THE HASHEMITE UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

GRADUATION PROJECT-(GP II)

Building an Interactive and Intelligent system Using Darwin OP2 Humanoid Robot

STUDENTS

Omar Maen Barham	1935515
Mohammad Khader Abu-Diab	1931678
Ahmad Issa Al-Najjar	1932534

Graduation Project Advisors

Dr. Khalil Yousef / Dr. Bassam Jamil

DEPARTMENT OF COMPUTER ENGINEERING

Academic Year

2022-2023

ACKNOWLEDGMENT

We begin by expressing our appreciation and gratitude to Allah for providing us with the strength and perseverance to complete this project.

We would also like to extend our heartfelt thanks to our supervisors, Dr. Khalil Yousef, and Dr. Bassam Jamil, for their invaluable guidance, motivation, and support throughout the duration of this project.

Finally, we are deeply grateful to our parents, friends, and well-wishers for their encouragement and support.

ABSTRACT

In recent years, Human-Robot Interaction (HRI) has found diverse applications across various fields such as medical treatment, education, and entertainment.

DARwin OP2 is a compact humanoid robot that offers an open platform for development. It features human-like joint actuators, a USB camera, multiple microphones, a speaker, and various I/O ports.

In this project, we utilized the Darwin OP2 robot to create a personalized assistant system designed to accompany students and individuals who spend extended periods of time performing office work duties while sitting on office chairs and desks. Our system comprises several components, including facial recognition, speech recognition, inter-device communication, text-to-speech conversion, bad posture detection, medical advice using reinforcement learning, web scraping, and entertainment. The majority of the data processing takes place on a separate computer. These implementations hold great potential for inspiring future HRI applications.

We conducted a survey to evaluate the system's performance. The survey results showed that the system has the potential to be highly beneficial for the intended audience.

Keywords: Robotis DARwin OP2, Personal Assistant, Human-Robot Interaction (HRI), Reinforcement Learning (RL), Speech Recognition, Robot Operating System (ROS), Computer Vision.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Number *Page*

EXECUTIVE SUMMARY

The objective of our project was to implement Human-Robot Interaction (HRI) using the DARwin OP2 robot. This involved incorporating features such as facial recognition, speech recognition, text-to-speech conversion, reinforcement learning-based medical advice, web scraping, and entertainment capabilities.

The project was divided into two distinct phases, each lasting for 12 weeks. During the first phase, we conducted literature research to gather relevant information. We learned how to interface with the robot, explored the Robot Operating System (ROS)[1], and gained expertise in data communication between devices and nodes. One of the challenges we faced was creating custom actions to enhance the robot's functionalities.

In the second phase, we delved into the field of Reinforcement Learning. We visited a medical rehabilitation facility to gather insights and data specifically related to medical advice. The software design was implemented in Python, leveraging various open-source libraries such as Speech recognition[2], gTTS[3], Gym[4], mediapipe[5], and OpenCV[6]. Inter-device communication was facilitated using ROS. Additionally, we utilized preinstalled software on the robot, including the action editor tool, to create customized actions.

As a result, we successfully developed an interactive robot personal assistant designed to accompany students and individuals during their prolonged desk work. Our system includes a wide range of functions, providing assistance and support in various tasks while also offering medical advice tailored to the user's needs.

Keywords: Robotis DARwin OP2, Personal Assistant, Human-Robot Interaction (HRI), Reinforcement Learning (RL), Speech Recognition, Robot Operating System (ROS), Computer Vision.

CHAPTER 1: INTRODUCTION

1.1 Problem Statement and Purpose

As robots become more prevalent in various fields, including teaching assistants, entertainment actors, delivery robots, home cleaning, and industrial robots, the limitations of their application have become more apparent. Most robots are designed to behave and feel like machines, rather than humans, which limits their potential uses.

By implementing Human-Robot Interaction (HRI), we can expand the range of applications for these robots. Our project focused on utilizing the DARwin OP2 robot to create an immersive and interactive personal assistant system that relies on audio and visual data.

By leveraging the robot's onboard USB camera and its internal microphone, we were able to capture real-world information, enabling the system to perform a wide range of actions based on the processed data. The implemented functionalities contained essential features such as face recognition, speech recognition, text-to-speech conversion, reinforcement learning-based medical advice, web scraping, and entertainment capabilities. These capabilities enabled the personal assistant system to provide a holistic user experience.

Reinforcement Learning (RL) was used in training an agent to give suitable advice on certain types of pain the user would experience. This is further explained in section (2.2).

Web scraping was used to make Darwin even more useful, with this, he can find answers to users' questions from the internet.

For more detailed information about the DARwin OP2 robot, please refer to Appendix A.

1.2 Project and Design Objectives

The primary goal of this project is to enable Human-Robot Interaction by employing the Darwin OP2 as a personal assistant robot, primarily targeting students and individuals who spend extended periods at their desks. The robot is designed to offer a wide range of services to enhance productivity and convenience.

The services provided by the robot include web searching, voice-note taking, playback, as well as providing time and date information. In addition to these features, if users experience any discomfort or pain in their back, shoulders, arms, legs, neck, or head, Darwin can provide medical advice tailored to their specific needs. Furthermore, Darwin is capable of offering topic summaries from Wikipedia, injecting humor with jokes to uplift one's mood, and delivering up-to-date weather information.

In addition to the mentioned services, the robot will also monitor the user's posture. If the user maintains a bad posture for an extended period, the robot will provide a timely warning and offer advice to encourage the user to improve their posture. This feature aims to promote ergonomics and help users maintain a healthier sitting position while working or studying.

1.3 Intended Outcomes and Deliverables

The project aims to build an Interactive and Intelligent system Using Darwin OP2 Humanoid Robot to:

- Enhance Human-Robot Interaction by employing the Darwin OP2 as a personal assistant robot, primarily targeting students and individuals working at their desks for extended periods.

- Offer a wide range of services, including accurate time and date information, web searching capabilities, voice-note taking, and playback.

- Provide personalized medical advice to address discomfort or pain in various body areas.

- Monitor the user's posture and, if poor posture is detected, issues a timely warning, and provides advice for improvement.

- Provide other features including topic summaries from Wikipedia, injecting humor through jokes, and delivering up-to-date weather information.

These comprehensive services aim to optimize productivity, convenience, and promote healthier work or study sessions.

1.4 Summary of the Used Design Process

During the project, we gained familiarity with the action editor through the Robotis OP e-manual. This involved modifying the motion file that contained the pages (actions), including adjusting individual values for each motor and incorporating elements of pre-defined actions. As a result, we were able to create eight custom actions.

We successfully implemented speech recognition in our system by establishing a connection between the robot and the laptop through socket programming. This allowed us to process the robot's microphone input and perform word or short phrase recognition as well as long speech recognition. By converting human speech into a transcribed format, our system achieved efficient and accurate speech recognition. This integration enhanced the overall functionality and responsiveness, enabling effective communication and interaction between the robot and the user.

We also implemented text-to-speech functionality in our system, allowing the robot to audibly communicate the results to its user. By converting the textual output into spoken words, we enable a more interactive and user-friendly experience.

We incorporated face recognition into our system using the "**face_recognition**"[7] library, which detects faces, extracts facial features, and encodes them numerically. The encoded features are compared to known faces in a database for identification or verification purposes. This implementation strengthens system security by acting as the initial authentication step, ensuring that only authorized users are granted access or functionality.

The project incorporated open-source code to detect bad posture, which served two important purposes. Firstly, it allowed for continuous tracking of the user's posture, providing timely warnings or advice if poor posture persisted. Secondly, when addressing neck or back pain, the robot considered the user's posture and adjusted its actions accordingly. By continuously analyzing the user's posture in real-time, the system could suggest and promote better posture habits and reducing discomfort or pain associated with prolonged desk usage. This ongoing monitoring prioritized the user's well-being and ergonomic considerations throughout their work or study sessions.

Command handling is a critical aspect of our system, governing the robot's response to user commands. Utilizing the "**word_finder.py**" script (see appendix C), which is a python code that looks for and finds certain words that the user says, and speech recognition, the robot can perform various functions based on recognized commands. These functions include self-introduction, providing time and date information, handling pain by offering appropriate responses, web scraping for retrieving relevant information, Wikipedia summaries based on user-defined topics, audio note recording and playback, and entertaining the user with jokes. With effective command handling, our system facilitates seamless interaction, offering a diverse range of functionalities to meet the user's specific needs and preferences.

During the system testing phase, we performed iterative tests to assess the performance of Darwin's functionalities and identify areas that needed improvement, those included

the team members trying to use Darwin's functions as they were needing them. Through this process, we identified specific aspects of the system that required enhancements. We subsequently made suitable optimizations to address these areas and improve the overall performance of the system. For more info about the system testing, please refer to section (3.1.11).

After designing the system, we conducted an evaluation to assess its performance. This evaluation allowed us to identify the system's strengths as well as areas that needed improvement and optimization. Given the nature of our system, which involves human-robot interaction and emphasizes a seamless and responsive experience, we decided to use a qualitative approach for evaluation. Based on the results of our evaluation, we determined that two areas, namely speech recognition and responsiveness, require further optimization and improvement. In our future plans, we aim to focus our efforts on enhancing these aspects to create a more responsive and seamless system experience. For more info about the system evaluation, please refer to section (3.1.12).

1.5 Summery of Report Structure

This report comprises six chapters, each focusing on different aspects of the project. The chapter breakdown and content overview are as follows:

The first chapter provides an overview of the report and includes sections such as the Problem Statement and Purpose, Project and Design Objectives, Intended Outcomes and Deliverables, Summary of the Used Design Process, and Summary of Report Structure.

The second chapter delves into the theoretical background of the project. It encompasses a Relevant Literature Search for GP-II and Mathematical Modeling specifically related to the q-learning component of the system.

Chapter three is dedicated to the Detailed Design of the Model/Prototype. It discusses various aspects such as Creating Custom Actions, Speech Recognition, Text-to-Speech,

Face Recognition, Bad Posture Detection, Command Handling, Project Management and Schedule, Testing and Improvement, System Performance Evaluation, and Engineering Standards Constraints. Additionally, this chapter explores Alternative Designs considered during the system design process.

The fourth chapter addresses Economic, Ethical, and Contemporary issues associated with the project. It covers topics such as Project Cost, Relevant Codes of Ethics, Ethical Dilemmas and Justification of Proposed Solution, and Relevant Environmental Considerations.

Chapter five summarizes the results of the system and presents the findings from the system evaluation. This section demonstrates the viability and effectiveness of the system by showcasing relevant results.

The final chapter includes the Conclusion section, providing a comprehensive overview of the project's final outcome. It also highlights the inclusion of the system on GitHub. The Future Work section outlines the goals and areas of improvement identified for future development, with a focus on enhancing functionality, responsiveness, and security.

By organizing the report into these chapters, we aim to provide a comprehensive understanding of the project, covering its introduction, theoretical foundations, detailed design, economic and ethical considerations, evaluation results, and a conclusion that reflects on the overall project outcome while outlining future work.

CHAPTER 2: THEORY

2.1 Relevant Literature Search for GP-II

Our project was initiated by conducting a comprehensive literature review on Human-Robot Interaction (HRI) and the development of robot personal assistants to gain a better understanding of the challenges and approaches used in this field. Through this research, we gained the following valuable insights:

The first paper presents the development of a voice-controlled personal assistant robot that can perform various movements[8], turns, start/stop operations, and object relocation. The robot receives voice commands remotely via a smart mobile phone, and the commands are processed in real-time using an online cloud server. The converted text commands are communicated to the robot through a Bluetooth network. The accent of the speaker does not impact the robot's operation since the voice commands are processed on a cloud server that is accent independent. The use of renewable energy sources for the robot's functioning is proposed as a cost-effective and eco-friendly approach. The developed robotic assistant has potential applications in industries and homes, demonstrating a signal processing application for voice-controlled robotics.

The second paper focuses on a robot designed to assist elderly or disabled individuals[9]. The robot consists of a separate robotic base for movement control, allowing for easy interchangeability. A TABLET PC serves as the user interface and facilitates application development. Both the TABLET PC and the main controller process data, with internet-related processing handled by the TABLET PC and sensor-related processing performed by the main controller. The physical creation of the robot is described, highlighting its telepresence and healthcare functionalities, enabling assistance in diagnostics and problem resolution.

The third research paper talks about Lio[10], which is a mobile robot platform equipped with a multifunctional arm specifically designed for human-robot interaction and personal care assistance tasks, particularly in healthcare facilities and home care. Its design takes into account the limitations of other robot platforms and aims to improve upon them. Lio utilizes in-house developed robot control and programming software, myP, in conjunction with algorithms based on the Robot Operating System (ROS) to create a node-based system functionality. Communication is established through ROS topics, allowing for the integration of additional modules into the system using standard ROS communication protocols and access to sensor data. myP offers robot pose and script databases, path planning, and machine learning capabilities, which can be accessed through the robot's user-friendly browser interface. Lio has several AI algorithms integrated into its system, including human pose estimation, object detection and recognition, object grasping, face detection and recognition, door opening and closing, voice recognition and synthesis. It is equipped with loudspeakers and a multidirectional microphone, allowing it to interact through voice and sound. Lio can understand voice commands and generate speech or play music, using a male voice as it is easier for elderly individuals to understand. However, Lio may struggle to understand some individuals with disabilities or elderly individuals and may have difficulty navigating cluttered environments or narrow spaces not suited for wheelchairs.

The fourth research paper talks about a magic act performed by Robotis DARwin OP2 robot that requires the application of artificial intelligence and robotics principles[11], such as perceiving a potentially error-prone environment, goal-directed reasoning, adapting to failure, and successful interaction, all of which must occur in real-time under difficult conditions. In this project, an Action Handler was developed using the libdarwin Framework, and audio input was recorded with a NESSIE Adaptive USB Condenser Microphone. The robot's video feed was obtained using OpenCV and a Logitech camera, and an API was created for playing card classification, speech recognition, and raw capturing of video and audio input.

The fifth research paper describes a pilot experiment involving human-robot interaction during English classes with pre-school children using a small humanoid robot as a teacher assistant[12]. The Robotis DARwin OP2 was used for communication through both verbal (speech) and non-verbal (gestures) channels, with the Robot Operating System (ROS) serving as the backbone of the software and the Gazebo simulator being used for testing algorithms before transferring them to the real robot. Despite its limitations in terms of mobility, vocal capabilities, and computational resources, the robot was able to effectively engage the children's attention and facilitate learning. However, the Robotis OP2's ability to interact with the internet offers additional features, including cloud computing which could be used to overcome some of the computational limitations. Overall, the robot garnered significant interest and successfully maintained the children's attention while helping them learn.

Our research shares several similarities with previous studies, including the use of the Robot Operating System (ROS) for communication. Also, our system is voice-controlled, incorporating speech recognition and text-to-speech functionalities. Our work platform, the Darwin OP2, is also similar to those used in previous papers. Darwin OP2 is capable of performing various physical actions. This aspect allowed us to leverage the robot's capabilities in providing medical advice. One of the papers discusses the use of cloud computing, which we ourselves used in our system to address some of the computational limitations of our platform. Specifically, we employed ROS to publish data from the robot, which was then processed by a laptop before being returned to the robot. This approach allowed us to overcome certain limitations and advance the state of the art in this field.

2.2 Mathematical Modeling

In this project, an important component involved Darwin providing advice to users experiencing various types of pain, including neck pain, arm pain, leg pain, back pain, and shoulder pain. To achieve this, q-learning algorithm were employed. These

algorithms enabled Darwin to learn and provide appropriate advice based on the user's specific complaint.

Q-learning is a reinforcement learning algorithm that helps an agent learn to make optimal decisions in a dynamic environment. It involves an agent exploring different actions in its environment and updating its knowledge based on the feedback received. The key idea behind Q-learning is to maintain a table, called the Q-table, which stores the expected rewards for different state-action pairs. The agent starts with an empty Q-table and gradually fills it through exploration and learning.

In simple terms, Q-learning is a method that allows an agent to learn how to make good decisions by exploring its environment, receiving feedback in the form of rewards, and updating its knowledge to maximize long-term rewards.

For our system, we defined the states and actions of our system as follows:

- States (rows of the Q-table):

    0. No pain (Terminal state: the state at which the agent terminates).
    1. Neck pain + Bad posture.
    2. Neck pain.
    3. Back pain + Bad posture.
    4. Back pain.
    5. Leg pain.
    6. Arm pain.
    7. Shoulder pain.

- Actions (columns of the Q-table):

    0. Neck exercise #1
    1. Neck exercise #2
    2. Arm exercise #1

3. Back exercise #1
4. Leg exercise #1
5. Back exercise #2
6. Arm exercise #2
7. Walk advice
8. Extend knee advice
9. Extend palm advice
10. Head side bending advice
11. Visit a doctor advice

We chose those pain types because they are common among people who spend a lot of time at their desks. We also selected exercises based on research we did to find the best ways to relieve those pains. Our goal was to create a system that can provide helpful advice and assistance for those specific pain types.

The user provides the initial state by expressing their discomfort, such as saying "I have neck pain." In this scenario, the q-learning system would assign it to state #2: Neck pain. Next, the system utilizes image processing to evaluate the user's posture. If the analysis detects bad posture, the state is updated to state #1: Neck pain + bad posture. It's important to note that the detection of bad posture only applies to state #2: neck pain and state #4: back pain.

Once the state has been identified, the agent proceeds to make optimal decisions using the Q-table. The objective is to reach the terminal state (state #0: No pain). These decisions involve a combination of physical actions and verbal advice that the agent takes by sending them to the robot to perform or speak out. The agent aims to assist the user in alleviating or eliminating their pain through these actions and advice.

Table 1 below shows the initial state of the Q-Table in our system.

Table 1: Q-table initial state

	Physical actions							Verbal advices				
Actions / States	neck_1 #0	neck_2 #1	arm_1 #2	back_1 #3	leg_1 #4	back_2 #5	arm_2 #6	walk #7	extend knee #8	extend palm #9	side bending #10	see a doctor #11
no pain #0	0	0	0	0	0	0	0	0	0	0	0	0
neck pain + bad posture #1	0	0	0	0	0	0	0	0	0	0	0	0
neck pain #2	0	0	0	0	0	0	0	0	0	0	0	0
back pain + bad posture #3	0	0	0	0	0	0	0	0	0	0	0	0
back pain #4	0	0	0	0	0	0	0	0	0	0	0	0
leg pain #5	0	0	0	0	0	0	0	0	0	0	0	0
arm pain #6	0	0	0	0	0	0	0	0	0	0	0	0
shoulder pain #7	0	0	0	0	0	0	0	0	0	0	0	0

During the learning process, the agent follows an exploration-exploitation strategy to select actions based on its current state. Exploration involves randomly choosing actions, while exploitation entails selecting the optimal action from the Q-table. Initially, the agent focuses on exploration and gradually transitions towards exploitation as the learning process progresses.

The learning process is divided into episodes, which represent complete system runs, and each episode consists of multiple steps. At each step, the agent first chooses an action based on the exploration-exploitation strategy. It then executes the chosen action, observes the resulting state, and receives a reward. The agent updates the Q-table by considering the obtained reward and the expected rewards in the next state.

For instance, if the agent is in state #2 (neck pain) and selects action #1 (neck exercise #2), and the user reports that it alleviates their pain, the agent transitions to state #0 (no pain) and receives a reward accordingly. As state #0 is a terminal state, the episode concludes, and the system resets for a new episode.

On the other hand, if the user indicates that the action did not alleviate their pain, the step concludes, and the agent proceeds to the next step within the same episode, unless it has reached the maximum number of steps allowed.

To update the Q-values for each pair of (state, action), we use the Bellman optimality equation:

$$q\ new(s, a) = (1-\alpha)\ \{old\ value\} + \alpha\ \{learned\ value\}$$

Where, α represents the learning rate, which is a predetermined value, has value in the range [0, 1]. The higher α, the more quickly the agent will adopt the new Q-value.

The agent determines whether it's going to do exploration or exploitation based on the exploration rate, for example, if the exploration rate in the current step is 0.8 then the agent has an 80% chance to do exploration, and 20% chance to do exploitation. The exploration rate is updated after each episode using the following equation:

$$\in = \in \times e^{-exploration\_decay\_rate \times episode\_number}$$

Both exploration rate (ϵ) and exploration decay rate are predetermined values.

The discount rate (γ) in Q-learning is a value that determines the importance of future rewards compared to immediate rewards. It represents the degree of "discounting" or reducing the value of future rewards. A higher discount rate emphasizes immediate rewards, while a lower discount rate places greater importance on long-term rewards. In essence, the discount rate influences how much an agent values future rewards when making decisions in the learning process.

By repeatedly updating the Q-table over many iterations, the agent learns to make better decisions in each state, aiming to maximize the total expected reward over time. Eventually, the agent converges to an optimal policy, knowing the best actions to take in each state to maximize its long-term reward. The Q-table in our system is only calculated during the learning process, and does not get updated when the system is running.

predetermined values for the training of our Q-learning system:

Learning rate (α): 0.1

The discount rate (γ): 0.05

exploration rate (ϵ): 1

exploration rate decay: 0.00008

Number of episodes: 100000

Max steps per episode: 12

To facilitate the learning process, we developed a function that simulates human user input based on the state and action taken by the agent. This simulation involves assigning probabilities to different outcomes. For instance, when in state #2 (neck pain) and the agent selects action #1 (neck exercise #2), there is a 95% probability of returning "yes" to indicate that the action successfully alleviated the pain.

Conversely, if the agent chooses action #4 (leg exercise #1) for the same state, the function immediately returns "no" to indicate that the action did not provide relief. By incorporating these probabilistic responses, we emulate user feedback to further enhance the agent's learning process. Table 2 shows the final form of the Q-table after we ran our learning process.

Table 2: Q-table state after learning process

	Physical actions							Verbal advices				
Actions / States	neck_1 #0	neck_2 #1	arm_1 #2	back_1 #3	leg_1 #4	back_2 #5	arm_2 #6	walk #7	extend knee #8	extend palm #9	side bending #10	see a doctor #11
no pain #0	0	0	0	0	0	0	0	0	0	0	0	0
neck pain + bad posture #1	0.801	0.981	0.049	0.047	0.049	0.048	0.048	0.723	0.049	0.047	0.898	0.205
neck pain #2	0.794	0.952	0.048	0.048	0.048	0.048	0.048	0.048	0.048	0.048	0.874	0.071
back pain + bad posture #3	0.045	0.046	0.631	0.972	0.045	0.786	0.045	0.582	0.045	0.046	0.045	0.313
back pain #4	0.045	0.045	0.62	0.873	0.045	0.837	0.045	0.044	0.044	0.045	0.045	0.338
leg pain #5	0.047	0.046	0.047	0.684	0.992	0.047	0.047	0.833	0.77	0.047	0.046	0.262
arm pain #6	0.047	0.047	0.896	0.047	0.047	0.649	0.623	0.047	0.047	0.941	0.048	0.125
shoulder pain #7	0.047	0.047	0.476	0.047	0.047	0.883	0.552	0.047	0.048	0.047	0.822	0.265

The implementation of the Q-learning algorithm has proven instrumental in developing a system capable of providing appropriate advice to users based on their specific pain and posture. This functionality holds particular significance for individuals who spend prolonged periods at their desks. By leveraging Q-learning, our system offers valuable guidance tailored to the user's needs, promoting better posture and alleviating pain associated with extended desk work.

CHAPTER 3: DETAILED DESIGN

3.1 Detailed Design of the Model/Prototype

Our design process for this project included many steps, which are:

1. Accessing the robot.
2. Learning how to use ROS and implementing publishers and subscribers.
3. Calling actions on the robot.
4. Creating custom actions.
5. Speech recognition.
6. Text to speech.
7. Face recognition.
8. Bad posture detection.
9. Command handling.
10. Project Management and Schedule.
11. Testing and Improvement.
12. System Performance evaluation.

3.1.1 Accessing the robot

During the design process, it was necessary to access the operating system of the robot to add or modify scripts, actions, and ROS launch files.

It is important to note that the OP2 robot we are utilizing has a dual boot system, with two operating systems installed:

- Lubuntu 12.04[13]: This operating system was pre-installed on the robot and was mainly used for the action editor (as described in section 3.2.5).

- Linux Mint 17.3 (Rosa)[14]: This operating system was installed at the recommendation of Robotis e-manual and was our primary platform due to its newer features and improved capabilities.

A graphical issue on the robot resulted in the external monitor not displaying any output when connected. However, we were able to successfully establish a remote connection to the robot using a Virtual Network Computing (VNC) session between the laptop and the robot, using the UltraVNC program[15]. To connect to the robot, we entered the robot's IP address and password in the required fields. As shown in Figure 1, remote access to the robot was successfully established using UltraVNC from a laptop.

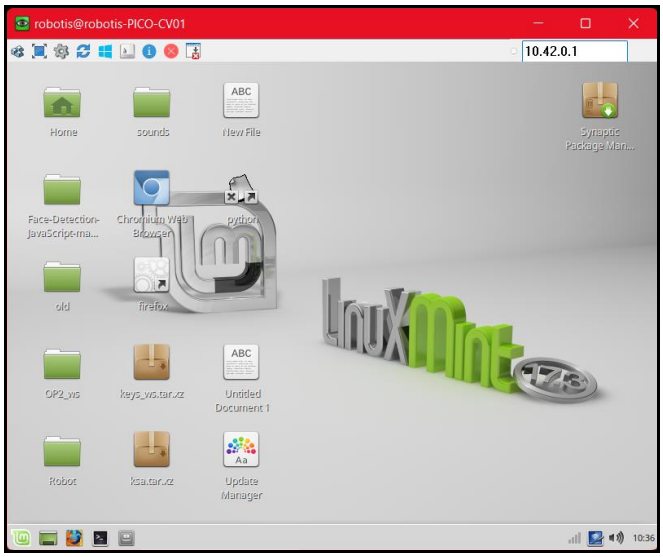


Figure 1: Accessing the robot using UltraVNC.

This enables us to access the graphical user interface (GUI) of the robot's operating system (Linux Mint 17.3 (Rosa)).

3.1.2 Learning how to use ROS

To learn about the Robot Operating System (ROS), we followed the well-documented tutorials on the ROS website[16].

We successfully installed ROS Noetic on our system and gained proficiency in navigating the ROS filesystem. Additionally, we created a dedicated ROS package named "**robot_personal_assistant_op2**" to encapsulate our work. We recognized the importance of version control and code backup, which prompted us to upload the package to GitHub[17]. This decision served two primary purposes. Firstly, it ensured that our code was securely stored and easily accessible for future revisions. Secondly, we aimed to provide a reference point for other individuals undertaking similar projects, allowing them to benefit from our work. Furthermore, we expanded our knowledge of ROS concepts, including nodes, topics, publishers, and subscribers, which enabled us to establish effective communication and coordination within the system.

We implemented our own image publishing and subscribing scripts, with the publisher node reading the webcam of the robot as input, converting it to image data, and publishing it on a specific ROS topic, called "**/webcam**". The subscriber node then reads the image message from the same topic, converts it to an image, and processes it for display or further processing.

3.1.3 Calling actions on the robot

There are two methods for calling actions on the robot, both of which require the OP2 manager to be running on the robot.

The first method is to use the included OP2 GUI demo, which is a ROS package that can be installed from the Robotis e-manual website[18]. Figure 2 illustrates the interface of the OP2 GUI demo.

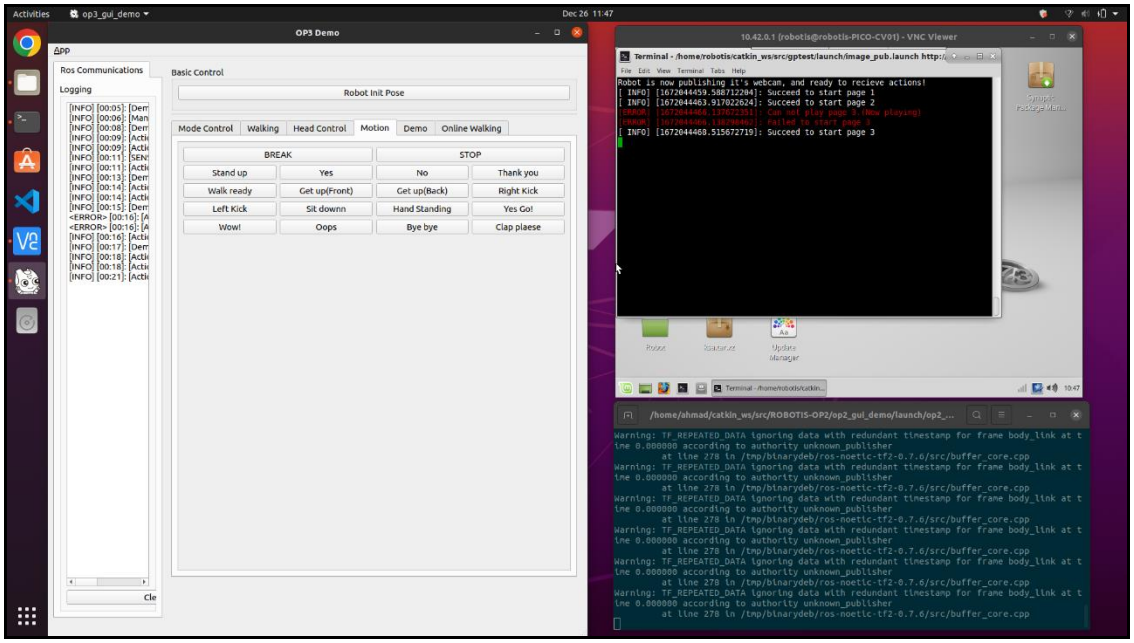


Figure 2: OP2 GUI demo.

The second method is to write a python script that publishes the desired actions. To do this, we need to enable the action module by sending the message "**action_module**" to the OP2 manager on a specific ROS topic called **"/robotis/enable_ctrl_module**". Once the action module is enabled, we can instruct the robot to perform an action by sending its page number (action number) on the "**/robotis/action/page_num**" ROS topic. The robot's actions are stored in an action file consisting of 256 pages, each representing one action and capable of storing up to seven stages (or steps) of action data. Figure 3 below shows the flowchart for the robot side.

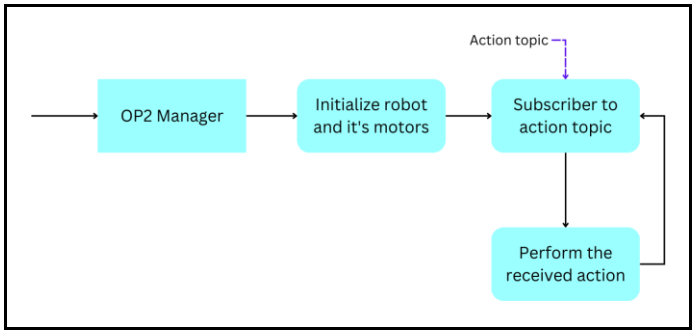


Figure 3: Action initalization flowchart.

Figure 4 below illustrates the robot performing an action.

Figure 4: Robot performing a welcoming action.

3.1.4 Creating Custom actions

One of the main challenges we encountered during the project was developing custom actions for the robot. After evaluating various approaches, we ultimately utilized the action editor provided by Robotis OP2. This tool was already installed on the robot and could be accessed through the Lubuntu 12.04 operating system, due to us facing challenges using the action editor from the Linux Mint 17.3 (Rosa). Figure 5 below presents the interface of the Action Editor. Which consists of 256 pages (actions) and each page consists of 7 steps. For each step, we have to define the values for each motor on the robot.

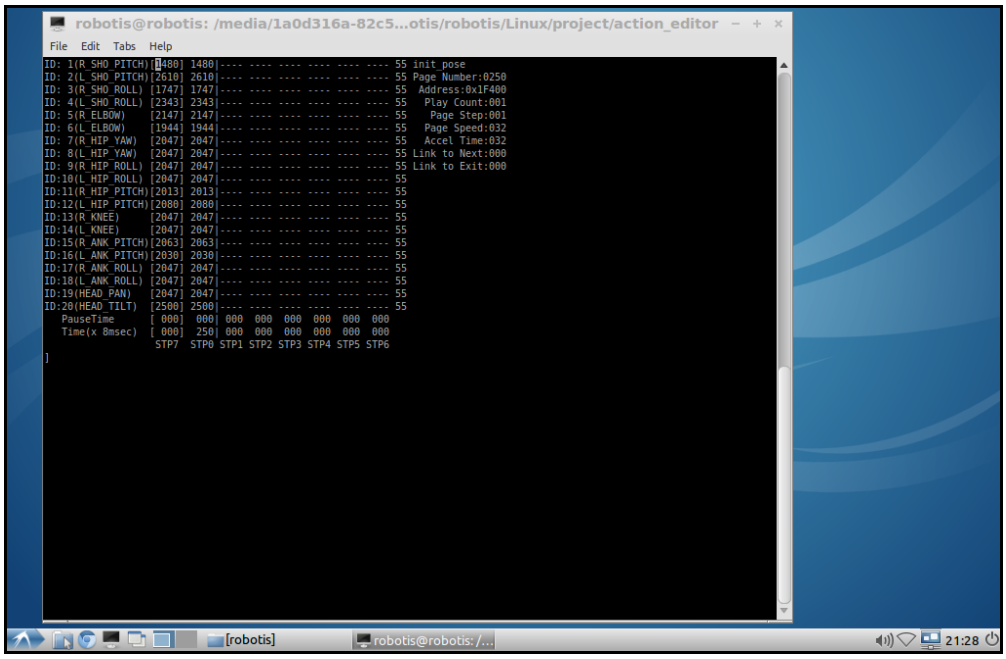


Figure 5: OP2 action editor.

During the project, we gained familiarity with the action editor through the Robotis OP e-manual[18]. This involved modifying the motion file that contained the pages (actions), including adjusting individual values for each motor and incorporating elements of pre-defined actions. As a result, we were able to create **eight** custom actions. They were as follows:

1. robot_initial_pose (100): initial position that the robot goes back to after every action, see Figure 6 - (a).
2. neck_1 (101): head pan (left and right), see Figure 6 - (b).
3. neck_2 (102): head tilt (up and down), see Figure 6 - (c).
4. arm_1 (103): robot raises its arms like a fork and holds for a few seconds, see Figure 6 - (d).
5. back_1 (104): standing lower back stretch, see Figure 6 - (e).

6. leg_1 (105): robot stands on one leg, raises its other leg to the back, then uses its arm to hold its extended leg, see Figure 6 - (f).
7. back_2 (106): robot makes its elbows 90 angle, then pulls its arms to the back, see Figure 6 - (g).
8. arm_2 (107): robot extends its arms to the front, then extends its arms to the sides, see Figure 6 - (h, k).

Figures 6 below shows the custom actions we created on Darwin:

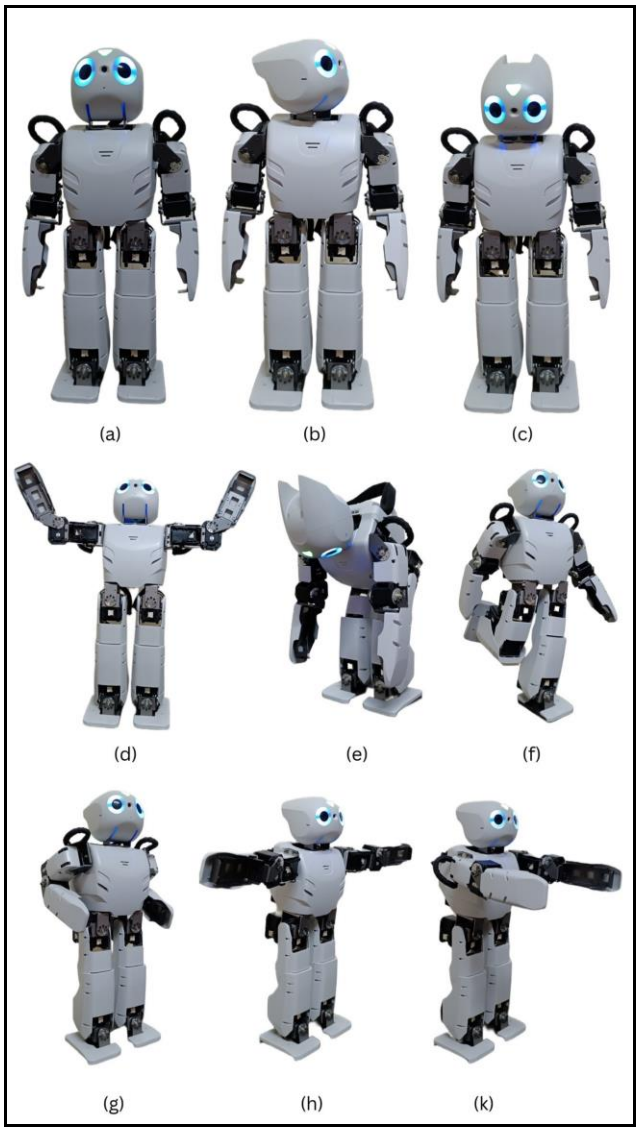


Figure 6: Custom actions on Darwin OP2

3.1.5 Speech Recognition

Speech recognition, or automatic speech recognition (ASR), is a technology that transforms spoken language into written text by analyzing and interpreting audio signals. It plays a significant role in our system since voice commands are the primary means of communication and interaction between the robot and the user. Through speech recognition, the system can accurately transcribe spoken words and phrases, enabling seamless and intuitive communication with the robot. This functionality allows users to control the robot, provide instructions, and engage in natural and efficient interactions using their voice.

To enable speech recognition in our system, we initially needed to transmit the robot's microphone input to the laptop where the processing would take place. This was achieved through socket programming, establishing a connection between the robot and the laptop. The microphone data was recorded in chunks and sent through the socket. The python script that was responsible for sending the audio is called "**audio_sender.py**" (see appendix C).

To implement speech recognition in our system, we needed to process the audio data received from the robot in two different ways: word or short phrase recognition and long speech recognition. These approaches served specific purposes within our system. For instance, we used word or short phrase recognition to detect specific phrases like "hey Darwin." On the other hand, long speech recognition was employed when we needed to record and transcribe user questions for online retrieval of answers. By utilizing these two modes, we ensured efficient and accurate speech recognition based on the intended use case.

Here are the steps we followed to implement word or short phrase recognition:

1. Create the "**audio_reciever_speech_recognizer.py**" (see appendix C) Python script to run on the laptop. It establishes a socket connection with the robot's audio sender and records the user's audio for 1.5 seconds. Immediately after recording, it creates a separate thread and passes the audio data captured in the 1.5-second interval.

2. The main script continues to record additional 1.5-second segments, ensuring a 0.5-second overlap with the previous recording. This overlap helps prevent cutting off any words and ensures a more complete audio capture, resulting in 2 seconds of continuous recording.

3. The thread responsible for receiving the audio data converts it into a suitable format and applies speech recognition using the "**speech_recognition**" open-source Python library[2]. When the thread obtains the output text from the speech recognition process, it publishes it as a string to the ROS topic "**/speech_recognition_output**".

4. Additionally, a parallel Python script named "**word_finder.py**" (see appendix C) subscribes to the same ROS topic. Upon receiving a string from the topic, it checks for specific words, including types of pains in certain cases. If any words of interest are found, the script returns the identified word and the corresponding pain type (if applicable).

The steps for short speech recognition are explained in the flowchart in Figure 7 below:

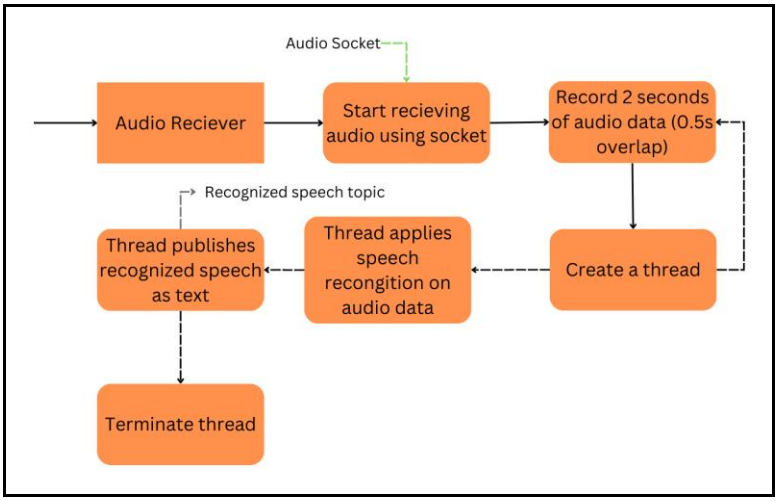


Figure 7: Short speech recognition flowchart.

By following these steps, we successfully implemented word or short phrase recognition in our system, enabling the identification of specific words and facilitating the detection of relevant pain types when necessary.

To implement long speech recognition in our system, we followed the following steps:

1. Firstly, we established a socket connection and connected to the audio sender on the robot, enabling us to receive the audio data.

2. We recorded the received audio data until the user remained quiet for a significant period. To determine user silence, we calculated the root mean square (rms) energy of the audio data chunks and compared it to a predetermined threshold. If we observed a substantial number of low-energy chunks (below the threshold), we considered the user to be quiet, indicating it was safe to stop recording their voice.

3. After recording, we converted the audio data into a suitable format and applied speech recognition using the "**speech_recognition**" open-source Python

library[2]. This allowed us to transcribe the user's speech into text form. We could then process the resulting string output as needed, such as using it to search for answers to user queries from the internet.

The steps for long speech recognition are explained in the flowchart in Figure 8 below:

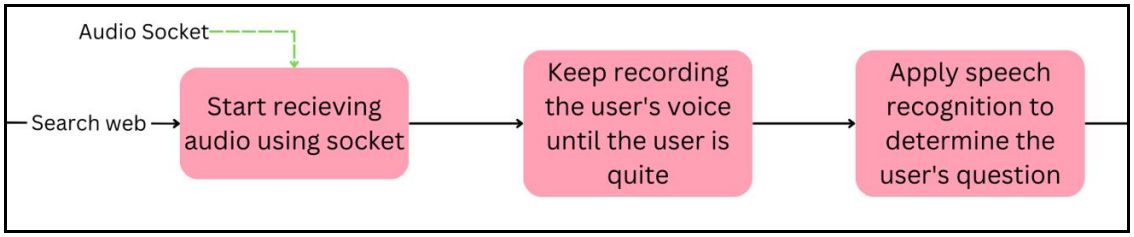


Figure 8: Long speech recognition flowchart.

By following these steps, we successfully implemented long speech recognition in our system, enabling us to capture and transcribe longer speech segments, thus enhancing the overall interaction and functionality of the robot with the user.

In summary, speech recognition played a significant role in our system, enabling us to establish a dependable and interactive experience between the robot and the user. By converting human speech into a transcribed format, we gained the ability to process and analyze the user's input, allowing the robot to interact and respond accordingly. This integration of speech recognition enhanced the overall functionality and responsiveness of our system, fostering effective communication and seamless interaction between the robot and its user.

3.1.6 Text to Speech

Text-to-speech (TTS) is a technology that converts written text into spoken words. It is an automated process that synthesizes human-like speech from text input, allowing computers or devices to “speak” the written content. This technology is utilized in

various applications, such as voice assistants, accessibility tools, navigation systems, and more, to enable information dissemination through audible speech.

Text-to-speech (TTS) is an essential component of our system as it allows the robot to audibly communicate the results to its user. By converting the textual output into spoken words, we enable a more interactive and user-friendly experience. To implement text-to-speech (TTS) functionality in our system, we followed these steps:

1. Initially, we created two Python scripts: "**text_to_speech_publisher.py**" (see appendix C) to run on the laptop and "**text_to_speech_subscriber.py**" (see appendix C) to run on the robot.

2. The publisher code accepts a string input and utilizes the open-source "**google text to speech (gTTS)**" Python library to generate speech audio from the provided string[3].

3. The generated speech audio is then converted into a suitable audio data format and published on a ROS topic named "**/tts**".

4. On the robot side, the subscriber script receives the audio data from the same ROS topic. It uses the "**pygame**" Python library to play the audio through the robot's speaker, enabling the user to hear the synthesized speech[19].

5. During this time, the publisher script waits for the subscriber script to finish playing the text-to-speech audio. It detects the completion by listening for a signal sent on a specific ROS topic named "**/finished_talking**".

The steps for text to speech are explained in the flowchart in Figure 9 below:

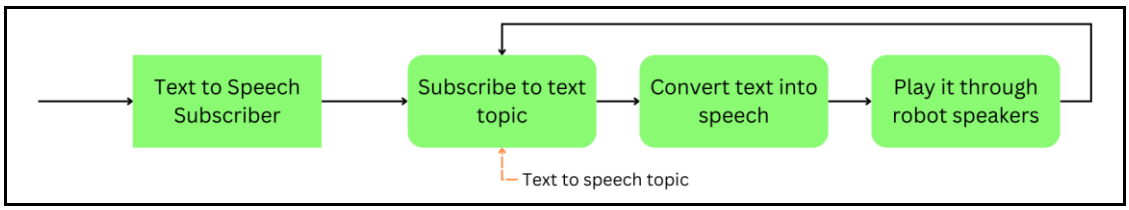


Figure 9: Text to speech flowchart

By following these steps, we successfully implemented text-to-speech functionality in our system, allowing the robot to convert textual information into spoken words, enhancing the interaction and communication experience with the user.

3.1.7 Face Recognition

Face recognition is a technology that involves identifying and verifying individuals based on their facial features. It uses algorithms to analyze patterns and unique characteristics of a person's face, such as the shape of the eyes, nose, and mouth, to match it against a database of known faces.

To implement face recognition in our system, we used an open-source python library called "**face_recognition**"[7]. The library detects faces in images or video frames, extracts unique facial features, and encodes them into numerical representations. These encodings are associated with known faces in a database. When presented with a face, the library computes its encoding and compares it against the encodings in the database to perform identification or verification tasks, providing outputs such as recognized identities or match results.

Facial recognition plays a crucial role in enhancing system security by serving as the initial verification step. The system ensures that only authenticated users are recognized before allowing further access or functionality. The steps involved in this process are as follows:

1. Load Face Encodings: The Python code loads the face encodings of known authentic users.

2. Receive Webcam Image: The code subscribes to the "**/webcam**" ROS topic and receives images from the robot's webcam.

3. Face Detection and Encoding: For each received image, the code detects faces, extracts unique facial features, and encodes them.

4. Compare Encodings: The face encodings of the detected faces are compared with the encodings of the known authentic users.

5. Match Found: If a match is found, the robot greets the user with the message "Hello [name], it's good to see you again!" and triggers an action by sending the value 100 to the action module, initiating the "**initial_robot_pose**" action.

6. No Match Found: If no authentic user is found, the code continues in the loop, capturing a new image from the webcam, and repeats the face recognition process.

By exiting the loop and entering the main code after one or more authentic users are detected, the system transitions to its normal operational state, where it awaits user commands and carries out the required tasks.

The steps for face recognition are explained in the flowchart in Figure 10 below:

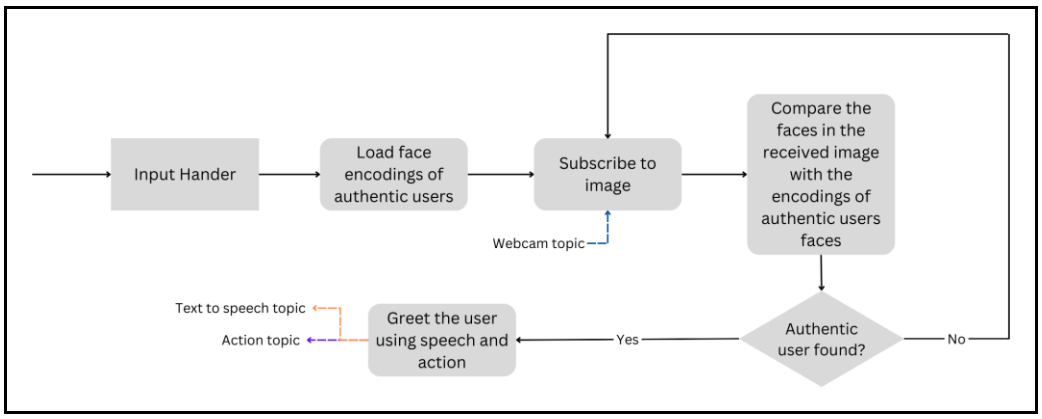


Figure 10: Face recognition flowchart

3.1.8 Bad Posture Detection

Posture detection, also known as pose estimation or pose detection, refers to the process of detecting and analyzing the body posture or pose of a person in an image or video. It involves identifying and tracking key body joints or landmarks to estimate the positions and orientations of different body parts.

In this project, we utilized open-source code for detecting bad posture[20]. This functionality holds significance for two primary reasons. Firstly, it enables the tracking of the user's posture, allowing for timely warnings or advice if they maintain a poor posture for an extended period. Secondly, when the robot aims to provide advice regarding neck or back pain, it first detects the user's posture. If a bad posture is detected, the robot suggests slightly different actions compared to cases without bad posture. This approach ensures that the robot delivers the most suitable advice based on the user's current condition. The bad posture detection in our system operates through the following steps:

- Receive Webcam Image: The code subscribes to the "**/webcam**" ROS topic and retrieves images from the robot's webcam.

- Perform Bad Posture Detection: The system analyzes the received images to determine if the user's posture is good or bad. It starts counting the number of seconds that the user has been in a bad posture.

- Send Bad Posture Time: The system sends the number of bad posture seconds on the "**/bad_posture_time**" ROS topic.

- Advice/Warning Mechanism: In a suitable state, the main code checks the bad posture time. If the duration exceeds 10 seconds (for testing), a counter is initiated. When the counter reaches a certain value the system then sends advice or warning to the user by converting the text into speech using the text-to-speech module. This way we ensure that the robot does not spam the user with warnings.

- Handling Neck/Back Pain: If the user reports neck or back pain, the system checks for bad posture. If the bad posture time is greater than 0, it treats the situation as both neck/back pain and bad posture occurring simultaneously.

Mediapipe pose is a high-fidelity body pose tracking solution that renders 33 3D landmarks and a background segmentation mask on the whole body from RGB frames[21], as shown in Figure 11 below.

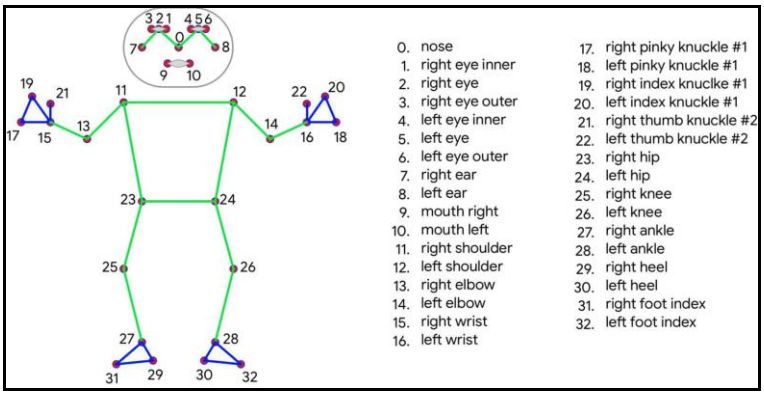


Figure 11: Mediapipe body pose landmarks[22]

The angle is the primary deterministic factor for the posture. We use the angle subtended by the **neckline** and the **torso line** to the y-axis. The neckline connects the shoulder and the eye. Similarly, the torso line connects the hip and the shoulder. Then, those angles are compared to set thresholds (40° for neckline and 10° for torso line), and if they exceed the thresholds, the system determines that the posture of the user is bad[20].

The steps for bad posture detection are explained in the flowchart in Figure 12 below:

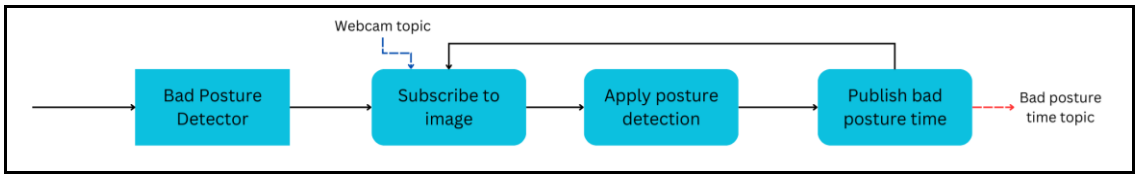


Figure 12: Bad posture detection flowchart

By running the code continuously, the system can effectively track and analyze the user's posture in real-time. This allows for timely interventions, such as providing advice or warnings, to encourage better posture habits and reduce the risk of discomfort, pain, or long-term injury associated with prolonged desk usage. The ongoing monitoring capability ensures that the user's well-being and ergonomic considerations are prioritized throughout their work or study sessions. Figure 13 shows the bad posture detection code working.

Figure 13: Bad posture detection.

3.1.9 Command Handling

Command handling is a crucial component of our system as it governs the robot's response to user commands. These commands are obtained through the "**word_finder.py**" (see appendix C) script, which relies on speech recognition. The robot is programmed to perform various functions based on the recognized commands; those commands are shown in table 3 below:

Table 3: Darwin's Commands

Command	Description
Self-introduction	The robot introduces itself, providing information about its identity and purpose.
Time/Date	The robot can inform the user about the current time or date.
Joke-telling	The robot has the ability to entertain the user by sharing humorous jokes.
Pain handling	If the user expresses discomfort or pain, the robot can provide appropriate responses or suggestions to alleviate or manage the pain.
Web scraping	The robot is capable of extracting relevant information from websites or online sources based on user queries or specific instructions.

Wikipedia summaries	By utilizing web scraping techniques, the robot can retrieve summarized information from Wikipedia based on user-defined topics.
Audio note recording and playback	The robot offers the functionality to record audio notes as per the user's request and can subsequently play them back.

Through effective command handling, our system ensures seamless interaction between the user and the robot, providing a range of functionalities tailored to meet the user's needs and preferences.

- **Self-introduction:**

When prompted with commands such as "**introduce yourself**", "**what can you do**", or "**who are you**", Darwin will provide a comprehensive paragraph introducing himself. This introduction aims to familiarize new users with Darwin's identity, purpose, and capabilities. Darwin will describe himself as a robot designed to assist and interact with users. He will mention his name, emphasizing his unique identity. Darwin will also highlight his primary purpose, which is to provide assistance and perform various functions to support users. Furthermore, he will elaborate on his capabilities, mentioning the range of tasks he can accomplish and how he can be helpful to users. This introductory paragraph serves as a helpful starting point for users to understand Darwin's role and how he can enhance their experience.

- **Time/Date:**

When the user requests the current time by saying "**what is the time**" or asks for today's date with "**tell me today's date**", Darwin will retrieve the time and date information from the operating system. He will then provide the accurate time or date to the user.

This feature is particularly useful for users who may not actively monitor the time or date on their devices. Darwin's ability to fetch and communicate this information allows users to conveniently stay updated without relying on their own devices or clocks.

- **Joke-telling:**

One delightful feature of Darwin is its ability to provide users with a joke. The joke is selected at random from a pool of jokes and then sent to the text-to-speech module to be spoken by the robot. This feature serves as a mood booster, bringing a smile to the user's face when needed. It adds an enjoyable touch to the interaction between the user and Darwin, providing a lighthearted moment of humor.

The steps for **Self-introduction, Time/Date, and Joke-telling** are explained in the flowchart in Figure 14 below:

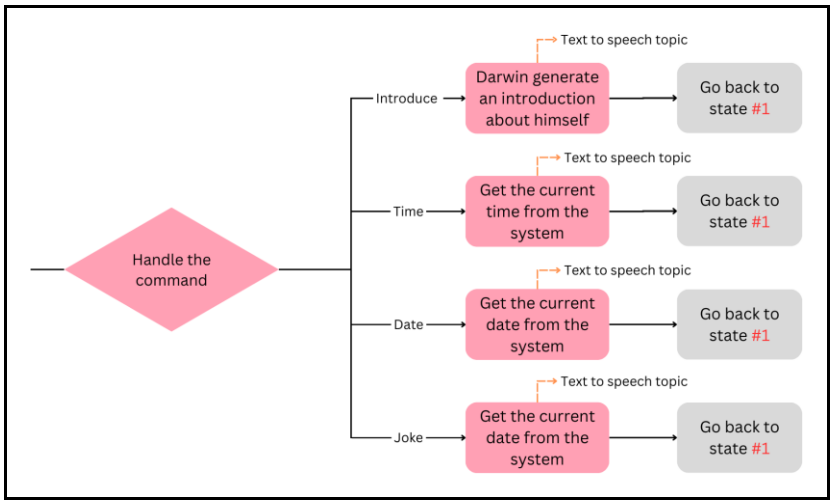


Figure 14: Self-introduction, Time/Date, and Joke-telling flowchart

State #1 represents the state where the system waits for user to give a command. So, after handling a command, the system waits for another command for a set amount of time. This can be seen in the flowchart in chapter 5.

- **Pain handling:**

Pain handling is a critical feature of Darwin, designed to support users who spend prolonged periods working or studying at their desks. These individuals often experience specific types of pain in their neck, back, legs, shoulders, arms, and head.

To address this, we utilized the Q-learning algorithm to train an agent capable of selecting optimal actions and advice based on the user's state. Q-learning part of our system is explained in detail in mathematical representations section (2.2).

Implementing pain handling involved the following steps:

1. First, the system waits for the user to say the words "**pain**" or "**hurt**" as an indication of their discomfort.

2. The "**word_finder.py**" (see appendix C) script also determines the type of pain if it matched the known types in the system.

3. If the pain type is unfamiliar, the robot suggests to the user that they should consider visiting a doctor if the pain persisted.

4. If the pain type is recognized, the "**pain_handler.py**" (see appendix C) script assigns a state based on the user's specified pain type and posture state.

5. Once the state is determined, the robot began providing advice and actions to the user based on the Q-values in the Q-table. The action with the highest Q-value was initially selected, followed by subsequent actions in descending order.

6. After providing each advice, the robot asks the user if it helped alleviate their pain. If the user responds positively, the robot ceased providing advice. However, if

the user indicates that the advice does not provide relief, the robot suggests the next best advice in the sequence.

7. The robot continues to offer advice until the user expresses satisfaction or until the "**pain_handler.py**" (see appendix C) code reaches the "**Visit a doctor advice**" action (action #11), in which case the code terminates.

The purpose of this process is to handle different pain types, provide appropriate responses or suggestions, and trigger additional actions if needed based on the determined pain state.

The steps for **Pain handling** are explained in the flowchart in Figure 15 below:

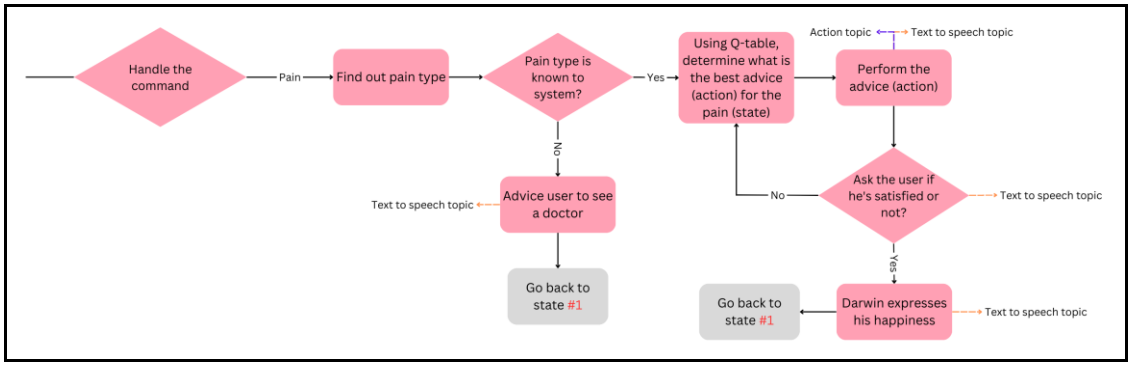


Figure 15: Pain handling flowchart

- **Web scraping:**

Web scraping is a method used to gather information from websites automatically. It involves extracting data such as text, images, or links from web pages. This capability is valuable for providing answers to user questions and enhancing Darwin's usefulness for individuals who spend extended periods studying or working at their desks.

To accomplish web scraping, the following steps are taken:

1. The system listens for the keywords "**search**" or "**Google**" as indicators that the user wants to search for an answer to a question.

2. The user's audio is recorded using the long speech recognition technique described in the speech recognition section (3.1.5).

3. Once the user finishes speaking their question, speech recognition is applied to extract the question as a string.

4. The question is then passed to the "**wolframalpha**" Python library[23], which attempts to find an answer.

5. If an answer is found, the robot communicates it audibly to the user.

This capability allows Darwin to effectively address user inquiries and further enhance its assistance for individuals engaged in long-term desk-based activities.

- **Wikipedia summaries:**

Wikipedia is an online encyclopedia with a vast amount of information on various topics. It is created and maintained by volunteers worldwide. One of the useful features of Darwin is its ability to retrieve summaries from Wikipedia using the "**wikipedia**" Python library. Here are the steps involved:

1. The system listens for the keyword "**wikipedia**" to identify when the user wants a summary on a specific topic[24].

2. The user's speech is recorded using the long speech recognition technique explained earlier in the speech recognition section (3.1.5).

3. After the user finishes speaking, speech recognition is applied to extract the topic as a text string.

4. The topic is then passed to the "**wikipedia**" Python library[24], which searches for a summary related to that topic.

5. If a summary is found, Darwin communicates it audibly to the user.

This functionality is particularly useful for individuals who engage in lengthy study sessions and need quick access to information from Wikipedia.

The steps for **Web scraping and Wikipedia summaries** are explained in the flowchart in Figure 16 below:

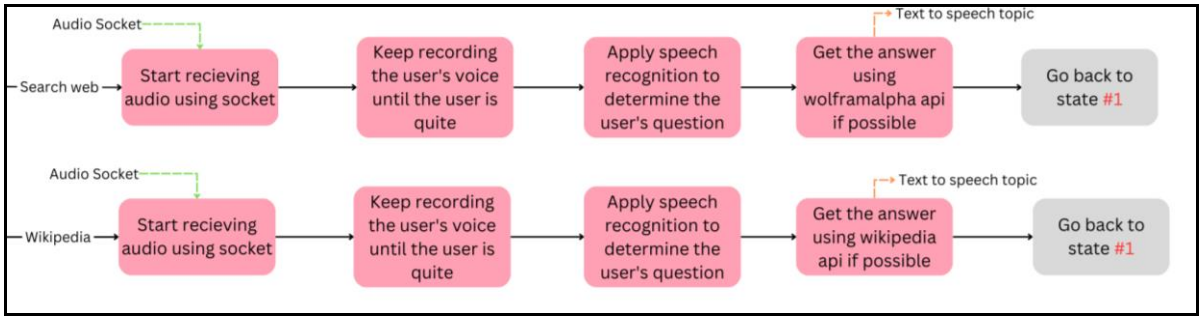


Figure 16: Web scraping and Wikipedia summaries flowchart

- **Audio note recording and playback:**

Audio note taking is a valuable feature of Darwin that enables users to create and play back voice notes using voice commands. The implementation involves the following steps:

Voice note taking:

1. The system detects keywords like "**note**" or "**record**" to identify when the user wants to create a voice note.

2. The user's speech is recorded, similar to the long speech recognition technique described earlier, but without applying speech recognition to the audio.

3. The recorded audio is saved as an mp3 file named "**recording{record_count}.mp3**".

Voice note playback:

1. The system listens for the keyword "**playback**" to determine when the user wants to listen to a specific voice note.

2. The system prompts the user to specify the number of the saved recording they wish to play back.

3. The audio data is loaded from the corresponding mp3 file, decoded into a string, and published on a ROS topic named "**/audio_topic**".

4. On the robot side, the "**audio_note_player.py**" (see appendix C) script subscribes to the same ROS topic and plays the received audio through the robot speakers.

5. Meanwhile, the audio note recorder waits for the subscriber to finish playing the audio recording. The completion signal is sent on the ROS topic "**/finished_talking**".

The steps for **Audio note recording and playback** are explained in the flowchart in Figure 17 below:

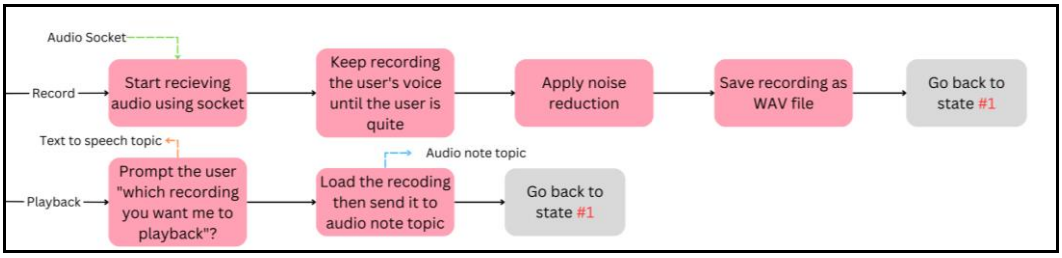


Figure 17: Audio note recording and playback flowchart

This functionality empowers users to conveniently record and listen to voice notes as needed, making it particularly beneficial for students and individuals seeking a reliable means of capturing and reviewing spoken information. For example, this feature can be used by students to prepare for oral exams.

3.1.10 Project Management and Schedule

Table 4: Time schedule of the project.

Week 1	Implemented speech-to-text functionality
Week 2	Developed text-to-speech functionality
Week 3	Implemented web scraping and date/time command handling
Week 4	Improved speech-to-text using multi-threading
Week 5	Implemented Q-learning system
Week 6	Implemented face recognition
Week 7	Implemented audio sending and receiving using socket programming for robot microphone
Week 8	Added bad posture detection
Week 9	Created custom actions for advice
Week 10	Expanded command handling with additional functions and optimizations
Week 11	Conducted system testing and performed further optimizations
Week 12	Conducted system evaluation and prepared this report

3.1.11 Testing and Improvement

During our system testing, we conducted iterative evaluations of Darwin's functionalities to assess their performance and identify areas for improvement. Through this process, we discovered certain aspects of our system that required enhancements.

Firstly, we initially utilized the "**pyttsx3**" Python library for text-to-speech generation[25]. However, we observed that the generated voice lacked clarity and it was challenging to comprehend English words. To address this, we explored alternative options and found that the "**gTTS**" library not only provided clearer speech but also better suited Darwin's overall appearance[3]. Moreover, this switch did not introduce any noticeable delays or additional processing time.

Secondly, for playing the generated speech, we initially employed the "**playsound**" Python library[26], which offered excellent sound quality. However, it exhibited significant delays, rendering it unsuitable for our real-time system requirements. To overcome this, we opted for the "**pygame**" library[19], which offered a reduced delay (approximately 30% less) while maintaining satisfactory sound quality, thereby improving the overall experience.

Furthermore, during testing, we identified several delays within our system that impacted its responsiveness. To mitigate these delays, we implemented two optimizations. Firstly, we downscaled the resolution of the transmitted webcam images, reducing both network utilization and processing time associated with image handling. Secondly, we ensured that the audio data socket was closed when not in use, allowing it to reopen only when audio data was required. This improvment minimized processing requirements on the robot side, resulting in improved speed and reduced network traffic.

Lastly, to further enhance the responsiveness of the text-to-speech module, we shifted the speech generation process from the robot side to the laptop side, leveraging its

greater processing power. Subsequently, the generated speech is transmitted to the robot via ROS. This modification reduced the processing demands on the robot and consequently reduced overall system delays.

These improvements, alongside other refinements, played a crucial role in minimizing delays within our system. By achieving a more responsive and seamless experience, we aimed to enhance the real-time interaction between humans and robots in our Human-Robot Interaction (HRI) system.

3.1.12 System Performance Evaluation

After completing the design process, it was essential to evaluate the system's performance. This evaluation helped us identify strengths and areas for improvement and optimization.

There are two methods for us to do system evaluation: quantitative and qualitative. Quantitative evaluation focuses on numerical data and measurements, using statistical analysis to assess performance. Qualitative evaluation, on the other hand, gathers non-numerical data through observations and interviews to understand user experiences and perceptions. Considering the nature of our system, which involves human-robot interaction and requires a seamless and responsive experience, we determined that a qualitative approach was more suitable for evaluating its performance.

To evaluate the system, we conducted tests with 31 student volunteers from our university. They were asked to interact with Darwin and rate various parts of the system on a scale of 1 to 10, with 1 being very bad and 10 being very good. Those tests included the following:

1. Hey Darwin -> **Introduce** yourself.
2. Hey Darwin -> What is the **time**?
3. Hey Darwin -> What is today's **Date**?

4. Hey Darwin -> I have [random] **pain** / my [random] **hurts** me.
5. Sit with bad posture for ~20 seconds.
6. Hey Darwin -> Tell me a **joke**.
7. Hey Darwin -> **Google** something for me / **search** something for me.
8. Hey Darwin -> **Record** my voice / record a **note** for me.
9. Hey Darwin -> **Playback** the recording.
10. Hey Darwin -> Give me a summary from **Wikipedia**.

Based on their feedback and observations, we obtained the following evaluation results:

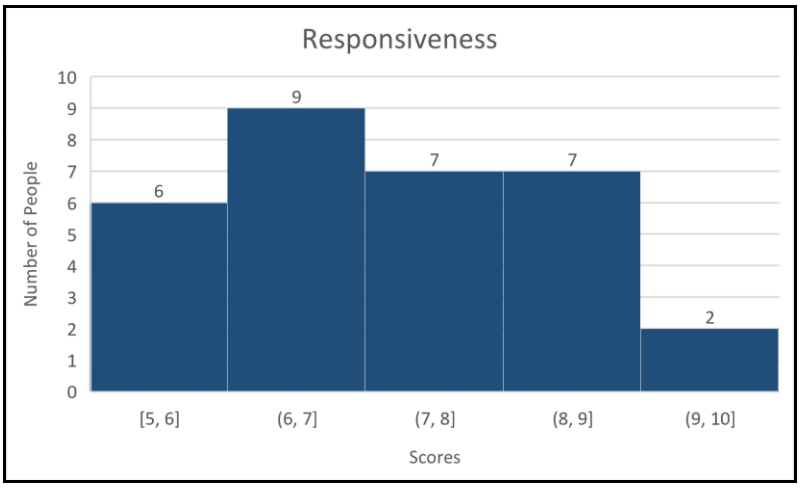


Figure 18: System responsiveness evaluation

The scores in Figure 18 represent the evaluation of system responsiveness. The variations in scores are attributed to the unpredictable evaluation environment, where challenges like poor internet connection occasionally affected the system's speed and responsiveness. Finding and maintaining the ideal evaluation environment is a complex task. While we acknowledge that our system is not flawless and requires further improvement in responsiveness, it remains an area of focus for future enhancements.

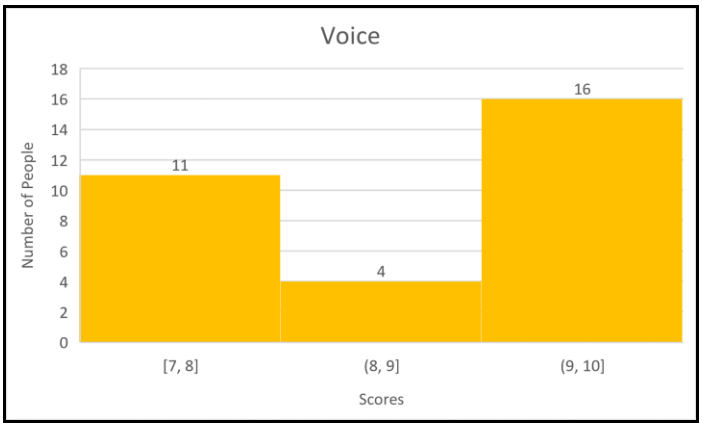


Figure 19: System voice evaluation

Figure 19 illustrates the evaluation scores for the system's voice. The results indicate that the voice used in our system is generally perceived as good, with the majority of users having no difficulty understanding it.

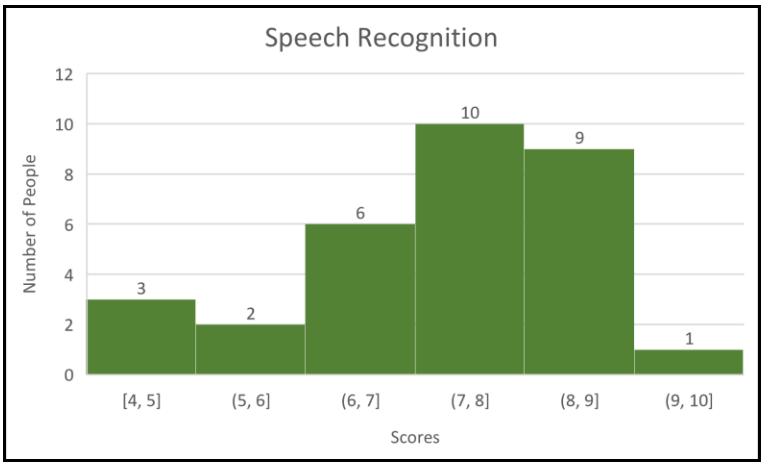


Figure 20: System speech recognition evaluation

Figure 20 presents the evaluation results for the speech recognition component of our system. The scores predominantly fall within the range of 7-8, although a few scores are as low as 5. This suggests that the speech recognition aspect of our system is moderately satisfactory but requires improvements to enhance its performance. We have plans to address these areas in the future, aiming to make the user experience with Darwin smoother and more seamless.

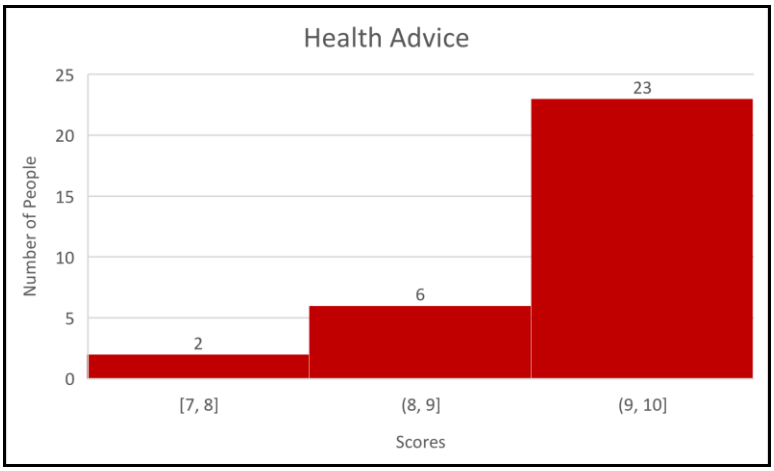


Figure 21: System health advice evaluation

In Figure 21, we observe the evaluation scores for the system's health advice. The results demonstrate that the advice provided by Darwin is consistently valuable and effectively alleviates user pain. This positive outcome is attributed to the implementation of Q-learning, which enables the selection of appropriate medical advice tailored to individual needs.

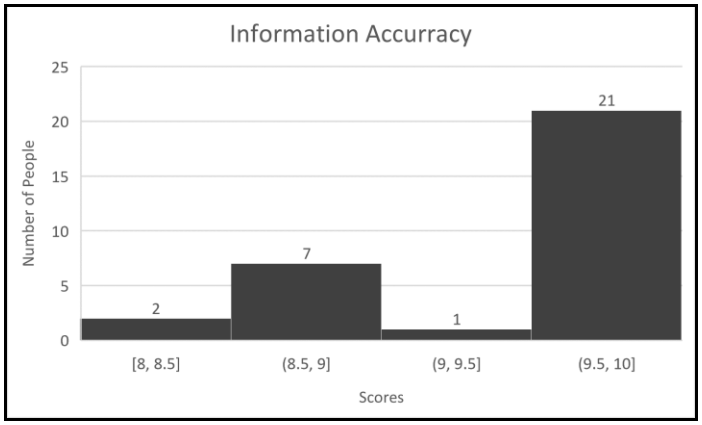


Figure 22: System information accuracy evaluation

Figure 22 displays the evaluation scores for the system's information accuracy. Overall, the results indicate a high level of accuracy, which can be attributed to the implementation of web scraping techniques. By extracting information from the internet, the system is able to provide reliable and precise answers to user questions.

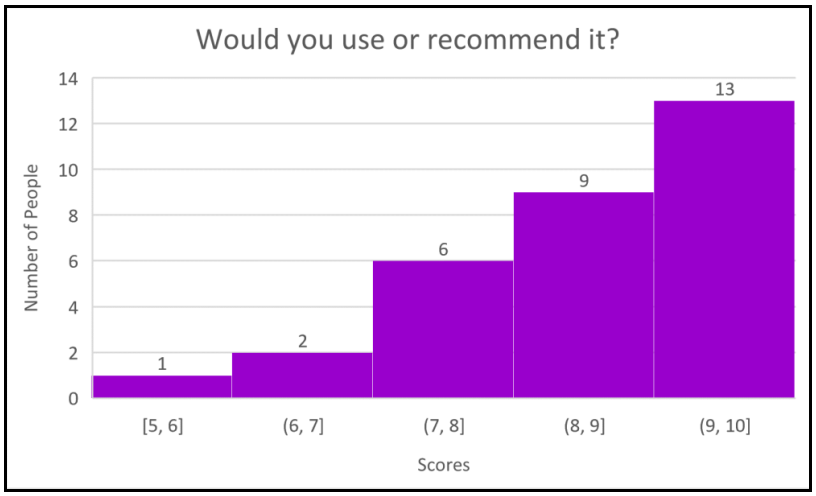


Figure 23: Users' opinion on Darwin Personal Assistant

Figure 23 showcases the scores for the final question in the system evaluation, which focused on users' willingness to use Darwin as a personal assistant in their daily lives and recommend it to others. The results indicate that users expressed a strong inclination towards using Darwin and recommending it to others. This signifies that our system holds great potential as a practical and versatile robot personal assistant, capable of fulfilling various functions.

Based on our system evaluation results, it is evident that two areas require further optimization and improvement: speech recognition and responsiveness. Our future plans involve dedicating efforts towards enhancing these aspects to achieve a more responsive and seamless system experience.

3.2 Engineering Standards

There were many engineering standards followed during this project, they were as follows:

Table 5: Design standards for this project.

IEEE 802.11	Wireless Networking – "Wi-Fi".
ISO 10218	Robots and Robotic Devices
IEEE 802.3	Ethernet
ISO 21500	Project Management
ISO 19794-5	A standard that specifies a framework for the representation of human facial features in data formats
IEEE P7017	Recommended Practice for Design-Centered Human-Robot Interaction (HRI) and Governance

3.3 Engineering Constraints

The design process of this project encountered significant engineering constraints primarily driven by the requirements of fast and reliable human-robot interaction. The following constraints emerged as crucial considerations:

3.3.1 Health and Safety constraints

During the course of this project, Darwin was programmed to provide medical advice related to various types of pain, such as neck pain, arm pain, leg pain, back pain, shoulder pain, and headaches. A critical engineering constraint we imposed was to

ensure that the advice provided by Darwin consisted solely of simple exercises that would not compromise the users' health. To gather the necessary advice, we conducted a field trip to Alhurriyah Physical Therapy Center and had the opportunity to meet with Dr. Hadeel Yousef[27], a knowledgeable physical therapist. During our visit, Dr. Hadeel guided us in understanding the appropriate exercises and medical advice to address each specific type of pain in our system. Figure 24 below shows Darwin in Alhurriyah Physical Therapy Center. We are immensely grateful for her expertise and contribution to our project. Additionally, in situations where Darwin lacked the necessary information to offer suitable advice, it was programmed to recommend users to seek medical attention and consult a doctor. This approach aimed to prioritize user safety and promote responsible healthcare decision-making.

Figure 24: Darwin in Alhurriyah Physical Therapy Center

3.3.2 Environmental Constraints

The development of our project, focused on the battery-powered humanoid robot Robotis OP2, necessitated careful consideration of environmental constraints. We

recognized the importance of proper battery disposal to minimize any adverse environmental effects. Throughout the project, we did not encounter the need for battery disposal. However, had it been necessary to dispose of or replace batteries, we would have prioritized safe and responsible practices to mitigate any potential environmental impact.

3.3.3 Technical: Manufacturability, Maintenance, Physical Constraints

During the design phase of this project, we encountered several physical constraints arising from the characteristics of the Darwin OP2 robot. These constraints posed unique challenges; they go as follows:

1. Balance: The top-heavy nature of the Darwin OP2 robot presented difficulties in achieving and maintaining balance. This posed challenges in creating and executing actions, as maintaining stability became a crucial consideration during the design process.

2. Limited Grip and Head Movements: One notable physical constraint of the Darwin OP2 robot was the absence of an arm grip and limited head movements. These limitations restricted the robot's ability to hold objects and perform certain tasks that required dexterous manipulation or a wider range of head motions.

3. Maintenance of Motors: The Darwin OP2 robot consists of multiple motors (actuators) that enable its movements. Like any mechanical component, these motors require regular maintenance to ensure optimal performance. Throughout the project, we encountered the need to perform maintenance on the motor located in the right elbow of the robot. This task posed significant challenges for us as we lacked experience in robot repairs, and the procedure was intricate, with delicate parts that risked breakage. Despite the complexity, we dedicated several

hours over two days to fix the motor successfully, restoring Darwin to its full functionality. We express sincere gratitude to our supervisors for their invaluable assistance during the maintenance process.

4. Battery Life: Darwin OP2 operates on a battery, allowing for a cable-free system experience. However, the battery capacity is limited to 1800mAh, resulting in a gradual depletion of power during system operation. In our tests, we observed an average battery life of approximately 1 hour. Although this is not optimal, it should be noted that the battery used during testing had already been in use for an extended period. With a new battery, we anticipate an increase in system runtime before requiring recharging. Furthermore, since our system is designed to be stationed at a desk, users should not encounter difficulty in conveniently plugging in a charger when needed.

Addressing these physical constraints necessitated innovative approaches and adaptations in the design process. By recognizing and working within these limitations, we were able to optimize the robot's functionalities while acknowledging the maintenance requirements and inherent physical characteristics of the Darwin OP2 robot.

3.3.4 Economic constraints

The completion of the project incurred no financial costs thanks to the generous provision of the Robotis OP2 humanoid robot by our project supervisor at no charge. Additionally, all software requirements were fulfilled using freely available resources and open-source code. The team utilized personal laptops for data processing, further minimizing expenses. As a result, this project was successfully accomplished without any financial burden.

3.3.5 Ethics Constraints

In the development of our personal assistant robot, the acquisition and utilization of environmental data, including voice data and webcam images, were essential. However, safeguarding user privacy remains our utmost ethical priority. We implemented rigorous measures to guarantee that no personal data was retained or stored. Once the data fulfilled its intended purpose and became unnecessary, we promptly deleted it to maintain user privacy. Throughout the design and implementation process, we remained acutely aware of ethical considerations and potential dilemmas. Our actions consistently adhered to established ethical standards and demonstrated deep respect for user privacy.

3.3.6 Time Constraints

The project was organized into two distinct phases, each spanning 12 weeks, imposing significant time constraints on our team. Within these time frames, we engaged in brainstorming sessions, conducted regular meetings, developed and refined design concepts, and translated them into functional code. Testing the system, troubleshooting issues, and finding creative solutions for challenges were integral aspects of our time-bound project.

Custom action creation proved to be particularly time-consuming, as it required meticulous attention to detail to ensure precision while safeguarding user and robot safety. We devoted considerable effort to create actions that were both effective and harmless.

Additionally, we considered the possibility of incorporating Arabic language support alongside English language in our system. This would have made Darwin even more versatile and accommodating to a broader range of users, enhancing its overall usefulness. However, due to time limitations, we were unable to pursue this idea and implement Arabic language functionality.

While time posed a challenge, our team remained focused and dedicated to achieving our project goals. Through effective time management and prioritization, we were able to capitalize on the available time and accomplish significant milestones.

3.4 Alternative Designs

During the design process, we evaluated the use of ROS2 as an alternative to ROS. ROS2 offers several benefits such as a modular and distributed architecture, support for a wider range of Quality of Service (QoS) policies, and improved performance and scalability. However, we ultimately decided to use ROS as it has a more established community and a larger number of online resources and tutorials. We chose to use ROS Noetic Ninjemys as it is the recommended active ROS distribution and we installed it on Ubuntu 20.04 LTS, which is the primary target platform for ROS Noetic Ninjemys.

Another alternative we considered was using Robotis OP3 instead of Robotis OP2. Robotis OP3 is a more advanced and capable humanoid robot, with a smaller and lighter body, more degrees of freedom for natural and human-like movements, a more powerful processor and more memory, and a higher-resolution camera and advanced sensor suite. While we were tempted to utilize Robotis OP3 for its superior capabilities, we ultimately decided to use Robotis OP2 due to cost constraints. Our supervisor was able to provide us with Robotis OP2 at no cost, whereas using Robotis OP3 would have significantly increased the budget for our project.

Additionally, we considered using C++ instead of Python for programming tasks. C++ is generally faster and more efficient than Python but has a more complex syntax. Python, on the other hand, has a simpler syntax and a larger and more active community of users and developers. Both languages can be used effectively for computer vision tasks. We ultimately chose to use Python due to its easier syntax, the availability of a large community for troubleshooting and access to open-source code, and our desire to gain more familiarity with Python as we already have a strong foundation in C++.

Moreover, we initially utilized the "**pyttsx3**" Python library for text-to-speech generation[25]. However, we encountered issues with the clarity and comprehension of the English words produced. To address this, we explored alternative options and discovered the "**gTTS**" library[3], which not only provided clearer speech but also better suited the overall appearance of our system's implementation on Darwin. Importantly, this switch to "**gTTS**" did not introduce noticeable delays or additional processing time, ensuring a seamless user experience.

Furthermore, for playing the generated speech, we initially employed the "**playsound**" Python library[26], which offered excellent sound quality. However, we found significant delays associated with this library, making it unsuitable for our real-time system requirements. To overcome this challenge, we made the decision to switch to the "**pygame**" library[19]. This alternative reduced the delays by approximately 30% while maintaining satisfactory sound quality, thereby significantly improving the overall experience for our users.

CHAPTER 4: ENOCOMIC, ETHICAL, AND CONTEMPORARY ISSUES

4.1 Preliminary Cost Estimation and Justification

The project was completed at no financial cost due to the availability of the Robotis OP2 humanoid robot, which was generously provided by our project supervisor at no charge. In addition, all software requirements were satisfied through the utilization of free resources and open-source code. The team utilized personal laptops for data processing. This allowed us to complete this project without incurring any financial costs.

However, if someone else needs to implement the system, they should consider the following costs:

1. The cost of acquiring a Darwin OP2 robot, which was priced at $9600 (Please note that Darwin OP2 is discontinued, so availability and pricing may vary).

2. The cost of a processing station for this project, which is estimated to be around $500. Alternatively, they can opt for cloud computing rental, the cost of which can vary depending on the specific requirements and duration of usage.

4.2 Relevant Codes of Ethics and Moral Frameworks

Two relevant codes of ethics to consider when working on a project involving the collection and handling of personal data are:

1. The International Data Privacy Principles (IDPPs), which require that personal data be collected only when necessary and not kept for longer than necessary[28].

2. The General Data Protection Regulation (GDPR), which mandates that personal data be collected and processed fairly and lawfully, and that it be kept secure and confidential[29].

Adhering to these codes of ethics can help ensure that personal data is handled responsibly and with respect for the privacy and rights of individuals.

4.3 Ethical Dilemmas and Justification of Proposed Solution

During the development of our personal assistant robot, capturing and utilizing data from the environment, such as voice data and webcam images, was necessary. However, to uphold user privacy as a top priority, we implemented strict measures to ensure that no personal data was saved or stored.

Once the data served its purpose and was no longer required, we promptly deleted it. Throughout the design and implementation process, we took great care to address any ethical considerations or dilemmas that could arise, ensuring that our actions were in line with ethical standards and respected user privacy.

4.4 Relevant Environmental Considerations

As a battery-powered humanoid robot, the Robotis OP2 raises several environmental considerations during the development of our project. Specifically, the proper disposal of used batteries is crucial to mitigate any negative impacts on the environment. During this project, no batteries were disposed of. However, in the event that battery disposal or replacement was necessary, we would have taken great care to ensure that it is done safely and responsibly.

4.5 Security Considerations

To ensure the security of our system and protect it from unwanted access, we implemented a security measure using face recognition, allowing only authorized users to access the system. While this is a good step towards security, it may not be sufficient. Since our system involves data communication between the robot and the processing computer/laptop, it becomes vulnerable to various cyber-physical attacks, which we must address and safeguard against.

To assess the security of our system, we collaborated with our colleagues under the supervision of Dr. Anas Al Majali, who worked on a project titled "Security Assessment and Penetration Testing Using Reinforcement Learning"[30]. Using their system, we conducted a security assessment primarily focusing on the robot side for two reasons:

1. The robot side is more critical, as cyber-physical attacks can have a greater impact on the robot and its overall security is of utmost importance.

2. The robot system operates on older software, specifically Linux Mint 17.3 (Rosa) and ROS Indigo Igloo[14], [31], both of which are considered outdated and more likely to have security flaws.

As a result of the assessment, we identified three common vulnerabilities and exposures (CVE) in our robot system. These vulnerabilities include CVE-2022-48198 (ROS)[32], CVE-2017-5638 (Linux Mint 17.3)[33], and CVE-1999-0524 (Linux Mint 17.3)[34]. Details of these CVEs are provided in Table 6 below:

Table 6: Security Vulnerabilities in Darwin OP2 System

Vulnerability	Description	Impact (CVSS)*
CVE-2017-5638	allows remote attackers to execute arbitrary commands via a crafted Content-Type, Content-Disposition, or Content-Length HTTP header.	10.0
CVE-2022-48198	allows attackers, who control the source code of a different node in the same ROS application, to change a robot's behavior.	9.8
CVE-1999-0524	ICMP information such as (1) netmask and (2) timestamp is allowed from arbitrary hosts.	N/A

*Common Vulnerability Scoring System (CVSS) is a method used to supply a qualitative measure of severity for security vulnerabilities.

Although we were unable to address the security issues in our system due to time constraints, we acknowledge their significance and importance. We plan to address these issues in the future and implement appropriate measures to enhance the security of our system.

CHAPTER 5: RESULTS AND DISCUSSION

After completing the design process, we conducted optimizations and evaluated the performance of our system. The system we developed is a robot personal assistant using the Robotis Darwin OP2 robot. It is designed to assist individuals who spend extended periods at their desks for studying or working.

Our personal assistant offers various functions, including web searching, voice-note taking and playback, providing time and date information, and offering medical advice for specific discomforts or pains. Additionally, it can retrieve topic summaries from Wikipedia, share jokes for entertainment, and provide up-to-date weather information. The flowchart for our whole system is show in Figure 25 below:

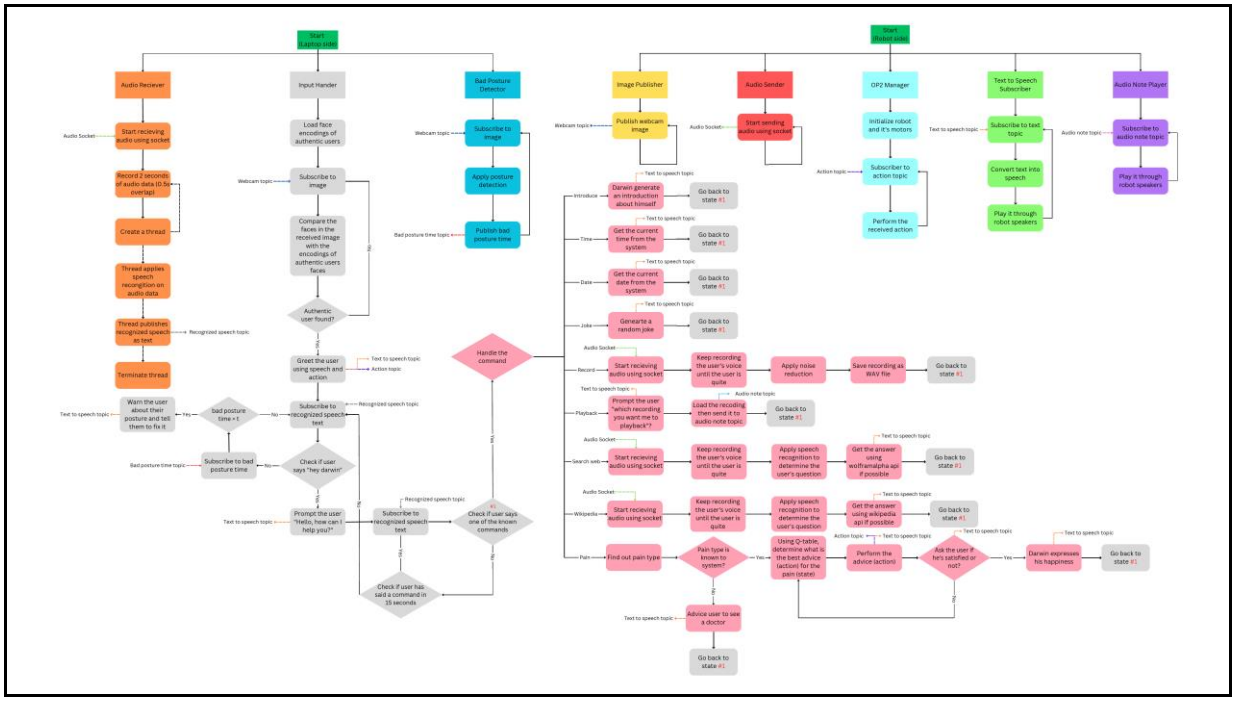


Figure 25: Whole system flowchart

Through the system evaluation, we determined that our personal assistant meets the requirements and expectations of users, making it a viable option for those who need

assistance in their work or study sessions, for more information about the system evaluation results, please refer to section (3.1.12).

Its physical presence as a robot personal assistant sets it apart from virtual assistants, providing a unique and interactive user experience. The graph presented in Figure 26 illustrates individuals' responses to the question, "Would you utilize Darwin as your personal assistant or recommend it to a friend or colleague?"

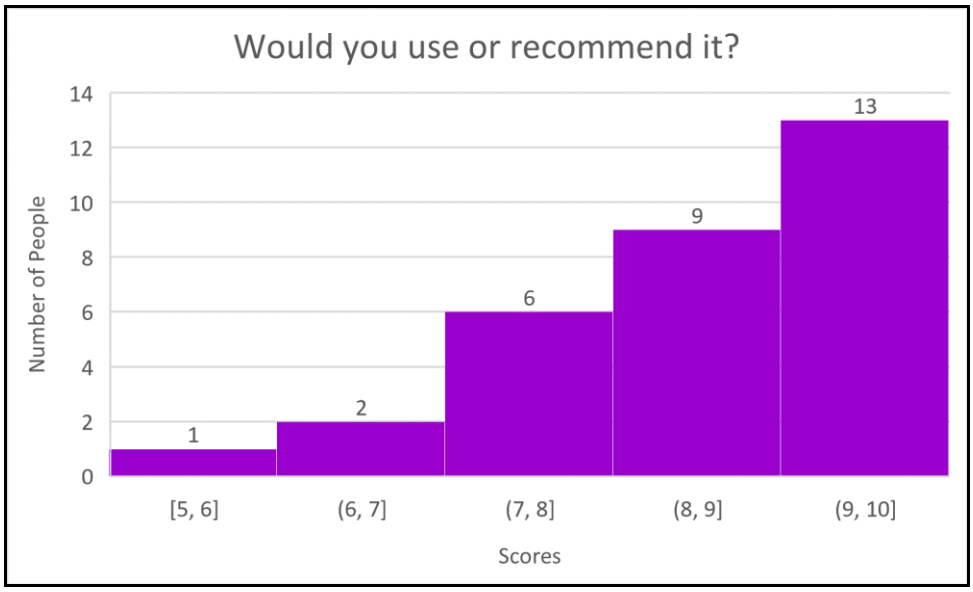


Figure 26: Users' opinion on Darwin Personal Assistant

Figure 26 showcases the scores for the final question in the system evaluation, which focused on users' willingness to use Darwin as a personal assistant in their daily lives and recommend it to others. The results indicate that users expressed a strong inclination towards using Darwin and recommending it to others. This signifies that our system holds great potential as a practical and versatile robot personal assistant, capable of fulfilling various functions.

CHAPTER 6: CONCLUSION AND FUTURE WORK

6.1 Conclusion

We have explored various aspects of a personal assistant system using the Robotis Darwin OP2 robot. The system incorporates features such as speech recognition, face recognition, posture detection, command handling, text-to-speech functionality, and iterative evaluations for performance optimization.

The implementation of speech recognition has greatly enhanced the interaction between the user and the robot, enabling efficient processing and analysis of user input. Face recognition serves as a crucial security measure, ensuring that only authenticated users are recognized. The posture detection feature provides timely warnings and advice to promote better posture habits and reduce discomfort associated with prolonged desk usage.

Command handling allows the robot to perform various functions based on recognized commands, such as self-introduction, time/date information, pain handling, web scraping, Wikipedia summaries, audio note recording/playback, and joke-telling. This comprehensive range of functionalities ensures a seamless and tailored user experience.

The system underwent iterative evaluations, leading to optimizations and improvements in its performance. Qualitative evaluation helped identify strengths and areas for further enhancement, particularly in speech recognition and responsiveness, which will be the focus of future efforts.

Overall, the personal assistant system using the Robotis Darwin OP2 robot demonstrates great potential for assisting individuals in their work or study environments. It combines

physical interaction, advanced functionalities, and user-oriented design to provide a valuable and engaging experience.

Acknowledging the significance of version control and code backup, we made the deliberate choice to upload our package to GitHub[17]. This decision served dual purposes: firstly, it guaranteed the secure storage and convenient access of our code for future revisions; secondly, it aimed to establish a valuable reference for fellow individuals embarking on similar projects, enabling them to leverage and build upon our work effectively. By sharing our code on GitHub, we aimed to foster collaboration and knowledge exchange within the community, ultimately contributing to the collective advancement of such endeavors. please refer to Appendix B for the GitHub project link.

6.2 Future Work

In our future endeavors, we have outlined several goals aimed at enhancing the capabilities and user experience of the system. These goals encompass the addition of new features, further reduction of delays, and implementation of enhanced security measures. The following objectives have been identified for future work:

1. Task scheduling and organization: One useful feature we could add to the functions of Darwin is to help the user manage their schedule using reminders. Such functionality could be very useful for students and people who have desk jobs.

2. Body language and custom actions: Expanding the range of body language and custom actions performed by the robot will enrich its interactive capabilities, allowing for more expressive and personalized interactions with users.

3. Enhanced security measures: To bolster the security of the system, we plan to implement advanced measures such as encrypting all network traffic between the robot and the laptop, ensuring the privacy and integrity of user data.

4. Robot to medical facility communication: To further ensure users' well-being, we plan to make Darwin report back to the user's doctor or medical facility. This could help the users to undergo medical treatment in case of need.

5. Using Deep Q-Learning (DQN) instead of Q-Learning: we plan to add even more states and actions, which would increase the complexity of the system. To make sure Darwin still can learn what actions to take, we plan to use DQN as it's suitable for complex systems.

6. Exploring design alternatives: We will explore alternative design approaches to introduce additional functionality or optimize existing features, seeking innovative solutions that enhance the overall system performance and user experience.

7. Overall optimizations: Ongoing optimizations will be undertaken to make the robot more responsive, ensuring quicker and smoother interactions between the user and the system.

By pursuing these future goals, we aim to continually improve the system, providing users with an enhanced and secure experience while further expanding the range of functionalities and optimizing performance.

REFERENCES

[1] "Documentation - ROS Wiki." https://wiki.ros.org/ (accessed Dec. 25, 2022).

[2] A. Zhang (Uberi), "SpeechRecognition: Library for performing speech recognition, with support for several engines and APIs, online and offline." Accessed: May 20, 2023. [MacOS :: MacOS X, Microsoft :: Windows, Other OS, POSIX :: Linux]. Available: https://github.com/Uberi/speech_recognition#readme

[3] P. N. Durette, "gTTS: gTTS (Google Text-to-Speech), a Python library and CLI tool to interface with Google Translate text-to-speech API."

[4] "Gym Documentation." https://www.gymlibrary.dev/ (accessed May 20, 2023).

[5] "Hands," *mediapipe*. https://google.github.io/mediapipe/solutions/hands.html (accessed Dec. 26, 2022).

[6] "Home," *OpenCV*. https://opencv.org/ (accessed Dec. 25, 2022).

[7] A. Geitgey, "face-recognition: Recognize faces from Python or from the command line." Accessed: May 20, 2023. [Online]. Available: https://github.com/ageitgey/face_recognition

[8] A. Mishra, P. Makula, A. Kumar, K. Karan, and V. K. Mittal, "A voice-controlled personal assistant robot," in *2015 International Conference on Industrial Instrumentation and Control (ICIC)*, May 2015, pp. 523–528. doi: 10.1109/IIC.2015.7150798.

[9] A. I. Alexan, A. R. Osan, and S. Oniga, "Personal assistant robot," in *2012 IEEE 18th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, Oct. 2012, pp. 69–72. doi: 10.1109/SIITME.2012.6384348.

[10] J. Mišeikis *et al.*, "Lio-a personal robot assistant for human-robot interaction and care applications," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5339–5346, 2020.

[11] K. J. Morris, V. Samonin, J. Baltes, J. Anderson, and M. C. Lau, "A robust interactive entertainment robot for robot magic performances," *Applied Intelligence*, vol. 49, no. 11, pp. 3834–3844, 2019.

[12] L. Gavrilova, V. Petrov, A. Kotik, A. Sagitov, L. Khalitova, and T. Tsoy, "Pilot study of teaching English language for preschool children with a small-size humanoid robot assistant," presented at the 2019 12th International Conference on Developments in eSystems Engineering (DeSE), IEEE, 2019, pp. 253–260.

[13] "Lubuntu 12.04 is now available," *lubuntu*, Apr. 26, 2012. https://lubuntu.net/lubuntu-1204-now-available/ (accessed May 27, 2023).

[14] "Linux Mint 17.3 'Rosa' - Linux Mint." https://linuxmint.com/edition.php?id=204 (accessed May 27, 2023).

[15] "Home - UltraVNC VNC OFFICIAL SITE, Remote Desktop Free Opensource." https://uvnc.com/ (accessed Dec. 25, 2022).

[16] “ROS/Tutorials - ROS Wiki.” https://wiki.ros.org/ROS/Tutorials (accessed Dec. 25, 2022).
[17] PersonalAssistantGradProject, “robot_personal_assistant_op2.” May 27, 2023. Accessed: May 27, 2023. [Online]. Available: https://github.com/PersonalAssistantGradProject/robot_personal_assistant_op2
[18] Y. Name, “ROBOTIS e-Manual,” *ROBOTIS e-Manual*. https://emanual.robotis.com/docs/en/platform/op2/ros_package/ (accessed Dec. 25, 2022).
[19] “pygame.” https://www.pygame.org/news (accessed May 20, 2023).
[20] “Body Posture Detection & Analysis System using MediaPipe.” https://learnopencv.com/building-a-body-posture-analysis-system-using-mediapipe/ (accessed May 20, 2023).
[21] “Home,” *mediapipe*. https://google.github.io/mediapipe/ (accessed May 27, 2023).
[22] “Pose landmark detection guide | MediaPipe,” *Google for Developers*. https://developers.google.com/mediapipe/solutions/vision/pose_landmarker (accessed May 27, 2023).
[23] “Wolfram|Alpha: Making the world’s knowledge computable.” https://www.wolframalpha.com (accessed May 20, 2023).
[24] J. Goldsmith, “wikipedia: Wikipedia API for Python.” Accessed: May 20, 2023. [Online]. Available: https://github.com/goldsmith/Wikipedia
[25] N. M. Bhat, “pyttsx3: Text to Speech (TTS) library for Python 2 and 3. Works without internet connection or delay. Supports multiple TTS engines, including Sapi5, nsss, and espeak.” Accessed: May 20, 2023. [MacOS :: MacOS X, Microsoft :: Windows, POSIX]. Available: https://github.com/nateshmbhat/pyttsx3
[26] T. Marks, “playsound: Pure Python, cross platform, single function module with no dependencies for playing sounds.” Accessed: May 20, 2023. [OS Independent]. Available: https://github.com/TaylorSMarks/playsound
[27] “Google Maps,” *Google Maps*. https://www.google.com/maps/place/%D9%85%D8%B1%D9%83%D8%B2+%D8%A7%D9%84%D8%AD%D8%B1%D9%8A%D8%A9+%D9%84%D9%84%D8%B9%D9%84%D8%A7%D8%AC+%D8%A7%D9%84%D8%B7%D8%A8%D9%8A%D8%B9%D9%8A%E2%80%AD/@31.8973611,35.9295371,15z/data=!4m6!3m5!1s0x151b5955908f8b4f:0x8b7d0f2dd03988db!8m2!3d31.8977991!4d35.9226004!16s%2Fg%2F11hytht7hx?entry=ttu (accessed May 27, 2023).
[28] W. Zankl, “The International Data Privacy Principles.” Harvard University, Cambridge, Massachusetts, USA, Oct. 2014. [Online]. Available: http://www.e-center.eu/static/files/moscow-dataprivacy-handout-russian.pdf
[29] “General Data Protection Regulation (GDPR) – Official Legal Text,” *General Data Protection Regulation (GDPR)*. https://gdpr-info.eu/ (accessed Jan. 08, 2023).
[30] A. Almajali *et al.*, “Vulnerability Exploitation Using Reinforcement Learning,” May 2023.
[31] “indigo - ROS Wiki.” http://wiki.ros.org/indigo (accessed May 27, 2023).

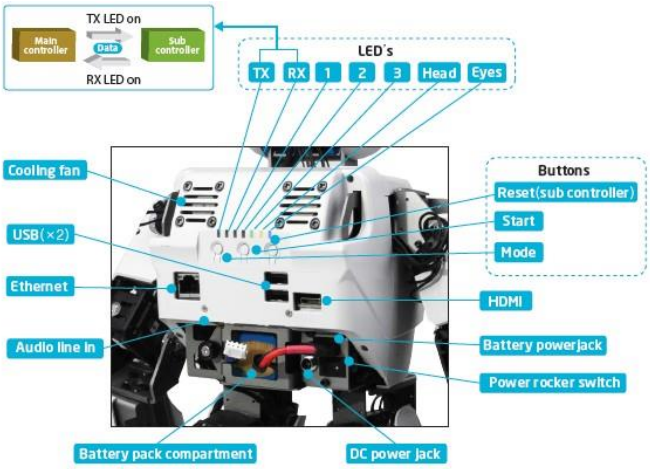
[32] “NVD - CVE-2022-48198.” https://nvd.nist.gov/vuln/detail/CVE-2022-48198 (accessed May 27, 2023).
[33] “NVD - CVE-2017-5638.” https://nvd.nist.gov/vuln/detail/CVE-2017-5638 (accessed May 27, 2023).
[34] “NVD - CVE-1999-0524.” https://nvd.nist.gov/vuln/detail/CVE-1999-0524 (accessed May 27, 2023).
[35] Y. Name, “Introduction,” *ROBOTIS e-Manual*. https://emanual.robotis.com/docs/en/platform/op2/getting_started/ (accessed Jan. 06, 2023).

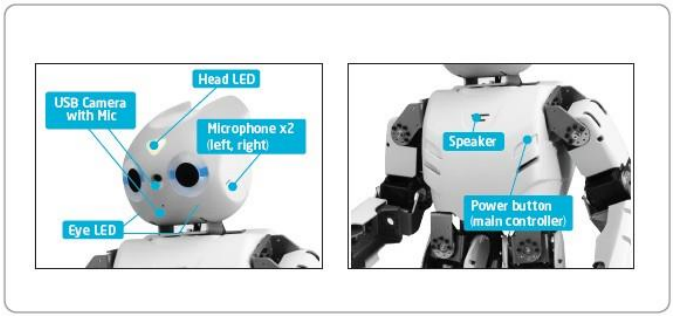
APPENDICES

Appendix A: DARwin OP2 Specifications

Here are the hardware specifications and the layout for Robotis DARwin OP2[35].

	DARWIN OP	ROBOTIS OP2
CPU	Intel Atom Z530 @1.6GHz single core	Intel Atom N2600 @1.6GHz dual core
RAM	1GB DDR2 (fi xed capacity)	up to 4GB DDR3 204-pin SO-DIMM module (user-replaceable)
Storage	4GB NAND flash IDE100 (fixed capacity)	half-size mSATA module (32GB) (user-replaceable)
LAN speed	100 Mbps	1 Gbps
Installable OS	Linux only (32-bit)	any Linux release (32-bit) any Windows release (32-bit)
wi-fi	802.11g	802.11n (2.4GHz-only)





Appendix B: GitHub Project

Link for our GitHub project repository :

https://github.com/PersonalAssistantGradProject/robot_personal_assistant_op2

Appendix C: Codes Used

word_finder.py

```
#!/usr/bin/env python

import rospy
import time
from std_msgs.msg import String,Int32
import random
import text_to_speech_publisher # text_to_speech_publisher.py

def init():
    global start_publisher
    global finished_publisher
    start_publisher =
rospy.Publisher('/start_recognition',String,queue_size=10)
    finished_publisher =
rospy.Publisher('/finish_recognition',String,queue_size=10)



def check_words(list_of_words):

    global start_publisher
    global finished_publisher
    start_publisher.publish("start")
    # subscribe to the rostopic

    rate = rospy.Rate(1)# 1Hz
    transcript = None


    def callback(data):
        nonlocal transcript
        # convert the recieved image into suitable format using CvBridge
```

```
        transcript = data.data

    past_transcript = ""
    past_past_transcript = ""

    # define subscriber on ROS topic '/speech_recognition_output' with
tts transcript
    rospy.Subscriber('/speech_recognition_output', String, callback)

    pain_type_found = ""









    if (list_of_words[0] == "darwin"):


        def callback1(data):
            nonlocal bad_posture_time
            # convert the recieved image into suitable format using
CvBridge
            bad_posture_time = data.data

        rospy.Subscriber('/bad_posture_time', Int32, callback1)


        bad_posture_time = None
        count = 10
        start_time = time.time()
        timeout = 30 # 30 seconds

        while not rospy.is_shutdown():



            if transcript is not None:
```

```
                print("- User said:",transcript)
                transcript_lower = transcript.lower()
                for word in list_of_words:
                    if word in transcript_lower:
                        finished_publisher.publish("finished")
                        return word, pain_type_found
                transcript = None

            if bad_posture_time is not None:

                if (bad_posture_time > 10):
                    if (count == 0):

                        rand_text = random.randint(0, 2)
                        if (rand_text == 0):
                            advice = ("I can see your posture needs a
little adjustment. Remember to ensure proper back "
                                      "support and consider changing
your position regularly.")
                        elif (rand_text == 1):
                            advice = ("I couldn't help but notice your
posture. It might be helpful to adjust how you sit, "
                                      "raise your shoulders, and ensure
proper back support.")
                        elif (rand_text == 2):
                            advice = ("Hey, I noticed your posture needs
attention. Try adjusting how you sit, raising your "
                                      "shoulders, and changing positions
every 30 minutes for better alignment.")
                        text_to_speech_publisher.publish_text(advice)
                        count = (count + 1) % 20
                    else:
                        count = (count + 1) % 20

            current_time = time.time()
            elapsed_time = current_time - start_time
            if elapsed_time >= timeout:
                word = "hey_darwin_timeout"
                return word, pain_type_found
            rate.sleep()
```

```
    elif (list_of_words[0] == "pain"):
        pain_types = ["back",
                      "neck",
                      "leg","foot","feet","knee",
                      "arm","wrist","hand",
                      "shoulder",
                      "head"]
        start_time = time.time()
        timeout = 15  # 15 seconds

        while not rospy.is_shutdown():
            if transcript is not None:


                print("- User said:",transcript)
                transcript_lower = transcript.lower()
                for word in list_of_words:
                    if word in transcript_lower:
                        if (word == "pain" or word == "hurt"):
                            for pain_type in pain_types:
                                if pain_type in transcript_lower +
past_transcript.lower() + past_past_transcript.lower():
                                    pain_type_found = pain_type

finished_publisher.publish("finished")
                                    return word, pain_type_found
                        finished_publisher.publish("finished")
                        return word, pain_type_found
                past_past_transcript = past_transcript
                past_transcript = transcript
                transcript = None


            current_time = time.time()
            elapsed_time = current_time - start_time
            if elapsed_time >= timeout:
                word = "command_timeout"
                return word, pain_type_found


    else:
```

```
        while not rospy.is_shutdown():

            if transcript is not None:

                print("- User said:",transcript)
                transcript_lower = transcript.lower()
                for word in list_of_words:
                    if word in transcript_lower:
                        finished_publisher.publish("finished")
                        return word, pain_type_found
                transcript = None
```

action_sender.py

```
#!/usr/bin/env python



import rospy
from std_msgs.msg import Int32

def init():
    global action_publisher
    action_publisher = rospy.Publisher('/robotis/action/page_num',
Int32, queue_size=10)
    return




def publish_action(action):
    global action_publisher
    rate = rospy.Rate(10)
```

```
    print("- Robot perfromed page num:", action)
    action_publisher.publish(action)
    rate.sleep()
    return
```

audio_receiver_speech_recognizer.py

```
#!/usr/bin/env python
import socket
import time
import speech_recognition as sr
import threading
import rospy
from std_msgs.msg import String






def create_thread(data_buffer):
    #print(f"Thread {threading.current_thread().name} is running, data
size is: {len(data_buffer)}")
    # Convert the accumulated data to an AudioData object
    audio_data = sr.AudioData(data_buffer, sample_rate=SAMPLE_RATE,
sample_width=SAMPLE_WIDTH)
    # Transcribe the audio data
    try:
        transcript = r.recognize_google(audio_data)
        text_publisher.publish(transcript)

    except sr.UnknownValueError:
        print("Could not understand audio")
```

```
    except sr.RequestError as e:
        print(f"Could not request results from Google Speech Recognition
service; {e}")




text_publisher =
rospy.Publisher('/speech_recognition_output',String,queue_size=1)
rospy.init_node('audio_reciever_speech_recognition',anonymous=True)

# Initialize the recognizer
r = sr.Recognizer()

# Set up a socket to receive audio data
HOST = ''  # Listen on all available interfaces
PORT = 5000  # Use a free port number
CHUNK = 1024  # Number of bytes to receive at a time
SAMPLE_RATE = 44100  # Sample rate of the audio data
SAMPLE_WIDTH = 2  # Sample width of the audio data
RECORD_SECONDS = 1.5  # Duration of audio to accumulate before
processing
OVERLAP_SECONDS = 0.5  # Duration of overlap between consecutive audio
segments


while not rospy.is_shutdown():


    start = None
    def callback(data):
        global start
        #nonlocal finished
        start = data.data

    rospy.Subscriber('/start_recognition', String, callback)
    while not rospy.is_shutdown():
        if (start is not None):
            break
```

```
    count = 0
    audio_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    audio_socket.bind((HOST, PORT))
    audio_socket.listen()
    print(f"Listening for audio data on {HOST}:{PORT}...")
    conn, addr = audio_socket.accept()


    with conn:
        print(f"Connected by {addr}")
        data_buffer = b''  # Initialize an empty buffer to store
received data
        start_time = time.time()  # Record the start time of the
recording
        end_time = start_time + RECORD_SECONDS
        overlap_buffer = None

        finished = None
        def callback_finished(data):
            global finished
            #nonlocal finished
            finished = data.data

        rospy.Subscriber('/finish_recognition', String,
callback_finished)

        while not rospy.is_shutdown():

            if (finished is not None):
                audio_socket.close()
                break

            # Receive audio data from the client
            data = conn.recv(CHUNK)
            if not data:
                break

            # Accumulate received data in the buffer
            data_buffer += data
```

```
            # Check if the accumulated data is greater than or equal to
RECORD_SECONDS
            elapsed_time = time.time() - start_time

            if elapsed_time >= RECORD_SECONDS + OVERLAP_SECONDS:


                if overlap_buffer is not None:
                    data_buffer = overlap_buffer + data_buffer
                    overlap_buffer = None



                if (count % 3 == 0):
                    thread0 =
threading.Thread(target=create_thread,args=(data_buffer,), name="Thread
0")
                    thread0.start()
                if (count % 3 == 1):
                    thread1 =
threading.Thread(target=create_thread,args=(data_buffer,), name="Thread
1")
                    thread1.start()
                if (count % 3 == 2):
                    thread2 =
threading.Thread(target=create_thread,args=(data_buffer,), name="Thread
2")
                    thread2.start()
                count = (count + 1) % 3

                # Update the buffer and start and end times for the next
recording
                overlap_seconds = RECORD_SECONDS * 0.15  # Calculate the
overlap duration
                overlap_buffer = data_buffer[-
int(overlap_seconds*SAMPLE_RATE*SAMPLE_WIDTH):]
                data_buffer = data_buffer[-int((RECORD_SECONDS-
overlap_seconds)*SAMPLE_RATE*SAMPLE_WIDTH):]
                start_time = end_time - overlap_seconds
                end_time += RECORD_SECONDS - overlap_seconds
```

```
    # Close the existing socket
    audio_socket.close()
```

text_to_speech_publisher.py

```
#!/usr/bin/env python

import rospy
from std_msgs.msg import String
from playsound import playsound
import os

def audio_callback(data):
    audio_data_str = data.data
    # Encode the audio data as bytes
    audio_data = audio_data_str.encode('latin-1')

    audio_file_path =
os.path.expanduser("~/op2_tmp/recordings/received_note_audio.wav")

    with open(audio_file_path, 'wb') as f:
        f.write(audio_data)

    # Play the audio using the playsound library
    playsound(audio_file_path)
    global finished_talking_publisher
    finished_talking_publisher.publish("finished!")
    rospy.loginfo("finished!")


def audio_subscriber():
    rospy.init_node('audio_note_subscriber', anonymous=True)
```

```
    rospy.Subscriber('/audio_topic', String, audio_callback)
    rospy.spin()

if __name__ == '__main__':
    try:
        finished_talking_publisher =
rospy.Publisher('/finished_talking', String, queue_size=10)
        audio_subscriber()
    except rospy.ROSInterruptException:
        pass
```

text_to_speech_subscriber.py

```
#!/usr/bin/env python


"""


This Python file creates a ROS node called "text_to_speech" which
subscribes
to the "/tts" ROS topic.

Whenever a string message is received on this topic, the node utilizes
gTTS
to convert the text to speech.

The resulting audio is then played through the robot's speakers, giving
the
impression that the robot is speaking the text.


"""
```

```
# imported libraries
from gtts import gTTS
import rospy
from std_msgs.msg import String
import os
import base64
import tempfile
import pygame



# callback function called when string data is recieved on '/tts'
def callback(data):
    # Decode the base64-encoded audio data
    audio_data = base64.b64decode(data.data)

    # Save the audio data to a temporary file
    temp_file = tempfile.NamedTemporaryFile(delete=False)
    temp_file.write(audio_data)
    temp_file.close()

    # Play the audio file
    pygame.mixer.music.load(temp_file.name)
    pygame.mixer.music.play()
    while pygame.mixer.music.get_busy():
        pass




    global finished_talking_publisher
    finished_talking_publisher.publish("finished!")
    rospy.loginfo("finished!")

    # Remove the temporary file
    os.remove(temp_file.name)



# The "speak" function accepts a string parameter named 'text'.
```

```
#
# It uses the gTTS library to convert the input text into speech.
#
# The generated audio is saved as an mp3 file, and then played through
the robot's speaker.




# spin() simply keeps python from exiting until this node is stopped
def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    pygame.init()
    rospy.init_node('text_to_speech', anonymous=True)
    rospy.Subscriber('/tts', String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()


if __name__ == '__main__':
    finished_talking_publisher = rospy.Publisher('/finished_talking',
String, queue_size=10)
    listener()
```

pain_handler.py

```
#!/usr/bin/env python
"""


This Python file contains a function named "process_state" which is
invoked by "input_handler.py". The purpose of this function is to
receive the user's state as input, and then instruct the robot to
```

```
execute the appropriate actions based on the state provided.


"""



import numpy as np
import rospkg
import os
from std_msgs.msg import Int32
import rospy
import random
import time
import text_to_speech_publisher # text_to_speech_publisher.py
import action_sender # action_sender.py
import word_finder # word_finder.py

# Function to print state number and name (used for testing)
def print_state(state):

    if(state == 0):
        print("- State 0: no pain")
    elif(state == 1):
        print("- State 1: neck pain + bad posture")
    elif(state == 2):
        print("- State 2: neck pain")
    elif(state == 3):
        print("- State 3: back pain + bad posture")
    elif(state == 4):
        print("- State 4: back pain")
    elif(state == 5):
        print("- State 5: leg pain")
    elif(state == 6):
        print("- State 6: arm pain")
    else:
        print("- State 7: shoulder pain")


# Function to print action number and name (used for testing)
def print_action(action):
```

```
    if(action == 0):
        print("- Action 0 performed: neck_1")
    elif(action == 1):
        print("- Action 1 performed: neck_2")
    elif(action == 2):
        print("- Action 2 performed: arm_1")
    elif(action == 3):
        print("- Action 3 performed: back_1")
    elif(action == 4):
        print("- Action 4 performed: leg_1")
    elif(action == 5):
        print("- Action 5 performed: back_2")
    elif(action == 6):
        print("- Action 6 performed: arm_2")
    elif(action == 7):
        print("- Action 7 performed: walk")
    elif(action == 8):
        print("- Action 8 performed: extend knee")
    elif(action == 9):
        print("- Action 9 performed: extend palm")
    elif(action == 10):
        print("- Action 10 performed: side bending")
    else:
        print("- Action 11: see a doctor")


# action 101 = 14s
# action 102 = 14s
# action 103 = 6s
# action 104 = 11s
# action 105 = 13s
# action 106 = 20s
# action 107 = 18s
def perform_action(action):

    text_num = random.randint(0,1)
    if(text_num == 0):
        thinking = "hmmmmmmmmmmmmmmmmmmmmmmmmm."
    elif(text_num == 1):
        thinking = "Hmmmmmm. Let me think."
```

```
    text_to_speech_publisher.publish_text(thinking)
    if (action == 0):

        # say this
        text_num = random.randint(0,2)
        if(text_num == 0):
            advice = "Try to move your neck to the right, then move it
to the left like this. Do it ten times."
        elif(text_num == 1):
            advice = "Here's an exercise for your neck. Tilt your head
to the right, then to the left, like this. Do it ten times."
        elif(text_num == 2):
            advice = "Let's do a neck stretch. Rotate your head to the
right, then to the left, like this. Repeat this motion ten times."
        text_to_speech_publisher.publish_text(advice,False)
        # send action 101
        time.sleep(5)
        action_sender.publish_action(101)
        time.sleep(14)

    elif (action == 1):

        # say this
        text_num = random.randint(0,2)
        if(text_num == 0):
            advice = "This neck exercise is quick and easy. Just move
your neck up and down like this, five times."
        elif(text_num == 1):
            advice = "To stretch your neck muscles, try moving your head
up and down like this, five times. It's a simple exercise."
        elif(text_num == 2):
            advice = "Could you please move your neck up and down like
this five times?"
        text_to_speech_publisher.publish_text(advice,False)
        # send action 102
        time.sleep(5)
        action_sender.publish_action(102)
        time.sleep(14)
    elif (action == 2):

        # say this
```

```
        text_num = random.randint(0,2)
        if(text_num == 0):
            advice = "Let's do a simple arm stretch. Raise your arms
like a fork, and hold it for ten seconds. It'll help relieve tension."
        elif(text_num == 1):
            advice = "This arm exercise is quick and effective. Raise
your arms like a fork, like this, and hold it for ten seconds."
        elif(text_num == 2):
            advice = "Try raising your arms like a fork like this. Hold
it for ten seconds."

        text_to_speech_publisher.publish_text(advice,False)
        # send action 103
        time.sleep(5)
        action_sender.publish_action(103)
        time.sleep(6)

    elif (action == 3):

        # say this
        text_num = random.randint(0,2)
        if(text_num == 0):
            advice = "Let's try a standing lower back stretch. Look at
me and follow along."
        elif(text_num == 1):
            advice = "This standing lower back stretch is quick and
effective. Watch me and try it yourself."
        elif(text_num == 2):
            advice = "You can do a standing lower back stretch. Look at
me how I am doing it."
        text_to_speech_publisher.publish_text(advice,False)
        # send action 104
        time.sleep(5)
        action_sender.publish_action(104)
        time.sleep(11)
    elif (action == 4):

        # say this
        text_num = random.randint(0,2)
        if(text_num == 0):
```

```
            advice = "Let's try a balance exercise. Stand on one leg,
lift your other leg behind you, and hold it with your arm for five
seconds."
        elif(text_num == 1):
            advice = "Can you try this simple standing balance exercise
with me? Lift one leg behind you and hold it with your arm for five
seconds, while standing on one leg."
        elif(text_num == 2):
            advice = "Try to stand on one leg, raise your other leg to
the back, then use your arm to hold your extended leg for five seconds.
"

        text_to_speech_publisher.publish_text(advice,False)
        # send action 105
        time.sleep(5)
        action_sender.publish_action(105)
        time.sleep(13)
    elif (action == 5):

        # say this
        text_num = random.randint(0,2)
        if(text_num == 0):
            advice = "Bend your arms to a 90-degree angle and pull them
back towards your body. Repeat this motion 10 times."
        elif(text_num == 1):
            advice = "Create a 90-degree angle with your elbows and pull
your arms back. Repeat this action 10 times."
        elif(text_num == 2):
            advice ="Try to make your elbows 90 angle, then pull your
arms to the back. Do it 10 times."

        text_to_speech_publisher.publish_text(advice,False)
        # send action 106
        time.sleep(5)
        action_sender.publish_action(106)
        time.sleep(20)

    elif (action == 6):

        # say this
        text_num = random.randint(0,2)
```

```
        if(text_num == 0):
            advice = "Can you do this arm exercise with me? Make a 90-
degree angle with your elbows, then extend your arms out to the sides.
Repeat ten times."
        elif(text_num == 1):
            advice = "Can you try this arm workout? Make a right angle
with your elbows, then stretch your arms out to the sides. Repeat ten
times."
        elif(text_num == 2):
            advice ="For this exercise, raise your elbows to shoulder
height and then extend your arms out to the sides. Repeat ten times."

        text_to_speech_publisher.publish_text(advice,False)
        # send action 107
        time.sleep(5)
        action_sender.publish_action(107)
        time.sleep(18)


    elif (action == 7):
        # say this
        text_num = random.randint(0,2)
        if(text_num == 0):
            advice = "You could try and walk around for a few minutes.
This could help reduce your pain"
        elif(text_num == 1):
            advice = "Maybe walking around for a few minutes could help
reduce your discomfort. Would you like to try it?"
        elif(text_num == 2):
            advice ="How about taking a short walk? It may help ease
your pain."
        text_to_speech_publisher.publish_text(advice)
        time.sleep(3)


    elif (action == 8):
        # say this
        text_num = random.randint(0,2)
        if(text_num == 0):
            advice = "Make sure your knee is fully extended and has a
slight bend of more than ninty degrees."
```

```
        elif(text_num == 1):
            advice = "Check that your knee is extended and has a
comfortable bend wider than ninty degrees."
        elif(text_num == 2):
            advice = "Please make sure that your knee is well extended.
It should have an angle a little wider than ninty degrees."

        text_to_speech_publisher.publish_text(advice)
        time.sleep(3)


    elif (action == 9):
        # say this
        text_num = random.randint(0,2)
        if(text_num == 0):
            advice = "Would you like to try extending your other arm
downwards for a few seconds? It could potentially reduce your
discomfort."
        elif(text_num == 1):
            advice = "You could try using your opposite arm to extend
downwards for a few seconds. It might help reduce your discomfort."
        elif(text_num == 2):
            advice = "Using your other arm. Extend your arm downwards
for a few seconds. This could reduce your pain."

        text_to_speech_publisher.publish_text(advice)
        time.sleep(3)


    elif (action == 10):
        # say this
        text_num = random.randint(0,2)
        if(text_num == 0):
            advice = "Side bend your head to the left and right, holding
each position for a few seconds. Do this three times on each side."
        elif(text_num == 1):
            advice = "Would you like to try side bending your head to
the left and right? Repeat the movement three times on each side."
        elif(text_num == 2):
            advice = "Try to side bend your head to the left and right.
Do it 3 times for each side."
```

```
        text_to_speech_publisher.publish_text(advice)
        time.sleep(3)


    elif (action == 11):
        # say this
        text_num = random.randint(0,2)
        if(text_num == 0):
            advice = "If your pain continues, it may be worth
considering consulting a doctor. I'm sorry that my suggestion didn't
help."
        elif(text_num == 1):
            advice = "If the pain persists, it might be a good idea to
see a doctor. I apologize that my suggestion didn't provide relief."
        elif(text_num == 2):
            advice = "I'm sorry that my advice wasn't helpful. It might
be worth considering seeing a medical professional if your pain
continues."

        text_to_speech_publisher.publish_text(advice)







def process_state(state):


    # load Q-table from "scripts/main/q_learning/q_table.npy"
    rospack = rospkg.RosPack()
    package_path = rospack.get_path('robot_personal_assistant_op2')
    q_table_path = os.path.join(package_path,
'scripts/main/q_learning/q_table.npy')
    q_table = np.load(q_table_path)

    #state = random.randint(1,7)
    bad_posture_time = None
```

```
    def callback(data):
        nonlocal bad_posture_time
        # convert the recieved image into suitable format using CvBridge
        bad_posture_time = data.data

    if (state == 2 or state == 4):
        rospy.Subscriber('/bad_posture_time', Int32, callback)
        while not rospy.is_shutdown():
            if bad_posture_time is not None:
                if (bad_posture_time > 0):
                    state -=1
                    break
                else:
                    break

    print_state(state)



    while True:

        action = np.argmax(q_table[state,:])
        # perform the action

        print_action(action)
        perform_action(action)
        if (action == 11):
            break



        time.sleep(2)
        text_num = random.randint(0,2)
        if(text_num == 0):
            text_to_speak = "Are you satisfied?"
        elif(text_num == 1):
            text_to_speak = "Have you noticed any changes in your pain
level?"
        elif(text_num == 2):
            text_to_speak = "Did your pain go away?"
        text_to_speech_publisher.publish_text(text_to_speak)
```

```
        list_of_words = ["yes","no"]
        found_word, pain_type = word_finder.check_words(list_of_words)

        if (found_word == "yes"):
            text_num = random.randint(0,1)
            if(text_num == 0):
                text_to_speak = "That's wonderful to hear!"
            elif(text_num == 1):
                text_to_speak ="I am really happy to hear that!"
            text_to_speech_publisher.publish_text(text_to_speak)
            break
        else:
            q_table[state,action] = 0
```

audio_note_player.py

```
#!/usr/bin/env python

import rospy
from std_msgs.msg import String
from playsound import playsound
import os

def audio_callback(data):
    audio_data_str = data.data
    # Encode the audio data as bytes
    audio_data = audio_data_str.encode('latin-1')

    audio_file_path =
os.path.expanduser("~/op2_tmp/recordings/received_note_audio.wav")

    with open(audio_file_path, 'wb') as f:
        f.write(audio_data)

    # Play the audio using the playsound library
    playsound(audio_file_path)
```

```
    global finished_talking_publisher
    finished_talking_publisher.publish("finished!")
    rospy.loginfo("finished!")


def audio_subscriber():
    rospy.init_node('audio_note_subscriber', anonymous=True)
    rospy.Subscriber('/audio_topic', String, audio_callback)
    rospy.spin()

if __name__ == '__main__':
    try:
        finished_talking_publisher =
rospy.Publisher('/finished_talking', String, queue_size=10)
        audio_subscriber()
    except rospy.ROSInterruptException:
        pass
```