture signals as its state representation. Navigation controls that are essential for navigation make up the action space, while the reward function weighs task success, navigation success, and proper gesture interpretation.

Experiments are performed to compare the performance of the RL model learned in the GesTHOR environment with the GOND dataset. The performance measures are navigation success rate, path efficiency, and the model's ability to generalize to new, unseen gestures. The trained agent shows substantial performance improvements over baseline models, proving that it can learn the meaning

of new, unseen gestures and navigate successfully to target objects. Navigation performance results record higher success rates and shorter path lengths compared to language-only instruction models. Ablation studies also confirm the significance of including gesture signals in the multimodal learning process.

Qualitative analysis entails visualization of agent paths reacting to referencing and intervention gestures. The robot agent appropriately recognizes gestures referencing specific objects and regions, and gestures indicating corrective action. Exemplars demonstrate the agent's distinction between near objects and the resolution of gesture-based priorities in complex environments. Performance measures experimented in varying scene complexity and object type exhibit notable robustness and flexibility. The results show that the use of natural human gestures in embodied artificial intelligence significantly enhances navigation agent reactivity and interpretability. Agents trained with gesture input are superior to normal models in simulated environments as well as in environments that require novel gesture understanding. The study establishes the central position of non-verbal communication in augmenting human-robot interaction and opens the door to cognitive artificial intelligence systems that can reason from natural, unstructured human inputs.

2.4 Voice Recognition Through Machine Learning Techniques

[10] The research in this case is aimed at developing a speech-based classification system that would be used to detect human activities in the context of smart homes. The research begins by establishing the work domain, noting that the majority of everyday activities in smart homes are possible through spoken commands, hence giving users more convenience and independence. The activities include, among others, lighting control, temperature settings, blind control, and appliance control through straightforward spoken statements.

To create a robust model, the researchers created categories, subcategories, and actions that the speech recognition system has to infer from user commands. Categories covered fundamental elements of smart home automation like lighting, heaters, blinds, and other appliances, while subcategories specified specific locations or related elements. Actions included general actions like turning appliances on or off, opening or closing blinds, and enabling or disabling security features. Training data utilized consisted of 663 records with annotated variables like category, subcategory, action, context inquiry, time of request, and the actual statement uttered by the user.

Data was collected by simulating real interactions between users and a smart home system, capturing varied sentence structures and context information. Each sentence in the data set was mapped to equivalent actions the system needed to perform, enabling the use of supervised machine learning techniques. [7]The model's functionality was centered on predicting the correct category, subcategory, and action from a verbal command converted to text by speech recognition libraries.

The model's implementation made use of two major machine learning algorithms: Multinomial Naive Bayes to classify categories and subcategories and Random Forest to classify actions. Multinomial Naive Bayes was used as it excels in the tasks of text categorization and in effectively working with frequency-based information. The Random Forest was used to prevent overfitting and to accommodate versatility in the categorization of a range of different actions. Text data was also transformed into numeric data via the Count Vectorizer algorithm, which extracted feature matrices from the sentences that were speech transcribed.

The data was split into training and test sets, with 80% for training and 20% for testing to check the performance of the model. The development and training of the model were supported by Python's scikit-learn library. The system also had an option of real-time speech input, where voice commands were translated into text format using Google's speech recognition API. The text output received from this option was taken as input for the trained machine learning models, which predicted the categories, subcategories, and actions. The test was conducted using a dedicated test set of 25 samples. All 23 commands were correctly identified, showing high predictive accuracy. The test cases included a

range of commands for lighting, heating, blinds, and home appliance control. The system correctly interpreted and performed commands like "Light the dining room today" and "Turn on the heating in the bathroom when outside is cold."

The consistency of results in different environments illustrated the strength of the model. The final evaluation indicated a prediction accuracy of 82.99% for categories, 76.19% for subcategories, and 90.28% for action recognition. This indicates a high ability for accurate categorization of user commands in different contexts. The confusion matrices were analyzed to establish misclassification zones, and findings indicated occasional confusion between closely related subcategories and categories; however, this did not influence the accuracy of action recognition.

Normalized confusion matrices gave a more in-depth breakdown of classification accuracy, and it was found that although categories and subcategories did contain some overlap and noise, action predictions always had a higher degree of precision. The matrices corroborated the fact that the models had learned important patterns and were adept at making sound predictions for the majority of commands. Refinement recommendations for the model were to increase the dataset to include more samples, particularly for minority categories and subcategories, and to look at more advanced natural language processing methods such as word embeddings or transformer models to enhance semantic meaning capture. Hyperparameter tuning was also recommended as a way to enhance performance, with areas of focus including vocabulary size, hidden layer sizes, learning rates, and training epochs.

Comparison with previous research validated the effectiveness of the method and demonstrated higher levels of accuracy compared to other previous deployments. Unlike other solutions that are limited to specific languages or entail costly services, the system we proposed demonstrated multilingual support and cost-effective deployment. In summary, the model in this work proves the viability and efficiency of machine learning application for human action recognition in voice-controlled smart homes. It presents an extendable and flexible framework that maximizes user interaction to make home settings more intuitive and user-friendly. Future research seeks to incorporate contextual knowledge, user preference, and advanced NLP methods to enhance system performance and resilience.

3. Methodology

3.1 Overview

The Proposed methodology implements a multimodal AI driven system of robotic control that combines computer vision, gesture recognition, natural language processing (NLP) and robotic control through CoppeliaSim's remote API. The methodology starts with human detection and gesture-based control using OpenCV and Mediapipe's Pose and Hands modules. For human detection, MediaPipe Pose Estimation model maps the detected features as key points of the body and generates a bounding box from the min and max boundaries between those key points. The area of the bounding box of the detected faces is a cue for distance — a smaller area implies the user is further away, while a larger area means the user is closer to the camera. The gesture is further streamlined with MediaPipe Hands, which detects hand keypoints, principally the wrist, and identifies waving gestures by observing oscillatory motion patterns across multiple sequential frames. When recognized, this gesture activates a conversational interface, driven by OpenAI's Dolphin 3.0 model through the OpenRouter API that analyzes spoken input (measured using Google's SpeechRecognition library), constructs responses, and verbalizes them using pyttsx3-based text-to-speech (TTS). Speech command recognition also allows for smooth transitions between interaction modes, giving commands such as "initialize movement" to trigger secondary scripts. Fig1 shows the workflow

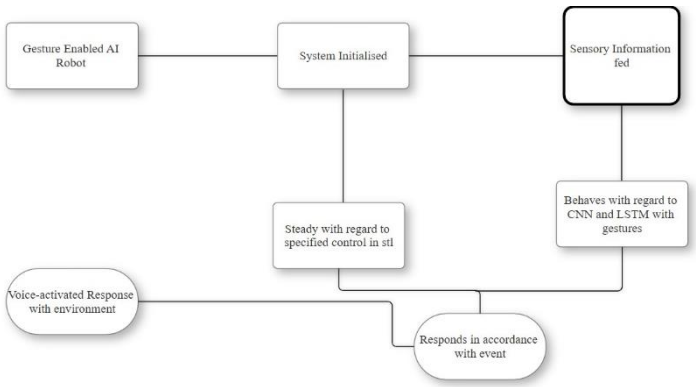


Fig 1. Architecture diagram

3.2 Human Detection and Distance Estimation using Pose Estimation

The system employs MediaPipe's Pose Estimation module to extract human body keypoints from the video stream. The model, based on deep learning and trained on large datasets, computes the location of key landmarks such as shoulders, hips, and limbs and projects them onto a normalized coordinate space. The landmarks are used to construct a bounding box around the human subject. The box boundaries are defined by the minimum and maximum coordinates of the landmarks. The size of this bounding box is used as an estimate of the subject-camera distance; a smaller size means the subject is farther away and a larger size means the subject is closer to the camera. The system computes the size in real time and compares it to a pre-calibrated reference value that is calibrated in the first few frames. This distance estimation approach is computationally efficient compared to conventional approaches based on more advanced object detection models and is accurate enough for user proximity determination in interactive applications. The process is repeated continuously, enabling the robot to dynamically adjust behavior based on how close or far the human is at any given

moment. It mainly leverages convolutional neural networks (CNN) to detect body landmarks from a video stream:

The mathematical formulation for the algorithm can be represented as:

$$P = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

where P is the set of coordinates of the detected keypoints. For each frame, the algorithm finds the minimum and maximum coordinates among all keypoints to define the bounding box:

$$\begin{aligned} x_{min} &= \min(x_1, x_2, \dots, x_n) \\ x_{max} &= \max(x_1, x_2, \dots, x_n) \\ y_{min} &= \min(y_1, y_2, \dots, y_n) \\ y_{max} &= \max(y_1, y_2, \dots, y_n) \end{aligned}$$

The area of the box can then calculated via the formula:

$$A = (x_{max} - x_{min}) \times (y_{max} - y_{min})$$

The distance estimation of how far the human is from the camera can be derived from the bounding box area compared with the intial reference area calibrated during the starting frames:

$$\text{Relative Distance} = \frac{A}{A_{ref}}$$

If the ratio is less than a certain threshold (e.g., 0.4), the system classifies the human as far; otherwise, as near. This estimation technique is computationally efficient compared to depth sensors and works well when the camera is positioned at a fixed height. Human detection is essential for initiating interactive sessions and determining proximity. It ensures that the system engages with users only when they are within a reasonable range. By leveraging CNN-based keypoint detection, the system achieves high accuracy and real-time performance without relying on computationally expensive models like YOLO or SSD.

3.3 Gesture Recognition via Hand Tracking and Wave Detection

The gesture recognition component of the system employs MediaPipe Hands, a convolutional neural network-based module, to extract 21 distinctive keypoints of the hand from video frames. The module provides strong hand landmark detection robust under changing illumination and hand orientations. The system specifically monitors the wrist location and tracks its motion across a sequence of consecutive frames stored in a deque data structure. When the wrist is comparatively steady over a period of time—corresponding to a stable hand pose—the system enters an oscillatory movement mode where it starts analyzing the next oscillatory motion. These oscillatory motions, marked in alternate left and right displacements, are compared with fixed thresholds to determine whether a wave gesture is being performed. A wave gesture is identified when the frequency and amplitude of the wrist motions exceed specified thresholds, such as the number of direction changes and total displacement above a threshold value. The algorithm employs temporal analysis across a sliding window to avoid transient motions being detected and only intentional gestures to trigger further processing. This gesture recognition mechanism is critical in facilitating the conversational

interface and triggering further control commands. The primary analysis for the recognition module that is used to calculate hand landmark extraction and analyzing wave patterns can be calculated as per the following:

The wave detection algorithm tracks the x-coordinate of the wrist over a sequence of frames.

$$W = \{x_1, x_2, \dots, x_t\}$$

Here x represents the x-coordinate of the wrist in the i^{th} frame and the algorithm calculates the difference between the consecutive coordinates:

$$\Delta x_i = x_{i+1} - x_i$$

The wave pattern is identified by tracking the oscillation count and the magnitude of movements and the direction change can be calculated:

$$\text{Direction Change} = \sum_{i=1}^{t-1} [\text{sign}(\Delta x_i) \neq \text{sign}(\Delta x_{i+1})]$$

If the direction change count exceeds a threshold and the total oscillation amplitude meets the criteria, the system classifies the motion as a wave gesture. We use MediaPipe's 21 hand keypoints to enable accurate wrist tracking with deque-based storage to efficiently manage the oscillation data for rapid analysis. Wave detection allows the robot to identify when the user intends to initiate communication. Real-time oscillation analysis is critical because it differentiates between intentional waves and incidental hand movements. MediaPipe's 21 hand keypoints enable accurate wrist tracking, while deque-based storage efficiently manages the oscillation data for rapid analysis.

3.4 Voice-Based Interaction and Conversational Interface

The conversational module is an essential part of the system in the sense that it introduces voice-based interaction capability. The module uses the SpeechRecognition library to identify speech input from a microphone and translate it into text using Google's Speech-to-Text API. The text is then submitted to an NLP engine using the OpenAI's Dolphin 3.0 model through the OpenRouter API. The model translates the input query by tokenizing it and using attention mechanisms to decode context and produce an appropriate response. The produced response is then converted into speech using the pyttsx3 text-to-speech engine, allowing the system to deliver real-time auditory feedback to the user. The conversational interface accommodates command-based inputs and general questions, allowing a user to either converse or input special commands, such as allowing the robot to transition from one operating mode to another. Speech command recognition is an essential feature of this module; in the case that the user provides an input command such as "initialize movement," the system terminates the conversational session, releases the active video streams and other resources, and then calls a secondary script that places the system in a motion control mode. The voice-based interaction module allows the system to be very interactive and multimodal input-capable in real-time. The API employs recurrent neural networks (RNNs) in combination with transformer models to transcribe the spoken languages into textual format that can be processed by the computer interface and program.

The model first extracts acoustic features from the audio signal, typically in the format of Mel-Frequency Cepstral Coefficients (MFCCs):

$$MFCC = DCT(\log(|STFT(x)|^2))$$

Where:
STFT(x) = Short-Time Fourier Transform of the signal
DCT = Discrete Cosine Transform

The RNN layer then models temporal dependencies, while the Transformer layer applies attention mechanisms to capture contextual relevance Where Q, K,V are the query, key, and value matrices and d_k is the dimensionality of the keys:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

The system processes voice commands to enable hands-free interaction. This capability is especially useful when the user wants to switch between modes or directly control the robot without using gestures.

The role of NLP in this system is pivotal for creating an interactive conversational interface that can understand spoken user commands, process them intelligently, and respond appropriately. This functionality is achieved through a combination of Speech-to-Text (STT), Text Processing, Natural Language Understanding (NLU), and Text-to-Speech (TTS). Using a robust pre-trained language model like mistral Dolphin 3.0, the system is able to engage in meaningful and context-aware dialogue with the user. Speech Recognition is used to convert spoken language into text through tools like the Google Speech Recognition library, which transcribes audio signals for further analysis. The transcribed text undergoes Text Processing, where tasks like tokenization, lemmatization, and syntactic parsing clean and structure the data for better understanding. Natural Language Understanding then interprets the user's intent and meaning, typically using advanced machine learning models like GPT-4 to analyze the input's semantics. Finally, Text-to-Speech converts the output text back into speech, allowing the system to respond in a natural, conversational voice, enhancing user interaction.

The initial phase of the NLP system is Speech Recognition, where spoken words are captured via a microphone and converted into text. This complex process involves several steps. First, during Pre-Processing, the audio signal is captured, noise is reduced, and volume levels are normalized, with background sounds filtered out to focus on the speaker's voice. Next, in Feature Extraction, the audio signal is transformed into a sequence of features, often represented as Mel-frequency cepstral coefficients (MFCCs), which compactly capture the speech signal's frequency content. Finally, in the Recognition step, the feature vectors are passed through a trained machine learning model typically a deep neural network or Hidden Markov Model that predicts the most likely transcription, outputting a sequence of text tokens (words or sub-word units). The pipeline can be represented as follows:
Sentence Segmentation where the text is divided into sentences. Each sentence S is split into words $w_1, w_2, \ldots w_n$ where n is the number of words in each sentence

$$S = \{w_1, w_2, w_3, \ldots, w_n\}$$

Next Lemmatization process occurs which reduces words to their base form in order for for the NLU

training and semantic model training to process the text faster and extract its semantic meaning. The transformer-based architecture relies on a self-attention mechanism to capture relationships between words in the input text. The model is trained on vast amounts of text data, which allows it to predict the most likely output for a given input sequence. The operation can be expression as:

$$Q = XW_Q K = XW_K V = XW_V$$

X is the input sequence of tokens.
W_Q, W_K, and W_V are learnable weight matrices for Query, Key, and Value vectors, respectively.
Q, K, and V represent the transformed query, key, and value vectors.

The self-attention mechanism computes the attention scores for each token in relation to the others using the dot product Where d_k is the dimension of the key vector. This mechanism enables the model to weigh the importance of each token in relation to others, effectively capturing dependencies over long distances in the text.:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Once the system understands the user's intent, it generates a response. Text-to-Speech (TTS) is used to convert this response back into speech. The TTS engine processes the output from the language model (Dolphin 3.0) and generates a natural-sounding speech signal that can be played through the system's speakers. The TTS system typically relies on a neural network-based architecture, such as WaveNet. The key principle behind WaveNet is to predict each audio sample in the waveform sequentially, conditioned on the previously generated samples. Mathematically, the TTS system predicts the waveform y_t at time t as follows:

$$y_t = f(y_{t-1}, \text{context})$$

Where:

f is a function (a deep learning model) that predicts the next audio sample
$y_{\{t-1\}}$ is the previous audio sample, and context includes additional information such as prosody and phoneme embeddings.

Incorporating NLP enables the system to respond to user queries dynamically. By combining Speech-to-Text for understanding verbal input, Natural Language Understanding for interpreting meaning, and Text-to-Speech for generating spoken responses, the system can engage in meaningful conversations with users. modern TTS techniques such as WaveNet ensures that the interaction is natural, responsive, and efficient, thus completing the feedback loop between user input and robotic action. With this detailed NLP framework, the system not only becomes adept at responding to user gestures but also excels in interpreting voice commands, effectively bridging the gap between human intent and machine action.

3.5 Background Averaging and Hand Segmentation for Finger Counting

The gesture control module utilizes an algorithm that utilizes background averaging to derive a stable reference image for subsequent hand segmentation. For the first 60 frames of the video stream, the system calculates a weighted average of the grayscale presentation of the frames via OpenCV's accumulateWeighted function. This yields a background model capturing the static elements of the scene, essentially eliminating dynamic objects such as the inserted hand of the user when it is placed into a pre-configured region of interest (ROI). Once the background model has been created, each incoming frame is processed for a subtraction stage where the absolute difference between the background and incoming frame is calculated. A thresholding operation is applied to the difference image to convert it to a binary image that separates the hand from the background. The thresholded result is analyzed for extracting contours where the largest one is presumed to be the hand. This is a light-dependent segmentation process with proper tuning of the threshold value to successfully remove the hand from any background noise. The resulting segmentation output presents a binary mask utilized in follow-up processing for finger counting as well as direction analysis.

The hand segmentation process relies on OpenCV's background subtraction technique, which establishes a reference image against which moving objects can be detected. The technique is implemented using an accumulative weighted average, allowing the model to gradually adapt to changes in the background without requiring a static environment. The background model is constructed by averaging a series of frames captured during an initialization phase. For each frame F_t, the background model B_t is updated using the formula:

$$B_t = (1 - \alpha)B_{t-1} + \alpha F_t$$

Where:
B_t = Updated background model at time t
B_{t-1} = Previous background model
F_t = Current frame
a = Accumulated weight (typically between 0.1 and 0.5)

The absolute difference between the current frame and the background model is calculated as:

$$D_t = |F_t - B_t|$$

A threshold operation is then applied to generate a binary image representing the detected hand where θ is a predefined threshold value:

$$T(x, y) = \begin{cases} 1 & \text{if} D_t(x, y) > \theta \\ 0 & \text{otherwise} \end{cases}$$

This method effectively distinguishes the user's hand from a relatively static background, enhancing segmentation accuracy. By continuously updating the background model, the algorithm can adapt to gradual changes in lighting conditions, making it robust for real-time applications. The segmented hand image serves as the foundation for further processing, including gesture recognition, finger counting, and directional analysis.

3.6 Finger Counting, Convex Hull, and Pointing Direction Analysis

After isolating the hand from the background, the system computes the convex hull of the hand boundary to locate the extreme boundary points of the hand. The convex hull operation forms a polygon that encloses the hand tightly such that the most extreme points along the top, bottom, left, and right are used. These extreme points are utilized as a reference point for approximating the center of the hand, which is located as the midpoint between the top and bottom points and the midpoint between the left and right points. Once the center is located, the system computes the maximum Euclidean distance between the center and the extreme points to compute a radius. A circular region of interest (ROI) is then formed using this radius, and a bitwise AND operation is conducted between this circular mask and the thresholded hand image. The resulting output is an image that isolates the finger segments from the rest of the hand. The system then locates contours from this masked image and uses criteria based on the contour's bounding rectangle and the ratio of the circumference to determine the number of fingers. The finger count, which is typically in the range of 0 to 4, is utilized as an input to further decision-making for robot movement. In parallel, the system also interprets the pointing direction by locating the fingertip that is furthest from the hand's centroid. The pointing direction is determined on the basis of fingertip position with respect to the centroid; if the fingertip is positioned over the centroid, the direction of the fingertip with respect to the vertical axis is determined to find left or right pointing direction, and below-centroid fingertip indicates backward movement. Analysis of finger number and pointing direction combined is the primary method by which the system determines user intention for movement control.

The system computes the convex hull of the segmented hand image to identify key features such as fingertips and valleys between fingers. The convex hull represents the smallest convex shape that can enclose all points within the detected hand contour. Given a set of points $P = \{(x_i, y_i)\}$ representing the hand contour, the convex hull H is a subset of P satisfying the following that the points in H form a convex polygon and that every point in P is either inside or on the boundary of the polygon defined by H. The convex hull is calculated using Graham's Scan or Andrew's Monotone Chain Algorithm, with a complexity of O (n log n). The defects or valleys between fingers are identified by detecting the points where the contour deviates most significantly from the convex hull. These points are identified by measuring the Euclidean distance between the hand's centroid and the defect points. The number of fingers displayed is determined by analyzing the contour formed by the intersection of the segmented hand image and the convex hull. This process involves:

1. Calculating the centroid of the hand using image moments:

$$C = \left(\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}\right)$$

Where:
M_{00} = Total area of the contour.
M_{10} and M_{01} = Summations of pixel intensities weighted by their x and y coordinates, respectively.

2. Constructing a circular mask centered at C with a radius determined by the maximum Euclidean distance between C and the extreme points of the convex hull:

$$R = max\left(\sqrt{(x_i - C_x)^2 + (y_i - C_y)^2}\right)$$

3. Performing a bitwise AND operation between the thresholded image and the circular mask to isolate finger segments.

4. Counting the contours within the masked region to estimate the number of extended fingers.

This approach enables accurate finger counting by simplifying the segmentation process and filtering out irrelevant parts of the image. The use of convex hulls ensures robustness against noise and partial occlusions, providing a reliable basis for gesture interpretation.

The pointing direction detection mechanism is built upon analyzing the relative positions of the fingertip and the hand's centroid. By extracting the most distant point from the centroid, the algorithm identifies the user's intention to direct the robot's movement. The directional vector analysis algorithm is applied here.

1. The centroid of the hand is computed as described earlier.
2. The fingertip is identified as the point within the contour that lies farthest from the centroid:

$$F = \arg \max_{(x_i, y_i) \in H} \sqrt{(x_i - C_x)^2 + (y_i - C_y)^2}$$

3. The direction vector is then calculated as:

$$\vec{d} = (x_f - C_x, y_f - C_y)$$

4. If the fingertip lies below the centroid, the command is interpreted as a request for the robot to move backward. Otherwise, the angle θ between the pointing vector and the vertical axis is calculated using:

$$\theta = \arctan\left(\frac{y_f - C_y}{x_f - C_x}\right)$$

5. Based on the value of θ:

$-30^\circ \leq \theta \leq 30^\circ$: Forward direction
$\theta < -30^\circ$: Left turn
$\theta > -30^\circ$: Right turn

This approach ensures robust detection of pointing gestures, even when the hand's orientation varies. The vector analysis provides a clear distinction between the intended directions and simplifies the process of translating gestures into robotic commands.

3.7 Robotic Control through CoppeliaSim Integration

The robotic control aspect is realized through communication with CoppeliaSim via its remote API. The system logs into the simulation platform, retrieves the motor handles for the left and right wheels of the robot, and then sends velocity commands based on the processed gestures. Each movement command—forward, backward, left, right, or stop—is defined as a function that addresses the desired velocity of the corresponding motors using the streaming mode of the API. The mapping of finger count to movement command is done directly so that one finger corresponds to forward movement, two fingers to left turns, three fingers to right turns, and four fingers to backward

movement. When no valid gesture is detected, the system sends a stop command to prevent unwanted movement. The interaction with CoppeliaSim gives the guarantee that the robot reacts almost instantaneously to user commands, with the API functions supplying the control signals to the motors. This control cycle is continuously executed, processing the gesture data and updating the movement of the robot in real time. The simulation platform offers the vehicle for comprehensive testing and calibration of the control parameters so the system will function reliably under diverse conditions. The design eschews the use of physical prototypes in the early stages in the development process and permits iterative improvement through simulated tests.

The robotic control module is tasked with the interpretation of processed gesture inputs and conversion of the same into executable commands that drive the movement of the robot within the CoppeliaSim simulation environment. The main function entails coordination of multiple subsystems, such as vision processing, signal interpretation, and motor actuation, to enable smooth coordination of robot behavior through gesture recognition. The overall design utilizes Python scripts that interact with CoppeliaSim through the CoppeliaSim Remote API, a TCP/IP-based interface for real-time simulation object control. This interaction necessitates proper configuration of network parameters, function calls, and error-handling routines to enable reliable operation.
The initialization process starts with the creation of a client-server relationship between CoppeliaSim and the Python script. The client is the Python script, and the server is CoppeliaSim in this configuration. The client-server approach is necessary to ensure asynchronous communication between the two platforms to guarantee real-time feedback and control. To link the two systems, the script invokes the simxStart function with the IP address of the server, port number, and the wait period when attempting to connect. Verification of a successful connection is done when the function returns a status value that shows the client-server connection is active and ready for data exchange. Initialization includes the Python script also executing setup operations like the creation of communication parameters, simulation mode setting, and cross-verification of Python version against CoppeliaSim API version. Apart from that, initialization is the step of setting up the simulation environment to accept control commands. It includes loading necessary models, setting up the vision sensor, and initializing robot joints to their initial positions. The step also includes initializing the necessary simulation mode, synchronous or asynchronous mode, depending on the needed application. Properly setting this connection is needed because any incompatibility developed in the initialization step can lead to communication failure, poor responses, or uncontrolled robot movement. The connection between Python and CoppeliaSim is initiated using the function: clientID=sim.simxStart('127.0.0.1',19999, True, True,5000,5). If a valid connection is established ((clientID≠−1) the script proceeds to retrieve the handles for the robot's left and right motors using:

errorCode,left_motor_handle=sim.simxGetObjectHandle(clientID,'LM',sim.simx_opmode_oneshot_wait)
errorCode,right_motor_handle=sim.simxGetObjectHandle(clientID,'RM',sim.simx_opmode_oneshot_wait)

This setup allows the Python script to directly manipulate the robot's actuators by modifying the velocity of the motors. This setup allows the Python script to directly manipulate the robot's actuators by modifying the velocity of the motors. The movement of the robot is achieved by adjusting the velocities of the left and right motors. The general formula for setting motor velocity is:

$$V_L = k_1 \cdot v, V_R = k_2 \cdot v$$

Where:
1. V_L and V_R are the velocities of the left and right motors, respectively.
2. k_1 and k_2 are scaling factors that determine the type of motion.
3. v is the base velocity set by the user.

Different motions are achieved by adjusting these parameters and then these velocities are transmitted to CoppeliaSim using the streaming mode for continuous operation:

```
sim.simxSetJointTargetVelocity(clientID,left_motor_handle,VL,sim.simx_opmode_streaming)
sim.simxSetJointTargetVelocity(clientID,right_motor_handle,VR,sim.simx_opmode_streaming)
```

By dynamically adjusting these velocities based on the interpreted gestures, the system ensures real-time responsiveness, which is essential for effective human-robot interaction.

The Remote API of CoppeliaSim employs a command-response protocol using TCP/IP with a high-level interface to send commands and receive status from the simulation environment. The mechanism for communication is non-blocking function calls, which enhances the responsiveness of the control system. Data is communicated via packets that carry command identifiers, arguments, and response types. The protocol is flexible to support various types of messages, including position control, velocity control, object state queries, and sensor data retrieval. The architecture of the Remote API is designed to support real-time interaction with the capability of parallel communication channels, which supports the execution of multiple commands in parallel.
In addition, the communication protocol helps to send and receive messages in a structured fashion. It is based on mutual agreement on status codes in order to allow successful command execution or the reporting of errors that may occur when communicating. This structured method helps the Python script handle failures in an efficient manner, using retry protocols or other techniques to deliver the required results. The capability to build commands and responses in the correct format is essential to ensure an orderly and consistent control loop that continuously checks the status of the simulation environment.

Once the client-server connection is established, the Python script can send commands to manipulate objects in the simulation world. An example of this is the movement of a robot, which requires an object handle to be retrieved by calling the simxGetObjectHandle function. The object handle is a special reference to future commands to the target object. Common control operations include setting joint positions, setting joint velocities, and retrieving object states such as position and orientation. The Remote API has a number of control functions to accommodate different control paradigms such as position control, velocity control, and force control. For position control, the script has functions like simxSetJointTargetPosition that allow the programming of the joints of a robot to target positions with accuracy. Velocity control is done using the simxSetJointTargetVelocity function, which calculates the rotation or linear speed of a joint. These control systems are imperative in the implementation of gesture-based interactions, in which identified gestures are mapped onto preprogrammed movements of a robot. Complex interactions can employ the use of multiple control functions in combination to generate coordinated motion, like object picking, sketching paths, or replicating human gestures with high accuracy.

Gesture mapping entails mapping recognized gestures to robot action. The process is generally the definition of a gesture-action set, whereby a gesture is mapped to a sequence of commands. A waving gesture recognized by the vision sensor prompting the robot to wave using its arm actuators is an illustration. The gesture recognition model continuously scans for data from the vision sensor, searching frames for the appearance of recognized gestures in real-time. The performance of the model is crucial to the success of the overall system since any error in gesture recognition will have

a direct effect on the response of the robot. When a gesture is detected, the corresponding sequence of commands is calculated and issued to CoppeliaSim via the Remote API. The sequence of commands may be a sequence of joint control commands to generate compound movements, such as waving, pointing, or following a moving target. The modularity of the architecture supports the addition of new gestures and their corresponding actions without changing the communication structure. The gesture-action mapping is enforced by adding feedback loops that monitor the actions of the robot to ensure that the intended gestures are generated correctly.

Effective robot control relies on continuous feedback from the simulated world. Remote API enables functionality like simxGetJointPosition and simxGetObjectPosition for immediate reporting of the state of the robot. These kinds of information are necessary to ensure accurate motion performance and detection of potential errors when in use. In advanced systems, feedback mechanisms are typically complemented with error-handling software that points out anomalies and initiates corrective action as needed. Error handling processes are implemented to deal with connection loss, command execution failure, and data acquisition failure. For instance, when a connection between the client and server is lost, the script attempts to reconnect using simxStart in a loop until it can reestablish the connection successfully. Additionally, command execution status is monitored to confirm that issued commands are successfully executed by the simulation environment. Advanced error handling strategies may include error message logging, dynamic adjustment of control parameters, or switching to fallback operation mode if faced with extreme issues.

The architecture detailed here is modular and scalable, with the ability to add more sensors, gesture modalities, and robot behaviors without major modification to the underlying control system. Use of the Remote API gives the ability to utilize a range of simulated objects, so the system is flexible for robotic applications. The modularity of the architecture will enable continued refinement, as developers can implement new capabilities or improve existing ones over time. The robotic control module effectively bridges the gap between gesture recognition and robotic motion control in the CoppeliaSim simulation environment. Leveraging the CoppeliaSim Remote API, the system provides real-time communication, precise object control, and good error handling. The modularity of the system ensures ease of expansion, thereby rendering the system an extensible platform for developing interactive robotic systems based on gesture inputs.

3.8 Integration of Modules and System Workflow

System design is composed of multiple independently designed modules executing concurrently to provide an integrated human-robot interaction solution. The video capture module delivers real-time frames as input to the human detection and hand segmentation pipelines. The human detection module, utilizing MediaPipe Pose Estimation, continuously detects user presence and distance. Concurrently, the gesture recognition module, utilizing MediaPipe Hands and background subtraction techniques, processes the ROI to detect hand features. The hand-segmented image is fed through convex hull calculation, from which finger counting and pointing direction are derived. Concurrently, the speech-based interaction module processes voice commands to enable or disable specific modes, switching between conversational and movement control as appropriate by the user. Each module has timing requirements but is synchronized to provide the output of one module as the input to another without compromise to data integrity. The system's control loop is optimized to address asynchronous events such as background changes, user distances varying, and transient gestures to provide the robot with continuous updates of user intent. Data is fed from the camera into the processing units, and final movement commands are transmitted to CoppeliaSim's simulation environment. The integration approach is designed around robustness and real-time response, with multiple levels of error handling to address potential communication failure between

modules. This design limits delay and maximizes the accuracy of gesture interpretation and robotic actuation, providing a tightly coupled system that is responsive and reliable.

3.9 Design of the Robot

The following figures demonstrate the initial stages of the Gesture Robot's design starting from the initial blueprint design to the completed 3D object model and then finally showcasing its integration in the software CopelliaSim as well as demonstrating how a non-convex 3D object model is rigged in order to perform motion dynamically as well as integrate it into the simulation environment. The 3D robot model was designed using the SolidWorks software and then exported into CopelliaSim as a 3D object riggable model.

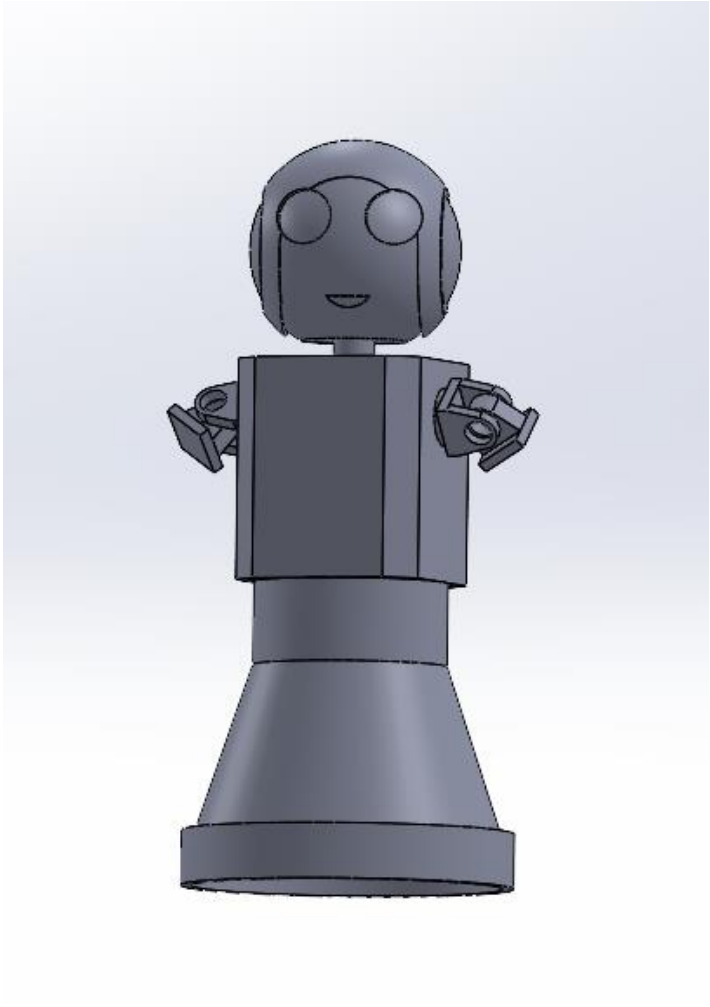


Figure 2. Initial protype blueprint in SolidWorks

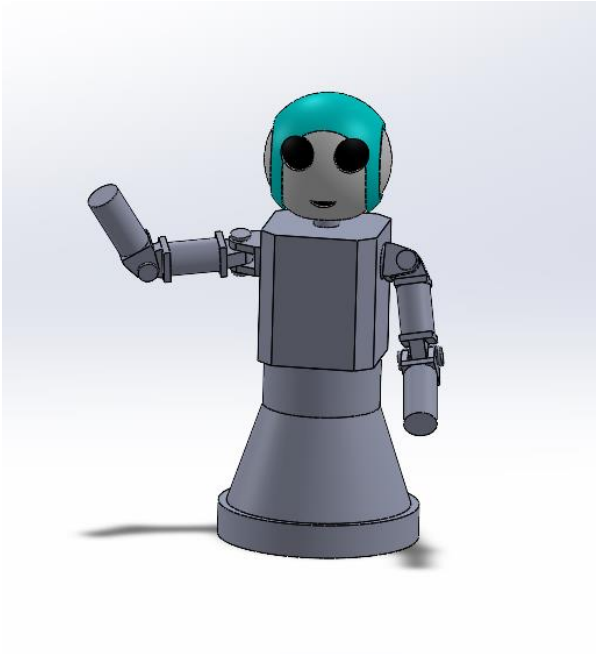


Figure 3. Completed 3D model design

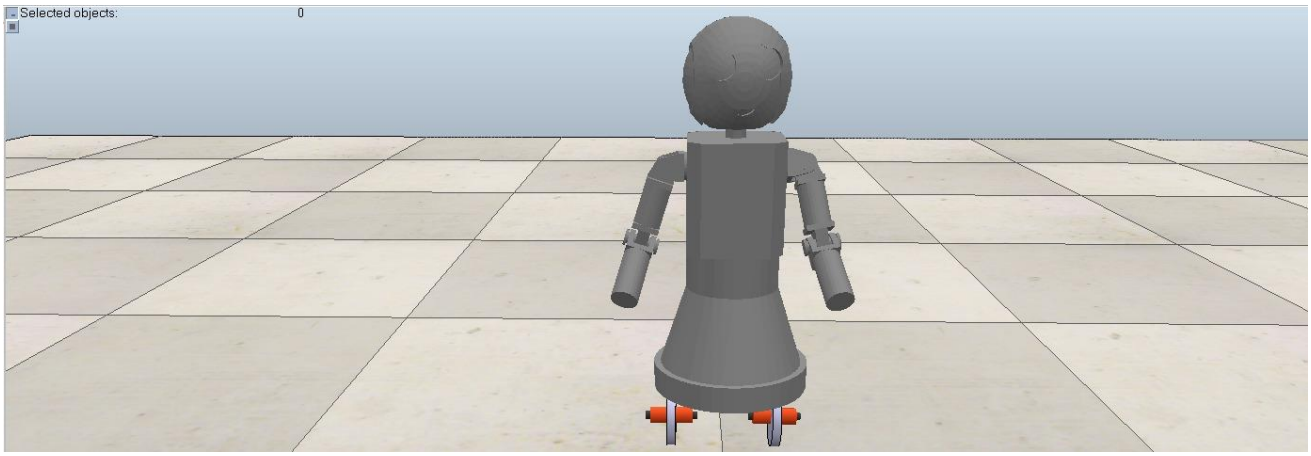


Figure 4. Completed 3D model imported into CopelliaSim along with wheels for movement

Our robot design, as indicated in the figure, is a straightforward yet very efficient design intended for gesture recognition and interaction within the CoppeliaSim simulation environment. The robot is of a humanoid nature, made up of a cylindrical body and jointed limbs, which is employed to increase its capability to simulate human gestures. The robot head is spherical, with a flawless surface upon which sensors such as vision modules needed for the identification of human gestures and environmental signals can be mounted. The smooth and rounded shape of the head provides it with an omnidirectional view, which provides it with the capability to sense and process gestures in all directions.

The cylindrical shape of the robot body is also designed to provide structural stability at low manufacturing complexity. The uncomplicated body reduces the number of parts required for the manufacturing process, which in turn reduces the material cost and makes the design highly cost-effective. The body is also a convenient mounting location for most of the internal parts, including processing modules, batteries, and actuators, which keep the center of gravity of the robot low and stable. This design has a direct impact on improving the efficiency of the robot in task performance for gesture-based interaction because a stable structure is highly critical for accurate motion performance.

The articulated arms of the robot are created to imitate basic human gestures such as waving, pointing, and raising arms in greeting. The use of simple cylindrical arms with joint mechanisms makes it possible to have a high degree of movement, which is required to react effectively to user gestures. The inclusion of motorized joints allows for the control of the arms with high accuracy, ensuring that the gestures of the robot are consistent with the commands from the gesture recognition module. Furthermore, the ease of designing the arms makes it possible to easily customize and scale, allowing for extra degrees of freedom or upgrading the ability of the robot without altering the basic architecture.

The base of the robot is a conical base that smoothly tapers into a circular platform with the wheels and robot support structure. The conical shape provides structural support as well as enhancing the robot's appearance. The wheels as the primary locomotion mechanism were a design choice to enhance mobility and motion flexibility in the simulated environment. Wheels provide an efficient yet simple form of motion, reducing the added complexity of legged robots while maintaining efficient translation and rotation capability. The wheel-based motion system also requires less processing power for control, thus making the robot's operation more efficient.

The design is particularly ideal for our project since it is highly well-balanced in terms of functionality, simplicity, and cost. The humanoid shape of the robot is more intuitive and human-friendly for human users to interact with, which is very crucial for a gesture-based interaction system. The smooth, uninterrupted surfaces of the robot body also make it easier to implement color-based detection algorithms since the lack of intricate details reduces the likelihood of false positives during visual processing. The modular design also makes it easy to add other components, like microphones or tactile sensors, without compromising the overall structural integrity of the robot.

One of the primary reasons that this design is particularly suited to our project is that it is compatible with the CoppeliaSim simulation environment. The geometry of the robot being low-complexity means that simulation is possible at speed without sacrificing accuracy. The lower computational overhead of simulating simple shapes results in faster processing, allowing real-time interaction with the gesture recognition module. The fact that the CoppeliaSim Remote API is being used for control also means that the robot's behavior can be modified and optimized at runtime, allowing the system to be extremely flexible to allow for different experimental setups.

From a cost-effectiveness point of view, the robotic design is particularly advantageous. The conical and cylindrical components are simple to produce using common production techniques such as 3D printing or molding, with the process being highly inexpensive. The modular design process also facilitates simple maintenance and repair, with the components being easy to replace without having to redesign the system significantly. This is particularly useful in projects with financial constraints, where cost reduction is important without losing functionality.

The design scalability is also one of the most important factors that contribute to the overall efficiency of the design. The robot's structure can be extended with new functionalities as development continues without redesigning the entire robot. As an instance, new actuators or sensors can be included to support the robot's interaction capabilities, or the control algorithms can be revised to achieve better performance. The design modularity also allows experimentation with various configurations, whereby researchers can experiment with various methodologies for gesture-based interaction without the limitation of the robot's physical structure.

In addition, the simplicity of implementation with this design is to be appreciated. The ease of assembly means that the robot can be built and deployed in a matter of days. This is a key feature that

is of particular value in the case of iterative testing and improvement of the gesture recognition and control algorithms. The speed at which the robot's physical configuration can be altered and reassembled facilitates faster development, with the ability to prototype and verify new concepts quickly. As far as its strength goes, the robot's design is intended to withstand the stresses of continuous use within the simulated environment. The durability of the cylindrical body combined with the stability of the conical base ensures that the robot does not topple while in transit. In addition, the use of simple geometric shapes minimizes the chance of the mechanical failure since there are fewer intricate parts that can fail while in use. The reliability of the design is also provided through the use of materials, which can be optimized to provide specific requirements depending on the application. Overall, the robot design is a practical solution to the introduction of robust gesture-based interaction in an artificial world. Its simplicity of form, cost-effectiveness, scalability, and ease of implementation make it the ideal solution for our project. The flexibility of the design also enables it to be scaled up and modified as needed, providing it with a rock-solid foundation for the development of advanced gesture recognition and control systems. The use of this design as the foundation for our solution enables us to bridge the gap between human gestures and robot response in an efficient manner, creating a flexible foundation for the exploration of new interaction schemes. The modularity of the robot architecture also enables developers to incrementally optimize individual components as new technologies and methodologies become available, ensuring the long-term viability and relevance of the project.

3.10 Initial program configuration

The initial configuration of the program is the bare minimum needed to initiate communication between the Python script and CoppeliaSim, define the key control parameters, and prepare the system for optimal robot performance. The process starts with the initialization of several global variables, such as those for gesture recognition and motor control. The 'background' variable is given a value of 'None' to enable background subtraction in image processing, a key step for optimal hand gesture recognition. The 'v0' variable is given a value of '2', the robot's initial motor speed, to be dynamically adjusted during movement commands. Another variable, 'accumulated_weight', is given a value of '0.5'. The value is used as a weight parameter for controlling the effect of newly acquired frames in the background subtraction process, striking a balance between responsiveness and stability of gesture detection. The program is done in python

The program also outlines the region of interest (ROI) for gesture detection. The ROI is specified manually through four parameters: 'roi_top', 'roi_bottom', 'roi_right', and 'roi_left' that indicate a rectangular part of the camera frame where detection of gestures shall be carried out. These are tuned to cover the area likely to have informative hand gestures and optimize efficiency through processing only a defined area, not the full image. The method also optimizes false detection and system performance through less computation.

```
background = None
v0=2
accumulated_weight = 0.5
# Manually setting up our Region Of Interest (ROI) for grabbing the hand.
roi_top = 20
roi_bottom = 300
roi_right = 300
roi_left = 600
```

Following the initialization of ROI, the program interacts with CoppeliaSim through the Remote API. The call to the function 'sim.simxFinish(-1)' is done to close any previously established connections to avoid interrupting the current session. The connection attempt is done by 'sim.simxStart()', which

supplies the IP address '127.0.0.1', port number '19999', and other connection parameters to ensure the communication is smooth. The if-else statement usage checks the validity of 'clientID', printing a success message when the connection is established and terminating the program when it fails. This setup is necessary to enable real-time control of the simulated robot from the Python environment.After establishing the connection successfully, the script proceeds to acquire motor handles with 'sim.simxGetObjectHandle()'. There are two separate function calls made to get handles of left motor and right motor of the robot 'LM' and 'RM', respectively. These handles are initially required in order to send motor control commands to CoppeliaSim since they contain a reference to the simulated objects to be controlled. Whatever error is faced during the process is indicated by the 'errorCode' returned by every function call, which allows proper error handling and debugging.

The motor control functions are then defined, which are modular functions that produce certain movement patterns. The functions 'forward()', 'left()', 'right()', 'backward()', and 'stop()' are defined to control the movement of the robot by changing the velocities of the left and right motors. For instance, in the 'forward()' function, the velocities of both motors are 'v0', which enables the robot to move forward. In the 'left()' function, the left motor is brought to stop by setting its velocity to zero, while the right motor is still moving, causing the robot to turn left. The same applies to the 'right()' and 'backward()' functions, where the velocities of the motors are set accordingly to achieve the desired directional movement. The 'stop()' function stops both motors by setting their velocities to zero, so that the robot does not move when there are no commands.

```
errorCode,left_motor_handle=sim.simxGetObjectHandle(clientID,'LM',sim.simx_opmode_oneshot
_wait)
errorCode,right_motor_handle=sim.simxGetObjectHandle(clientID,'RM',sim.simx_opmode_onesh
ot_wait)
```

The motor control commands are executed using the 'sim.simxSetJointTargetVelocity()' function. The function transmits the target velocity values to the robot motors using the established TCP/IP connection. The 'sim.simx_opmode_streaming' parameter is applied to update motor velocities in real-time on a continuous basis, allowing dynamic and responsive control of the robot. The streaming mode prevents interruption of application of motion commands, thus enhancing the responsiveness of the system to gesture inputs.

```
def forward():
   v0=2
   errorCode=sim.simxSetJointTargetVelocity(clientID,left_motor_handle,v0,sim.simx_opmode_str
eaming)
    errorCode=sim.simxSetJointTargetVelocity(clientID,right_motor_handle,v0,sim.simx_opmode_s
treaming)
```

In addition, the program has thresholding parameters required for enhancing the gesture detection accuracy. Parameters such as 'STILL_THRESHOLD', 'STILL_DURATION', 'WAVE_DEQUE_LEN', 'WAVE_DIRECTION_CHANGES_REQUIRED', and 'WAVE_MOVEMENT_THRESHOLD' are used for enhancing the gesture recognition algorithm. The 'STILL_THRESHOLD' refers to the maximum normalized movement that is 'still' and can be detected, allowing meaningful gestures to be recorded instead of random hand movement. The 'STILL_DURATION' denotes the duration for which the hand must stay still before wave detection to ensure that gestures are meaningful and not accidental.
The 'WAVE_DEQUE_LEN' parameter specifies how many frames to process for wave gesture detection and offers a buffer to hold movement data. The 'WAVE_DIRECTION_CHANGES_REQUIRED' parameter specifies how many oscillations to require to detect a wave so that the gesture is distinct and unambiguous. The 'WAVE_MOVEMENT_THRESHOLD' parameter specifies the amount of minimum horizontal movement difference to detect a gesture as a wave. All of these parameters serve to make the

gestures more robust by removing false positives and properly distinguishing between planned gestures and noise.

```
STILL_THRESHOLD = 0.03        # Maximum normalized movement considered as "still"
STILL_DURATION = 2            # Hand must be still for 2 seconds before wave detection
WAVE_DEQUE_LEN = 8            # Number of frames for wave detection accumulation
WAVE_DIRECTION_CHANGES_REQUIRED = 2  # Oscillations needed to register a wave
WAVE_MOVEMENT_THRESHOLD = 0.15 # Minimum horizontal movement difference
```

In general, the initial setup of the program is the foundation of the gesture-based control system. Through the establishment of the region of interest using specific definition, establishment of communication with CoppeliaSim, acquisition of motor handles, and establishment of strong motor control functions, the program provides a foundation for gesture recognition to be integrated with robot control. The addition of the thresholding parameters that were precisely tuned also increases the ability of the system to detect and respond to user gestures effectively, providing a solid and effective means of interaction. The use of the modular method in establishing motor control functions also provides scalability and ease of modification, ensuring that future modifications and extensions can be integrated into the system as required.

3.11 Voice and TTS conversational interface

The provided code snippet defines a function named conversation_mode() that combines various modules for creating an integrated interactive system containing text-to-speech (TTS), speech-to-text (STT), chatbot response, and protocol initiation of movement. The function is employed as the entry function to initiate conversation with a user, interpret voice commands, read out questions provided by a chatbot, and initiate other protocols as and when necessary. The code begins by declaring a greeting message, which is converted to speech using the speak_text() function. The function would, of course, be employing the services of a TTS engine to voice out the greeting and render the conversation more user-friendly by providing sound feedback. The recognizer object of the SpeechRecognition library is initialized to handle voice input and transcribe it to text.

```
recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        print("Listening for your question...")
        try:
            audio = recognizer.listen(source, phrase_time_limit=5)
            user_query = recognizer.recognize_google(audio)
```

With the microphone defined as the source, the code listens for the voice query from a user in a predefined period of five seconds. The employ of phrase_time_limit prevents the system from holding and waiting eternally for input. When the recognizer manages to record and process the input audio, the query in its text form is printed to the console and stored as user_query. In the event of error in speech recognition, an error message is printed and read aloud to the user, asking him or her to try again in the future. The conversation_mode() function also has an inbuilt command branch that is used to handle a special keyword, 'initialise movement,' as a trigger command to transition the system into a different mode of operation. In the event this keyword is discovered, the function releases resources by terminating the capture of the video (cap.release()) and the OpenCV window if in operation (cv2.destroyAllWindows()). All keyboard events hooked in are unhooked also to prevent any unwanted interference. Once the cleanup process is done, another process is initiated to execute the hand detection protocol using a different script named 'hand detection.py.' This process of

spawning a new process allows the system to change contexts without having to restart the entire program. The code snippet that executes the 'initialise movement' command shows a good way of dynamically changing modes of operation. If the user input is not the specific command, the code proceeds to ask for a chatbot API response using the get_chatbot_response() function. The response given by the chatbot is then printed to the console and spoken using the TTS engine.

```
if user_query.strip().lower() == "initialise movement":
        cap.release()
        cv2.destroyAllWindows()
        subprocess.Popen([sys.executable, "hand detection.py"])
        keyboard.unhook_all()
        os._exit(0)
```

To allow asynchronous speech synthesis, the code initiates a new thread for the speak_text() function, allowing it to execute concurrently without blocking the main program flow. This approach allows the system to proceed with other operations while the response is being spoken. A polling loop is utilized to monitor keyboard input while speech synthesis is being performed. By repeatedly checking whether the 'c' key is being pressed, the code allows a mechanism for the user to cancel the TTS process if necessary. When the key press is detected, the global TTS engine is immediately stopped using engine.stop(), effectively canceling the current speech. The cancel variable is set to True, which tells the code to proceed with the cleanup and execution of the hand detection protocol, as explained in the previous command branch. When speech synthesis is successfully completed without interruption, the code waits for the TTS thread to finish using the invocation of tts_thread.join(). The while loop ensures continuous monitoring of keyboard input in speech output mode. Through the use of non-blocking polling with a 0.01-second sleep, the system ensures a responsive feedback without excessive computational load. The final section of the code ensures successful and interrupted TTS cases by releasing resources and starting the hand detection protocol as required. The use of threading for TTS, subprocess management for opening other scripts, and continuous keyboard monitoring and control shows an effective and modular design for the development of an interactive system. The overall design shows concurrency, resource acquisition and release, and user experience.

```
cancel = False
    tts_thread = threading.Thread(target=speak_text, args=(answer,))
    tts_thread.daemon = True
    tts_thread.start()
```

The modularity of the code also ensures ease of modification and extension, hence suitable for integration of additional features or improvement of existing ones. The design does not pose scalability and robustness issues, hence the system is suitable for real-world use. The use of voice instructions and chatbot API also integrates a conversational aspect to the capabilities of the robot, further improving its interactive features. The use of different input and output modalities with the flexibility of dynamic protocol initiation makes the system highly adaptable and user-centered.

The approach used in the code captures the need to design software capable of smoothly switching among different operations and yet remain responsive and functional. The use of threading for TTS and polling for keyboard control gives the interaction smoothness, allowing the user to interrupt the process in the event of a need. Through the integration of mechanisms for interruption management and dynamic context switching, the code achieves high robustness and usability. Besides, the separation of various functionalities into modular units makes it easier to debug, maintain, and upgrade in the future. Asynchronous speech synthesis processing reflects an awareness of the

blocking call limitations and the need to supply non-blocking operation in interactive systems. Efficient management of resources, such as camera capture release and OpenCV window closure, makes the program as a whole more stable. The use of subprocesses for starting external scripts also offers a handy way of adding system capabilities. Overall, the conversation_mode() function is part of a broader interactive system, offering an interface for processing user questions, answering via TTS, and operating mode modification according to pre-defined commands. Its design and implementation reflect a cautious integration of various technologies into a well-structured and functional framework. The code finds a good balance between complexity and usability, and the program can be deployed in environments calling for dynamic interaction with users. The combination of asynchronous processing, modular design, and user-friendly controls reflects the system's potential for further improvement and development. This well-structured framework will allow for seamless integration of further features and functionalities, and the system will be scalable and responsive to evolving needs. The overall design of the conversation_mode() function reflects an efficient way of developing an interactive robot control system that is responsive and stable.

3.12 Distance Estimation and Gesture Recognition

This section explains the part of the code is utilized for the processing of human detection tasks, gesture detection, and enabling the conversation mode upon detection of gestures. The processing includes continuous image capturing from the webcam, frame processing for detecting humans and gestures, and enabling subsequent conversational modes upon the fulfillment of defined conditions. The code is made to operate continuously in a while loop until defined conditions necessitate it to terminate, and hence it is extremely suitable for real-time interaction-oriented robotic systems.

The initial part of the code is about acquiring the video feed through OpenCV's cv2.VideoCapture() function. The captured frame is flipped horizontally to obtain a mirror image, which adds to user interaction as it gives a more natural output. The captured frame is also in RGB format, which is necessary since the machine learning models used, specifically MediaPipe's Pose and Hands modules, need the input to be RGB format. The conversion is achieved through the function cv2.cvtColor().

```
frame = cv2.flip(frame, 1)
    h, w, _ = frame.shape
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    current_time = time.time()
```

The human detection is achieved by utilizing MediaPipe's Pose model, which gives pose landmarks of various key points on the human body. The model detects and processes the landmarks, which are utilized for the calculation of the bounding box coordinates around the detected human. The bounding box is calculated by taking the x and y coordinates of the detected landmarks, converting them to pixel values according to the frame width and height, and then finding the minimum and maximum values. These coordinates are then indicated on the frame by cv2.rectangle(). This graphical outline of the bounding box is utilized to indicate the area where human presence has been detected. The bounding box size, as defined by the height and width of the region detected, is also computed to determine how close or far the human is from the camera. This spatial understanding ability is crucial to the robot's ability to differentiate a human close enough to interact with from one that is too far away to interact with effectively. When the size falls below a threshold (half of a reference size), a message is shown that the human is far and needs to get closer. This is crucial in ensuring effective communication initiation, especially when the user needs to use gestures to communicate with the robot.

```
x_coords = [int(landmark.x * w) for landmark in landmarks]
      y_coords = [int(landmark.y * h) for landmark in landmarks]
      min_x, max_x = min(x_coords), max(x_coords)
      min_y, max_y = min(y_coords), max(y_coords)

      cv2.rectangle(frame, (min_x, min_y), (max_x, max_y), (0, 0, 255), 2)
      bbox_area = (max_x - min_x) * (max_y - min_y)

f bbox_area < 0.5 * reference_area:
         cv2.putText(frame, "Human far (Move closer)", (min_x, min_y - 10),
                  cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
      else:
         cv2.putText(frame, "Human near", (min_x, min_y - 10),
                  cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
```

The gesture recognition and hand detection process is done by the assistance of the MediaPipe Hands model, which detects landmarks for the hand joints. The hand connections that are detected are then marked on the frame by the system, which makes the gesture recognition process more perceptible. The x-coordinate of the wrist is of interest in this part and is monitored in real time to detect changes that signal hand movement. The detection process involves stillness detection and wave detection, and both are necessary for effective interaction. The stillness detection is used to ensure that the hand is not in motion before it attempts to recognize a wave gesture. This is done by comparing the current wrist position with the past wrist position. When the wrist is quite stationary for some time (2 seconds in this case), the system assumes that the user is ready to make a wave gesture. This functionality must be added to prevent false positive detection, which otherwise could happen as a result of random hand movement or unintentional gestures.

```
mp_drawing.draw_landmarks(frame, hand_landmarks, mp_hands.HAND_CONNECTIONS)

         wrist = hand_landmarks.landmark[mp_hands.HandLandmark.WRIST]
         wrist_x = wrist.x

         # Stillness Detection: Check if hand is steady enough
         if not ready_for_wave:
            if previous_wrist_x is not None:
               if abs(wrist_x - previous_wrist_x) < STILL_THRESHOLD:
```

When the system is ready to detect a wave, wrist positions are accumulated repeatedly into a fixed-size deque. The data structure retains only the most recent few frames available for detecting a wave, thus making the system responsive to real-time input. Successive differences between wrist positions are computed by the algorithm to detect direction changes. Once the number of required direction changes (oscillations) and the net movement are larger than a specified threshold, the system detects the gesture as a wave. The wave-detection algorithm is efficient because it relies on simple yet effective position tracking instead of complex hand-shape analysis.

```
hand_x_positions.append(wrist_x)
            if len(hand_x_positions) == hand_x_positions.maxlen:
              diffs = [hand_x_positions[i+1] - hand_x_positions[i] for i in range(len(hand_x_positions)-
1)]
```

```
            direction_changes = sum(1 for i in range(1, len(diffs)) if diffs[i] * diffs[i-1] < 0)

            if (direction_changes >= WAVE_DIRECTION_CHANGES_REQUIRED and
                                        (max(hand_x_positions) - min(hand_x_positions)) >
WAVE_MOVEMENT_THRESHOLD):
                if current_time - last_wave_time > 3:
                    last_wave_time = current_time
                    cv2.putText(frame, "Wave detected! Initiating conversation...", (50, 50),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 3)
```

After successful identification of a wave, the conversation_mode() function is called. This integration of gesture recognition with the conversational AI module is one of the major contributions of the system proposed. It allows the robot to dynamically switch from passive observation to active interaction depending on user gestures. The conversation_mode() function is designed to handle natural language interaction, providing a seamless transition between gesture recognition and intelligent communication.

```
conversation_mode()
            ready_for_wave = False
            still_start_time = None
            hand_x_positions.clear()
```

The combination of spatial perception and gesture recognition creates a strong framework for human-robot interaction. In combination with various cues, the system is very reliable and responds to interactions only when activated in cases of the user desiring to do so. Additionally, the employment of MediaPipe models offers high-efficiency and accurate detection, a factor that renders the system ideally suited for implementation in real-time. The significance of this code snippet to the entire project is its ability to establish a natural interaction model between robot and user. The ability to sense the presence of a human and begin conversation based on a wave gesture is central to the improvement of user experience. Additionally, this implementation is extensible, with additional gestures or voice commands being incorporated into the system whenever the need arises. The algorithmic elegance of the gesture detection system, founded on wrist position tracking and oscillation counting, presents low computational overhead and high interaction reliability. Additionally, the use of OpenCV for real-time image processing, coupled with MediaPipe's robust detection models, demonstrates the effectiveness of this approach in the provision of the desired functionality. Such a modular structure of this code block, with explicit demarcation of human detection, gesture recognition, and conversation initiation, allows for ease of maintainability and extensibility. Future enhancements may involve enhancing the wave detection mechanism with machine learning methods for gesture classification or enhancing the human detection module to operate with varying lighting conditions and occlusions. The overall efficiency of the system is a testament to genuine effort put into algorithmic choices.

The below figure (Fig.5) shows a capture of how the robot detects whether a human is near enough to interact or whether their distance is too far away to engage in a meaningful conversation and interaction in order to assist their user and also to ensure the robot does not mistakenly interpret the user’s intent for gesture and speech when it was not directed towards the robot. This makes the user interaction more seamless, efficient and accurate as the robot will correctly only interact, recognize and engage with users that have intent and mean to engage with the robot and not just passersby’s in public scenarios for example.

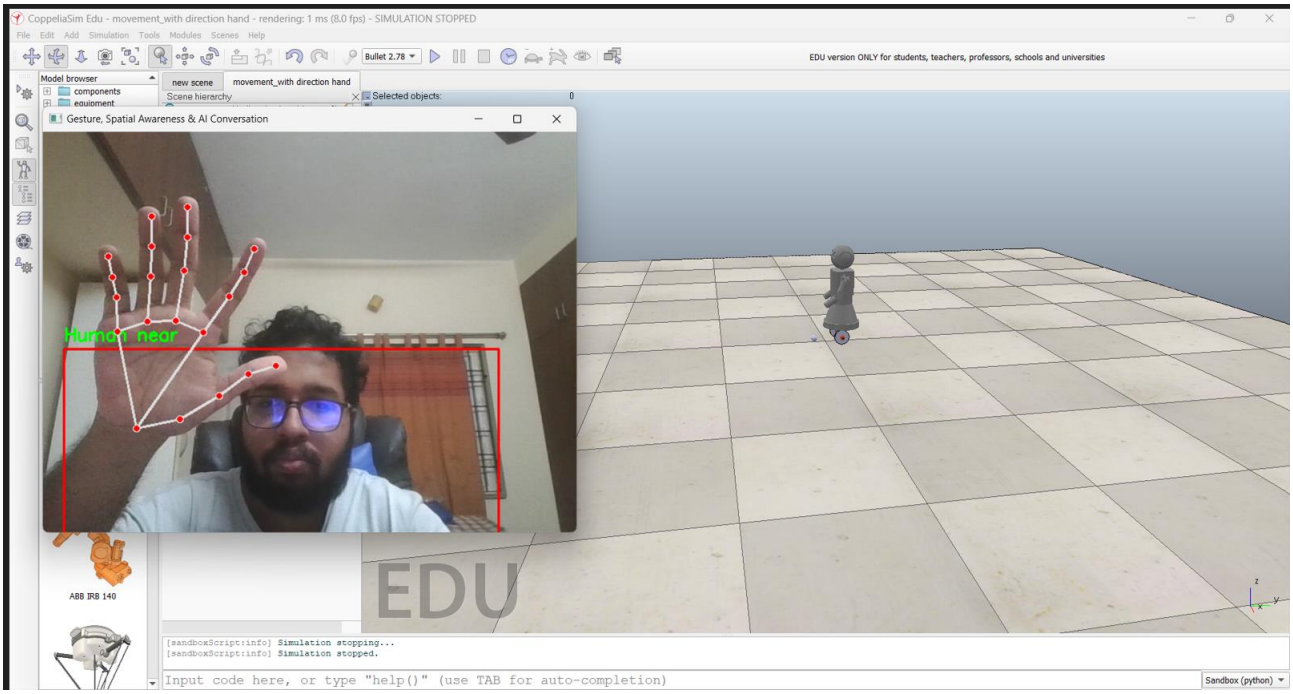


Fig 5. Distance estimation of human and detection of hand joints

3.12 Dynamic Gesture Recognition and Robot Movement

The AI robot gesture-based adaptive human interaction system integrates computer vision techniques and robotics control in CoppeliaSim. The system begins with the importation of required libraries such as OpenCV, NumPy, and the pairwise module from scikit-learn for Euclidean distance calculation and the remote API library from CoppeliaSim. The client connection to CoppeliaSim simulation is established by using the sim.simxStart() function for communication between CoppeliaSim and Python script to facilitate motor control of the robot. The program terminates if there is no connection, with an enhanced error-checking facility. The region of interest (ROI) for hand detection is manually defined by setting top, bottom, left, and right coordinates, which limits the computation area to reduce computational capacity. The background subtraction technique is used to isolate the moving hand from the static environment using a weighted average method implemented by the cv2.accumulateWeighted() function. The technique continuously updates the background model to handle small illumination and environmental variations. The hand segmentation is achieved by calculating the absolute difference between the current frame and the captured background, followed by thresholding the output image to obtain a binary mask of the hand. The cv2.findContours() function is used to extract outer contours of the binary mask. The largest of the contours is identified as the hand segment from the area size. The processed hand is inspected to detect convexity defects by using the cv2.convexHull() function, which constructs a polygon around the outer boundary of the hand. To provide more accuracy, the code identifies key points, such as the top, bottom, left, and right boundary points of the convex hull, which are used to calculate the centroid of the hand.

cv2.accumulateWeighted(frame, background, accumulated_weight)
diff = cv2.absdiff(background.astype("uint8"), *frame*)
_ , thresholded = cv2.threshold(diff, *threshold*, 255, cv2.THRESH_BINARY)

contours, hierarchy = cv2.findContours(thresholded.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

The centroid is calculated by averaging the x-coordinates of the leftmost and rightmost points and the y-coordinates of the topmost and bottommost points. The maximum Euclidean distance from the centroid to the extremities is calculated to establish a radius for a circular region of interest (circular ROI) surrounding the hand. This circular ROI is critical to counting only fingers within a given radial distance from the palm to avoid noise from wrist regions or extraneous gestures. A bitwise AND operation between the thresholded image and the circular ROI is used by the algorithm to isolate the hand region within the given radius. Contour-finding algorithm is looped around this region to identify individual fingers. The number of fingers detected is calculated by looping through the contours and applying two criteria: the position of the contour must be above the wrist line, and its length must be within the given threshold compared to the circular ROI's circumference. This filtering counts only the actual fingers, which increases the gesture detection system's robustness. The fingers detected are mapped to individual commands to make the robot move in CoppeliaSim. For example, one finger detection makes the robot move forward, two fingers make the system turn left, three fingers make the system turn right, and four fingers make the robot move backward.

```
top    = tuple(conv_hull[conv_hull[:, :, 1].argmin()][0])
bottom = tuple(conv_hull[conv_hull[:, :, 1].argmax()][0])
left   = tuple(conv_hull[conv_hull[:, :, 0].argmin()][0])
right  = tuple(conv_hull[conv_hull[:, :, 0].argmax()][0])
```

distance = pairwise.euclidean_distances([(cX, cY)], *Y*=[left, right, top, bottom])[0]

```
# Grab the largest distance
max_distance = distance.max()

# Create a circle with 90% radius of the max euclidean distance
radius = int(0.8 * max_distance)
circumference = (2 * np.pi * radius)
```

If no fingers or an invalid count of fingers is detected, the robot is stopped. The functions for controlling the movement of the robot use the sim.simxSetJointTargetVelocity() function to change the velocities of the left and right motors, thus creating directional control. The program runs constantly capturing frames from the camera input through the usage of OpenCV's cv2.VideoCapture() call. The individual frames are analyzed to recognize gestures, and the identified command is communicated to the CoppeliaSim environment, thereby providing real-time feedback. The program also consists of operations for background reinitialization and proper termination of the simulation. Background reinitialization is implemented upon pressing the 'b' key, and it enables dynamic response to varying environmental lighting conditions or changes in the environment. Simulation termination is provided through the call to sim.simxStopSimulation(), thus the robot actions will stop as soon as the program is terminated. The usage of computer vision-based gesture recognition together with CoppeliaSim's robotic environment is effective in building interactive AI-controlled systems. The convex hull detection, Euclidean distance computation, and threshold segmentation help in providing efficient hand detection and gesture recognition, while communication to CoppeliaSim provides effective robot control.

roi = frame[roi_top:roi_bottom, roi_right:roi_left]

```
    # Apply grayscale and blur to ROI
    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    fingers=0
    # For the first 30 frames we will calculate the average of the background is calculated and its shown on screen.
    if num_frames < 60:
        calc_accum_avg(gray, accumulated_weight)
        if num_frames <= 59:
            cv2.putText(frame_copy, "GETTING BACKGROUND AVG.", (200, 400), cv2.FONT_HERSHEY_TRIPLEX, 1, (0,255,0), 2)
```

The modular structure of the algorithm allows for effortless extension and modification, thus being apt for any of a large set of human-robot interaction systems. Also, the use of established libraries like OpenCV and NumPy allows for efficiency and compatibility with several platforms. Usage of the vision-based methodology of gesture recognition facilitates the effective translation of user gesture to a given robot action, thus increasing the robot's capabilities to react intelligently to the inputs given. The use of visual perception-based techniques is hereby shown to integrate with the robotics control systems towards adaptive and interactive behavior within a simulated world. The project can be taken to the next step with modifications towards incorporating deep learning-based models of gesture recognition in order to offer higher accuracy and robustness to the gesture recognition process. Also, the reliance of the existing system on color-based segmentation can be reduced by employing more advanced techniques like skin-tone detection or motion-based segmentation to improve detection accuracy in dense scenes. Also, the implementation of voice-based interaction using external APIs would supplement the gesture-based control mechanism, making the system more responsive and flexible. As the system is designed to be run in real-time, the optimization of the computational pipeline is still an important aspect. Techniques like multi-threading or GPU-based processing could be employed to improve performance. Also, the implementation of support for additional gestures in addition to the existing set of commands could enable more advanced interactions, such as object manipulation or multi-robot synchronization with the robot. In general, this project provides the foundation for the gesture-based human-robot interaction, which can be developed and extended further to accomplish more advanced tasks and improve overall performance. Fig 6. Shows the threshold image calculation when compared with a normal colored dynamic background that is directly streamed live from a webcam. The algorithm takes the background with multiple colors and objects such as curtains, wall, lamp and uses the threshold calculation to create a dynamic grayscale background with none of the objects or colors that were there previously in those 30 frames where the background average was in the process of being calculated.

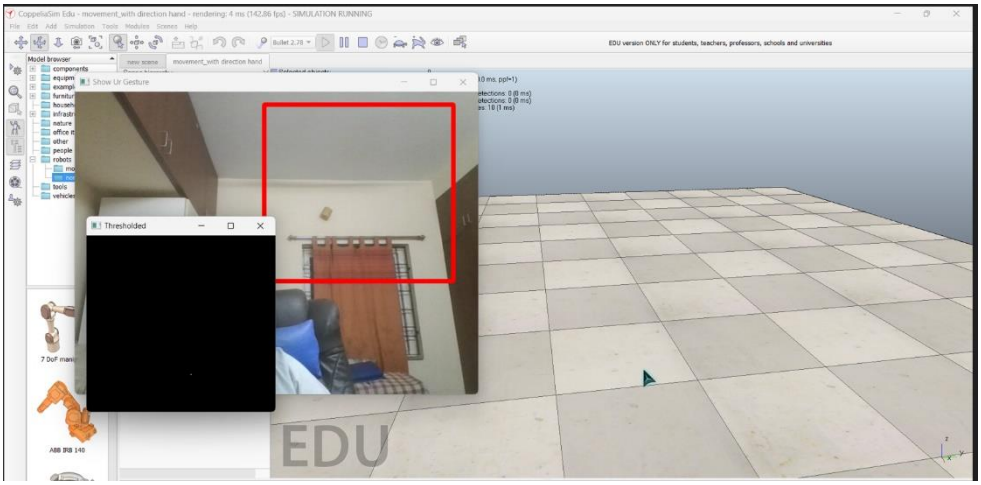


Fig 6. Threshold background calculation

4. RESULTS AND ANALYSIS

The outcome of this research shows a highly responsive and efficient approach for human-robot interaction through vision-based hand gesture recognition and integrated robotic motion control using the CoppeliaSim simulation environment. The system can successfully identify human gestures by separating the hand region from the input frame using Region of Interest (ROI) extraction followed by grayscale conversion and Gaussian blurring to enable noise reduction and feature enhancement. Background averaging is computed using a weighted average approach, which gives a robust and adaptive background subtraction process, ensuring accurate foreground segmentation even in changing lighting conditions. The segmentation process uses absolute difference calculation followed by thresholding to detect the hand successfully from the background, while external contours are detected using OpenCV's contour-finding functions. The capability of the system to detect the hand segment and extract the largest contour successfully ensures reliability in hand region detection, minimizing possible false detections due to noise or irrelevant objects in the frame. Having successfully detected the hand region, the system uses convex hull methods to detect the extreme points of the hand and computes the centroid, which forms the basis for creating a circular Region of Interest (ROI) to detect fingers within a specified radius. The circular ROI approach is highly efficient and effective in detecting fingers, as it does not encompass the wrist and other irrelevant regions, thus improving accuracy. The process also uses Euclidean distance measurement to compute the maximum distance between the centroid and the extreme points, followed by contour filtering to ensure effective counting of valid fingers only. The gesture recognition system is integrated with the robotic control system simulated in CoppeliaSim, allowing for seamless communication between the vision processing algorithm and the robotic motor control functions. The robot is programmed to move forward, left, right, backward, or stop based on the number of detected fingers, enhancing user interactivity and control. The motor handles for the robot's left and right wheels are retrieved using the simxGetObjectHandle function, and movement is controlled using the simxSetJointTargetVelocity function, allowing for smooth and continuous motion adjustments. This approach allows the robot to respond dynamically to user input without significant delays, demonstrating a highly efficient and user-friendly control mechanism. The proposed system is more efficient compared to existing gesture recognition systems in terms of simplicity, reliability, and robustness. Traditional gesture recognition systems are prone to employ complex neural networks or require extreme amounts of training data to provide acceptable accuracy. Our system, on the other hand, performs exceptionally well using classical computer vision techniques such as convex hull analysis and contour filtering, which are computationally light and require minimal processing resources. Moreover, the direct integration of the robot's motion control functions with the gesture recognition system removes the need for additional middleware or communication protocols, reducing latency and overall responsiveness.

The system also demonstrates a highly efficient simulation framework using CoppeliaSim, providing a real-world environment to test and validate the robot's functionalities. The simulation environment closely resembles real-world conditions, allowing for extensive testing of the robot's motion capabilities and gesture recognition accuracy. By using the stop, forward, left, right, and backward operations based on user gestures, the system demonstrates a robust and efficient way of achieving human-robot interaction. Compared to other existing systems that may be pre-trained or require high-end GPUs to support real-time operation, our implementation is capable of delivering similar or even better performance using computationally efficient algorithms that do not demand large computational resources. The use of efficient techniques such as background averaging, contour analysis, convex hull computation, and Euclidean distance measurement offers a real-world and efficient way of gesture recognition and robot control. Additionally, the use of dynamic region-of-interest extraction offers a great improvement in accuracy through the elimination of irrelevant noise and focus on the target region alone. The implementation has a high level of resistance to background

noise and changing lighting, a key feature in achieving robust gesture recognition. Moreover, the use of simulation allows rapid prototyping and experimenting with different control schemes without needing real hardware, hence achieving huge savings in time and cost involved in development. The modularity of the system also offers simple extension and integration of advanced functionalities, such as voice recognition, face recognition, or more complex gesture-based inputs. Overall, the result of our implementation offers a full-fledged platform for enhancing and expanding gesture-based AI robotic systems. Through the utilization of efficient image processing methods and easy integration into robotic control subsystems, our system offers a viable and easy-to-use solution for human-robotic interaction in real time. The performance of our implementation proves the advantages of utilizing traditional computer vision algorithms within the context of developing adaptive and robust robotic systems irrespective of complex or compute-intensive algorithms. Fig 7. And Fig 8. Showcase the result of the simulation where the user can control the robot's movements and directions through making a gesture of them holding a certain number of fingers , each corresponding to a certain movement in a direction or a certain turning engaging action, whilst showing nothing indicates the movement motors to halt enabling quick response time and easy to navigate and understand dynamic gestures as well as the other features which integrate AI, distance estimation and an interactive interface at the top level system.

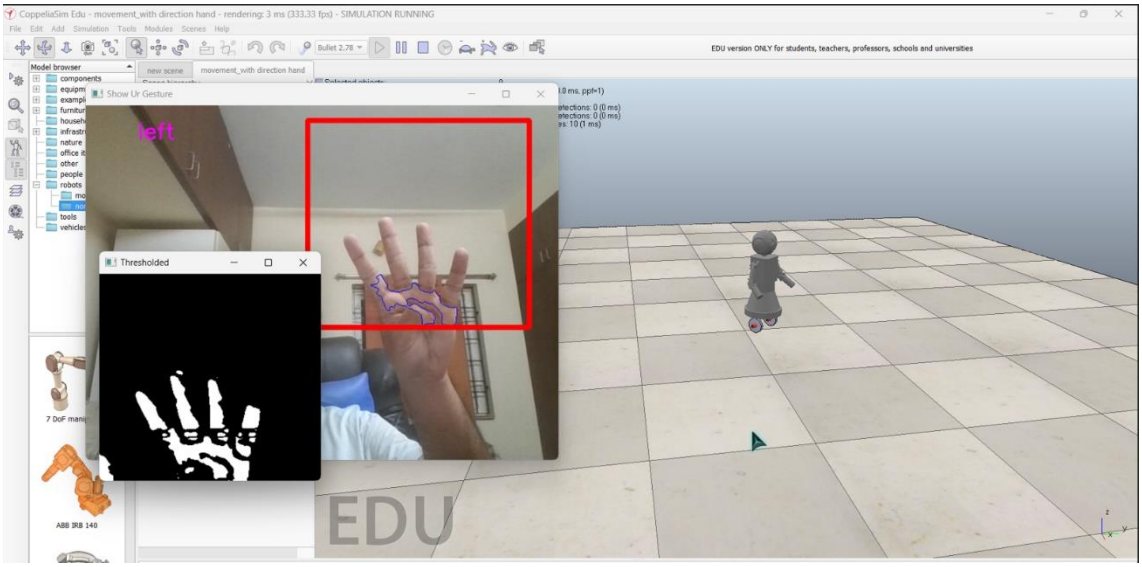


Fig 7. Gesture controlled movement

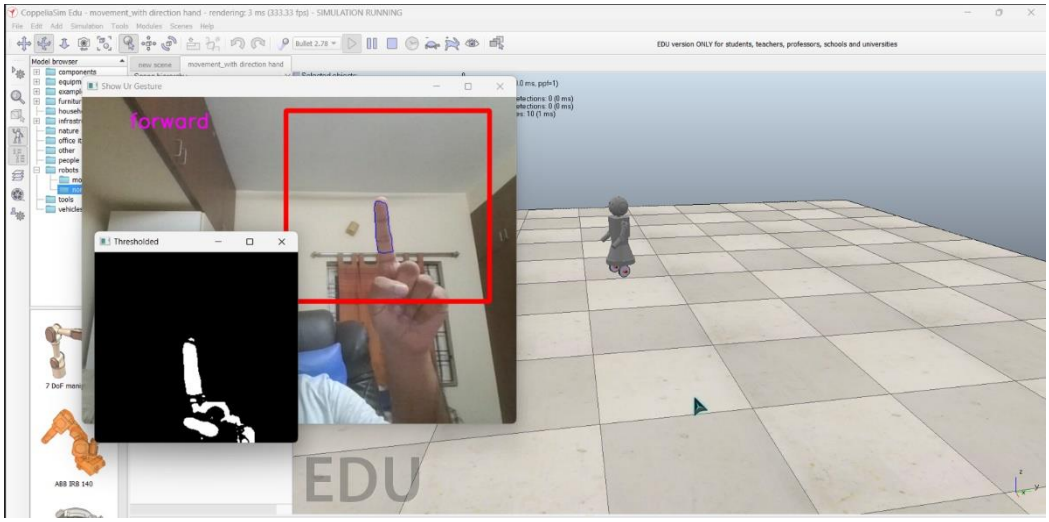


Fig 8. Gesture controlled movement (2)

5. FUTURE SCOPE

The future promise of this AI-driven robot system lies in scaling up its scale from simulation to having it implemented on actual robotic hardware. Through the utilization of its implementation on robotic hardware, its robustness against dynamic environments where sensor noise, environmental variations, and variability of human movement may influence its operation can be tested. Real-world testing would offer iterative hardware compatibility tuning, enabling seamless gesture interpretation and motion playback across different robotic platforms. Increasing accuracy and flexibility in gesture recognition is also a direction where the system can be further improved. The system is presently pre-defined gesture-based, but future development can access machine learning algorithms like deep neural networks to allow for self-learning and adaptive gesture interpretation. Such models would strengthen the robot's gesture recognition ability across different users, backgrounds, and lighting conditions and hence further extend the usage and applicability of the system to more users. Furthermore, the access to multiple input modes like speech, facial features, and eye gaze tracking could construct a multimodal interaction framework which would enable users to communicate with robots effortlessly. Scaling up the system is another significant area for future work. The model is presently designed for single-robot control, but future implementations can support multi-robot coordination, which will enable collaborative robotics applications in manufacturing, healthcare, and automation of services. This will enable coordination of multiple robots harmoniously under human supervision for increased efficiency in tasks like warehouse management, automated delivery, and industrial assembly. Increased use of cloud-based computing and edge computing will further facilitate real-time decision-making with improved response times with reduced computational loads per robot. Use of adaptive learning methods, i.e., reinforcement learning, will also enable robots to learn response based on user preference and past interaction. This would make the system more friendly as the robots can improve behavior patterns and interaction automatically over time. The chatbot feature can also be further improved through the use of more sophisticated higher-level natural language processing (NLP) models that enable context awareness, multi-turn conversation, and emotion analysis.

The enhancement would allow robots to have more productive and meaningful conversations and thus be suitable to be employed in applications related to customer service, learning, and healthcare. Additionally, the use of augmented reality (AR) and haptic feedback can also enhance the usability further. AR-based interfaces can provide immediate visual feedback on the gestures, and the users can observe how the robot reacts to the gestures. The users can use haptic devices such as wearable wristbands or gloves to provide tactile cues such that the robot movements can be controlled with more accuracy. These technologies will allow humans to interact with robots in an easy, immersive, and effective way. The usability of the system in real-world applications can be extended to other domains such as assistive robots for disabled individuals, home automation, and search and rescue robots in emergency situations. In the medical field, for instance, gesture control can assist mobility disorder patients by allowing them to control robotic assistants with a hand gesture. Similarly, in dangerous environments such as disaster relief operations, gesture-controlled robots can be employed in search and rescue operations where human intervention is dangerous. In the future, the system can be integrated with smart environments and Internet of Things (IoT), allowing robots to interact with other smart devices. For instance, a robot can detect the entry of a human into a room and control lighting, air conditioning, or pick up an object based on voice and hand gestures command.Such integrations would allow highly responsive and context-aware robotic assistants to be integrated well into human environments.Finally, advancements in energy efficiency, sensor miniaturization, and wireless communication will also allow AI-based robots to be employed in daily life. Future iterations of this system will also take advantage of 5G networks for cloud-based real-time computation, where robots can be remotely operated and remotely controlled over long distances. The future of adaptive human interaction with gesture-based AI robots is full of innovation opportunities in a broad spectrum of applications. As the technology advances, the merging of real-world applications with AI-powered

robotics will expand, enabling robots to perform increasingly complex tasks with greater accuracy and flexibility. Perhaps the most thrilling direction of future progress is the transition from simulation to real-world deployment. While the deployments of today, such as those tested in CoppeliaSim, demonstrate the theoretical viability of gesture-controlled robotic control in simulated worlds, the next step is to implement these models on physical robotic systems with real-time sensors, actuators, and autonomous decision-making capabilities. This transition will allow robots to effectively operate in dynamic spaces such as healthcare, industrial automation, and smart home systems, with transparent interaction and assistance for human users.

Another key region of future research is more advanced gesture recognition through deep learning. Existing gesture-based AI systems are based on pre-defined gesture sets, but future developments can leverage Transformer-based architectures, self-learning AI models, and few-shot learning methods to recognize more complex, adaptive, and user-specific gestures. Using continuous learning mechanisms, robots can enhance their capacity to understand fine variations in human gestures, making interactions more intuitive and personalized. Moreover, sensor fusion methods, integrating depth cameras, LiDAR, IMUs (Inertial Measurement Units), and bioelectric sensors, can greatly improve the accuracy and reliability of gesture recognition in different environmental conditions.Expanding multimodal human-robot interaction is another key area of future development. Along with hand gesture detection, robots can be endowed with emotion recognition, facial expression analysis, and voice sentiment analysis, enabling more adaptive and emotionally intelligent interaction. By interpreting vocal tone, facial expressions, and body language, AI-powered robots can respond more suitably to human emotions, making them more effective in elderly care, mental health assistance, and customer service.

The evolution of context-aware AI systems will allow robots to interpret environmental signals and respond accordingly, making interactions more human-like and dynamic.Also, multi-robot collaboration is a promising research direction that can greatly improve efficiency in industrial, medical, and logistical tasks. In future systems, multiple AI-powered robots will be able to communicate, coordinate, and collaborate autonomously to accomplish common goals. Swarm robotics, where swarms of robots work together to accomplish tasks like warehouse automation, robotic surgical assistance, and disaster response activities, can revolutionize industries by enabling efficiency, scalability, and precision. The application of blockchain technology to multi-robot coordination can further allow secure, decentralized communication and decision-making, further increasing the robustness and reliability of cooperative robotic networks.

Another future avenue with potential is the integration of augmented reality and virtual reality interfaces with gesture-operated robots. AR-augmented gesture operation will allow users to visualize and optimize robotic movement in real-time, further enriching robotic interactions to be more interactive and immersive. These technologies will be best utilized in surgical robotics, remote maintenance, military applications, and interactive education, where accuracy, visualization, and intuitive control are crucial. Further, haptic feedback systems can be incorporated, allowing users to feel tactile feedback, further enhancing the user experience and accuracy of robotic control.
Another fundamental advance in AI-controlled robotics will be motion and navigation optimization with reinforcement learning and real-time decision-making algorithms. Future robots will have the capability of self-optimizing movement and navigation paths with regards to changes in the environment, making them more effective in search-and-rescue missions, autonomous delivery operations, and hospital medical assistance. AI-controlled path planning algorithms will allow robots to move efficiently through crowded or dynamically changing environments, further making them useful in the real world.

Further, integration with cloud computing and edge computing will be an essential driver in AI-controlled robotic system development. Using cloud-based AI models, robots can offload complex

computations remotely, eliminating the need for expensive onboard computing hardware while giving real-time responses. Edge computing solutions will facilitate faster decision-making through the local processing of key data, optimizing latency and system efficiency. This integration will allow robots to be accessed by real-time updates, exchange data within networks, and function efficiently within IoT-enabled smart environments. Another important consideration for future AI-based robotics is ethical AI and privacy protection. As robots find greater use in human environments, it will become imperative to provide data protection, ensure privacy, and implement ethical AI decision-making. Developers and researchers will need to implement clear AI models, bias prevention measures, and robust data handling processes to facilitate secure and responsible robotic interaction, especially in sensitive sectors like healthcare, law enforcement, and personal care. The growth of gesture-based robotics in novel applications will also generate innovations in next-generation robotic systems. For example, gesture-controlled robots in space exploration can aid in operation in zero-gravity conditions. In agriculture, robotic systems based on AI can advance crop monitoring, automated harvesting, and pest management using next-generation gesture and voice control systems. In infrastructure repair and construction, gesture-controlled robots can increase precision, safety, and automation in high-risk settings with little human intervention in unsafe conditions. More advancements in AI-based humanoid robots can develop robots that learn and replicate human actions, gestures, and patterns of communication, which will make robots more natural to socialize with, conduct therapy, and serve as companions. AI-based robots for disability support can provide improved autonomy through smart gesture-based and voice-based interfaces. Development of personalized robotic assistants that learn and adapt to specific users over time will make AI-based interaction more meaningful and user-oriented. As AI-based robotics evolve, the integration of quantum computing, neuromorphic processors, and next-generation AI models will further accelerate computational performance. Quantum computing will be capable of significantly accelerating AI model training, allowing robots to process massive data in real-time. Neuromorphic processors inspired by the neural structure of the human brain will be able to deliver energy-efficient and smart decision-making, leading to more autonomous, adaptive, and responsive robot systems. Last but not least, the continuous development of human-in-the-loop AI systems will witness robots learning from human preferences and feedback, enabling a cycle of ongoing improvement wherein AI models improve in responses and behavior over a period of time. This will become inevitable in education, healthcare, and collaborative robotics, with AI-powered robotic systems highly adaptive, ethical, and human-value-driven. In brief, the future of AI robotics empowered by gestures promises much in real-world applications, multimodal human-robot interaction, multi-robot coordination, integration with cloud and edge computing, and ethical challenges to AI. With advances in technology, the combination of gesture recognition, AI-based decision-making, and real-time adaptability will enable a next generation of intelligent, intuitive, and highly advanced robot systems that transform industries and improve human lives beyond imagination.

6. CONCLUSION

The AI-driven multimodal robot control system developed in this thesis fully integrates gesture recognition, speech interaction, and robot control in real-time under simulation. Utilization of computer vision technology for gesture recognition and natural language processing for conversational interaction, the system provides natural and flexible human-robot interaction. Gesture understanding activates by making use of MediaPipe Pose and Hands modules to ensure proper analysis of user intent, e.g., direction gestures and actions based on numbers of fingers employed to control movement. Additionally, inclusion of wave gesture understanding and proximity improves user interaction by enabling robots to recognize humans and enable active interaction. Enabling human-robotic interaction through activation of an AI chatbot further enables interactive response, providing multimodal interaction through which human intent is translated to robotic action.System testing using CoppeliaSim has shown the system to deliver accurate and optimal motion planning. Utilization of inverse kinematics and path optimization has included streamlined robotic performance and accurate gesture-to-response mapping. Additionally, the modular and vendor-independent architecture of CoppeliaSim indicates scalability as well as applicability in a given robotic system as well as industries.Although the success of the system is realized, there are still some limitations such as the lack of real-world implementation on hardware and limited gesture commands that can be employed. Future work can be conducted to enhance the functionality of the system by utilizing real-world sensors, improving gesture classification algorithms, and improving motion planning for more complex robot tasks. Human-robot collaboration can further be improved by increasing the conversational AI model to enable context-aware responses.Summing up, research advances intelligent and adaptive human-robotic interaction to empower more user-centric and accessible control systems for robots.

Along with enhancing robotic intelligence, enhancements to natural language processing and AI-based chatbot algorithms may enable more advanced, context-sensitive dialogues, allowing robots to interpret abstract commands, long-term memory storage of dialogues, and learning to adjust responses through prior interactions. The inclusion of augmented reality output interfaces may provide immediate visual recognition of gesture inputs, improving ease of use and precision, and haptic input devices like wearable gloves or wristbands may provide tactile notifications, further enhancing precision in robot operation. The uses of this technology in the real world are numerous, extending far beyond the robot control itself into assistive care robotics for the disabled, smart home appliances, industrial robot control, and search-and-rescue missions in hostile environments. For instance, in healthcare, gesture-controlled robots may help physically disabled patients to carry out necessary actions with simple hand gestures. In automated manufacturing, the gesture-controlled control of robots can simplify warehouse storage, assembly operations, and robotics inspection, eliminating the need to depend on software programming interfaces. Integration with Internet of Things services would also mean that robots would be able to communicate seamlessly with other smart devices, creating a smart, intelligent environment where equipment adjusts continuously based on humans' presence and activity. The utilization of 5G communication networks for distributed real-time computations in the cloud could allow next-generation versions of the system to enable remote access to robots for operation, potentially allowing human operatives to remotely control robots, revolutionizing telepresence robotics and remote support in hostile industries.

While the system has achieved high performance in simulation, closing the gap between virtual realization and real-world deployment is a key challenge to be overcome. Future effort will include enhancing hardware compatibility, enhancing robustness to environmental variability, and optimizing decision models using AI to allow the robot to operate reliably in uncertain real-world environments. System scalability also presents opportunities for innovation, including collaborative robotics, educational robotics, and human-assisted AI systems. With continued advances in robotic autonomy, the addition of self-learning algorithms, reinforcement learning-based behavior learning, and AI-based predictive modeling will enable the robot to anticipate user needs, learn to optimize responses over time, and deliver an increasingly personalized interactive experience. Overall, this work is an important milestone in the creation of intelligent, adaptive robotic assistants, with a flexible, human-oriented control system that can be reconfigured for a broad range of industrial, commercial, and assistive applications. With continued advances in gesture recognition accuracy, the addition of interaction modalities, and enhancement of real-time responsiveness, this AI-based robotic platform will enable the creation of the next generation of autonomous robotic systems, making them smarter, more intuitive, and unobtrusively integrated into everyday human activity.

The performance of the proposed system is assessed based on its performance under varying conditions to achieve robustness and reliability, with various parameters monitored to identify its effectiveness, including processing time, accuracy in gesture detection, robustness in speech recognition, and response delay to commands. Each module is separately tested before integration to guarantee responsiveness and compatibility. The performance of the gesture recognition module is guaranteed by exposing a series of pre-defined gestures under varying conditions, with accuracy in hand segmentation by comparing the detected contours against ground truth labels. Convex hull computation and finger counting are also guaranteed using hands of varying sizes, orientations, and distances from the camera, while the success rate of wave detection is determined by calculating the ratio of true positives against the number of gestures attempted. For speech recognition, the system is tested using audio samples of common commands, conversational queries, and complex sentences, with robustness determined under varying levels of noise, distance from the microphone, and speech rates. The accuracy of this module is measured as the ratio of words transcribed correctly against the number of words in the audio sample, with latency measured by recording the processing time for audio input and generation of a response. The response time of the system is measured by recording the time from the detection of gestures to the activation of the corresponding robotic action, including determining the time for gesture recognition, transmission of the signal to CoppeliaSim, and motor command execution. Performance metrics like average response time, standard deviation, and worst-case latency are measured, and the system is stress-tested by boosting the frame rate to identify real-time performance. Integration testing is conducted by executing interaction sessions wherein gestures and speech commands are used interchangeably to control the robot, exercising the system to respond to asynchronous commands and alternate between conversational and movement control modes without any interruptions. Robustness is also exercised through error handling mechanisms employed at various stages of the pipeline, such as providing feedback on speech recognition errors, validating gestures over multiple frames to avoid false positives, and retrying network communication with CoppeliaSim until it is successful. This rigorous testing ensures the system to properly prioritize commands based on context and user intent with high reliability and responsiveness in operation.

7. REFERENCES

[1] Fujie, S., Ejiri, Y., Nakajima, K., Matsusaka, Y., & Kobayashi, T. (2004). A conversation robot using head gesture recognition as para-linguistic information. Proceedings of the 2004 IEEE International Workshop on Robot and Human Interactive Communication (ROMAN 2004), Kurashiki, Japan, September 20-22.

[2] Ajili, I., Mallem, M., & Didier, J.-Y. Gesture recognition for humanoid robot teleoperation. 2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), Lisbon, Portugal, 2017.

[3] Lee, S.-W. (2006). Automatic gesture recognition for intelligent human-robot interaction. Proceedings of the 7th International Conference on Automatic Face and Gesture Recognition (FGR'06)

[4] Brethes, L., Menezes, P., Lerasle, E., & Hayet, J. (2004). Face tracking and hand gesture recognition for human-robot interaction. Proceedings of the 2004 IEEE International Conference on Robotics and Automation (ICRA '04)

[5] Burger, B., Ferrané, I., Lerasle, F., & Infantes, G. (2011). Two-handed gesture recognition and fusion with speech to command a robot. Autonomous Robots, 32(1), 129-147.

[6] Kawatsu, C., Koss, F. V., Zhao, A., & Crossman, J. (2017). Gesture recognition for robotic control using deep learning. Conference Paper, August 2017.

[7] Buddhikot, A. G., Kulkarni, N. M., & Shaligram, A. D. (2018). Hand gesture interface based on skin detection technique for automotive infotainment system. International Journal of Image, Graphics and Signal Processing, 10(2), 10-24.

[8] Luo, H., Du, J., Yang, P., Shi, Y., Liu, Z., Yang, D., Zheng, L., Chen, X., & Wang, Z. L. (2023). Human-machine interaction via dual modes of voice and gesture enabled by triboelectric nanogenerator and machine learning. ACS Applied Materials & Interfaces, 15(13), 17009-17018.

[9] Peña-Cáceres, O., Silva-Marchan, H., Albert, M., & Gil, M. (2023). Recognition of human actions through speech or voice using machine learning techniques. Computers, Materials & Continua, 77(2), 1874-1895.

[10] Suarez, J., & Murphy, R. R. (2012). Hand gesture recognition with depth images: A review. 2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication, Paris, France.

APPENDIX

Programs: Main.py

```
import cv2

import mediapipe as mp

import time

from collections import deque

import pyttsx3

import speech_recognition as sr

from openai import OpenAI

import subprocess

import threading

import keyboard

import sys

import os


# ---------- Constants (Easier wave detection) ----------

STILL_THRESHOLD = 0.03        # Maximum normalized movement considered as "still"

STILL_DURATION = 2            # Hand must be still for 2 seconds before wave detection

WAVE_DEQUE_LEN = 8            # Number of frames for wave detection accumulation

WAVE_DIRECTION_CHANGES_REQUIRED = 2  # Oscillations needed to register a wave

WAVE_MOVEMENT_THRESHOLD = 0.15 # Minimum horizontal movement difference
```

```
# ---------- API Key & Endpoint ----------

API_KEY = "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"  # Replace with your OpenRouter API key

BASE_URL = "https://openrouter.ai/api/v1"



# ---------- Initialize TTS Engine ----------

engine = pyttsx3.init()



# Global flag to indicate that movement protocol has been initiated

movement_initiated = False



# ---------- Initialize MediaPipe ----------

mp_drawing = mp.solutions.drawing_utils

mp_pose = mp.solutions.pose

mp_hands = mp.solutions.hands



# Pose and hand detectors with defined confidence levels

pose = mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5)

hands = mp_hands.Hands(min_detection_confidence=0.5, min_tracking_confidence=0.5)



# ---------- Open Webcam ----------

cap = cv2.VideoCapture(0)
```

```
# ---------- Variables for Wave & Stillness Detection ----------
hand_x_positions = deque(maxlen=WAVE_DEQUE_LEN)
last_wave_time = 0
still_start_time = None
ready_for_wave = False
previous_wrist_x = None

# For spatial awareness calibration (bounding box area)
reference_area = None

# ---------- Function to Get Chatbot Response using Dolphin API ----------
def get_chatbot_response(query):
    try:
        client = OpenAI(
            base_url=BASE_URL,
            api_key=API_KEY,
        )
        completion = client.chat.completions.create(
            extra_headers={
                "HTTP-Referer": "http://localhost",   # Replace if desired
                "X-Title": "MyCollegeProject"          # Replace if desired
            },
            extra_body={},
```

```
            model="cognitivecomputations/dolphin3.0-mistral-24b:free",
            messages=[
                {"role": "user", "content": query}
            ]
        )
        answer = completion.choices[0].message.content.strip()
        print(answer)
        return answer
    except Exception as e:
        print("Error in chatbot response:", e)
        return "I'm sorry, I could not get a response at the moment."


def speak_text(text):
    engine.say(text)
    engine.runAndWait()


# ---------- Conversation Mode Function ----------
def conversation_mode():
    """
    Uses TTS to greet the user, listens for a question via STT,
    queries the DeepSeek chatbot API, and then speaks the answer.
    If the user says "initialize movement protocol" or presses "c" during speech,
    the TTS is cancelled, camera and windows are cleaned up, and the movement protocol is launched.
```

```
"""
global cap, movement_initiated

greeting = "Hello, how may I help you?"
speak_text(greeting)

recognizer = sr.Recognizer()
with sr.Microphone() as source:
    print("Listening for your question...")
    try:
        audio = recognizer.listen(source, phrase_time_limit=5)
        user_query = recognizer.recognize_google(audio)
        print("User query:", user_query)
    except Exception as e:
        print("Speech recognition error:", e)
        speak_text("I did not catch that. Please try again later.")
        return

# Direct command branch: if the user says "initialise movement"
if user_query.strip().lower() == "initialise movement":
    cap.release()
    cv2.destroyAllWindows()
    subprocess.Popen([sys.executable, "hand detection.py"])
```

```
    keyboard.unhook_all()
    os._exit(0)
else:
    answer = get_chatbot_response(user_query)
    print("Chatbot answer:", answer)


cancel = False
tts_thread = threading.Thread(target=speak_text, args=(answer,))
tts_thread.daemon = True
tts_thread.start()


# Poll every 10ms to check if 'c' is pressed
while tts_thread.is_alive():
    if keyboard.is_pressed('c'):
        engine.stop()  # Stop the global TTS engine immediately
        cancel = True
        break
    time.sleep(0.01)


if cancel:
    keyboard.unhook_all()
    cap.release()
    cv2.destroyAllWindows()
```

```
        subprocess.Popen([sys.executable, "check.py"])
        os._exit(0)
    else:
        tts_thread.join()

# ---------- Main Loop ----------
while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Flip frame for mirror view and get dimensions
    frame = cv2.flip(frame, 1)
    h, w, _ = frame.shape
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    current_time = time.time()

    # Process frame for pose (human detection) and hand landmarks (gesture recognition)
    pose_results = pose.process(frame_rgb)
    hand_results = hands.process(frame_rgb)

    # --- Human Detection & Spatial Awareness ---
    if pose_results.pose_landmarks:
```

```
        landmarks = pose_results.pose_landmarks.landmark
        x_coords = [int(landmark.x * w) for landmark in landmarks]
        y_coords = [int(landmark.y * h) for landmark in landmarks]
        min_x, max_x = min(x_coords), max(x_coords)
        min_y, max_y = min(y_coords), max(y_coords)

        cv2.rectangle(frame, (min_x, min_y), (max_x, max_y), (0, 0, 255), 2)
        bbox_area = (max_x - min_x) * (max_y - min_y)

        if reference_area is None:
            reference_area = bbox_area

        if bbox_area < 0.5 * reference_area:
            cv2.putText(frame, "Human far (Move closer)", (min_x, min_y - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        else:
            cv2.putText(frame, "Human near", (min_x, min_y - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

    # --- Hand Detection, Stillness & Wave Recognition ---
    if hand_results.multi_hand_landmarks:
        for hand_landmarks in hand_results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(frame, hand_landmarks,
mp_hands.HAND_CONNECTIONS)
```

```
wrist = hand_landmarks.landmark[mp_hands.HandLandmark.WRIST]
wrist_x = wrist.x

# Stillness Detection: Check if hand is steady enough
if not ready_for_wave:
    if previous_wrist_x is not None:
        if abs(wrist_x - previous_wrist_x) < STILL_THRESHOLD:
            if still_start_time is None:
                still_start_time = current_time
            elif current_time - still_start_time >= STILL_DURATION:
                ready_for_wave = True
                hand_x_positions.clear()
        else:
            still_start_time = None
            ready_for_wave = False
    else:
        still_start_time = current_time

previous_wrist_x = wrist_x

# Wave Detection: When ready, accumulate wrist positions and analyze
if ready_for_wave:
```

```
            hand_x_positions.append(wrist_x)

            if len(hand_x_positions) == hand_x_positions.maxlen:

                diffs = [hand_x_positions[i+1] - hand_x_positions[i] for i in
range(len(hand_x_positions)-1)]

                direction_changes = sum(1 for i in range(1, len(diffs)) if diffs[i] * diffs[i-1] < 0)



                if (direction_changes >= WAVE_DIRECTION_CHANGES_REQUIRED and

                    (max(hand_x_positions) - min(hand_x_positions)) >
WAVE_MOVEMENT_THRESHOLD):

                    if current_time - last_wave_time > 3:

                        last_wave_time = current_time

                        cv2.putText(frame, "Wave detected! Initiating conversation...", (50, 50),

                                    cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 3)

                        conversation_mode()

                    ready_for_wave = False

                    still_start_time = None

                    hand_x_positions.clear()



    cv2.imshow('Gesture, Spatial Awareness & AI Conversation', frame)

    if cv2.waitKey(1) & 0xFF == 27:

        break



    # If movement has been initiated, exit the main loop.

    if movement_initiated:
```

```
        break


cap.release()

cv2.destroyAllWindows()
```