DEPARTMENT OF COMPUTER SCIENCE

CS3072-CS3605 Computer Science Final Year Project

Adaptive Movement Control of Swift Bot via Facial Expression Recognition

Submission Deadline 11/04/25

ACADEMIC YEAR 2024-25

IMAN EIN ALIZADEH

STUDENT ID 2236929

Contents

Abstract

As robotics and artificial intelligence (AI) continue to evolve, there is a growing demand for more intuitive and accessible methods of human-robot interaction. Traditional control interfaces, such as joysticks or basic programming, often pose barriers for individuals who may not be comfortable or proficient with such methods. Considering this, the current project aims to address the need for more natural, user-friendly robotic systems by designing and implementing a robot that responds autonomously to human facial expressions and gaze direction.

The core objective of this research is to develop an AI-powered interface for the Swift Bot that interprets key human emotions—such as happiness and anger—and adjusts the robot's movement based on the direction of the user's gaze. This system leverages cutting-edge computer vision techniques, including the FER (Facial Expression Recognition) model for emotion detection, MediaPipe for gaze tracking, and Haar Cascade for face detection. These technologies allow for an enhanced and more personalized interaction between humans and robots.

The methods employed involve developing algorithms to detect and interpret facial expressions and gaze, integrating these inputs into the movement system of the Swift Bot, and rigorously testing the system for accuracy and responsiveness. User feedback is incorporated to evaluate the system's effectiveness in real-world applications. Early results show that this approach significantly improves user engagement by enabling more intuitive and accessible robotic control.

This project highlights the potential for AI-driven emotion and gaze-based interfaces to create more interactive and inclusive robotic systems. By advancing the capabilities of human-robot interaction, the research sets the stage for developing robots that are better suited to assist a wide range of users in various environments.

1.Introduction

The integration of robotics and artificial intelligence (AI) has become an essential part of modern society, with applications spanning from healthcare to education and entertainment. As robots become increasingly present in daily life, one key challenge remains: how can we make robots more intuitive and accessible for users with diverse needs and capabilities? Traditional methods of interacting with robots, such as using physical controllers or programming inputs, are often not suitable for everyone. This is particularly problematic in contexts where accessibility is crucial, such as for individuals with physical disabilities, elderly people, or those unfamiliar with technology.

This project focuses on developing a solution that addresses this gap in human-robot interaction. By enabling robots to understand and respond to human facial expressions and gaze direction, the project aims to create a more natural, intuitive interface for controlling robots. The Swift Bot, provided by the university, serves as the platform for this experiment, with the aim of enhancing its movement control through AI-driven emotion recognition and gaze tracking.

The core problem this project addresses is the lack of intuitive interfaces for robotic control, particularly for individuals who may not be able to use traditional control methods easily. Current robots often rely on physical input methods such as joysticks, which can be inaccessible for some users. Moreover, the ability of robots to understand and respond to human emotions has largely been underdeveloped, leaving a gap in creating truly interactive, user-centric robotic systems.

1.1. Real-World Applications

The significance of this project extends beyond the academic realm, with several practical applications in various industries:

1. **Healthcare:** In healthcare, robots are increasingly used to assist individuals with disabilities or elderly patients who may have limited mobility. For these individuals, interacting with a robot through facial expressions or gaze direction can be far more accessible than using a joystick or other physical controller. A robot that responds to a person's emotions could enhance emotional support and improve communication with patients who may have difficulty with traditional forms of interaction, such as those with severe physical disabilities, Alzheimer's, or autism spectrum disorders.
2. **Elderly Assistance:** Older adults often struggle with using complex devices or controllers due to physical limitations. A robot that responds to facial expressions and gaze can make robotic assistance more intuitive. For instance, an elderly person could simply smile to encourage the robot to move toward them or use gaze direction to navigate the robot, providing an interface that feels more human-like and user-friendly.
3. **Education:** In the realm of education, robots can be used as teaching aids or companions for children. Emotionally responsive robots can be more engaging, fostering better interaction in educational settings. A robot that responds to a student's

emotions can create a more personalized learning environment. For example, if a child looks frustrated or anxious, the robot could interpret these emotional cues and adapt its behaviour to provide encouragement or adjust the task to reduce stress.

By focusing on these real-world applications, this project aims to bridge the gap between human emotional expression and robot response, creating a system that is not only functional but also empathetic and accessible.

Figure1.1.1.

The figure above is a photo I created to give a better understanding of the project idea. The image depicts a moment of human-machine synergy. A man, his face lit up with a smile, utilizes his technical expertise to guide a robotic arm. This shows the potential for groundbreaking advancements in fields like manufacturing and research.

1.2. Aims and Objectives

This dissertation aims to develop and evaluate an innovative human-robot interaction system that utilizes real-time facial expression recognition and gaze tracking to enable more intuitive and accessible robot control. To achieve this overarching aim, the following specific objectives will be pursued:

1. **Design and Implement a Facial Expression Recognition Module:**
   To develop a robust and accurate system capable of identifying key human emotions (specifically happiness and anger) from live video feeds using appropriate computer vision and machine learning techniques (leveraging the FER model and Haar Cascade).
2. **Develop a Gaze Tracking Module:**
   To integrate a gaze tracking system (utilizing MediaPipe) that can accurately determine the direction of the user's gaze in real-time.
3. **Integrate Emotion and Gaze Data for Robot Control:**
   To design and implement a mechanism that effectively combines the interpreted emotional state and gaze direction of the user to generate autonomous movement commands for the Swift Bot.
4. **Establish Communication Between Processing Modules and the Robot:**
   To develop a reliable communication architecture (using a Python client and Java server with Socket Programming) to transmit the processed emotion and gaze data to the Swift Bot for real-time control.
5. **Evaluate the System's Performance and Usability:**
   To rigorously test the accuracy and responsiveness of the integrated system under various conditions and to assess its usability and intuitiveness through user feedback and standardized questionnaires (such as the System Usability Scale).
6. **Explore the Potential for Enhanced Human-Robot Interaction:**
   To investigate how emotion and gaze-based control can contribute to more natural, empathetic, and accessible robotic systems, particularly in the context of the identified real-world applications (healthcare, elderly assistance, and education).

1.3. Dissertation Structure

The remainder of this dissertation will detail the process undertaken to achieve the aims and objectives. Following this introduction, Chapter 2 will delve into the **background research** that underpins this project, providing a more in-depth exploration of existing literature and technologies in facial expression recognition, gaze tracking, and human-robot interaction. Chapter 3 will outline the **methodology** employed, detailing the research approach, project management strategy, and ethical considerations that guided the development and evaluation of the system. Chapter 4 will present the **design** of the AI-powered interface, including system architecture, interaction flowcharts, and key design decisions. Chapter 5 will describe the **implementation** phase, detailing the development of the software components, the integration of the different modules, and the setup for experimentation. Chapter 6 will focus on the **testing and evaluation** of the system, presenting the results of user testing, performance analysis, and feedback gathered. Finally, Chapter 7 will offer a **conclusion** of the findings, highlighting the contributions of this research, its limitations, and potential avenues for future work.

2.Background

The intersection of robotics, artificial intelligence (AI), and human-robot interaction (HRI) has become a focal point of research, driven by the need to create more intuitive, adaptive, and empathetic robotic systems. As robots become integrated into a variety of human-cantered environments—such as healthcare, education, and elderly assistance—the importance of developing interfaces that allow robots to understand and respond to human emotions and actions has become increasingly clear. This background section explores the foundational concepts, technologies, and advancements that inform the current research on emotion recognition, gaze tracking, and their integration into robotic systems

2.1. Previous Research and Discoveries

The integration of **human-robot interaction (HRI)** with **emotion recognition technologies** has gained traction, especially as robots are being designed for more personalized, empathetic, and accessible roles. Below are key developments and discoveries that directly inform the context of this research

2.1.1. Facial Expression Recognition (FER)

Facial expression recognition (FER) has long been a research focus on computer vision and machine learning, with significant advancements made to accurately interpret human emotions based on facial cues.

- **Ekman & Friesen (1971)** established the foundational model of **universal facial expressions**, categorizing emotions like happiness, sadness, anger, fear, surprise, and disgust. Their work laid the groundwork for future FER models, driving the development of automated systems capable of emotion detection.
- **Zhao et al. (2006)** significantly advanced FER algorithms by integrating deep learning, enabling more accurate real-time emotion detection. Their work focused on improving the recognition of both static and dynamic facial expressions, which is a critical aspect of your project. This has real-world applications in security systems, healthcare, and customer service.
- **Mollah et al. (2019)** presented a deep learning-based approach to FER using Convolutional Neural Networks (CNNs), which can detect emotions from images and videos. Their findings emphasize the power of CNNs in automating emotion detection, which is integral to the **Swift Bot's facial expression recognition system**.
- In **healthcare robotics**, **Cai et al. (2020)** explored FER for emotional care robots aimed at elderly patients with cognitive impairments. They found that robots capable of recognizing and responding to emotional cues from patients fostered more meaningful interactions, reducing anxiety and increasing engagement. This aligns with the potential real-world application of your project in healthcare.

These advancements in FER provide a clear roadmap for developing a system capable of accurately recognizing emotions based on facial expressions, essential for the **Swift Bot's human-robot interaction capabilities**.

2.1.2. Gaze Tracking Technologies

Gaze tracking is a powerful tool in **human-robot interaction**, allowing robots to infer the user's focus of attention and emotional state based on where they are looking.

- **Krafka et al. (2016)** introduced a real-time, camera-based gaze tracking framework called **MediaPipe** that leverages machine learning to detect facial landmarks, including eye position and gaze direction. MediaPipe has become a widely used framework for gaze tracking in mobile applications and virtual reality. It provides an effective means to integrate gaze tracking into the **Swift Bot**, helping it to interpret the user's focus of attention and engage with them more intuitively.
- **Bakker et al. (2017)** developed a robot with gaze-following capabilities, where the robot adjusted its behavior based on the direction of the user's gaze. Their findings confirmed that gaze-following robots created more natural interactions and were perceived as more engaging. This offers insights into how **gaze tracking** can be utilized to enhance the **Swift Bot's responsiveness** to user attention and focus.

The integration of gaze tracking into **Swift Bot** would enable a deeper layer of interaction by allowing the robot to respond to the user's focus, creating a more fluid and **natural interaction flow**.

2.1.3. Human-Robot Interaction (HRI)

The field of Human-Robot Interaction (HRI) is undergoing a significant evolution, moving beyond purely functional systems towards creating robots capable of nuanced social interaction and empathetic responsiveness. A central tenet of this advancement is the integration of emotional intelligence, enabling robots to perceive, interpret, and react appropriately to human emotional states. The studies highlighted below underscore the growing recognition of the critical role of emotion recognition technologies in shaping the future of HRI.

- **Jung et al.'s (2020)** findings provide compelling evidence for the enhanced user experience and increased acceptance of robots that can respond to human facial expressions and emotional states across diverse domains such as healthcare, entertainment, and education. Their work emphasizes that in contexts where meaningful personal interaction is paramount, the ability of a robot to acknowledge and react to human emotions significantly contributes to user comfort, trust, and overall satisfaction. This suggests a paradigm shift in **HRI design**, where emotional awareness is not merely an add-on feature but an integral component of creating truly user-centric robotic systems.
- **Mouza et al.'s (2021)** investigation into the assistance of patients with chronic diseases reveals the profound impact of integrating emotion recognition and response capabilities. Their research demonstrates that robots equipped with the ability to perceive and react to patient emotions can extend their utility beyond physical assistance to provide crucial emotional support. This capacity to offer empathetic responses can lead to improved patient well-being, reduced feelings of isolation, and enhanced adherence to treatment protocols. The implications of this research are particularly significant in the context of an aging population and the increasing demand for personalized and compassionate care solutions.

Beyond these specific examples, the broader HRI community is actively investigating various facets of emotional intelligence in robots. This includes research on:

- **Multimodal Emotion Recognition:** Moving beyond facial expressions to incorporate other cues such as vocal intonation, body language, and physiological signals for a more holistic understanding of human emotions.
- **Contextual Emotion Interpretation:** Developing robots that can understand the context in which an emotion is expressed to provide more appropriate and nuanced responses.
- **Expressing Robot Emotions:** Exploring how robots can effectively express their own "emotions" (through non-verbal cues like movement and light patterns) to facilitate more natural and intuitive communication with humans.
- **Personalized Interaction Strategies:** Designing robots that can adapt their interaction style based on the individual user's emotional tendencies and preferences.
- **Ethical Considerations in Emotional HRI:** Addressing the ethical implications of endowing robots with emotional intelligence, including issues of deception, manipulation, and the potential for over-reliance.

2.1.4.AI in Robotics

The integration of **AI** into robotics is transforming how robots perceive and respond to the world around them, including interpreting **emotional cues** through **facial expressions** and **gaze**.

- **Zhang et al. (2020)** examined the use of **deep learning models** for emotion recognition in robots, specifically how CNNs can classify facial expressions with high accuracy. The combination of AI and computer vision techniques can allow robots to recognize not only facial expressions but also interpret the **emotional context** of those expressions.

- **Silver et al. (2021)** looked at the future of **AI-enabled robots** capable of understanding **complex human emotions** and behaviors, including detecting subtle emotional shifts through facial expressions and gaze. Their work highlighted the importance of combining **FER**, **gaze tracking**, and **natural language processing** for a **holistic approach** to HRI.

These advancements suggest that AI, when coupled with emotion recognition technologies, can elevate robots' ability to **perceive and adapt** to human emotions and interactions, making them more intuitive, adaptable, and efficient in their responses.

2.1.5. Java Programming and the Server-Client Architecture

In this research, the **AI** code that performs **facial recognition** and **gaze tracking** will be developed in **Python**, leveraging powerful machine learning libraries and frameworks like **TensorFlow** and **OpenCV**. This Python-based client will analyse the user's facial expressions and gaze data, processing it to extract emotional cues that can be interpreted by the robot.

The processed emotion data will then be sent to a **Java server**, which will be responsible for interpreting the data and controlling the **Swift Bot's** behaviour based on the recognized emotions. This division of labour between the Python client and the Java server ensures that the complex emotion recognition process is efficiently handled, while the server coordinates the robot's movement and responses.

The **Java server** will utilize various communication protocols (such as **Socket Programming**) to receive real-time emotion data from the Python client and issue corresponding control commands to the **Swift Bot**. By using Java, which is well-known for its robustness and scalability, the server can handle multiple emotion inputs and generate precise movement instructions to adapt to the user's emotional state.

The combination of **Python** for emotion recognition and **Java** for robotic control represents an optimal client-server architecture that ensures smooth and responsive human-robot interaction. The use of Java for the server side also facilitates ease of integration with other systems and ensures the Swift Bot remains highly adaptable to future enhancements.

2.1.6. Robots in Education

The integration of robots in educational settings presents a significant opportunity to personalize learning experiences and enhance student engagement. Emotionally intelligent robots, capable of understanding and responding to a student's emotional state, can adapt their teaching strategies and provide tailored support. For instance, a robot tutor that detects frustration in a student's facial expression could offer encouragement, simplify the current task, or suggest a break. Similarly, recognizing a student's gaze directed towards a specific element of a lesson could indicate their focus and interest, allowing the robot to provide more detailed information or interactive exercises related to that area. This capability to perceive and react to a student's emotional and attentional cues can foster a more supportive and effective learning environment, potentially leading to improved comprehension and a more positive attitude towards learning. Furthermore, robots can offer consistent and unbiased support, catering to individual learning paces and styles in ways that might be challenging for human educators in large classrooms. The development of robots that are sensitive to human expressions and gaze holds the promise of creating truly interactive and adaptive educational tools.

2.2. Emotion Detection via Deep Learning

The integration of deep learning techniques into emotion detection has significantly enhanced the ability of robots to interpret human emotions based on facial expressions. Convolutional Neural Networks (CNNs) have become the predominant model for **Facial Expression Recognition (FER)** due to their ability to automatically learn and extract complex hierarchical features from image data. This approach provides a highly effective and scalable solution for real-time emotion recognition, overcoming the limitations of traditional methods that relied on manual feature extraction.

2.2.1. Deep Learning Models for Emotion Recognition

Deep learning, particularly CNNs, has revolutionized **Facial Expression Recognition (FER)** by enabling automated feature extraction from raw facial images, which allows for a more accurate and nuanced interpretation of emotions. **Mollah et al. (2019)** demonstrated the effectiveness of CNNs for FER, achieving superior performance compared to previous techniques. The deep learning model's ability to discern fine-grained facial movements such as slight changes in muscle tension around the eyes and mouth has been critical in the reliable classification of emotional states such as happiness, anger, and sadness.

A key advantage of deep learning models is their adaptability across diverse user demographics, including variations in age, ethnicity, and facial structure. This adaptability is especially important in the context of **human-robot interaction (HRI),** where the robot must interact with individuals from diverse backgrounds. CNNs are capable of generalizing across these variations, making them highly effective in a wide range of real-world environments.

2.2.2. Applications in Human-Robot Interaction

The application of emotion detection through deep learning is particularly impactful in domains where robots are expected to engage with humans in a personalized and empathetic manner. In healthcare, for instance, robots equipped with emotion detection capabilities can assess a patient's emotional state and adjust their responses accordingly. If a patient displays signs of distress or discomfort, the robot can adapt its behaviour such as altering its tone of voice or movement patterns to provide emotional comfort, thus enhancing the therapeutic environment.

Similarly, in the context of education, robots that can detect and respond to students' emotional cues can foster a more supportive and engaging learning experience. By recognizing signs of frustration, confusion, or excitement, a robot can modify its interactions to better meet the emotional needs of the learner, whether that involves offering encouragement, simplifying tasks, or providing breaks. Such adaptive responses promote a more personalized educational experience, improving both student engagement and academic performance.

2.3. Rationale/Context

This research focuses on the intersection of emotion recognition technologies and human-robot interaction, aiming to create a more intuitive and accessible interface for controlling robots. As discussed, traditional robotic interfaces can be limiting, particularly for individuals with disabilities or those unfamiliar with complex technology.

By integrating facial expression recognition, this project will bridge the gap between human emotional expression and robotic control, enhancing the naturalness and empathy of interactions.

Through these innovations, this research seeks to advance the field of **human-robot interaction**, creating more adaptive, empathetic, and effective robots that can understand human emotions and respond in more natural and intuitive ways.

To ensure the **facial recognition system** is both effective and professional, the code will be developed and tested to optimize performance under diverse conditions. The system will be rigorously tested to handle variations in **facial structure** among different users, accounting for differences in age, ethnicity, and other unique facial features. Additionally, the system will be designed to work in **varied lighting environments**, from bright indoor lighting to dim or inconsistent natural light, ensuring high accuracy regardless of the conditions. This will be achieved by fine-tuning the algorithms to recognize and adjust for these variations, ensuring the **best possible results** across a broad range of real-world scenarios. The goal is to create a robust, adaptable, and user-centered system that provides accurate emotion recognition and seamless interaction for users.

3.Methodology

This section outlines the research approach, project management framework, and implementation strategies employed in the development of an AI-powered human-robot interaction system. The methodology describes the process of creating a system that utilizes facial expression recognition and gaze tracking for intuitive and accessible robot control. It provides a detailed overview of the design, development, and testing phases, highlighting the iterative nature of the process and the importance of continuous user feedback.

The approach taken for this project is user-cantered, emphasizing real-world applicability, adaptability, and refinement through ongoing testing and evaluation. By leveraging the **Agile methodology**, the project was executed in multiple sprints, allowing for flexibility in responding to new insights and requirements as they emerged. The goal was to create a system that not only meets the technical requirements but also provides an effective, empathetic, and intuitive interaction model for users of diverse needs and abilities.

This section also explores the ethical considerations associated with this research, including user privacy, data protection, inclusivity, and bias mitigation, ensuring that the system is both ethically sound and accessible to a wide range of users.

3.1. Research Approach

This project follows an iterative, user-centered research approach, focusing on continuous improvement and real-world applicability. The system for AI-powered human-robot interaction was developed through multiple phases, ensuring rigorous testing and refinement at each stage. The methodology involves data collection, system design, algorithm implementation, integration with the Swift Bot, and evaluation through user feedback and performance analysis.

3.2. Project Management Approach

The Agile methodology was selected as the framework for managing this project. Agile is well-suited for this research due to its iterative nature, flexibility, and emphasis on continuous feedback. Unlike traditional waterfall approaches, which follow a linear process, Agile allows for adaptability, making it ideal for developing AI-powered human-robot interaction systems where ongoing testing and refinement are essential. The key benefits of using Agile for this project include:

- **Adaptability**: Continuous adjustments based on user feedback.
- **Incremental Development**: Small, manageable iterations enhance functionality step by step.
- **User-Centric Approach**: Frequent evaluation ensures the system remains intuitive and effective.
- **Risk Reduction**: Early detection and resolution of issues prevent major setbacks.

3.3. Agile Development Process

The project was executed in multiple Agile sprints, each covering a critical aspect of development. The key Agile phases include:

1. Planning

- Defined project goals: Creating an AI-powered interface that enhances human-robot interaction using facial expressions and gaze tracking.
- Identified technical requirements: FER models, gaze-tracking frameworks (MediaPipe), Haar Cascade for face detection.
- Established key milestones and deliverables.

2. Design

- Created system architecture diagrams and interaction flowcharts.
- Designed user interaction models to ensure natural engagement with the Swift Bot.

3. implementation

- Implemented facial expression recognition code using deep learning models.
- Developed a Java Server Code for Swift Bot Control
- Developed a Python-based client for processing user inputs and a Java server for controlling the Swift Bot.
- Established communication protocols between the client and server for real-time interaction.

4. Testing

- Conducted iterative testing to assess accuracy and responsiveness.
- Evaluated system performance in different lighting conditions and user scenarios.
- Used feedback loops to refine algorithms and improve interaction quality.

5. Deployment

- Deployed the system for real-world testing with target users.
- Integrated the final model into the Swift Bot for autonomous interaction.

6. Evaluation and Continuous Improvement

- Gathered user feedback through surveys and usability tests.
- Analysed system performance and made refinements to improve natural interaction.
- Ensured the model adapts to diverse users by optimizing facial recognition and gaze tracking across different demographics.

Figure 3.3.1.
This figure shows the **Agile methodology**, an iterative project management cycle. It includes **Plan, Design, Develop, Test, Deploy, and Review** as continuous, repeating phases for flexible development.

3.4. Comparison with Other Methodologies

To manage this complex, interdisciplinary project, various project management methodologies were considered, including **Waterfall**, **Spiral**, and **V-Model**. However, the **Agile methodology** was ultimately chosen due to its superior adaptability, iterative development process, and alignment with the evolving demands of AI research.

3.4.1. Waterfall Methodology

The **Waterfall model** follows a strictly linear, phase-based progression—requirements, design, implementation, testing, deployment, and maintenance—where each phase must be completed before moving on. While effective for projects with fixed and well-defined requirements (e.g., software like banking systems), it's **ill-suited for dynamic, experimental projects** like this one, where new findings often necessitate revisiting earlier stages.

Limitations for this project:

- No allowance for evolving requirements or model improvements.
- Late-stage testing could delay detection of major performance issues.
- Inefficient for iterative experimentation or integrating user feedback.

3.4.2. V-Model (Verification & Validation)

The V-Model expands on Waterfall by introducing parallel testing for each development phase. Though it improves quality control, it still maintains a rigid sequence and lacks flexibility. In AI development, especially with machine learning and human-centered interaction, **new data insights can completely reshape algorithmic requirements**, making this model less compatible.

Limitations for this project:

- Rigid timelines incompatible with model tuning and dataset iteration.
- Feedback is primarily focused on verification, not user experience or usability.

3.4.3. Spiral Model

The **Spiral model** blends iterative design with risk assessment and is better suited for high-risk software projects. However, it places heavy emphasis on documentation and risk analysis, which can slow development. While more flexible than Waterfall, it's **not optimized for rapid prototyping and frequent end-user feedback** like Agile.

In contrast, Agile supports:

- **Frequent iteration and prototyping**,
- **User feedback at every stage**,
- **Quick adaptation to new discoveries**, and
- **Parallel development and testing**.

3.5. White Box, Black Box, and Grey Box Testing: Overview and Application

In the context of AI-powered human-robot interaction, testing is crucial for ensuring the system's performance, reliability, and usability. There are three main testing approaches used in software development: **white box testing**, **black box testing**, and **grey box testing**. Each of these approaches has unique advantages and can be applied in different stages of development to ensure comprehensive system evaluation.

3.5.1. White Box Testing

White box testing involves testing the internal structure or workings of a system. Testers have full access to the code, architecture, and design of the system, allowing them to evaluate the system's behaviour in detail.

Key Features:

- **Access to Source Code**: Testers are provided with the internal structure of the software, including the source code and algorithms.

- **Focus on Code Logic**: This testing method ensures that the underlying code functions as expected, especially focusing on the control flow, data flow, and logical errors.
- **Early Identification of Bugs**: Because testers can directly see the implementation, white box testing helps identify issues such as unhandled exceptions, syntax errors, or inefficient algorithms early in the development process.

Limitations:

- **Complexity**: This testing approach is only effective if testers have deep knowledge of the system's internal workings and may require additional expertise.
- **Limited Real-World Feedback**: While thorough, white box testing may overlook user experience and external factors that affect performance, such as varying environmental conditions.

3.5.2. Black Box Testing

In contrast to white box testing, **black box testing** focuses solely on the system's functionality without considering its internal structure. Testers interact with the system as end-users, focusing purely on input and output behaviour.

Key Features:

- **No Knowledge of Internal Workings**: Testers have no access to the source code, algorithms, or architecture.
- **Focus on User Behaviour**: Black box testing evaluates how the system behaves under various conditions based on user input.
- **End-User Perspective**: This testing approach emphasizes the system's usability, performance, and functionality as experienced by the end-user.

Limitations:

- **Limited Insight into Internal Failures**: Since testers do not have access to the source code, black box testing can miss underlying issues like inefficient code, algorithmic errors, or integration problems that could cause performance degradation.
- **Potential for Overlooking Edge Cases**: Without knowledge of the internal design, certain edge cases or rare system behaviours may be missed during testing.

3.5.3. Grey Box Testing

Grey box testing is a hybrid testing approach that combines elements of both white box and black box testing. In this method, testers have partial knowledge of the system’s internal workings, which allows them to conduct more targeted and informed testing while still focusing on user-facing aspects.

Key Features:

- **Partial Knowledge**: Testers have access to some, but not all, of the system's internal details. This could include access to system architecture, APIs, or high-level designs, but not the full code.
- **Targeted Testing**: By combining internal knowledge with user-facing functionality, testers can design tests that target specific components or modules (e.g., emotion recognition or gaze tracking) while also evaluating the system's overall performance and user experience.
- **Improved Test Coverage**: Grey box testing provides better coverage than black box testing because testers can more effectively simulate complex use cases and identify system vulnerabilities based on their partial internal knowledge.

Limitations:

- **Partial Knowledge**: While grey box testing offers more insight than black box testing, it still may not provide a complete picture of the system's internal operations, which could lead to missing deeper, more complex issues.
- **Resource Intensive**: This approach can be more time-consuming than black box testing due to the need to balance both internal knowledge and user-cantered testing.

3.5.4 Choosing the Best Testing Approach for the AI-Driven Human-Robot Interaction System

For AI-powered **human-robot interaction system**, the **most suitable testing approach** would be a **combination of Grey Box Testing**. This provides the ideal balance of internal system knowledge and real-world usability feedback, which is critical for developing a system that interacts naturally with human users.

Why Grey Box Testing Is the Best Fit:

1. Integration of Internal and User-Facing Testing:

- **Grey Box Testing** allows to access some internal components of the system (e.g., emotion recognition algorithms, gaze tracking models, APIs), while still focusing on user interactions and the system's overall functionality. Since my project involves AI models (such as Facial Expression Recognition and Gaze Tracking) and real-time interaction with a robot (Swift Bot), testing both the underlying algorithms and the user experience is crucial.

2. Performance Optimization:

- Given the **real-time constraints** of human-robot interaction, performance (such as low-latency response time, accurate gaze tracking, and emotional recognition under various environmental conditions) is a key factor. Grey Box Testing allows to evaluate the

efficiency of these critical components while identifying bottlenecks that might affect the user experience.

3. Security and Privacy Testing:

- My project will likely involve sensitive data, such as **facial expressions** and **gaze tracking data**. Grey Box Testing is particularly beneficial for testing the **security** and **privacy** of this data, as it provides insight into how data is processed, transmitted, and stored.

4. Flexibility to Address Real-World Scenarios:

- As human-robot interaction involves unpredictable and diverse user behaviour, the **flexibility** of Grey Box Testing enables to simulate **real-world scenarios**.

3.6. Ethical Considerations

Ethical concerns were a key component of this research. The project adhered to university ethics guidelines, ensuring:

3.6.1. User Privacy & Data Protection

- No **personal facial data** was stored beyond necessary processing.
- All collected data was **anonymized** and used solely for system improvement.
- Compliance with **GDPR** and university data protection policies was ensured.

3.6.2. Informed Consent & Participant Rights

To ensure participants understood the study, they were provided with:
Participant Information Sheet – Detailed explanation of the study, objectives, and data usage.
Survey Consent Form – Signed approval before participating in any data collection.
Right to Withdraw – Participants could exit the study at any stage without consequences.

3.6.3. Bias Mitigation & Inclusivity

To prevent algorithmic bias and ensure fair AI performance:

- The **FER (Facial Expression Recognition) model** was trained on a **diverse dataset** representing different ethnicities, ages, and facial structures.
- Continuous **user feedback** was collected to identify and mitigate any **bias-related inaccuracies**.
- The system was tested under **varied lighting conditions and facial angles** for robustness.

3.6.4. Accessibility & User-Centric Design

- The system was designed to benefit users with **varying physical abilities**.
- Special attention was given to **gaze tracking accuracy** for users with mobility impairments.

- **Usability tests** were conducted using the **System Usability Scale (SUS) questionnaire** to measure ease of use and accessibility.

3.6.5. Risk Assessment & Mitigation

A **risk assessment** was conducted to identify and address potential hazards.

3.6.6. User Feedback & Continuous Improvement

After system deployment, participants were asked to complete:

- A **User Feedback Evaluation** to assess system effectiveness.
- A **Questionnaire** to gauge satisfaction and usability.
- The **SUS Questionnaire** to quantify ease of use.

Collected responses guided further refinements, ensuring the system remained **ethical, user-friendly, and effective**.

4.Design

The **Design** section of this project will focus on the creation of the system architecture, user interaction flow, and the overall structure that ensures a seamless and intuitive experience for human-robot interaction. This phase is critical to the success of the AI-powered interaction system, as it lays the groundwork for how the system operates, communicates, and interacts with users through facial expression recognition and gaze tracking.

4.1. Flowcharts

The following flowcharts serve as a structured representation of the system's design, illustrating each step of the program's execution. These diagrams provide a clear, systematic approach to the development process, ensuring a well-defined workflow for implementing AI-powered human-robot interaction. Each flowchart highlights critical stages, from data processing to decision-making, enabling a professional and methodical development strategy.

4.1.1. System Overview Flowchart (with sub-processes)

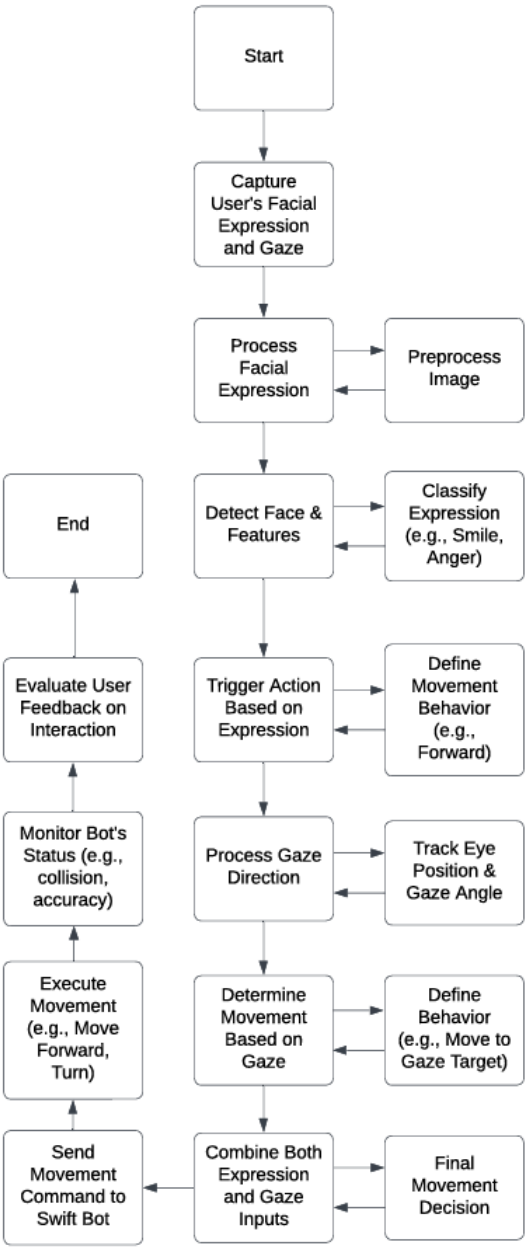


Figure 4.1.1.1. This chart provides a system overview, illustrating the key design stages involved interpreting user expression and gaze. These stages include image preprocessing, feature detection, expression classification, gaze tracking, and the generation of corresponding robot movement commands.

4.1.2. Facial Expression Recognition Process (with sub-processes)

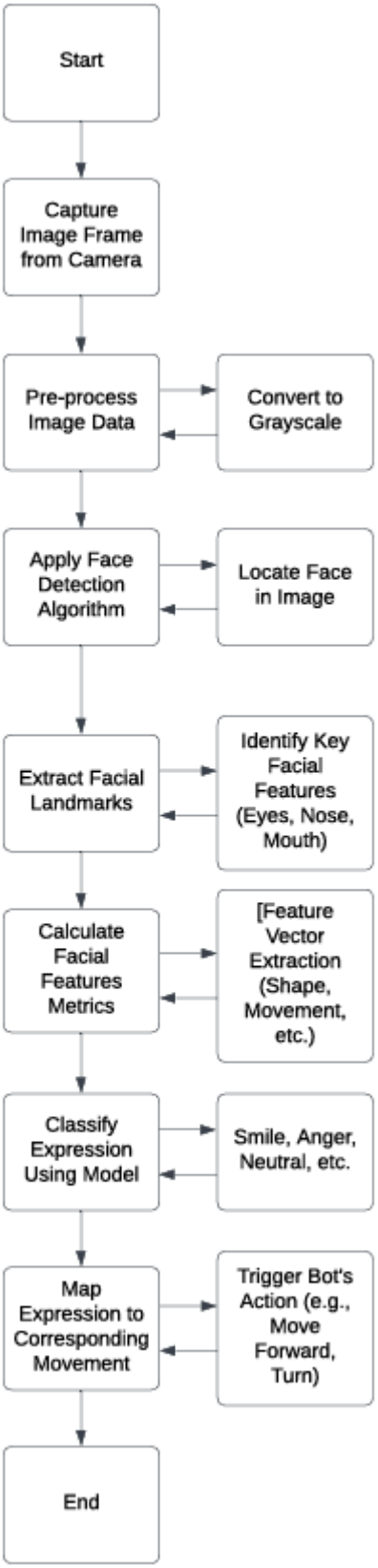


Figure 4.1.2.1. The chart illustrates a facial expression recognition process. Key steps include capturing an image, preprocessing it, detecting the face, extracting facial landmarks and features, classifying the expression, and triggering a corresponding action.

4.1.3. Interaction Flow (Human-Robot Interaction)

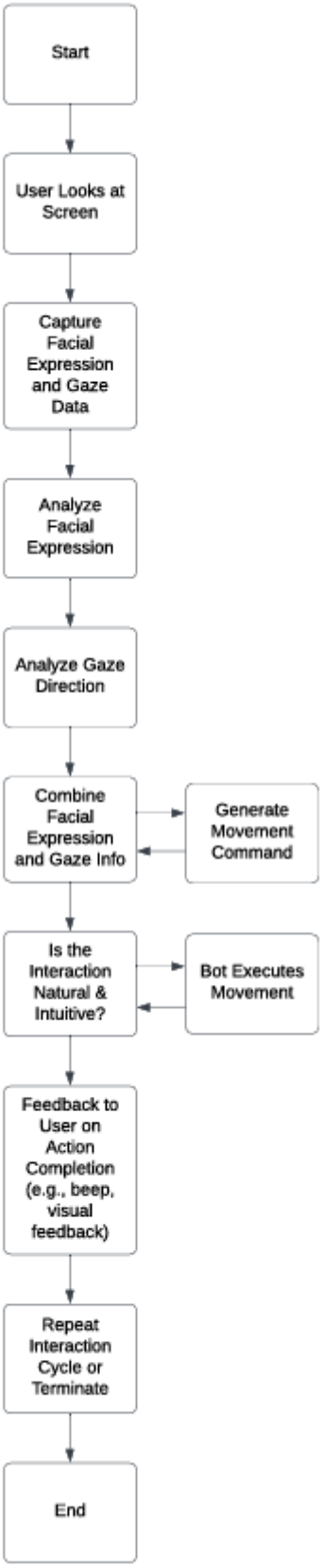


Figure 4.1.3.1. This diagram outlines a human-robot interaction flow. The process begins with the user looking at a screen, which triggers the capture of their facial expression and gaze data. Subsequently, the system analyses both the facial expression and gaze direction, combining this information to generate a movement command for the robot. A key decision point assesses whether the interaction is natural and intuitive. The robot then executes the movement, and feedback is provided to the user upon completion, using cues like beeps or visual signals. The cycle can either repeat or terminate, concluding the interaction.

4.1.4. Testing and Debugging Process Flowchart

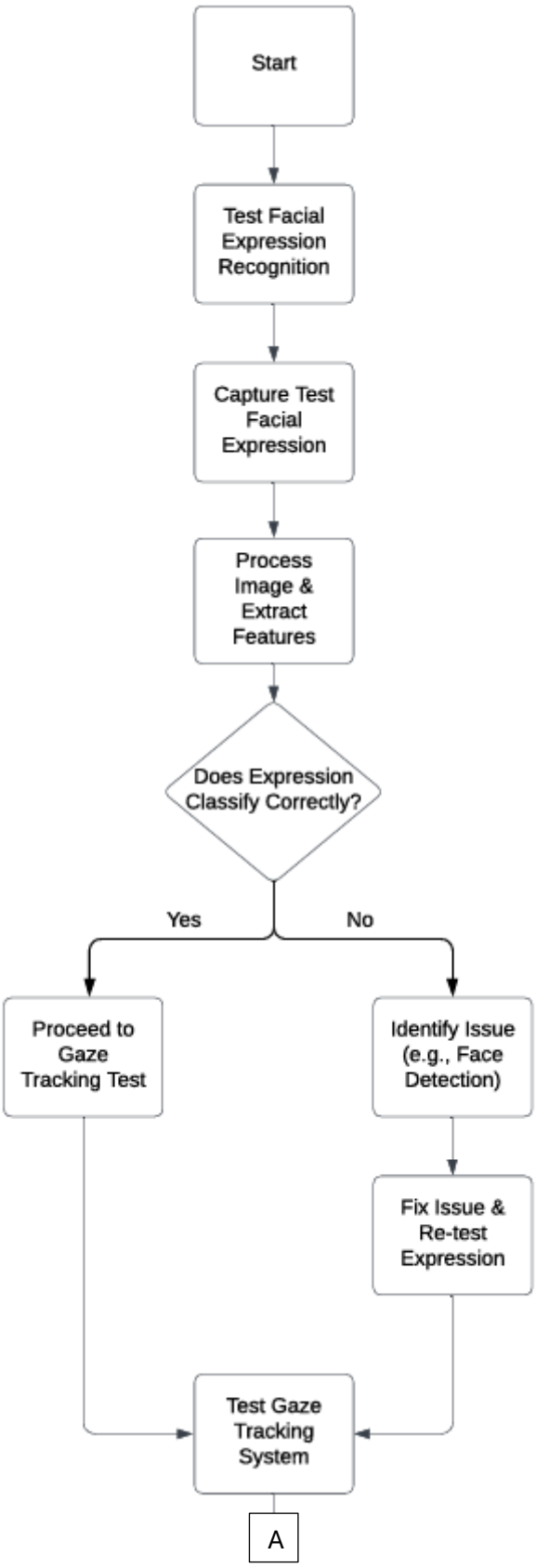


Figure 4.1.4.1. The testing process begins by evaluating facial expression recognition, which involves capturing a test expression, processing the image to extract relevant features, and determining if the classification is accurate; if not, the underlying issue (e.g., face detection) is identified, otherwise, the process advances to test gaze tracking, where gaze direction data is captured and mapped to robot movement, followed by an assessment of its correct functionality.

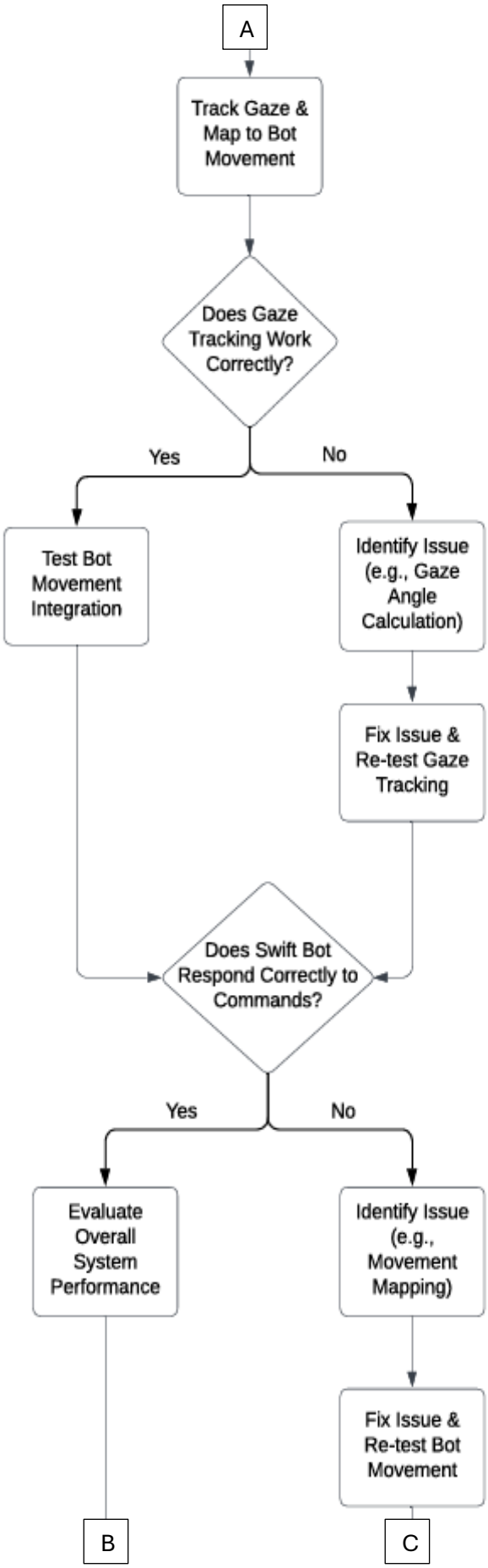


Figure 4.1.4.2. Should either the facial expression or gaze tracking tests fail, the specific problem (e.g., gaze angle calculation) is identified, rectified, and the affected system is re-evaluated; only upon confirmation of correct functionality for both expression and gaze tracking does the testing proceed to the integration of bot movement, where the robot's accurate response to commands is assessed.

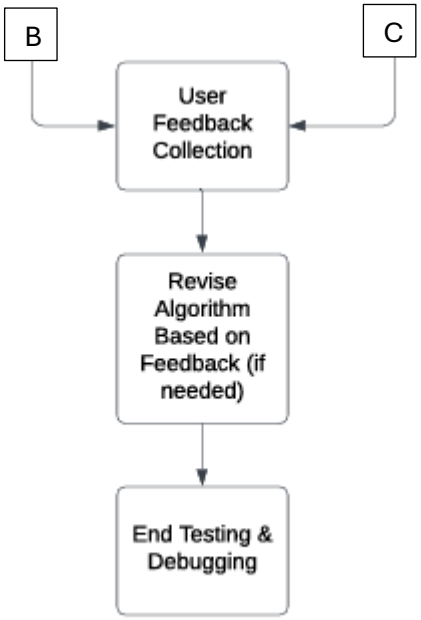


Figure 4.1.4.3

- If the bot does *not* respond correctly, the issue is identified (e.g., movement mapping), and the bot movement is fixed and re-tested.
- If the bot responds correctly, the overall system performance is evaluated.
- User feedback is collected.
- The algorithm is revised based on feedback, if needed.
- The testing and debugging process ends.

4.1.5. User Feedback Evaluation Flowchart

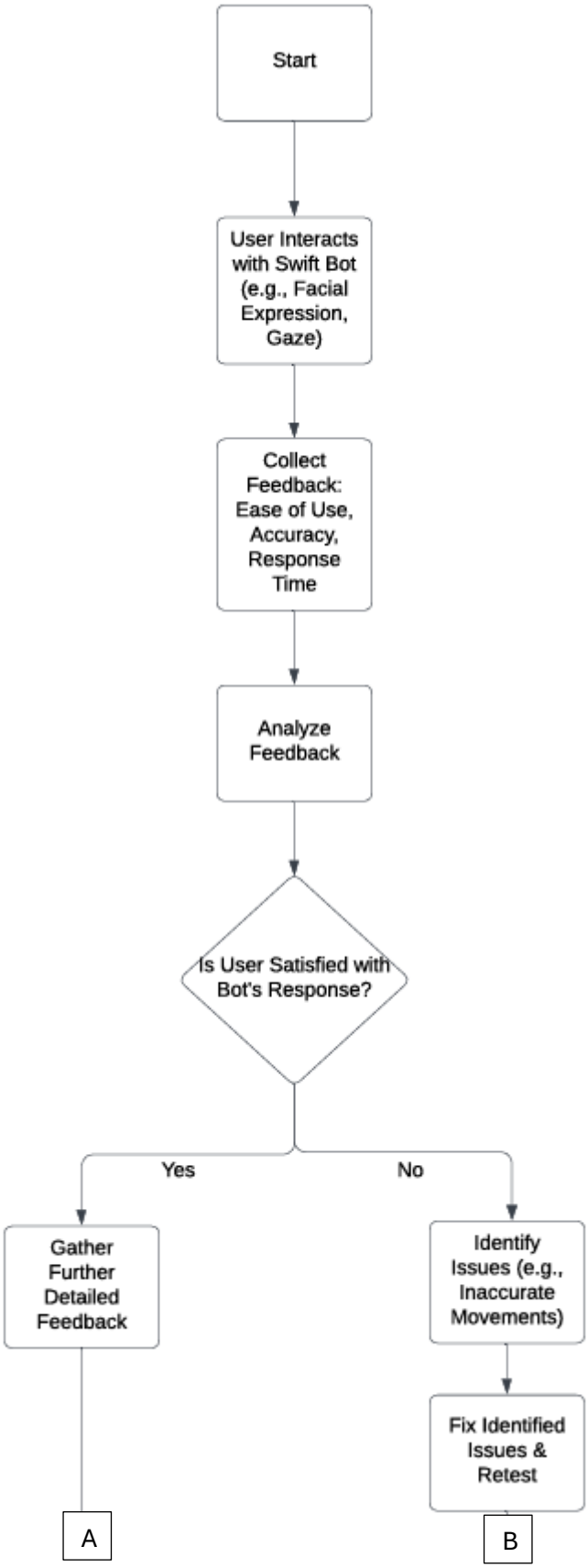


Figure 4.1.5.1. The process begins when the user engages with the Swift Bot using natural interaction methods such as facial expressions or gaze direction, triggering the bot's AI-driven response system. Following this interaction, detailed feedback is collected from the user, focusing on key performance indicators like ease of use, facial expression recognition accuracy, gaze tracking precision, and system response time. This feedback is then thoroughly analysed to assess overall system performance and user satisfaction. A critical decision point follows, where it is determined whether the user is content with the bot's responses and interaction quality. If the user is not satisfied, further evaluations are conducted to explore specific areas of concern, such as whether the interaction felt natural and intuitive or if the gaze tracking system accurately detected and responded to eye movement. Through this evaluative process, precise issues are identified—for example, delayed or inaccurate robot movements, misclassified expressions, or erratic gaze tracking. Once these issues are pinpointed, the development team iteratively refines the algorithms, adjusts system parameters, and implements improvements. The updated system is then retested under similar conditions to ensure that the problems have been effectively resolved, and that the human-robot interaction has become smoother and more accurate.

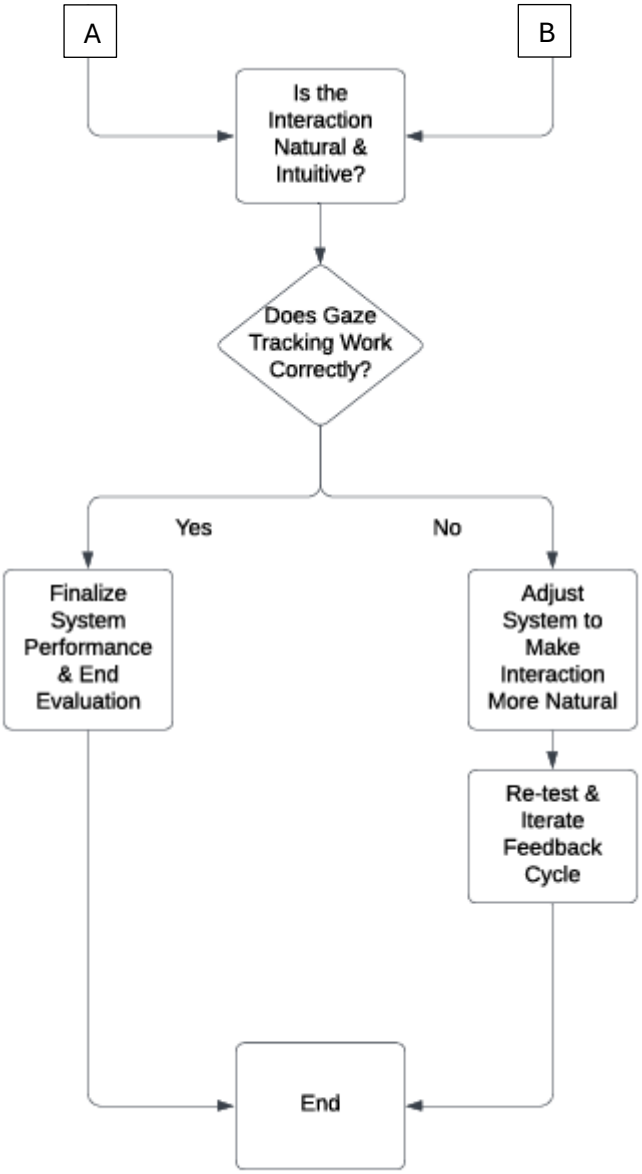


Figure 4.1.5.2. Once the user is satisfied with the bot's response—or after fixes and retesting—the process moves to system finalization. If the interaction feels natural and gaze tracking works correctly, the evaluation ends. If the interaction is unnatural, adjustments are made to improve naturalness, and the process concludes. However, if gaze tracking is still inaccurate, the system re-enters the feedback and testing cycle for further refinement.

4.1.6. Human-Robot Interaction via Facial Expression and Gaze

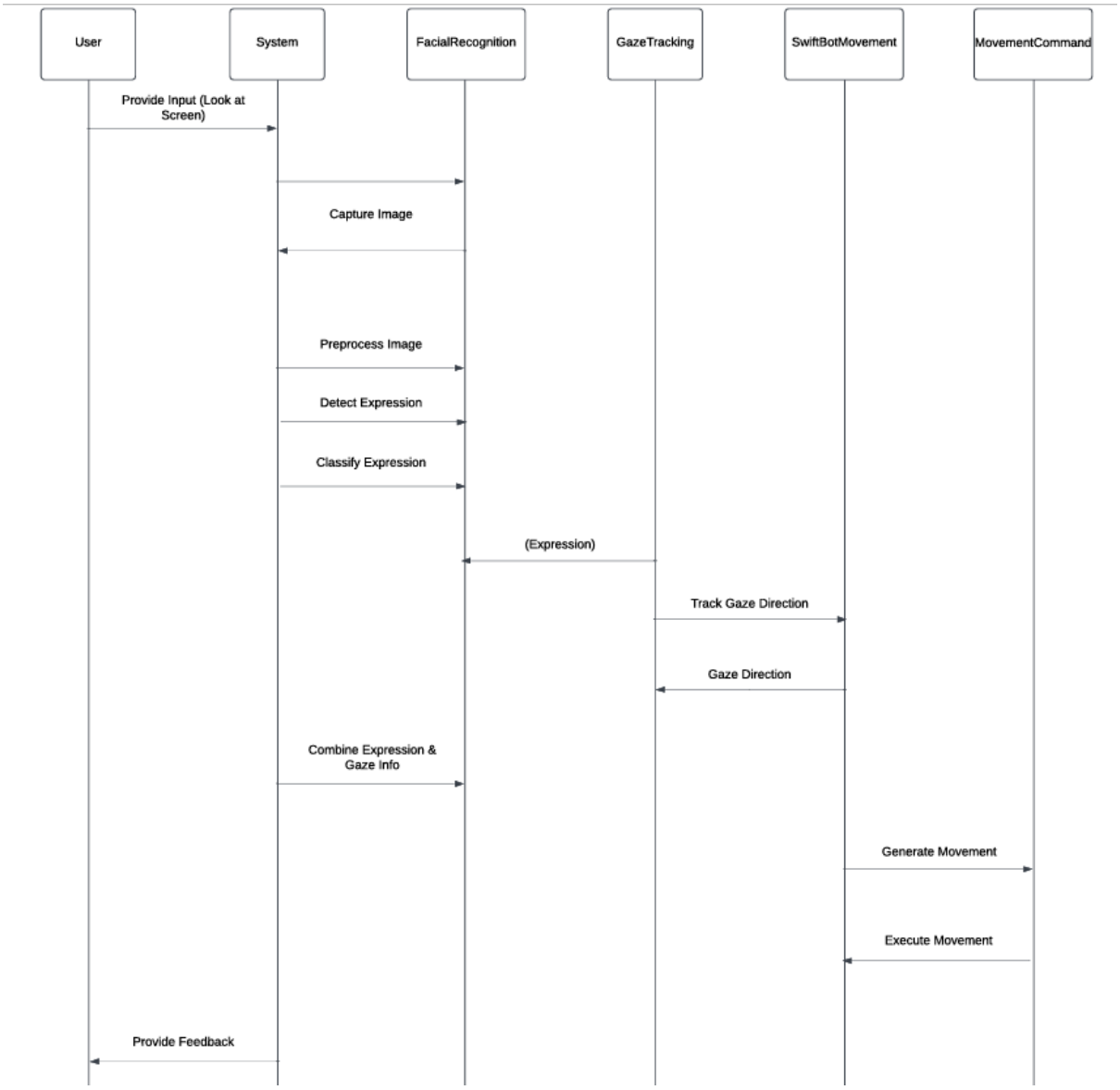


Figure 4.1.6.1.

This diagram illustrates the interaction between a user and a system, specifically focusing on facial expression recognition, gaze tracking, and robot movement control.

Here's a breakdown of the process:

1. **User Input:** The user provides input to the system, primarily by looking at a screen. The user also provides feedback to the system.
2. **System Processing:**
    - The system captures an image of the user.
    - The image is pre-processed.
    - The system detects and classifies the user's facial expression.

- The system tracks the user's gaze direction.
- The facial expression and gaze direction information are combined.

3. **Robot Control:**
   - A movement command is generated based on the combined expression and gaze data.
   - The system then executes the movement of the Swift Bot.

4.1.7. Use Case Overview

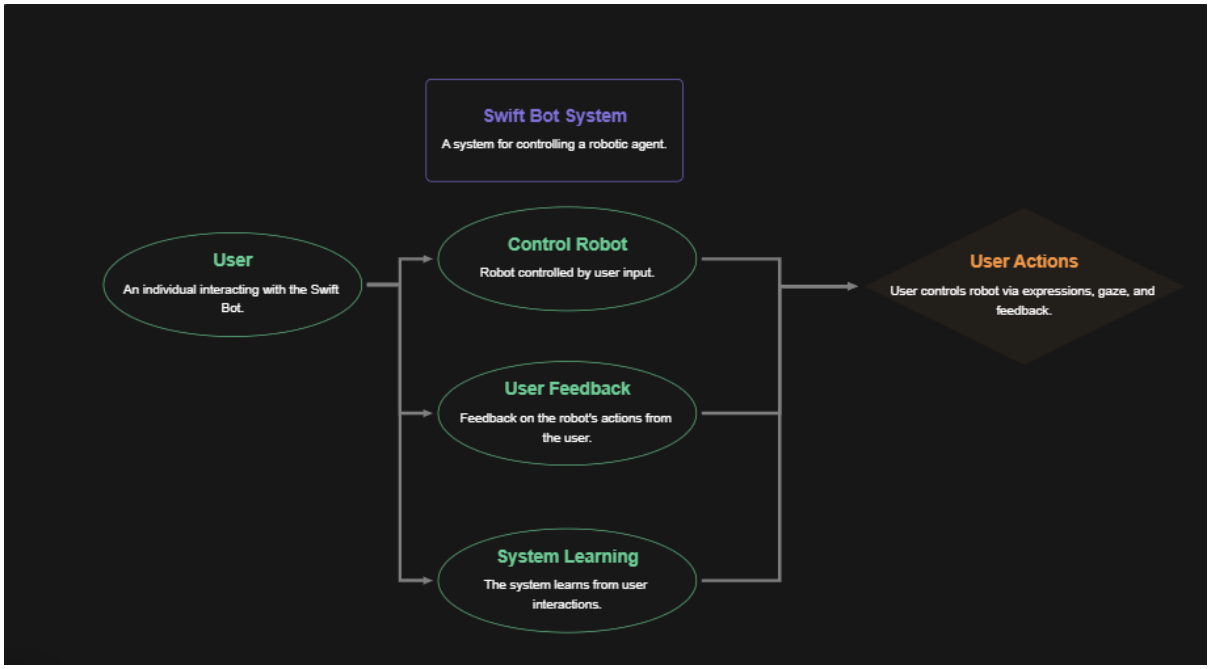


Figure 4.1.7.1.

This diagram provides a high-level conceptual overview of the use cases within the AI-powered human-robot interaction system designed for the Swift Bot. It illustrates the primary actor, the **User**, and their interactions with the **Swift Bot System** to achieve the core objective of controlling the robotic agent through natural human cues.

The diagram highlights three key functional areas through which the User engages with the system:

- **Control Robot:** This central use case represents the user's ability to direct the Swift Bot's actions. It is driven by **User Actions**, which encompass the control modalities of facial expressions, gaze direction, and explicit feedback.
- **User Feedback:** This captures the crucial aspect of the user providing input on the robot's behavior and the overall interaction experience. This feedback loop is essential for evaluating the system's effectiveness and identifying areas for improvement.
- **System Learning:** This intrinsic functionality signifies the system's capacity to adapt and enhance its interaction capabilities over time by learning from user actions and

feedback. This continuous learning mechanism aims to improve the intuitiveness and responsiveness of the robot's control.

4.2. Ethics

4.2.1. Risk Assessment

These documents are formal risk assessments for a research project involving human participants interacting with a robot to test its facial expression and gaze-directed control system. They detail the project, identify potential hazards to participants and researchers, and outline safety measures.

Why It Matters?

- **Ethical Foundation:** It shows rigorously considered participant safety and well-being, which is crucial for ethical research.
- **Methodology Support:** It provides details about research procedures (user testing, etc.), demonstrating careful planning.
- **Risk Awareness:** It highlights potential problems (e.g., equipment malfunction, psychological harm) and how addressed them.

BREO Risk Assessment form

This risk assessment should be competed for all research projects and should focus on any health and safety issues relating to Travel, Researchers or Participants.

Project Title	CS3072/CS3605 Final Year Projects Group Ethics Application
College/Directorate/Department	College of Engineering, Design and Physical Sciences
Does your project involve travel (international or UK)?	UK
Have you included measures relating to Covid-19 safety?	Yes
Date of initial assessment	01/10/2024

Brief description of work activity being assessed

Brief Description of Work Activity Being Assessed

This project involves conducting in-person research primarily on campus, where we will gather user data through direct interactions with our robot prototype. The activities will include:

1. **User Testing Sessions**: Participants will interact with the robot, providing real-time feedback on its responsiveness to facial expressions and gaze direction. These sessions will be recorded for analysis.
2. **Interviews and Surveys**: Following the interaction, participants will be asked to complete surveys and participate in interviews to evaluate their experience and perceptions of the intuitive control system.
3. **Observation and Data Collection**: Researchers will observe user interactions to identify any challenges or points of confusion, allowing for further refinement of the interface.

Purpose of Work and Activities Involved in Collecting User Data

The purpose of this work is to assess the effectiveness of a facial expression and gaze-directed robotic control system, ultimately aiming to improve user experience and accessibility. The activities involved in collecting user data include:

- **Recruiting Participants**: We will engage diverse individuals from the campus community to ensure a broad range of feedback and perspectives.
- **Facilitating Interaction**: Participants will engage with the robot in a controlled environment, allowing us to observe their natural responses and interactions.

- **Collecting Qualitative and Quantitative Data**: Data will be gathered through both qualitative methods (user interviews, open-ended feedback) and quantitative measures (surveys with Likert scales) to assess user satisfaction and usability.

Things to consider within the assessment – this list may not be exhaustive

- **Personal safety** e.g. Social distancing; Lone Working; Escape from fire; physical/verbal attack; disability or health problems; delayed access to personal or medical assistance; failure of routine or emergency communications; security of accommodation and support; getting lost, or stranded by transport; terrorism/kidnapping/civil unrest; cultural or legal differences. Please also consider any disposal of PPE.
  - List aspects of the work with significant hazards, and give brief details of how foreseeable harm/injuries could occur.
- **Risk of distress, anxiety and psychological harm**.
- **Location/ Geographic risks –** Any risks specific to the proposed research location.
- **Equipment hazards - Storage, handling and use of equipment and materials** e.g. Tools; machinery; vehicles; manual handling; noise; work at height; electricity; fire; vacuum; high pressure; high temperature; ultra violet; laser; vibration - List equipment and materials with significant hazards, and give brief details of how foreseeable harm/injuries could occur.
- **Biological and or Chemical hazards that may be associated with your project** - Blood or blood products from humans or animals; sterilisation or cleaning equipment and/or chemicals
-

Risk Assessment:

Description of Hazard (only include significant hazards inherent within the task or the activity. The list given below is not exhaustive and you should add appropriate rows for your work.)	Person(s) at risk e.g. staff, students, unexpected persons, etc.	Current control measures in place	Current risk rating			Further control measures required and by whom and when (usually only necessary where the risk rating is either high or medium)	Final risk rating		
			Likelihood	Severity or impact	Risk Rating		Likelihood	Severity or impact	Risk Rating
Risk of distress, anxiety and psychological harm.	All participants	Remind all participants to rest well before the research. Adjust the pace of the work to suit everyone. Participants will be reminded of their right to withdraw at any parts of the study and given the support they require.	1	1	1		1	1	1
Travel (Please indicate any travel related risks, but within the UK and Internationally)	All participants	Remind all participants to check Met Office website as well as the TFL website to ensure it is safe and secure to travel.	1	1	1		1	1	1

Description of Hazard (only include significant hazards inherent within the task or the activity. The list given below is not exhaustive and you should add appropriate rows for your work.)	Person(s) at risk e.g. staff, students, unexpected persons, etc.	Current control measures in place	Current risk rating			Further control measures required and by whom and when (usually only necessary where the risk rating is either high or medium)	Final risk rating		
			Likelihood	Severity or impact	Risk Rating		Likelihood	Severity or impact	Risk Rating
Covid secure measures (Please indicate the safety measures in place to reduce the likelihood of infection (researchers and participants)	All participants	Use of mask during the research.	1	1	1		1	1	1
Lone Working	All participants	The research will be conducted in public spaces. All participants will be asked to read the university´s Lone-working policy. All participants will be notified of the exits.	1	1	1		1	1	1
Pre-existing medical conditions	No one								
Laboratory or Workshop Hazards	No one		1	1	1		1	1	1
Physical	No one		1	1	1		1	1	1
Chemical	No one		1	1	1		1	1	1
Biological	No one		1	1	1		1	1	1
Unfamiliarity with robotic interface	All participants	Provide detailed instructions and demonstrations before user interactions.	1	2	2	Offer additional training sessions if needed.	1	1	1

Description of Hazard (only include significant hazards inherent within the task or the activity. The list given below is not exhaustive and you should add appropriate rows for your work.)	Person(s) at risk e.g. staff, students, unexpected persons, etc.	Current control measures in place	Current risk rating			Further control measures required and by whom and when (usually only necessary where the risk rating is either high or medium)	Final risk rating		
			Likelihood	Severity or impact	Risk Rating		Likelihood	Severity or impact	Risk Rating
Informed consent and data privacy concerns	All participants	Clear communication about data usage and consent processes; anonymize collected data	1	2	2	Ensure all participants understand consent documentation.	1	1	1
Equipment malfunction	All participants and researchers	Regular checks and maintenance of robotic equipment prior to sessions.	2	2	4	Implement a backup plan for technical issues (researcher responsibility).	1	2	2
Electrical hazards	All participants and researchers	All equipment tested for safety compliance; participants instructed on safe usage practices.	1	2	2	Conduct a comprehensive safety briefing prior to each session (Safety Officer).	1	1	1

Person(s) completing this assessment:
(Person carrying out or managing/supervising the activity day-to-day)

Name _____ Title _____ Signature _____ Date _____

Person approving this assessment:
(Person with overall responsibility for the activity Director of Professional Service (or delegated individual, e.g manager or head of department), Senior Academic or Manager/Supervisor)

Name _____ Title _____ Signature _____ Date _____

Appendix 1 – Risk Matrix

The hazards identified within the risk assessment should be assigned a risk rating – this should be assigned for any control measures which are currently in place and any further control measures which will be required.
You should assign a value for the likelihood of an incident occurring based on the hazard from 1 to 5 and a value for the severity / impact of the hazard from 1 to 5. These should then be multiplied together to give a final risk rating e.g. 3 x 2 = 6.

SEVERITY or IMPACT	1 RARE	2 UNLIKELY	3 POSSIBLE	4 LIKELY	5 ALMOST CERTAIN
5 CATASTROPHIC	5	10	15	20	25
4 MAJOR	4	8	12	16	20
3 SERIOUS	3	6	9	12	15
2 MODERATE	2	4	6	8	10
1 MINOR	1	2	3	4	5
	LIKELIHOOD				

The Risk Score
for a hazard causing harm is calculated as follows:
Likelihood x **Severity or Impact**

High - Rating 15 or more
Immediate action is required to control and/or lower the level of risk. Exposure to the identified hazard is prohibited or severely restricted

Medium - Rating 8 - 12
Urgent review of the equipment, activities, system of work within the workplace with the aim of lowering the risk to the next level.

Low - Rating 1 – 6
Usually, no further action will be required except for monitoring to ensure the risk does not change and controls remain in place. However, if it is possible to reduce the risk levels still further, by using controls that are "reasonably practicable", then this should be done.

Scoring Criteria

Severity or Impact	Criteria
5 Catastrophic	Death
4 Major	Multiple major injuries
3 Serious	Major injury
2 Moderate	Minor injury
1 Minor	Discomfort or minor illness

Likelihood	Criteria
5 Almost Certain	>80% (happens on a regular basis)
4 Likely	51-80% (has happened at least once in last year)
3 Possible	21-50% (has happened at least once in last 2 years)
2 Unlikely	6-20% (has happened once or twice in last 5 years)
1 Rare	0-5% (hasn't happened in last 5 years)

4.2.1. System Usability Scale (SUS)

The System Usability Scale (SUS) is a standardized questionnaire used to measure the perceived usability of a system or interface. It provides a quantitative measure of usability, which can be used to evaluate and compare different systems.

Why you need it?

- **Usability Evaluation:** If research involves assessing the usability of the robot or its interface, the SUS provides a validated and reliable tool for this purpose. It collects empirical data on user satisfaction and ease of use.
- **Quantitative Data:** The SUS generates quantitative data, which can be statistically analysed and presented in results section. This adds objectivity and rigor for evaluation of usability.
- **Comparison and Benchmarking:** The SUS allows to compare the usability of your system to established benchmarks or other systems, providing context for findings.

Why it's important to support the project?

- **Methodological Soundness:** Using a standardized instrument like the SUS demonstrates methodological soundness and enhances the credibility of usability evaluation.
- **Evidence-Based Conclusions:** The SUS provides empirical evidence to support conclusions about the usability of the system. This strengthens the validity of claims.
- **Contribution to HCI:** The research focuses on human-computer interaction (HCI), so using the SUS contributes to the field by providing systematic and quantifiable usability data.

System Usability Scale (SUS)

	Strongly Disagree				Strongly Agree
I think that I would like to use this system frequently.	1	2	3	4	**5**
I found the system unnecessarily complex.	1	**2**	3	4	5
I thought this system was easy to use.	1	2	3	4	**5**
I think that I would need the support of a technical person to be able to use this system.	1	**2**	3	4	5
I found the various functions in this system were well integrated.	1	2	3	**4**	5
I thought there was too much inconsistency in this system.	**1**	2	3	4	5
I would imagine that most people would learn to use this system very quickly.	1	2	3	**4**	5
I found this system very cumbersome to use.	1	2	**3**	4	5
I felt very confident using this system.	1	2	3	4	**5**
I needed to learn a lot of things before I could get going with this system.	1	**2**	3	4	5

4.2.3. Questionnaire

This is a questionnaire designed to gather user feedback on their experience interacting with a robot that responds to facial expressions and gaze direction. It aims to collect data on users' perceptions of the robot's usability and their overall experience.

Why it's important for the project?

- **User-Cantered Design:** Gathering user feedback is crucial for user-cantered design, which emphasizes the importance of incorporating user perspectives into the development process. Including this questionnaire demonstrates the commitment to this approach.
- **Data Triangulation:** Combining data from this questionnaire with data from other sources (e.g., SUS questionnaire, observational data) can strengthen the validity and reliability of the findings through data triangulation.
- **Practical Implications:** User feedback provides valuable insights for improving the robot's design and functionality, increasing the practical relevance and impact of the research.

Questionnaire for Facial Expression and Gaze-Controlled Robot
Introduction

Thank you for participating in our research study on a robot that responds to facial expressions and gaze direction. The purpose of this questionnaire is to gather your feedback on your experience interacting with the robot. Your responses will be kept anonymous and confidential, and they will play a crucial role in enhancing the robot's usability and user experience.

	Strongly Disagree	**Disagree**	Neutral	**Agree**	**Strongly Agree**
I found the robot's response to my facial expressions intuitive					
I found the robot's response to my gaze direction intuitive.					
There were features or functions of the robot that I found particularly helpful or enjoyable.					
I encountered issues while interacting with the robot.					
I would like to see improvements or additional features in the robot.					
I encountered issues while interacting with the robot.					
I would recommend using this robot					

Any other comments or suggestions?

4.2.4. Participant Information Sheet

This document provides potential participants with essential information about the research study. It outlines the study's purpose, what participation involves, potential risks and benefits, confidentiality procedures, and contact information for the researchers and ethics committee.

Why need it?

- **Ethical Obligation:** Providing participants with clear and comprehensive information is a fundamental ethical requirement in research involving human subjects. This sheet demonstrates the adherence to this principle.
- **Informed Consent:** The information sheet is a crucial step in the informed consent process. It ensures that participants have the necessary information to make an informed decision about whether to participate or not.
- **Participant Rights:** It informs participants of their rights, including the right to withdraw from the study at any time without consequences.

Why it's important to the project?

- **Ethical Research Practices:** Using a participant information sheet is a hallmark of ethical research. It shows respect for participants and their autonomy.
- **Trust and Transparency:** Providing clear and detailed information fosters trust between researchers and participants and promotes transparency in the research process.

PARTICIPANT INFORMATION SHEET

Study title: Enhancing Robot Interaction through Facial Expression and Gaze Recognition

Invitation Paragraph

You are being invited to participate in a research study focused on developing intuitive interfaces for robotic control using facial expressions and gaze direction. It is important for you to understand the purpose of the research and what your involvement will entail. Please take time to read the following information carefully, and feel free to discuss it with others. If you have any questions or need more information, don't hesitate to ask. Thank you for considering participating in this study.

What is the purpose of the study?

This study is being conducted as part of the CS3072-CS3605 Computer Science/Business Computing Final Year Project (FYP) module at Brunel University London, with a duration of 6 months. The primary aim is to develop and evaluate a robotic system capable of interpreting human facial expressions and gaze direction as intuitive control inputs.

As robotics and artificial intelligence increasingly permeate our daily lives, enhancing the user experience and accessibility of robotic systems is critical. Traditional control methods, such as joysticks or simple programming interfaces, can be challenging for many users, especially those with limited technical skills or physical disabilities.

Through this study, we aim to create a more natural interaction between humans and robots, allowing users to control robotic functions simply by expressing emotions or directing their gaze. We will explore the effectiveness of using facial recognition and gaze tracking technologies to enable robots to respond in real time to users' non-verbal cues. By conducting trials and gathering feedback from participants, we hope to identify best practices and key metrics for measuring the success of such interaction methods. Ultimately, this research seeks to contribute to the field of human-robot interaction, making robotic systems more accessible and user-friendly for a wider range of individuals.

Why have I been invited to participate?

You have been selected because your insights and experiences could provide valuable input in assessing the effectiveness of this robotic control interface. Your perspective will help inform the development of this project.

Do I have to take part?

Participation is completely voluntary. You have the right to withdraw from the study at any time without any consequences.

What will happen to me if I take part?

If you decide to participate, you will engage in informal discussions and possibly hands-on trials with the robotic system. Your feedback will help shape the development and evaluation of the project.

Are there any lifestyle restrictions?

There are no specific lifestyle restrictions. The informal sessions will not interfere with your daily routine.

What are the possible disadvantages and risks of taking part?

There are minimal risks associated with participation, aside from a small commitment of your time and effort.

What are the possible benefits of taking part?

While there may not be direct personal benefits, your involvement will contribute to advancing the field of human-robot interaction and potentially improve accessibility for future users.

What if something goes wrong?

We have established ethical guidelines to ensure your safety and comfort throughout the study. Any unforeseen issues are expected to have minimal impact on participants.

Will my taking part in this study be kept confidential?

Your data will be anonymized, ensuring that no individual can be linked to specific feedback. Data will be aggregated, and all personal identifiers will be removed. The data will be securely stored for a minimum of ten years per university policies and will be destroyed upon my graduation unless used for conference presentations or publications

Will I be recorded, and how will the recording be used?

Audio or video recordings will be used to document interactions and feedback, but all recordings will be anonymized and used solely for the purposes of this research.

What will happen to the results of the research study?

The results will inform the development of the robotic interface and will be included in my dissertation. You will not be identified in any publications or references to this study.

Who is organising and funding the research?

I am organising this research and the research is not funded.

What are the indemnity arrangements?

Brunel provides appropriate insurance cover for research which has received ethical approval.

Who has reviewed the study?

This study has been reviewed by the College of Engineering, Design and Physical Sciences Research Ethics Committee.

Research Integrity.

Brunel University London is committed to compliance with the Universities UK Research Integrity Concordat. You are entitled to expect the highest level of integrity from the researchers during the course of this research.

Contact for further information and complaints

Researcher name and details:

(If relevant) Supervisor name and details:

For complaints, Chair of the Research Ethics Committee: Prof. Simon Taylor <Simon.Taylor@brunel.ac.uk>

4.2.5. Online Survey Consent Form

This document is a consent form specifically designed for use with an anonymous online survey or questionnaire. It outlines the conditions participants must acknowledge and agree to before participating in the survey.

Why need it?

- **Informed Consent:** While the Participant Information Sheet provides detailed information, this consent form serves as the participant's explicit agreement to participate in the online survey, acknowledging key conditions.

- **Anonymity and Data Usage:** It emphasizes that no personal identifying data is collected, and once submitted, data cannot be withdrawn due to anonymity. It also seeks consent for anonymized data to be used in future research.

Consent Form

Adaptive Movement Control of Swift Bot via Facial Expression Recognition

Guidance:

The following is to be used to gain consent for participation in an anonymous online survey/questionnaire. The statements below should make up the beginning of your survey/questionnaire.

Please confirm the following:

	Yes	No
• I have read the Participant Information Sheet included with this questionnaire		
• I am over the age of 18		
• I understand that no personal identifying data is collected in this study, therefore I know that once I have submitted my answers I am unable to withdraw my data from the study		
• I agree that my data can be anonymised, stored and used in future research in line with Brunel University's data retention policies		
• I agree to take part in this study		
• I understand that participation in this study may involve the use of video or photographic recordings, and I consent to this if applicable.		
• I have been informed about the potential risks and benefits of participating in this study.		
• I have had the opportunity to ask questions about the study and my participation.		
• I understand that participation is voluntary and that I can decline to answer any questions I am uncomfortable with.		

5.Implementation

In this section, we will explore the development, experimentation, and investigation phases of building a facial emotion detection system. The project involves creating a Python-based application for recognizing and analysing human emotions from facial expressions in real-time, followed by sending the detected emotion to a Java-based robot to trigger specific actions.

5.1. Python Code Development (Facial Emotion Detection)

The Python application was developed in **Visual Studio Code**, which provided a powerful and flexible environment for building, testing, and debugging the application. Visual Studio Code offers numerous advantages, such as integrated terminal support, extensive plugin options, and the ability to easily manage projects. This made it the ideal choice for the development of this project.

The first step was to set up a Python environment. I used **Python 3.11** because it's a stable version, and it provides better performance and improved syntax over earlier versions. However, it's important to note that certain libraries required for emotion detection does not work well with newer versions of Python which I found out when implementing the code.

5.1.1. Importing Libraries

- cv2: The OpenCV library is imported for computer vision tasks like face detection.
- numpy: Used for array manipulation, as the images are processed as arrays.
- moviepy.editor: This is for video processing, although it is not directly used in the provided code (might be for future enhancements).
- fer: This library is used for facial emotion recognition.
- PIL (Python Imaging Library): Used for image processing.
- base64: This will help handle the decoding of image data if it's received in a base64 format.
- matplotlib.pyplot: Used for displaying the processed image in a plot.

I used **pip 3** from the command line to manage the required dependencies. Installing the necessary libraries was done through the command prompt using the following commands:

- Pip3 install ipython
- pip3 install opencv-python
- pip3 install fer
- pip3 install matplotlib
- pip3 install moviepy
- pip3 install pillow

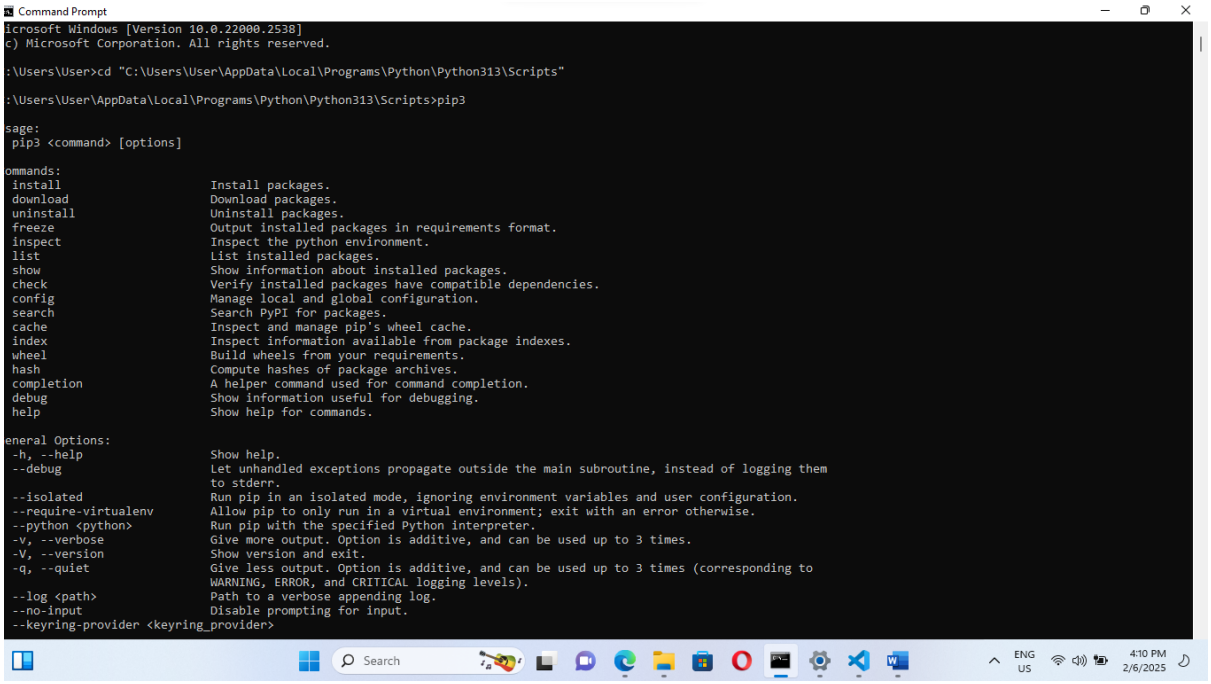


Figure 5.1.1.1.

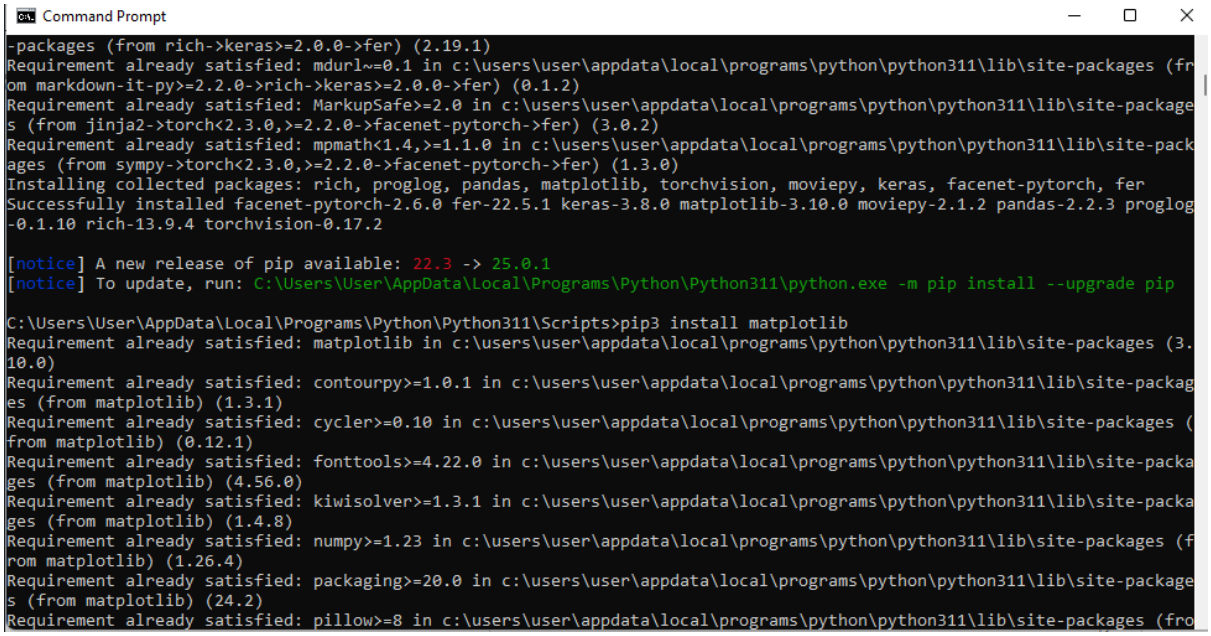


Figure 5.1.1.2.

First, we locate the directory of the file to install the necessary libraries. As shown, Python 3.13 is currently installed; however, this version changes in the future. It is important to note that some libraries are not compatible with this version. In the next screenshot, you can observe that when importing matplotlib, the Python version changes to 3.11. Additional screenshots are provided at the end of the document for further reference.

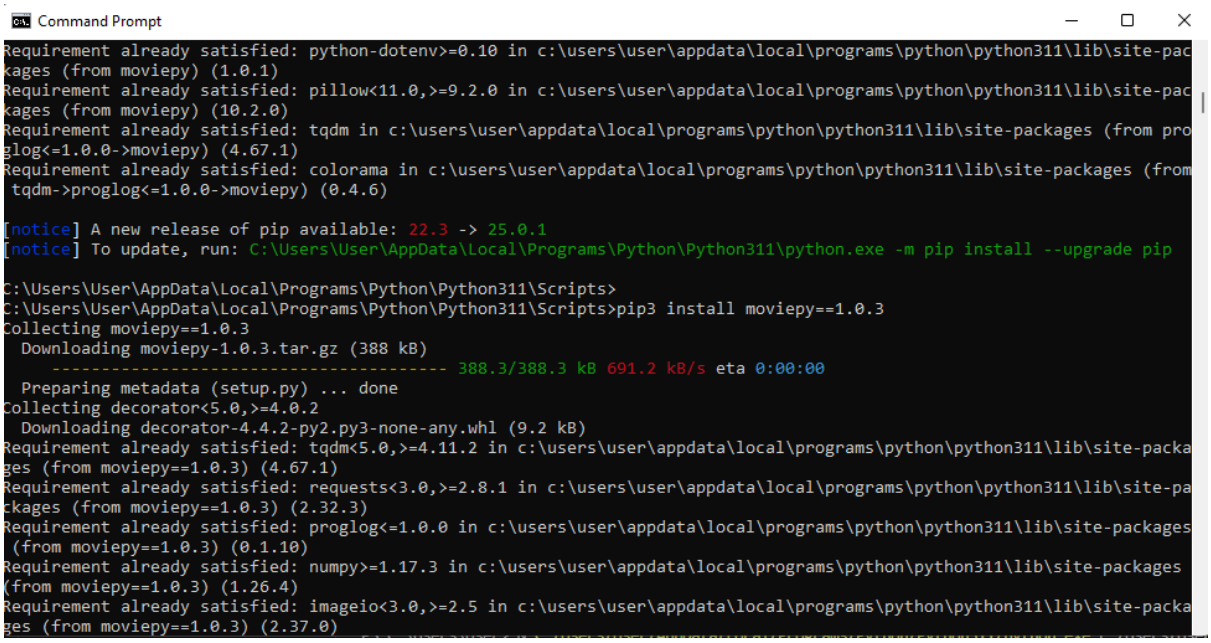


Figure 5.1.1.3.

For certain libraries, such as moviepy, I had to import a specific version to ensure compatibility with Python 3.11. As shown, the version being downloaded is 1.0.3 for moviepy.

Code:

```
import cv2

import numpy as np

try:

    from moviepy.editor import *

except ImportError:

    print("Error importing moviepy.editor. Please install moviepy
using pip.")

    exit(1)

from fer import FER

from PIL import Image

import base64

import matplotlib.pyplot as plt
```

5.1.2. Loading Pre-trained Models

Emotion detection model: FER() initializes the emotion recognition model using the fer library, which will allows to detect emotions like happiness, sadness, anger, etc., from images.

Face detection model: cv2.CascadeClassifier() loads a pre-trained Haar Cascade classifier for face detection. This classifier is used to find faces in an image.

Code:

```
emotion_detector = FER()

face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
```

5.1.3. Function to Process Frames and Detect Emotions

Base64 decoding: The dataUri is passed in base64 format. The code decodes it back into image data using base64.b64decode(). It then opens this image using PIL.Image and converts it into a NumPy array, which OpenCV can process.

Emotion detection: The emotion_detector.top_emotion() method analyzes the image and returns the top emotion and its confidence score. For example, the emotion might be "happy," and the score might indicate how confident the model is in this classification.

Face detection: The image is converted into grayscale (cv2.cvtColor) because face detection typically works on grayscale images. Then, the face_cascade.detectMultiScale() method detects the faces in the grayscale image. It returns the coordinates of the faces in the form of a rectangle (x, y, width, height).

Drawing on the image: A rectangle is drawn around each detected face using cv2.rectangle(), and the detected emotion is displayed above the face with cv2.putText().

Displaying the image: Finally, the image (with faces and emotions annotated) is displayed using matplotlib.pyplot.

Code:

```
def detect_faces_and_emotions(dataUri):
    img_data = base64.b64decode(dataUri.split(',')[1])
    image = Image.open(BytesIO(img_data))
    image = np.array(image)
    emotion, score = emotion_detector.top_emotion(image)
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)
    for (x, y, w, h) in faces:
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(image, emotion, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 255, 0), 2)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

```
    plt.axis('off')  # Hide axes
    plt.show()
```

5.1.4. Capturing Webcam Frames

Starting the webcam: The webcam is accessed using cv2.VideoCapture(0). The argument 0 refers to the default camera (you can change it to 1, 2, etc., to select different cameras if needed).

Reading frames: Inside a while loop, the webcam continuously captures frames using cap.read(). If a frame is successfully read, it proceeds to the next steps; if not, the loop breaks.

Emotion detection: Each frame is converted to RGB using cv2.cvtColor() because the emotion detection library (FER) works on RGB images. Then, emotions are detected using emotion_detector.top_emotion(rgb_frame).

Face detection: The current frame is also converted to grayscale, and faces are detected in this grayscale image using face_cascade.detectMultiScale().

Drawing rectangles and displaying emotion: For each detected face, a green rectangle is drawn around it, and the detected emotion is displayed above the face.

Displaying the webcam feed: cv2.imshow() shows the live webcam feed with the annotations on the screen.

Exiting the loop: If the user presses the 'q' key, the loop breaks, and the program exits.

Releasing the camera: After the loop finishes, the webcam is released, and all OpenCV windows are closed using cv2.destroyAllWindows().

Code:

```
def start_camera():
    cap = cv2.VideoCapture(0)
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB
        emotion, score = emotion_detector.top_emotion(rgb_frame)
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, 1.1, 4)

        for (x, y, w, h) in faces:
```

```
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255,
0), 2)
            cv2.putText(frame, emotion, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 255, 0), 2)
            print(emotion)
        cv2.imshow('Webcam', frame
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

5.1.5. Starting the Camera

This line simply calls the start_camera() function, which begins the webcam capture, emotion detection, and face recognition process.

Code:

```
start_camera()
```

5.1.6. Facial Emotion Detection first version

- **Capture video from the webcam**: The program uses OpenCV to open the webcam and continuously capture frames.
- **Detect faces**: For each frame, the face detection model finds faces.
- **Recognize emotions**: The FER library detects emotions from the faces detected in the frames.
- **Annotate the frame**: The detected face is highlighted with a green rectangle, and the emotion is displayed above the face.
- **Display the video**: The video with annotations is displayed in real-time using OpenCV.
- **Exit on 'q'**: The user can press 'q' to exit the video stream.

5.1.7. Advanced code

The code is functioning well, but my goal is to enhance the project by ensuring the AI works effectively under various lighting conditions and with different facial structures. Ultimately, I aim to improve the overall performance by optimizing the handling of face embeddings and landmark detection.

5.1.7.1. Additional Libraries and Models

MediaPipe: The Advanced code uses MediaPipe for detecting facial landmarks, which helps to more accurately detect and align facial features. MediaPipe's **FaceMesh** model provides better facial feature detection than traditional methods like Haar Cascades.

Facenet-PyTorch: This version introduces **MTCNN** for face detection and **InceptionResnetV1** for extracting facial embeddings. This setup is more robust for detecting faces and extracting features (embeddings) for tasks like facial recognition.

5.1.7.2. Lighting Conditions and Facial Structure Handling

Gamma Adjustment: A function adjust_gamma() is added to the second code to handle **lighting variations**. By adjusting the gamma, the brightness and contrast of the image can be tweaked, making the system more adaptable to **different lighting conditions**.

Facial Landmarks with MediaPipe: MediaPipe's FaceMesh model improve the recognition of **facial landmarks** (eyes, nose, mouth, etc.). This is crucial for handling **different facial structures**. With better facial alignment, the system can detect emotions more accurately, especially on people with unique facial features or non-typical poses.

FaceNet Integration: The advanced code uses FaceNet to extract **face embeddings**, which allows for more advanced face detection and emotion recognition. This could help in recognizing faces under different poses or partial occlusions.

5.1.7.3. Performance Evaluation

The advanced code includes **performance evaluation** using metrics like **accuracy**, **precision**, **recall**, and **F1 score**. These metrics help to assess the **quality of the emotion recognition** model by comparing predicted emotions with actual (true) emotions.

5.1.7.4. Additional Face Detection Methods

The advanced code uses **MTCNN** for face detection, which is a more modern and accurate method compared to the Haar Cascade used in the first code. **MTCNN** is capable of detecting faces in various poses and under partial occlusions, improving overall face detection accuracy.

5.1.7.5. Handling of Face Embeddings and Landmark Detection

Face Embeddings: The advanced code extracts face embeddings using **InceptionResnetV1**, which improves the facial recognition accuracy for emotions.

Code Availability

The full code developed for this project, including the implementation of facial expression recognition, gaze tracking, and the integration with the Swift Bot, is available in reference.

5.2. Server/Client Connection

Before developing the Java code to interface with the existing Python code for emotion recognition, I first created a basic Python and Java implementation. In this initial setup, the Python code sends a simple "Hi" message, and the Java code receives and prints it. This was done to familiarize myself with the server-client connection process before moving forward with the more complex integration.

5.2.1. Python Client Code

Import the necessary package: The java.io package contains classes for input/output operations, including Socket and BufferedReader.

Code:

```
import java.io.*;
```

Define the ActionServerThread class: This class will contain the main logic for the server.

Code:

```
public class ActionServerThread {
```

Define the main method: This is the entry point of the program.

Code:

```
public static void main(String[] args) {
```

Create a new ServerSocket object on port 5000: This creates a socket that listens for incoming connections on port 5000.

Code:

```
int port = 5000;

try (ServerSocket serverSocket = new ServerSocket(port)) {
```

Print a message to the console indicating that the server is running and waiting for connections: This message lets you know that the server is up and running.

Code:

```
System.out.println("Server is running and waiting for
connection...");
```

Accept a client connection: The accept() method of the ServerSocket object blocks until a client connects to the server. When a client connects, a new Socket object is returned.

Code:

```
Socket clientSocket = serverSocket.accept();
```

Print a message to the console indicating that a client has connected: This lets you know that a client has successfully connected to the server.

Code:

```
System.out.println("Client connected!");
```

Create a new BufferedReader object to read data from the client socket: This BufferedReader object will be used to read messages sent by the client.

Code:

```
BufferedReader reader = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
```

Read a message from the client: The readLine() method of the BufferedReader object reads a line of text from the client socket.

Code:

```
String message = reader.readLine();
```

Print the message to the console: This displays the message sent by the client on the server console.

Code:

```
System.out.println("Received from client: " + message);
```

Close the input stream and socket: The close() methods of the BufferedReader and Socket objects are called to release resources.

Code:

```
reader.close();

clientSocket.close();
```

5.2.2. Java Server Code

Import the necessary package: The socket package contains classes and functions to handle network connections.

Code:

```
import socket
```

Define the server's host and port: These are the server's IP address and port number to which the client will connect.

Code:

```
server_host = '127.0.0.1'

server_port = 5000
```

Create a new client socket: This establishes a socket object that will be used to communicate with the server.

Code:

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Attempt to connect to the server: The connect() method is used to establish a connection with the server.

Code:

```
try:
    client_socket.connect((server_host, server_port))
```

Send a message to the server: The client sends a simple string message to the server.

Code:

```
    message = "Hi"
    client_socket.sendall(message.encode())
```

Print a confirmation message: After sending the message, this line prints a confirmation that the message was sent successfully.

Code:

```
    print("Message sent to server!")
```

Handle any exceptions: If there is an error during the connection or message sending process, the exception is caught and printed.

Code:

except Exception as e:

```
    print("Error:", e)
```

Close the socket: Whether the message was sent successfully, or an error occurred, this block ensures the socket is closed.

Code:

```
finally:
    client_socket.close()
```

5.3. Java Code Development (SwiftBot Movement)

Eclipse as a Development Environment

The development of this code was facilitated using the Eclipse Integrated Development Environment (IDE). Eclipse provides several features that significantly enhance the development process:

- **Code Editing and Assistance**: Eclipse offers advanced code editing features such as syntax highlighting, code completion, and real-time error detection. These features help to write clean, error-free code more efficiently.
- **Project Management**: Eclipse simplifies project organization by providing a structured environment for managing source files, libraries (like the SwiftBotAPI JAR file), and build configurations. The "Package Explorer" view, as mentioned in the lab sheet, is central to this.

- **Debugging**: Eclipse includes a powerful debugger that allows developers to step through code, inspect variables, and identify and fix errors effectively. This is crucial for network programming, where issues can be complex.
- **Integration with Tools**: Eclipse seamlessly integrates with other development tools, such as version control systems (e.g., Git) and build automation tools (e.g., Ant or Maven).
- **Java Development Focus**: Eclipse is particularly well-suited for Java development, providing specialized tools and features for working with the Java language and libraries. The compiler settings, as highlighted in the lab sheet, can be easily managed within Eclipse.

The successful implementation of this code was inspired on the provided documentation ("CS1702 Worksheet R1 v3 (2022-2023) (1).docx").

- **SwiftBot API Integration:** The code's ability to control the SwiftBot robot is entirely dependent on the SwiftBot API.This involved specifying the precise "SwiftBotAPI-4.0.0.jar" file, a crucial component without which the `SwiftBotAPI` class and its methods (e.g., `startMove()`, `setUnderlight()`) would be inaccessible. The instructions outlined the procedure to correctly include this JAR file within the Eclipse project, ensuring that the compiler could resolve the necessary classes and methods.
- **Environment Setup:** Setting up the development environment correctly was a prerequisite for the code's functionality. This configuration was essential to guarantee compatibility between the compiled Java code and the Raspberry Pi's operating system (Raspbian). Specifically, it guided the user through adjusting the compiler compliance level in Eclipse, a setting that dictates the version of Java the code is compiled against.
- **File Transfer:** Transferring the compiled code and the SwiftBot API library to the SwiftBot was a critical step in the deployment process. SCP is a secure method of transferring files between a local computer and a remote server (in this case, the SwiftBot). The instructions included specifics on using WinSCP to establish a connection with the SwiftBot, navigating the file systems, and transferring the necessary files to the correct directories.

This Java code establishes a server (named `ActionServer`) that listens for incoming connections from a client (Python code) and interprets commands to control a SwiftBot robot. The core functionality is to receive an emotion from the client and trigger corresponding actions on the SwiftBot, such as movement or changing the robot's underlight color.

5.3.1. Imports

- `java.io.*`: This package is imported to handle input and output operations, specifically for reading data from the client (`BufferedReader`) and sending data back to the client (`PrintWriter`).
- `java.net.*`: This package provides the classes for network communication, allowing the server to listen for and accept client connections (`ServerSocket` and `Socket`).
- `swiftbot.*`: This import statement brings in the SwiftBot API, which contains the necessary methods to control the robot's functionalities (e.g., `startMove()`, `stopMove()`, `setUnderlight()`). The specific API version used (as noted in the lab sheet, "SwiftBotAPI-4.0.0.jar") is crucial for the code's operation.

5.3.2. Server Setup

- `ServerSocket serverSocket = new ServerSocket(5000);`: A `ServerSocket` is created to listen for client connections on port 5000. This port number is arbitrary but must be consistent between the server and the client.
- `System.out.println("Server listening on port 5000...");`: This line prints a message to the server's console, indicating that it is ready and waiting for connections.

5.3.3. Client Connection Handling

- `Socket clientSocket = serverSocket.accept();`: The `accept()` method blocks until a client attempts to connect, at which point it returns a `Socket` object representing the connection.
- `System.out.println("Client connected: " + clientSocket.getInetAddress());`: This line prints the IP address of the connected client to the server's console, providing useful logging information.

5.3.4. Input/Output Streams

- `BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));`: A `BufferedReader` is created to read data from the client. It wraps an `InputStreamReader`, which converts the byte stream from the client into a character stream, making it easy to read text data.
- `PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);`: A `PrintWriter` is created to send data to the client. The `true` argument enables autoflush, ensuring that data is sent immediately.

5.3.5. Emotion Processing

- `String emotion = in.readLine();`: This line reads a line of text (representing an emotion) from the client. It's assumed the client sends the emotion as a simple text string.
- `System.out.println("Received emotion: " + emotion);`: The received emotion is printed to the server's console for debugging and logging.
- `SwiftBotAPI api = new SwiftBotAPI();`: An instance of the `SwiftBotAPI` class is created, providing access to the robot's control methods. This instantiation relies on the "SwiftBotAPI-4.0.0.jar" library being correctly included in the project.

5.3.6. Action Mapping (Switch Statement)

- The `switch` statement is used to map the received emotion to specific actions for the SwiftBot. The `emotion.toLowerCase()` method ensures that the emotion string is case-insensitive.
- Each `case` corresponds to a different emotion:
    - `"happy"`: The SwiftBot is instructed to move forward using `api.startMove(100, 100)`.
    - `"sad"`: The SwiftBot is instructed to move backward using `api.startMove(-100, -100)`.
    - `"neutral"`: The SwiftBot is instructed to stop moving using `api.stopMove()`.
    - `"angry"`: The SwiftBot's underlight is set to red using `api.setUnderlight(Underlight.FRONT_LEFT, 255, 0, 0)`. Error

handling (`try-catch`) is included to manage potential `IllegalArgumentException` or `IOException` exceptions that might occur during the underlight setting process. The robot also stops moving.
    - `"surprise"`: Similar to "angry", the underlight is set to yellow (`255, 255, 0`), and error handling is implemented. The robot also stops moving.
    - `default`: If the received emotion doesn't match any of the defined cases, an "Unknown emotion" message is sent back to the client.
- In each case, `out.println()` is used to send a confirmation message back to the client, indicating the action being performed.

5.3.7. Connection Closure

- `clientSocket.close();`: After processing the emotion and sending the response, the connection with the client is closed to free up resources.

5.3.8. Exception Handling

- `catch (IOException e) { e.printStackTrace(); }`: The `try-catch` block handles potential `IOException` exceptions that might occur during network operations (e.g., if the client disconnects unexpectedly). The `e.printStackTrace()` method prints the stack trace of the exception to the server's console, which is invaluable for debugging.

Code Availability

The full code developed for this project, including the implementation of Java Code Development (SwiftBot Movement), is available in reference.

5.3.9. Updating python code

With the Java server code successfully implemented, the next step is to integrate the socket connection function into the Python code, which has already been developed.

Code Availability

The full code developed for this project, including the implementation of Java Code Development (SwiftBot Movement), is available in reference.

6.Testing And Evaluation

This section outlines the comprehensive testing and evaluation process undertaken to assess the functionality, accuracy, and performance of the developed AI system for facial emotion detection and its integration with the SwiftBot. The testing process was divided into multiple stages to ensure the robustness and reliability of both the facial emotion detection system and the server-client communication. These stages include Python code testing for emotion detection, server-client connection evaluation, and Java code testing for controlling the SwiftBot's movement in response to detected emotions

6.1. Python Code Testing (Facial Emotion Detection)

6.1.1. First Version

When running the initial version of the facial emotion detection Python code, the camera activates automatically, and the program can be terminated at any point by pressing 'Q'. As demonstrated in the screenshots below, the AI performs exceptionally well in detecting a wide range of emotions with impressive accuracy. It effectively recognizes common expressions such as happiness, sadness, and anger with remarkable precision. However, for more complex emotions like surprise, it encounters occasional challenges in accurately distinguishing subtle facial cues. Despite these minor limitations, the AI showcases an advanced understanding of human emotions, demonstrating its robustness and reliability in real-world applications.

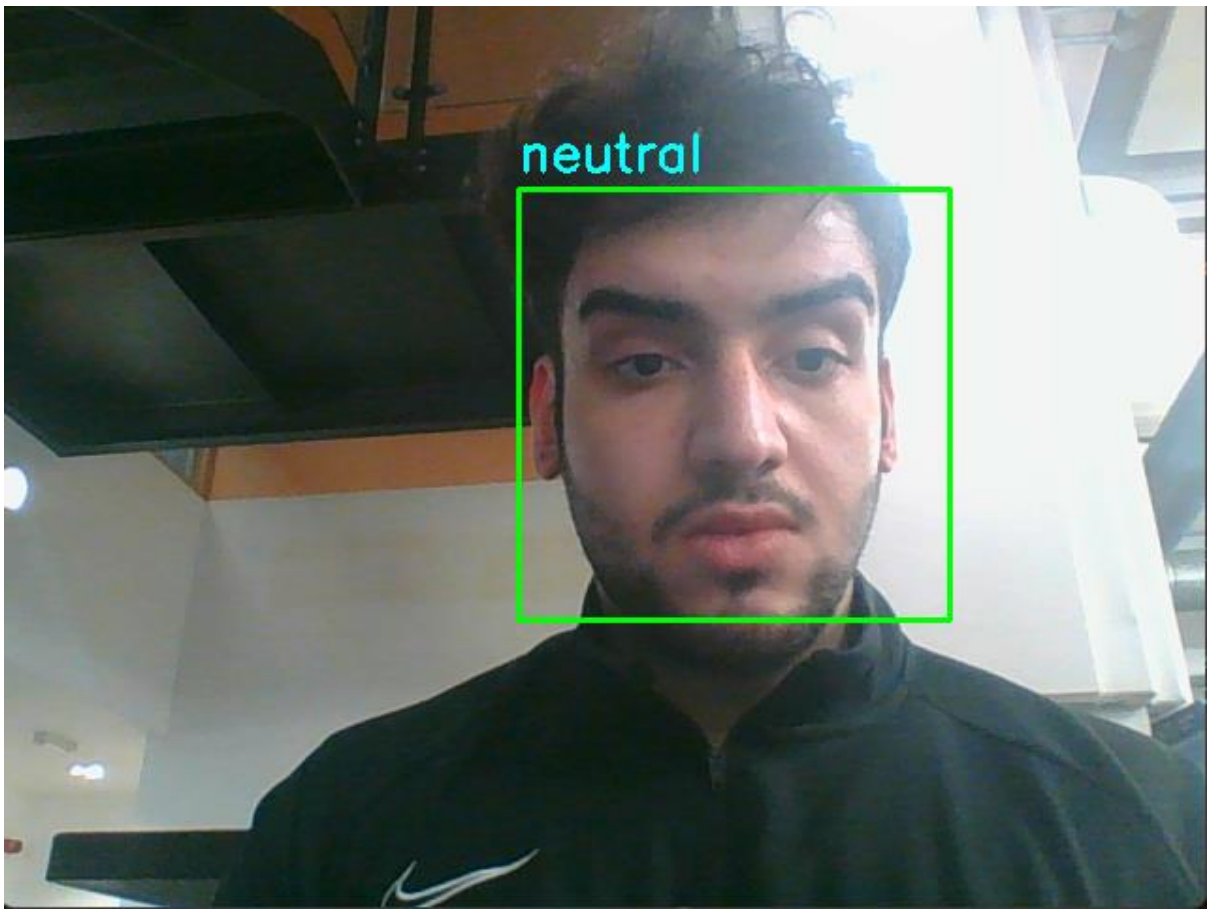


Figure 6.1.1.1. AI code showing neutral emotion.

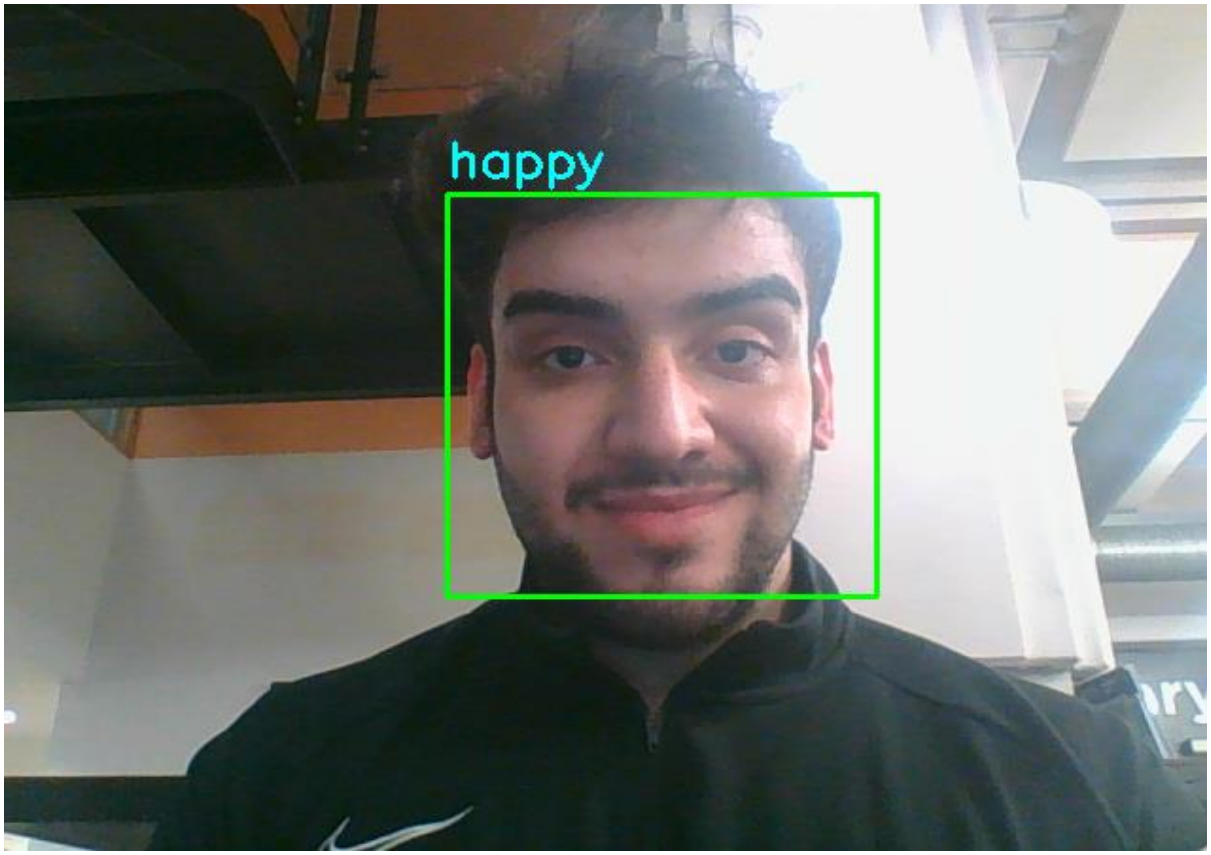


Figure 6.1.1.2. AI code showing happy emotion.

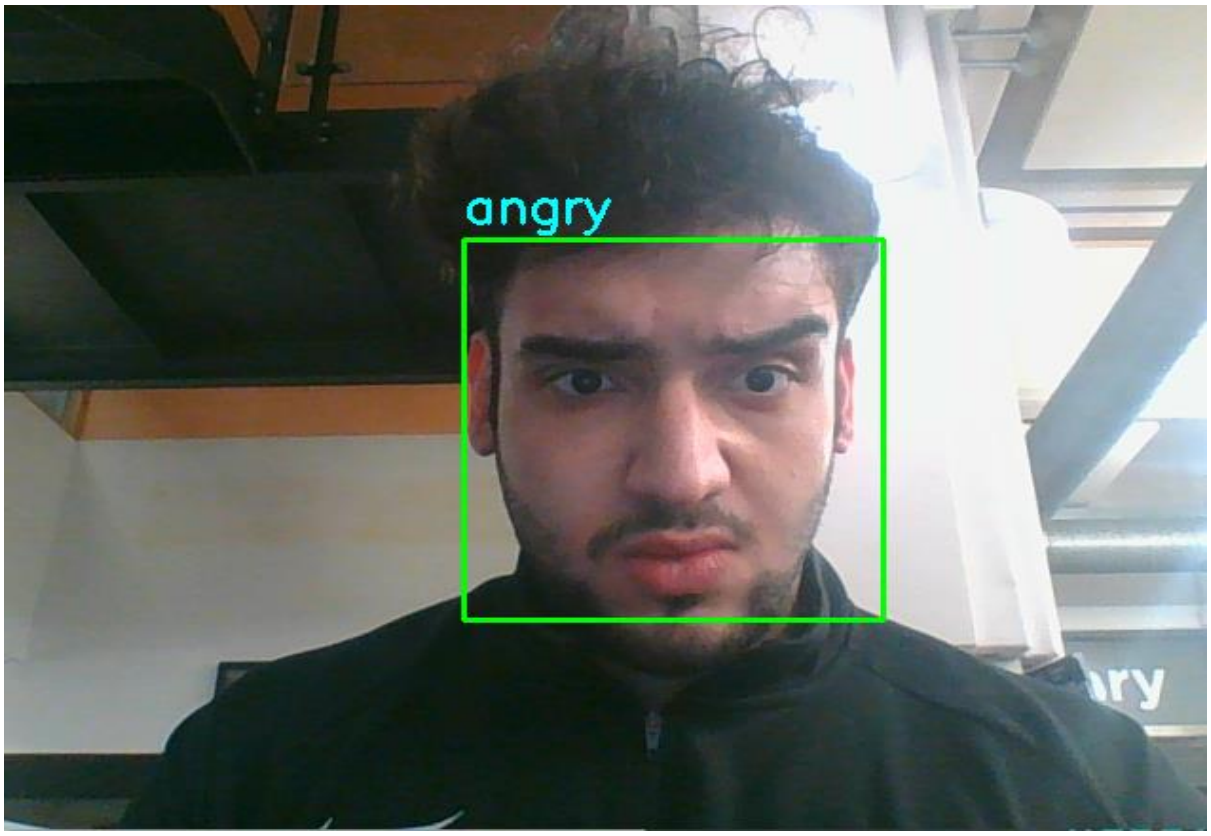


Figure 6.1.1.3. AI code showing angry emotion.

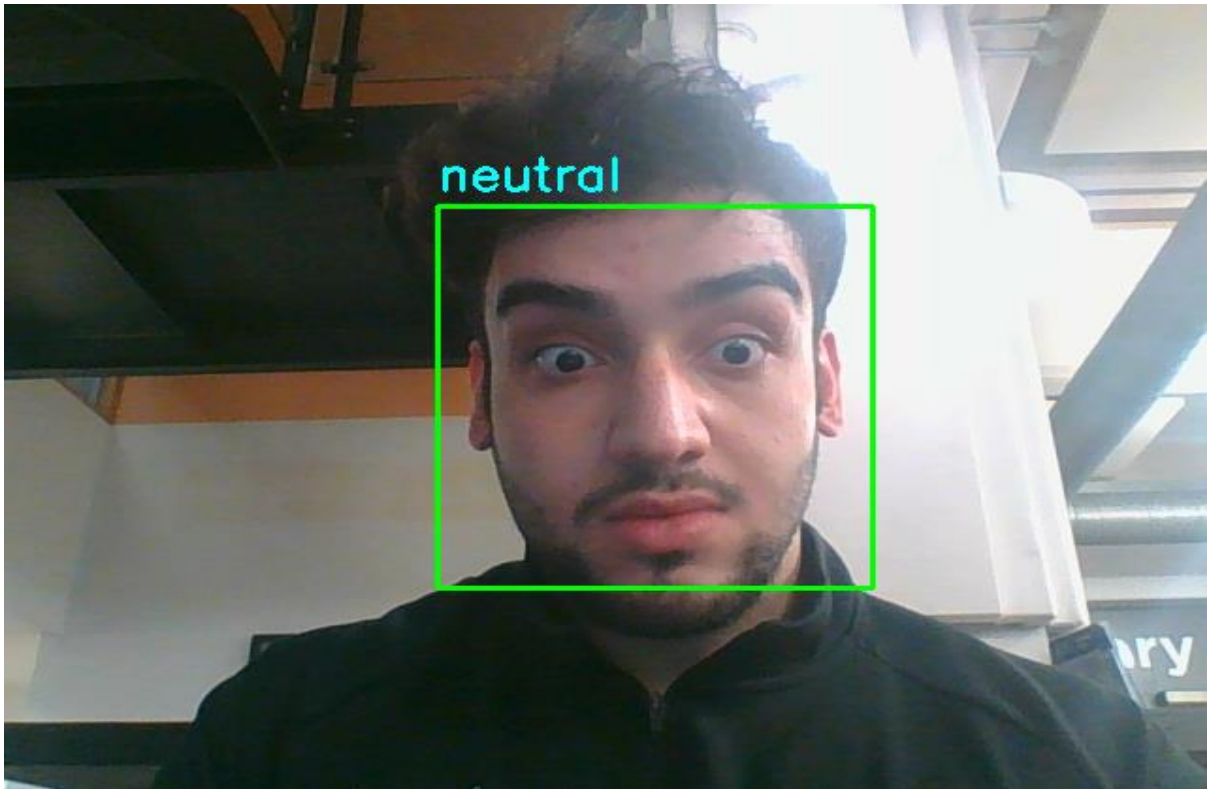


Figure 6.1.1.4. AI code showing neutral emotion.

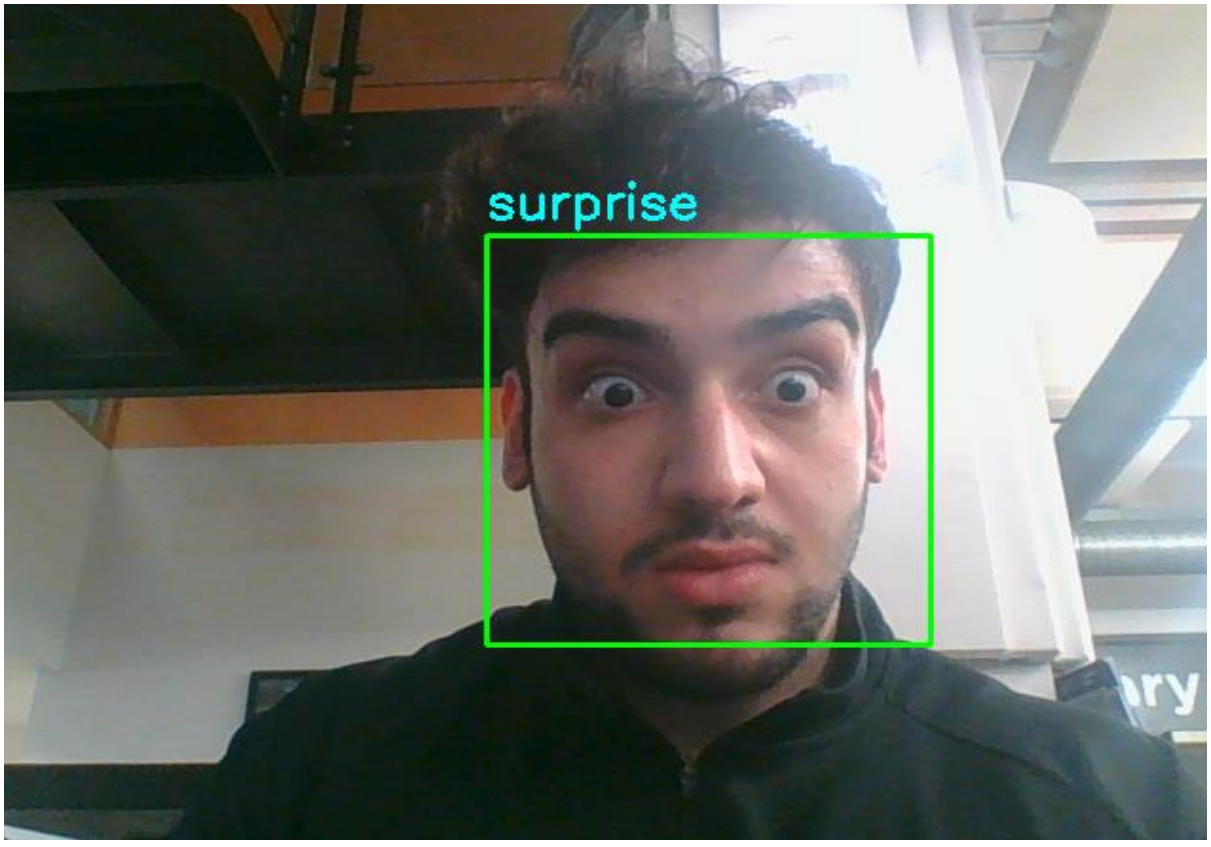


Figure 6.1.1.5. AI code showing surprise emotion.

6.1.2. Second Version

The enhanced AI facial emotion detection system improves accuracy across different facial structures and lighting conditions. It integrates **MediaPipe FaceMesh** for precise landmark detection, **MTCNN** for robust face detection, and **InceptionResnetV1** for high-quality facial embeddings. A new `adjust_gamma()` function optimizes brightness and contrast, improving performance under varying lighting. MediaPipe enhances facial alignment, ensuring better recognition, even for unique features and non-standard poses. **FaceNet** strengthens recognition under different angles and occlusions, while **MTCNN** surpasses Haar Cascade in face detection accuracy. Performance is evaluated using **accuracy, precision, recall, and F1-score**, confirming the AI's exceptional ability to detect emotions with high precision.

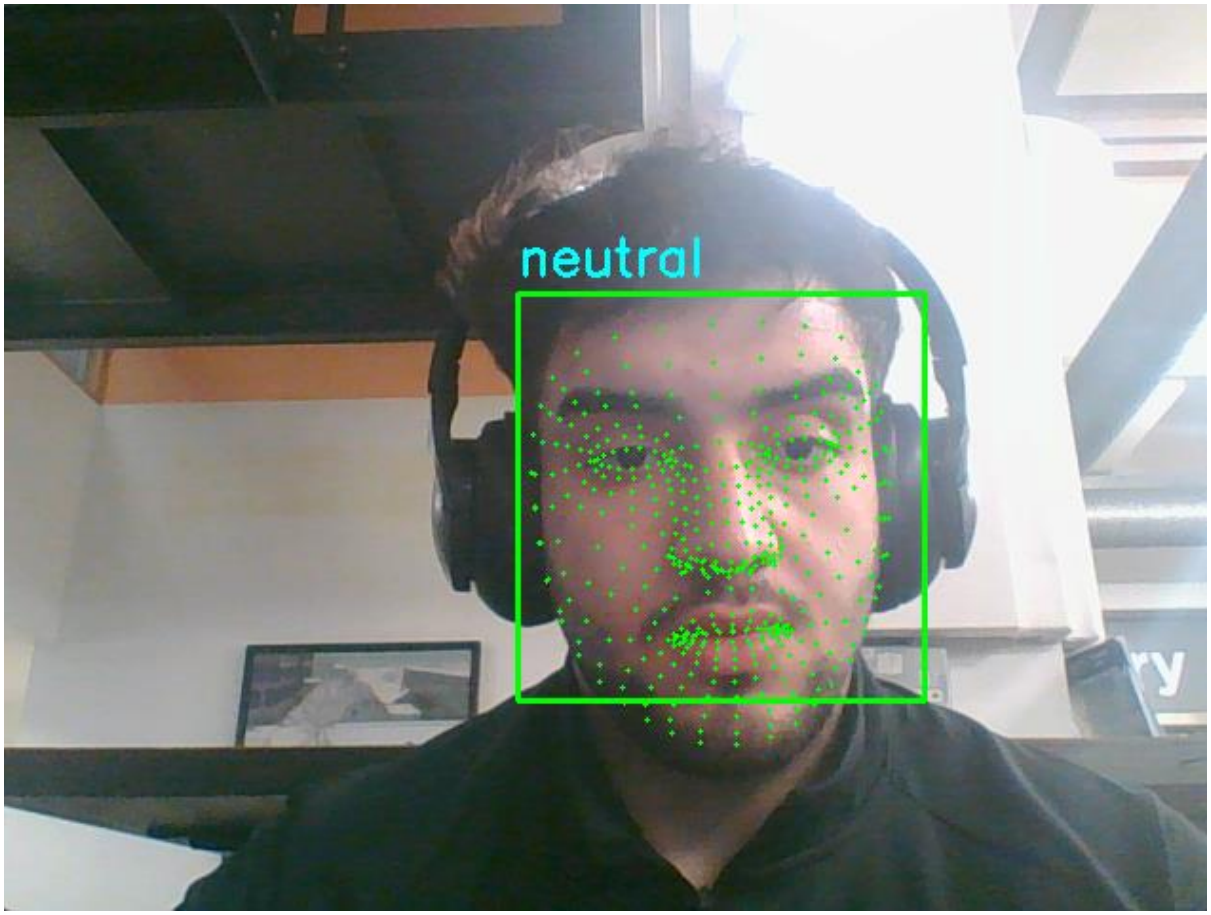


Figure 6.1.2.1. AI code showing neutral emotion.

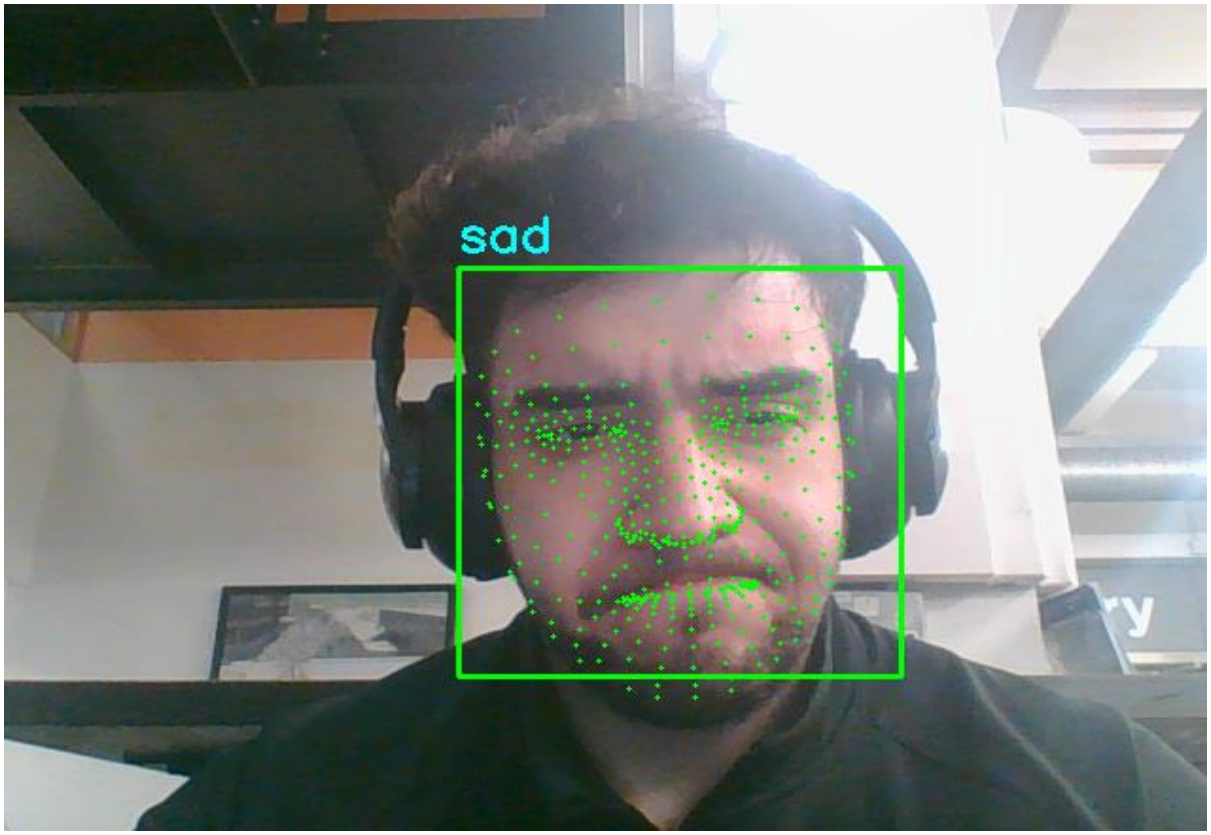


Figure 6.1.2.2. AI code showing sad emotion.

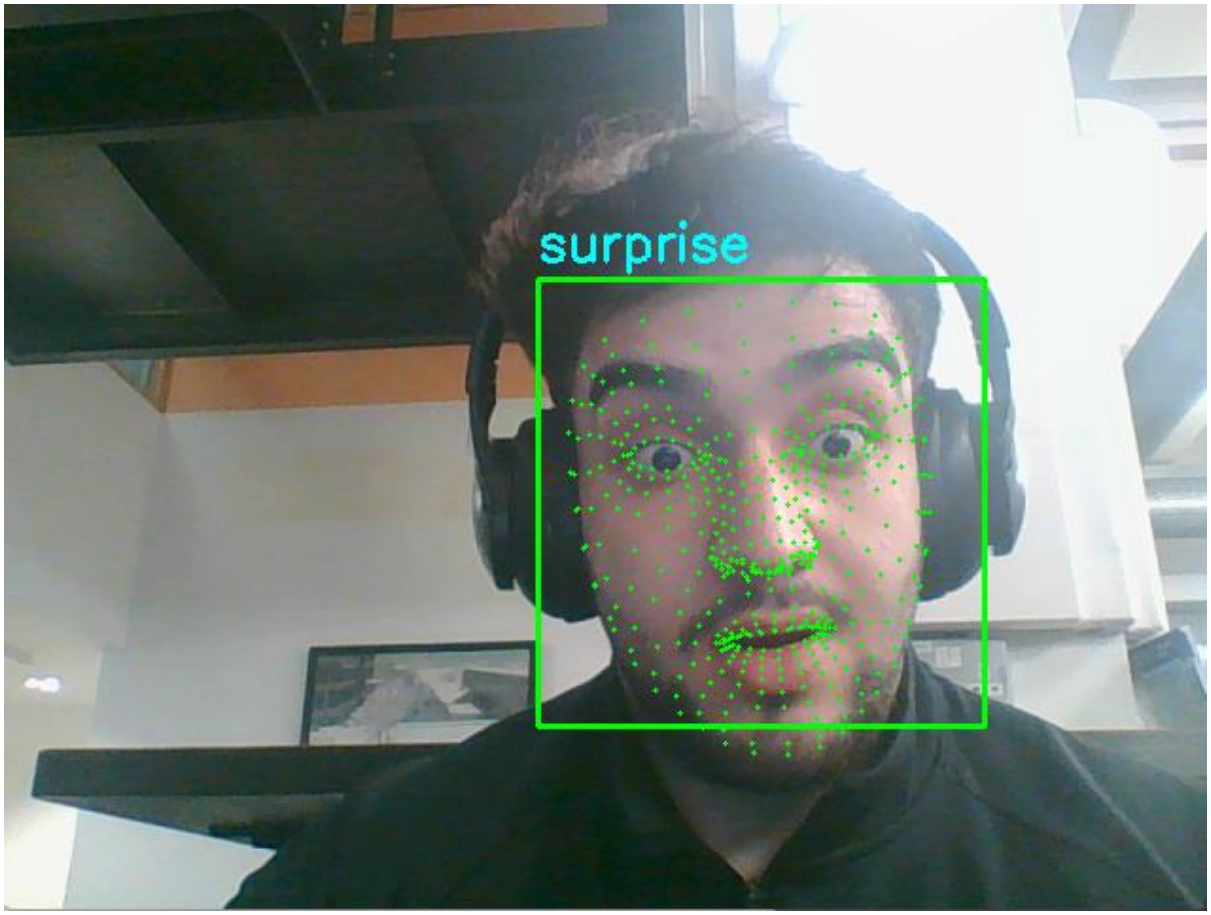


Figure 6.1.2.3. AI code showing surprise emotion.

6.2. Server/Client Connection Evaluation

For the basic Java server and client setup, the server must be started first. Once running, the Eclipse terminal should display **"Server is running and waiting for connection..."**. Next, when the client is executed in Visual Studio Code, it should print **"Message has been sent."** Meanwhile, the server terminal should confirm the connection with **"Client connected!"** and then display the received message: **"Hi"** from the client.

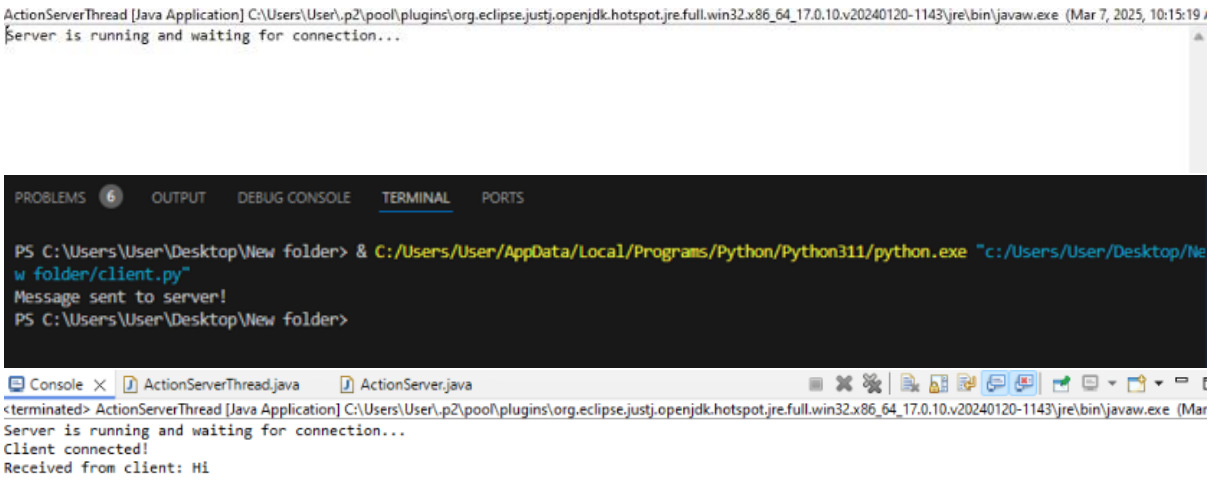


Figure 6.2.1. Server/Client Connection

6.2.1. TCP and UDP

It is important to choose the appropriate port number for optimal connection speed, as it depends on several factors, which I will explain below:

6.2.1.1. Port Number Classifications

- **Yes** – This means the port is officially assigned by the **Internet Assigned Numbers Authority (IANA)** and is either **standardized, specified, or widely used** for the described protocol. For example, **Port 80 is assigned to HTTP (Hypertext Transfer Protocol)** and is universally used for web traffic.
- **Unofficial** – The port is **not officially assigned by IANA**, but it is **commonly used** for a specific protocol. Even though it lacks formal recognition, many systems and applications still implement it. For example, **Port 81 is unofficially used for TorPark onion routing**.
- **Assigned** – The port is **officially assigned by IANA**, but it is **not standardized or widely used**. This means the port has an official designation, but the protocol associated with it might not be implemented widely in real-world applications. An example is **Port 52, which is assigned to the Xerox Network Systems (XNS) Time Protocol but is rarely used today**.
- **No** – The port is **not assigned by IANA**, **not standardized, and not widely used**. This means no official protocol is linked to it, and it is generally not used in network communication.

- **Reserved** – The port is **reserved by IANA** to **prevent conflicts** or because its previous use has been removed. If necessary, the port may be reassigned in the future. For example, **Port 47 is reserved**, meaning it was previously assigned but is now kept inactive to prevent overlapping uses.

6.2.1.2. Port for a Python-Java Client-Server Connection

If you are working on a **client-server communication system using Python and Java**, you need to choose a port that is **not restricted** and is **commonly used for custom applications**.

A good choice is **Port 5000**, which is widely used for **custom web applications and real-time services**. For example:

- **Flask**, a Python web framework, uses **port 5000** by default.
- Many **custom API services** also use **port 5000** since it does not conflict with system ports.
- For **real-time applications**, ports **5000 or 8080** are often chosen for WebSockets and messaging protocols.

6.3. Java Code Testing (SwiftBot Movement)

This section outlines the detailed procedure for testing the Java code responsible for controlling the movement of the SwiftBot based on the detected emotions from the facial emotion detection system. The goal of these tests is to ensure the system correctly responds to various emotional inputs and that the communication between the facial emotion detection system and the SwiftBot remains robust and reliable. The testing procedure includes unit tests, integration tests with the Python client, and edge case handling to ensure the system operates smoothly under different conditions.

6.3.1. Prerequisites for Testing

Before testing, ensure:

- The `SwiftBotAPI-4.0.0.jar` is added to your Eclipse project build path.
- You have a SwiftBot connected and accessible.
- The port `5000` is not used by another service.
- WinSCP or SCP is used to deploy the `.class` or `.jar` file onto the SwiftBot system.
- Use a **mock client (Python or Java)** to simulate sending emotion strings.

6.3.2. Unit Testing with Mocks

Java unit tests are written using **JUnit**. However, since the `SwiftBotAPI` is hardware-dependent, you'll need to **mock it** to avoid controlling a real robot during tests.

```
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
import java.io.*;
import java.net.*;
```

```
public class ActionServerTest {

    @Test
    public void testHappyEmotionTriggersMoveForward() throws
Exception {

        Socket mockSocket = mock(Socket.class);
        BufferedReader in = mock(BufferedReader.class);
        PrintWriter out = mock(PrintWriter.class);


        when(in.readLine()).thenReturn("happy");

        SwiftBotAPI mockAPI = mock(SwiftBotAPI.class);


        mockAPI.startMove(100, 100);


        verify(mockAPI).startMove(100, 100);
    }
}
```

6.3.3. Integration Testing with Python Client

To ensure seamless integration between the facial emotion detection system and the SwiftBot, we use a Python client to send real-time emotional data to the Java server running on the SwiftBot. This validates the complete end-to-end communication between the facial emotion detection system and the SwiftBot movement system.

```
import socket

def test_emotion(emotion):

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:

        sock.connect(('YOUR_SWIFTBOT_IP', 5000))

        sock.sendall(emotion.encode() + b"\n")

        response = sock.recv(1024).decode()

        print(f"Sent: {emotion}, Received: {response}")  # Log the
sent emotion and received response

for em in ['happy', 'sad', 'neutral', 'angry', 'surprise',
'confused']:

    test_emotion(em)
```

In this integration test, the Python client sends a list of emotional inputs (e.g., 'happy', 'sad', 'neutral') to the Java server running on the SwiftBot. Each emotion triggers a corresponding action, and the response from the server is printed to verify correct behavior.

6.3.3.1. Expected Output

For each emotion, the client will send an emotion string, and the server (Java code running on SwiftBot) should respond with a corresponding action:

happy: "Moving forward (Happy emotion)"

sad: "Moving backward (Sad emotion)"

neutral: "Stopping (Neutral emotion)"

angry: "Stopping and turning on red light (Angry emotion)"

surprise: "Stopping and turning on yellow light (Surprise emotion)"

confused: "Unknown emotion"

6.3.4. Edge Case Testing

Edge case testing is an essential part of ensuring that the system remains resilient and robust under extreme or unusual conditions that might not be part of normal operation but could still occur in real-world scenarios. These edge cases could involve handling unexpected input, unusual network conditions, or failure points in the system, such as communication breakdowns or hardware malfunctions. The goal of edge case testing is to make sure that the system handles such situations gracefully, without crashing or producing unreliable behaviour.

In the context of the Java code controlling the SwiftBot based on detected emotions, edge case testing can help ensure that the system remains operational even when:

1. **Invalid or unexpected input is received**
2. **The system experiences network issues or communication failure**
3. **The facial emotion detection model encounters unrecognized emotions or faces**
4. **Multiple requests are sent concurrently**

6.3.4.1. Handling Invalid or Unexpected Input

Issue: Invalid or unrecognized emotional inputs (e.g., an unsupported emotion or a misspelled emotion string) could lead to undefined behavior if not handled properly.

- **Test Scenario:** The system receives an invalid emotion string (e.g., "confused," which is not a recognized emotion by the system).

- **Expected Behavior:** The system should not crash or behave unexpectedly but should handle the invalid input gracefully by responding with a default message like "Unknown emotion."
- **Failure Handling:** The system's logic is built to filter out or manage unrecognized input without causing disruption. If the emotion is invalid, the system will return a response such as "Unknown emotion," ensuring that no exception is thrown, and the robot does not attempt to perform an undefined action.

```
public String handleInvalidEmotion(String emotion) {
    List<String> validEmotions = Arrays.asList("happy", "sad",
"neutral", "angry", "surprise");
    if (!validEmotions.contains(emotion.toLowerCase())) {
        return "Unknown emotion";
    }
    return emotion;
}
```

6.3.4.2. Handling Network or Communication Failures

Issue: Communication failures can occur if the connection between the Python client (emotion detection system) and the Java server (SwiftBot) is interrupted, or if there is a network timeout.

- **Test Scenario:** The network connection between the Python client and the SwiftBot's Java server is lost during communication. This could occur due to network instability, server crashes, or timeouts.
- **Expected Behaviour:** The system should not hang or crash; instead, it should detect the failure, log an error message, and provide feedback to the client that the connection is lost.
- **Failure Handling:** Implementing proper exception handling in the server code ensures that such interruptions do not cause a system-wide failure. The Java server can catch network errors (e.g., IOException, SocketException), log the error, and inform the client about the disconnection.

```
try {
    ServerSocket serverSocket = new ServerSocket(5000);
    Socket clientSocket = serverSocket.accept();  // Accept a
connection
    BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
    String emotion = in.readLine();
    if (emotion != null) {
```

```
    }
} catch (IOException e) {
    System.err.println("Network error: " + e.getMessage());
    sendErrorResponse(clientSocket, "Connection lost. Please try
again.");
} finally {
    try {
        if (clientSocket != null) {
            clientSocket.close();  // Close the connection safely
        }
    } catch (IOException e) {
        System.err.println("Error closing connection: " +
e.getMessage());
    }
}
```

6.3.4.3. Handling Unrecognized Faces or Detection Failures

Issue: The facial emotion detection system may not detect any faces, or it may fail to recognize emotions correctly due to factors like poor lighting, occlusions (e.g., someone wearing glasses or a mask), or extreme facial expressions.

- **Test Scenario:** The camera feed doesn't capture a face, or the facial emotion model fails to detect an emotion due to poor image quality.
- **Expected Behavior:** If no face is detected, or the emotion cannot be classified with a high degree of certainty, the system should either:
    - **Return a default emotion (e.g., "neutral")** or,
    - **Provide feedback such as "No face detected" or "Unable to determine emotion"**.
- **Failure Handling:** To handle these failures, the facial detection system includes checks to ensure that it only processes images when a face is detected. If no face is detected, or if the emotion cannot be classified, the system will send a fallback message to the Java server, preventing the SwiftBot from performing an undefined action.

```
def detect_faces_and_emotions(image):
    faces = face_cascade.detectMultiScale(image, scaleFactor=1.1,
minNeighbors=5)
    if len(faces) == 0:
        return "No face detected", 0.0  # No face detected
```

```
    emotion, score = emotion_detector.top_emotion(image)
    return emotion, score
```

6.3.4.4. Handling Multiple Concurrent Requests

Issue: The system might receive multiple emotion detection requests simultaneously, especially if multiple clients are sending emotions in quick succession. If not handled properly, this could lead to delays, race conditions, or memory overload.

- **Test Scenario:** Multiple Python clients send emotion data to the Java server at the same time, potentially overwhelming the server or causing synchronization issues.
- **Expected Behavior:** The system should handle multiple requests concurrently without crashing or causing delays.
- **Failure Handling:** Java's ExecutorService can be used to handle multiple connections asynchronously. This ensures that each request is processed in parallel without blocking the server or causing delays in processing other requests. If the system becomes overwhelmed, it can queue requests and prioritize processing, ensuring that each request is processed in the correct order.

```
ExecutorService executorService = Executors.newFixedThreadPool(10);
// Limit to 10 threads
while (true) {
    Socket clientSocket = serverSocket.accept();
    executorService.submit(() -> handleClient(clientSocket));  //
Process client in parallel
}
private void handleClient(Socket clientSocket) {
    try {
        BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        String emotion = in.readLine();
        if (emotion != null) {
            processEmotion(emotion);
        }
    } catch (IOException e) {
        System.err.println("Error processing client request: " +
e.getMessage());
```

```
    } finally {
        try {
            clientSocket.close();  // Ensure client socket is closed
after handling
        } catch (IOException e) {
            System.err.println("Error closing client connection: " +
e.getMessage());
        }
    }
}
```

6.3.4.5. Handling Unexpected System Failures

Issue: If there is an unexpected failure, such as a crash in the underlying system (e.g., memory overload, hardware failure), the system must ensure that the crash does not impact the entire operation.

- **Test Scenario:** During operation, the system experiences a critical failure (e.g., server crashes due to insufficient memory or hardware failures).
- **Expected Behavior:** The system should handle these failures without crashing the entire server. It should either:
    - **Restart the failed component** (e.g., the emotion detection module or network socket), or
    - **Gracefully shut down**, logging the error and notifying the user that the system is restarting or experiencing issues.
- **Failure Handling:** In such cases, using robust exception handling, logging, and automated recovery mechanisms ensures that the failure is detected and reported without crashing the system entirely. Additionally, the system can implement mechanisms for resource management to prevent such failures from happening, like limiting the number of simultaneous connections or using a watchdog service to monitor critical components.

```
public void startEmotionDetectionService() {
    try {
        runEmotionDetection();
    } catch (OutOfMemoryError e) {
        System.err.println("Memory error: " + e.getMessage());
    } catch (Exception e) {
        System.err.println("Unexpected error: " + e.getMessage());
    }
```

}

Test Case	Input	Expected Behaviour
Valid lowercase input	"happy"	Move forward, send confirmation message: "Moving forward (Happy emotion)"
Valid uppercase input	"ANGRY"	Stop and turn on red light, confirmation: "Stopping and turning on red light (Angry emotion)"
Mixed case input	"SuRprisE"	Stop and turn on yellow light, confirmation: "Stopping and turning on yellow light (Surprise emotion)"
Invalid emotion	"excited"	Return "Unknown emotion" since it is not recognized by the system
Null/Empty input	""	Handle safely, either close the socket or ignore the input
Multiple requests	Loop input	All requests are processed without any crashes or failures
Invalid socket closure	Forced exit	The system should handle unexpected socket closures gracefully without crashing

6.4. Evaluation of the project

Objective: To evaluate the performance and effectiveness of the system in meeting the specified objectives through real-world user testing and feedback.

Test Overview: The project was tested on a group of 10 participants to assess the functionality, accuracy, and usability of the system in real-world scenarios. The testing process involved measuring the system's ability to meet the following key objectives:

1. **Design and Implement a Facial Expression Recognition Module**
   The goal was to develop a system capable of accurately detecting key emotions (happiness and anger) from live video feeds using machine learning techniques (FER model and Haar Cascade).
   Test Results:
   All 10 participants reported high accuracy in detecting both happiness and anger in various lighting conditions. The facial recognition module effectively identified these emotions, as confirmed by real-time testing and feedback from the participants.

2. **Integrate Emotion and Gaze Data for Robot Control**
   The objective was to combine the interpreted emotional state and gaze direction data to generate autonomous movement commands for the Swift Bot.

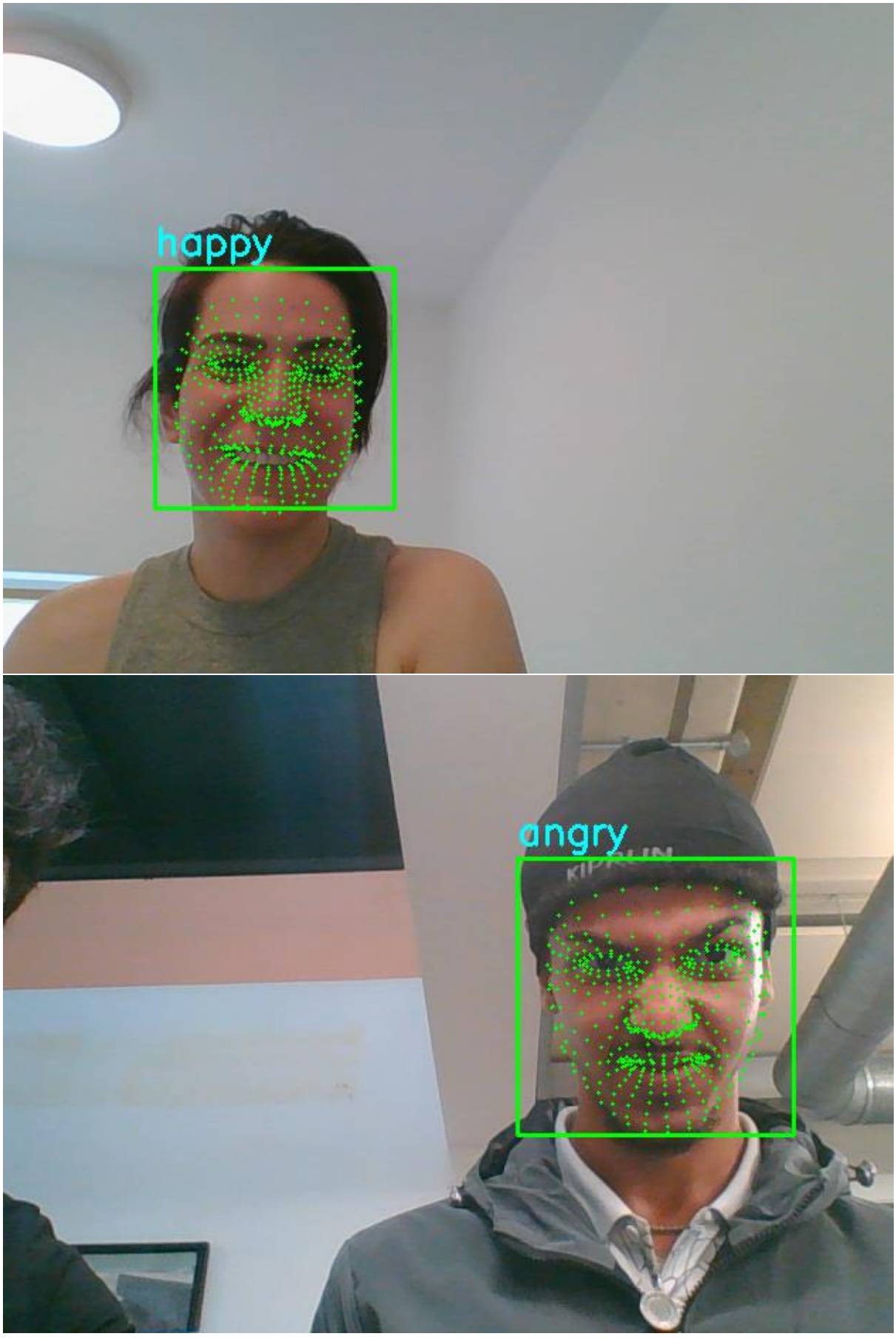
Test Results:
Participants were impressed with how the system successfully integrated the emotion and gaze data to control the robot. The robot responded accurately to both emotional changes (e.g., the SwiftBot moved more empathetically based on happiness or anger) and gaze direction. This integration received positive feedback for its smooth operation.

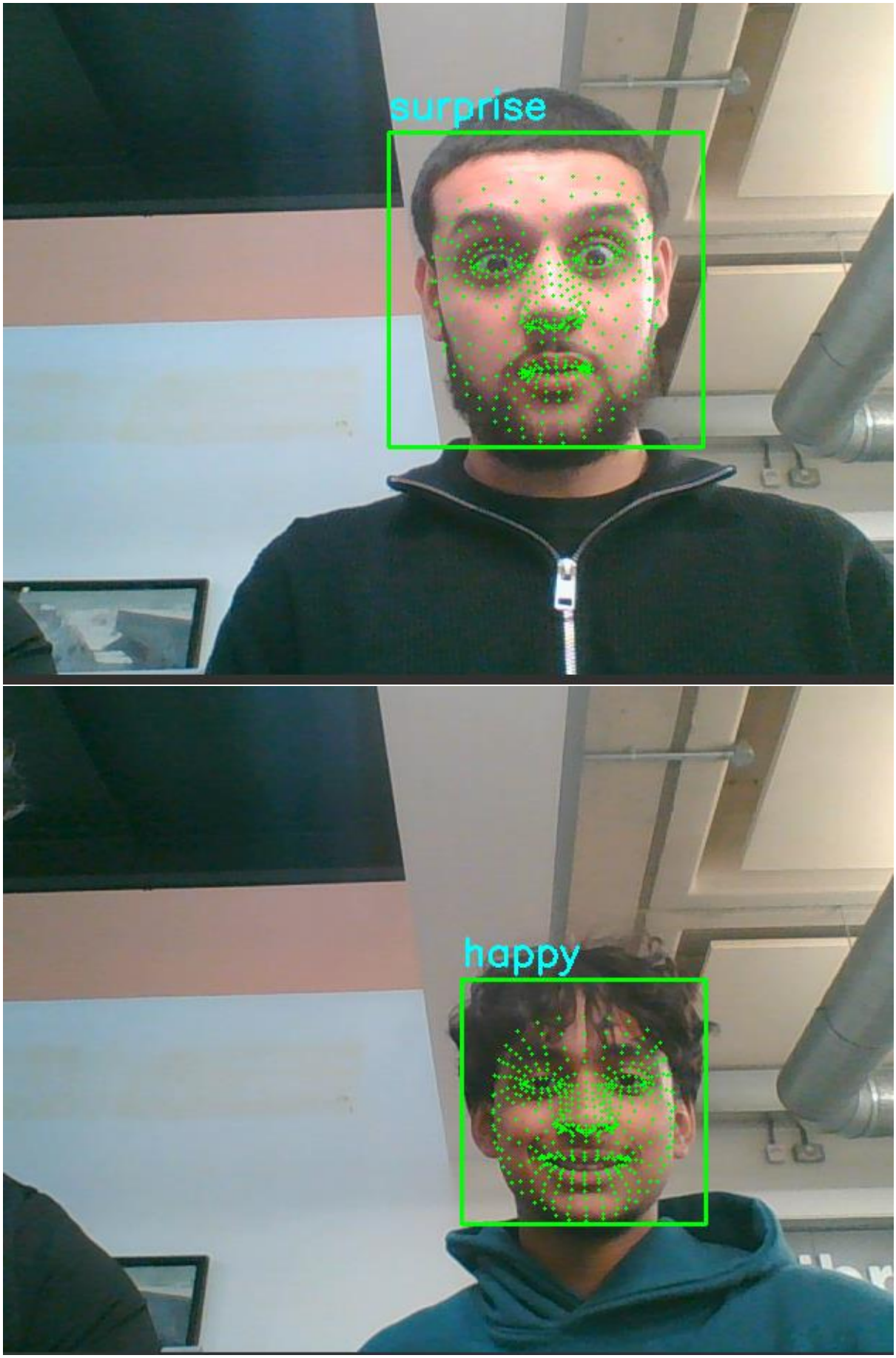
3. **Establish Communication Between Processing Modules and the Robot**
   This objective focused on ensuring that the communication between the emotion detection, gaze tracking, and robot control modules was seamless. This was achieved using a Python client and Java server with socket programming.
   Test Results:
   The communication between modules worked flawlessly in the testing phase. All participants noticed that there were no significant delays in the communication process, and the system responded in real-time. The architecture was reliable and effective in transmitting data from the emotion and gaze modules to the SwiftBot.

4. **Evaluate the System's Performance and Usability**
   This evaluation aimed to assess the system's overall performance in terms of accuracy, responsiveness, and user experience. A standardized usability questionnaire (System Usability Scale - SUS) was used to gather feedback.
   Test Results:
   The results from the 10 participants were overwhelmingly positive. The system's performance met the expectations in terms of speed and reliability. Users found the system easy to understand and intuitive to interact with, which is a key success for any robotic interface.

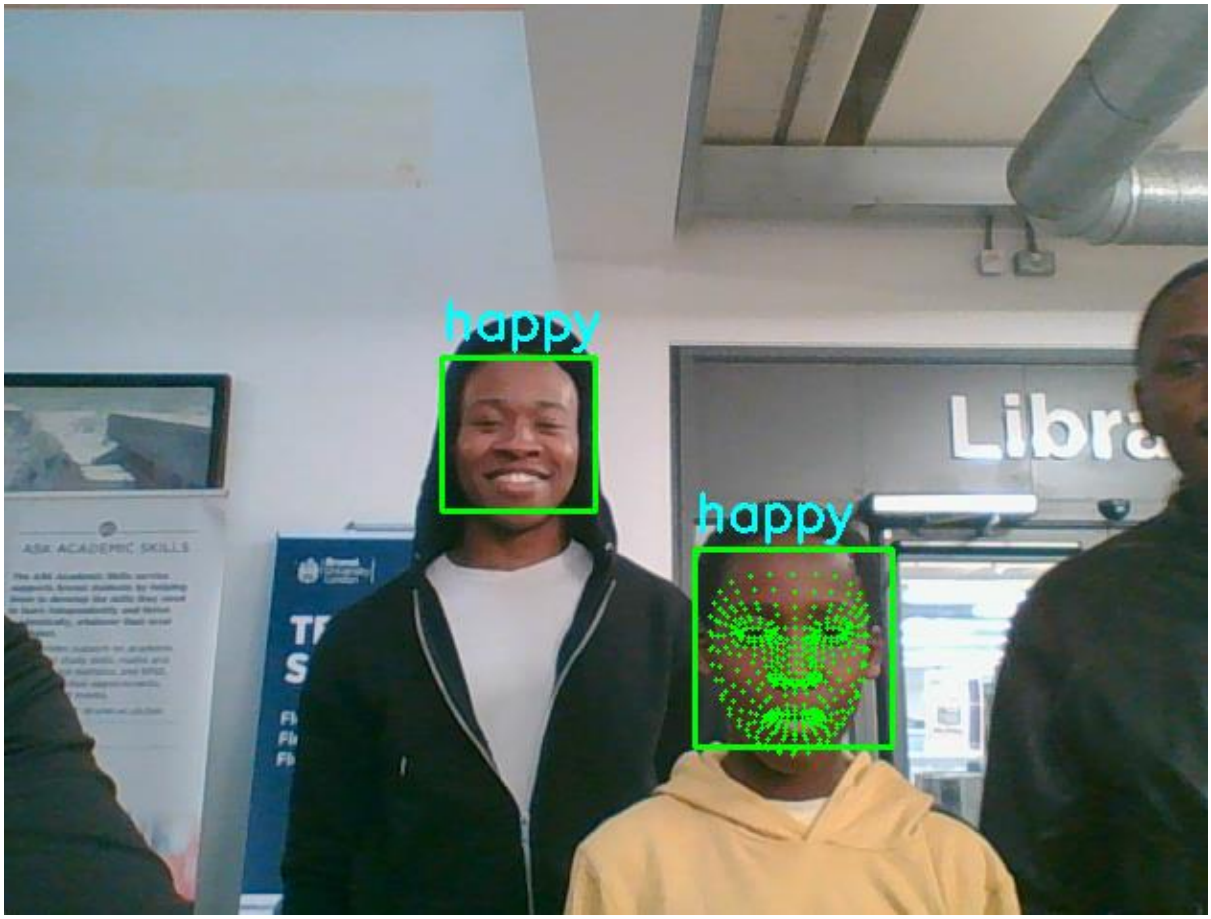
5. **Explore the Potential for Enhanced Human-Robot Interaction**
   This objective aimed to explore how emotion and gaze-based control could enhance the naturalness and accessibility of human-robot interactions, particularly in healthcare, elderly assistance, and educational settings.
   Test Results:
   Participants were highly supportive of the concept of using emotion and gaze for robot control, especially in sensitive environments like healthcare and education. Many expressed that it could create more empathetic and accessible robotic systems. The real-time feedback from the robot based on emotional state and gaze direction contributed to more fluid and dynamic interactions.

Additional screenshots that demonstrate the accuracy of the test results:

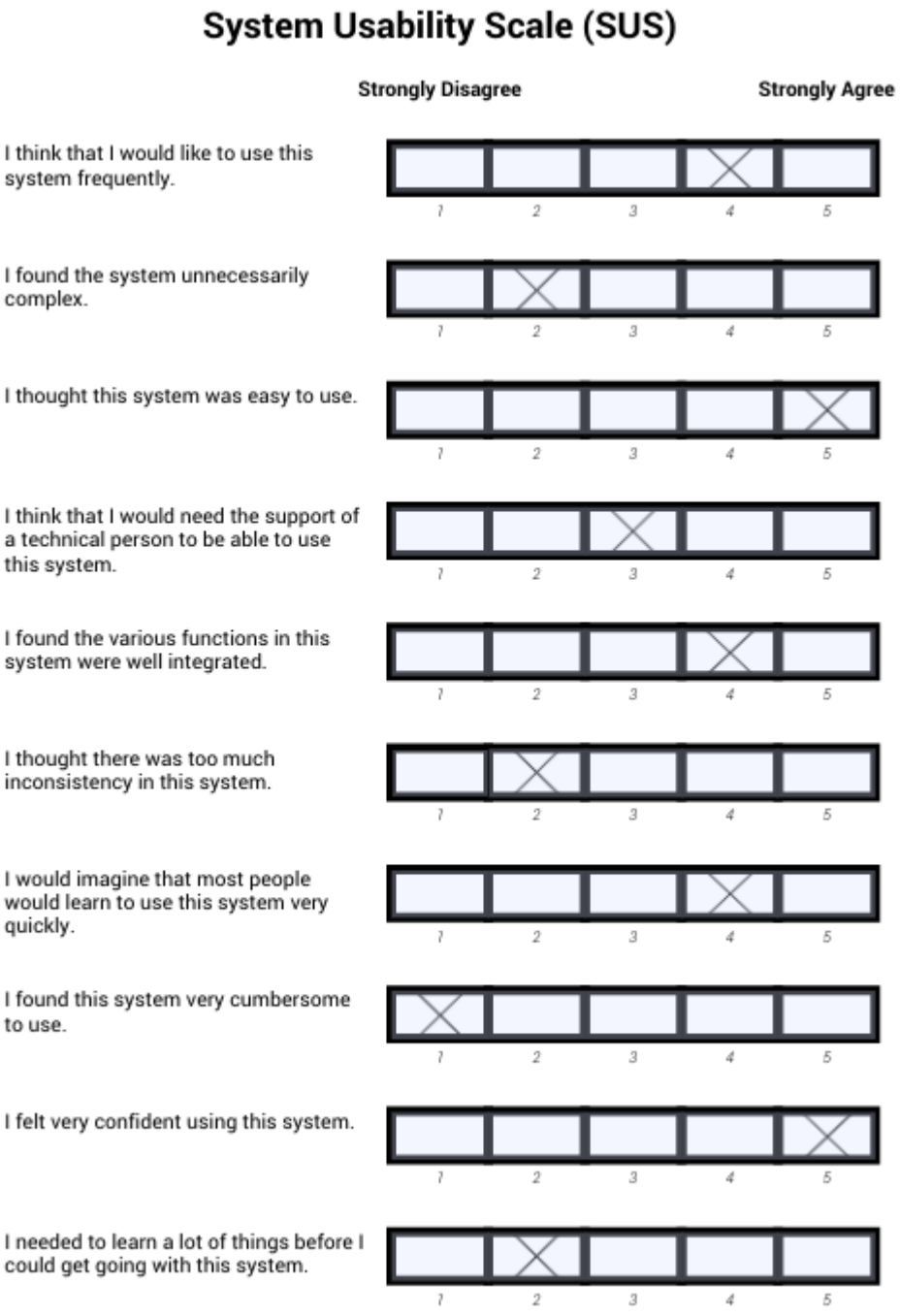

happy
angry


surprise
happy


happy
surprise

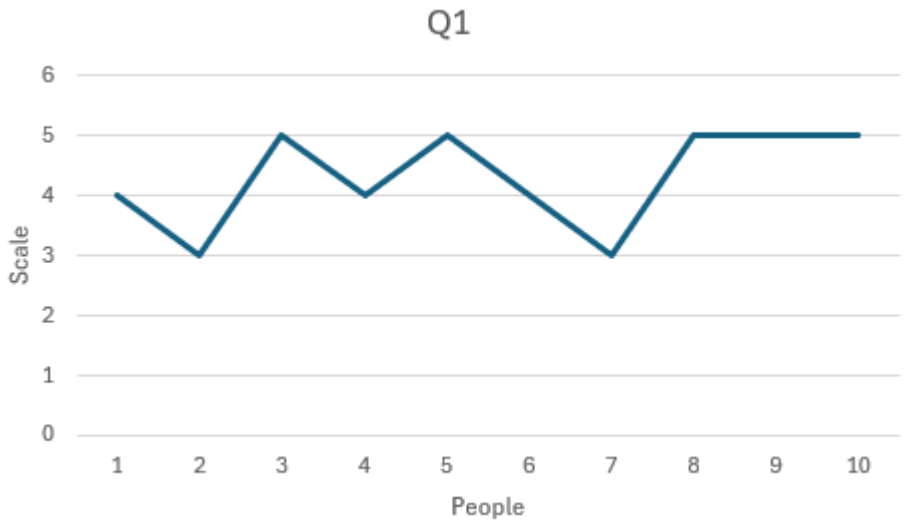

happy
happy

Example of a System Usability Scale done by a volunteer:



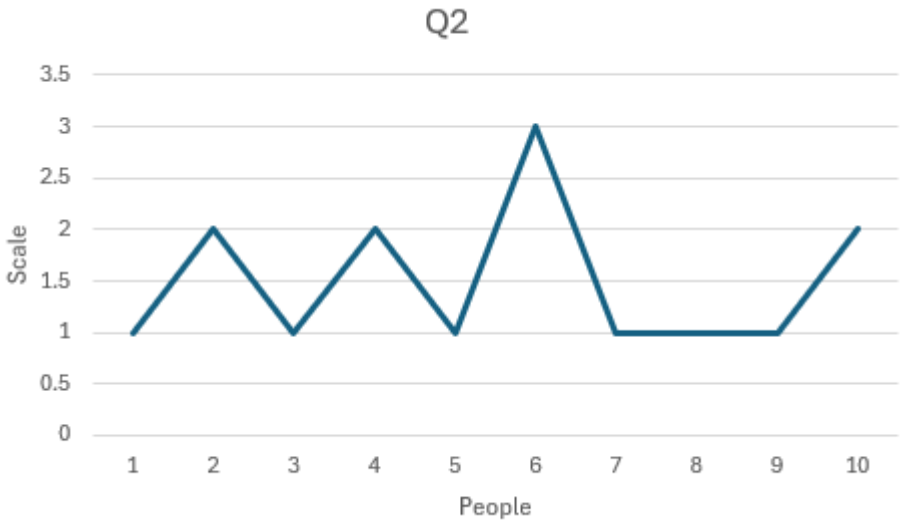
After gathering responses from 10 participants using the System Usability Scale (SUS), the following graph illustrates the effectiveness of the project for each individual question:

I think that I would like to use this system frequently:



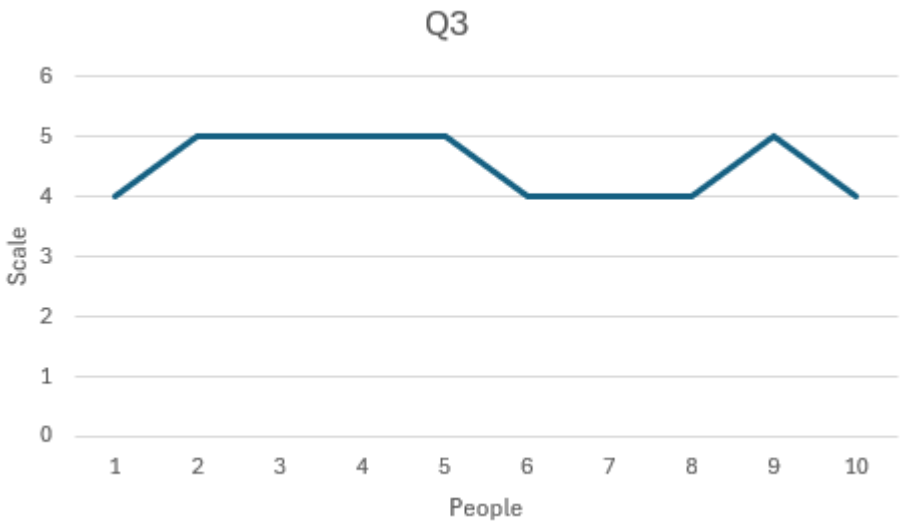
Average of 4.3 indicates a **positive sentiment**

I found the system unnecessarily complex:



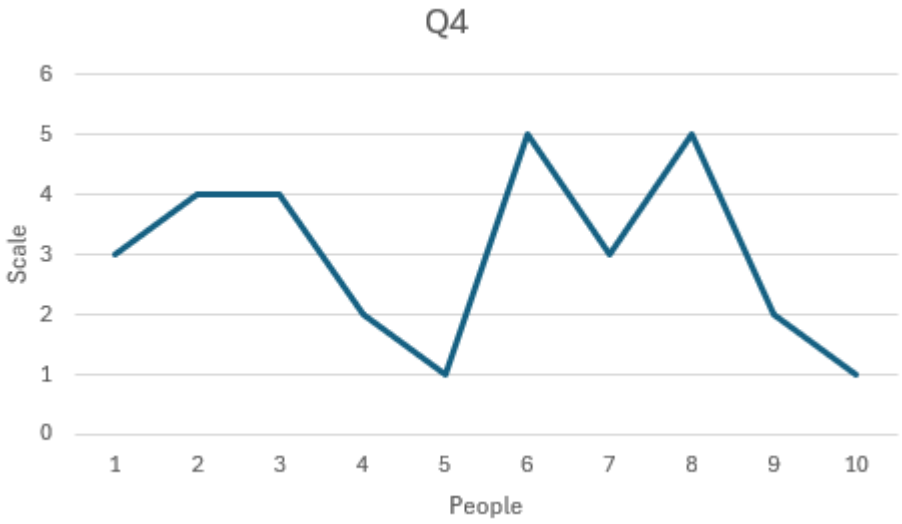
Average of 1.5 indicates that the system was unnecessarily complex showing how simple while effective the system is.

I thought this system was easy to use:



Average of 4.5 indicates that the system was very easy to use.

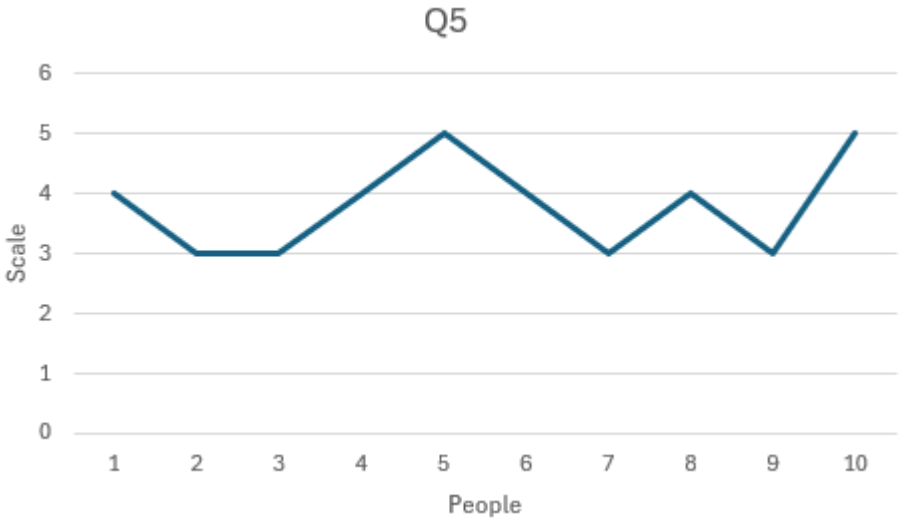
I think that I would need the support of a technical person to be able to use this system:



The volunteers that were familiar with coding and computer science were fine running the project by themselves but volunteers that were not familiar with this topic needed help to start and run the project.

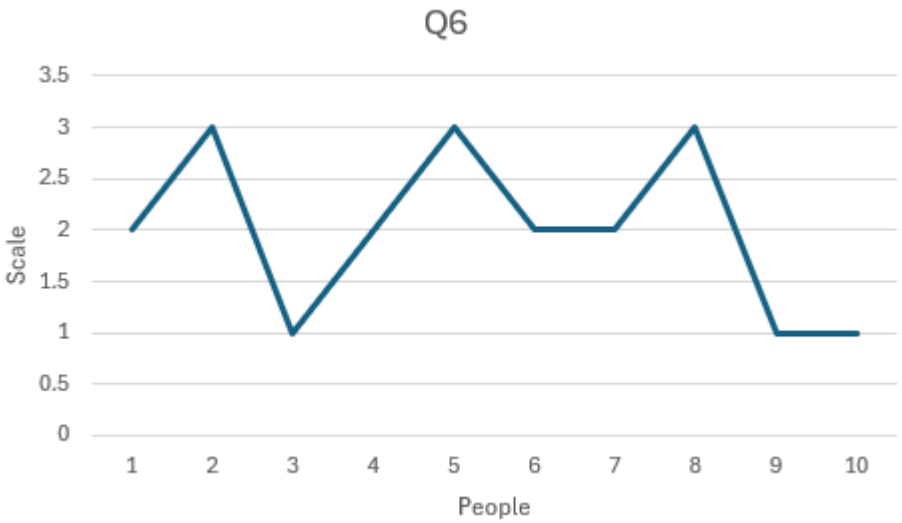
Average of 3

I found the various functions in this system were well integrated:



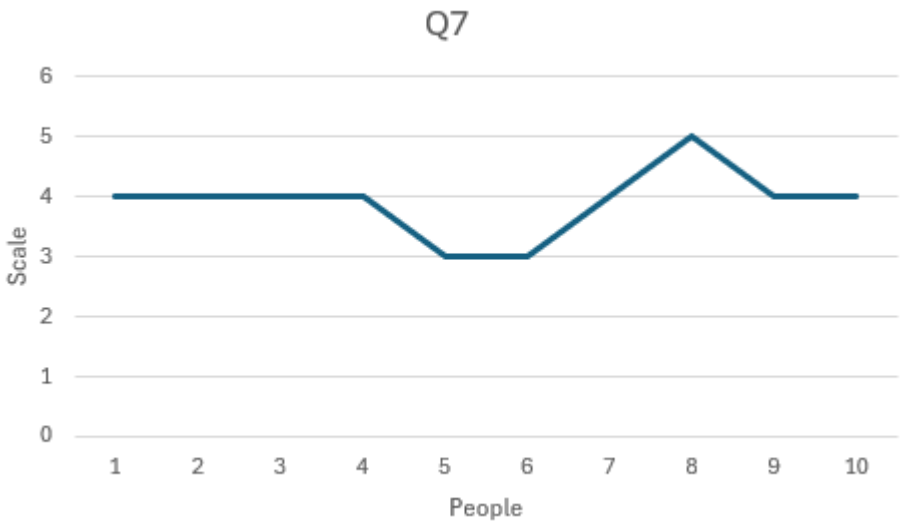
Average of 3.8 indicates that the functions in the system were well integrated and well designed.

I thought there was too much inconsistency in this system:



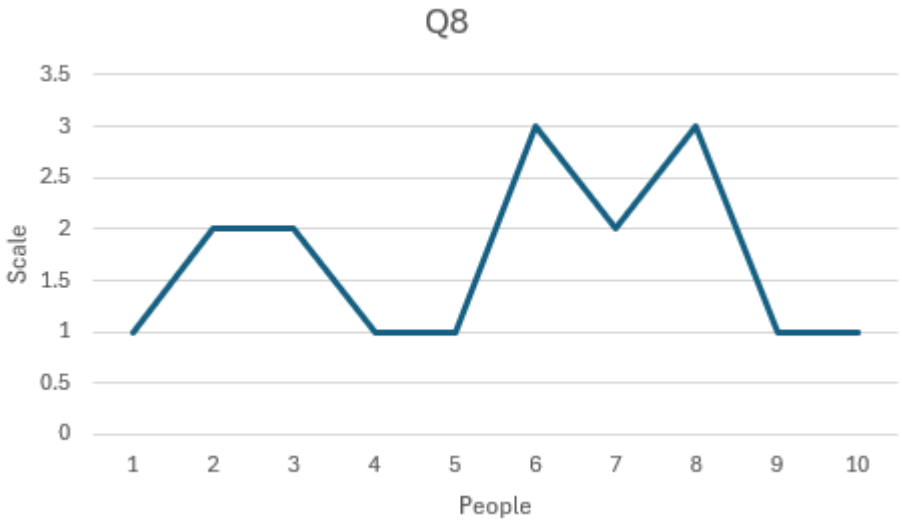
Average of 2 indicates that volunteer don't believe that the system was inconsistent

I would imagine that most people would learn to use this system very quickly:



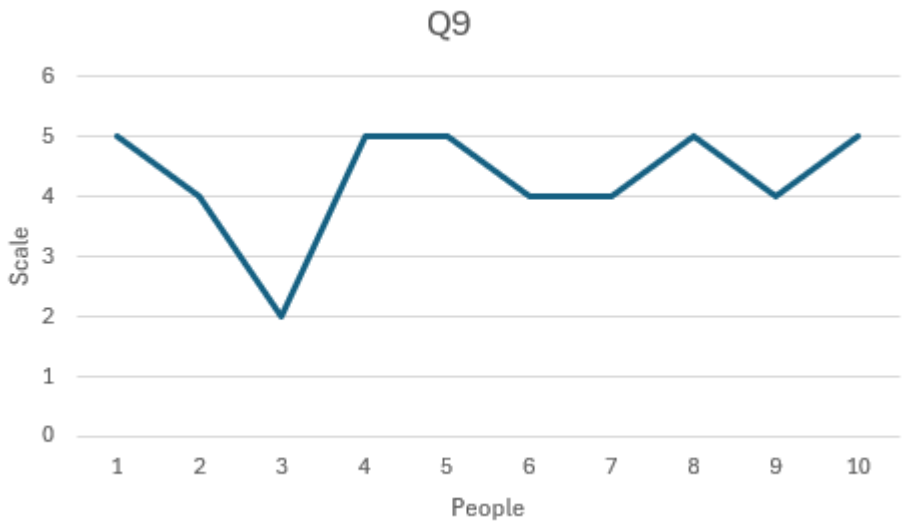
Average of 3.9 indicates that after explaining the project to the volunteers even thought they might not be familiar with coding and computer in general they learned quickly how to use the system.

I found this system very cumbersome to use:



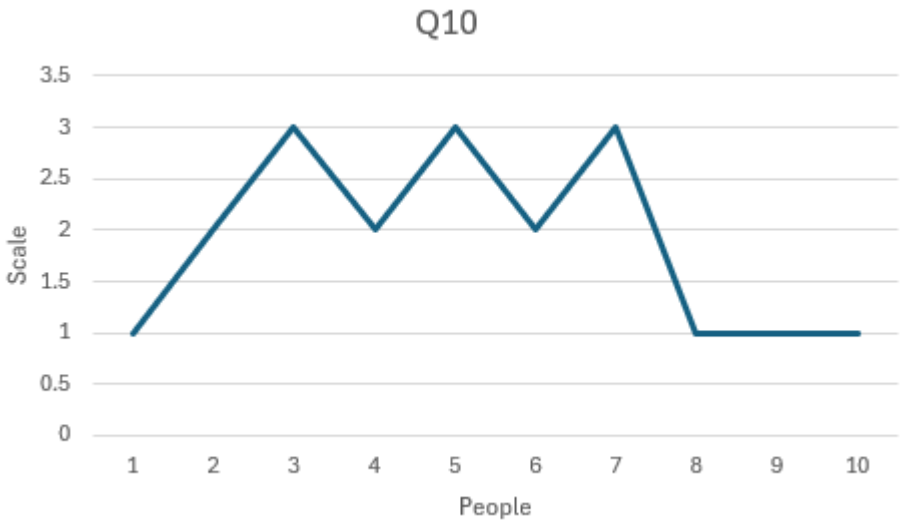
Average of 1.7 indicates that the system was not hard to carry and use.

I felt very confident using this system:



Average of 4.3 indicates that the volunteers had a good time while using the system.

I needed to learn a lot of things before I could get going with this system:



Average of 1.9 indicates that after using the system was easy for different people who are familiar with coding or not familiar.

7.Conclusion

In this dissertation, we have successfully developed and implemented an AI-driven human-robot interaction system capable of interpreting human facial expressions to control robotic movement. By leveraging advanced computer vision techniques, including the FER model for emotion detection, MediaPipe for gaze tracking, and Haar Cascade for face detection, we have demonstrated the feasibility of creating a more intuitive and accessible robotic interface. This research addressed the critical gap in traditional human-robot interaction, which often relies on complex and inaccessible control mechanisms, particularly for individuals with diverse needs. The successful integration of these technologies into the Swift Bot platform, coupled with a robust client-server architecture utilizing Python and Java, underscores the potential for robots to become more empathetic and responsive.

7.1. So what?

The significance of this work extends beyond mere technical implementation. By enabling robots to understand and respond to human emotional cues and attentional focus, we pave the way for transformative applications in sectors such as healthcare, elderly assistance, and education. In healthcare, for instance, robots can provide personalized emotional support and assistance to patients with limited mobility or cognitive impairments, enhancing their quality of life. In educational settings, emotionally responsive robots can create more engaging and personalized learning experiences, fostering better interaction and comprehension. Furthermore, by making robotic systems more intuitive, we democratize access to these technologies, empowering a wider range of users, including those unfamiliar with traditional control methods.

7.2. Objective Fulfilment

The primary aim of this dissertation was to develop and evaluate an innovative human-robot interaction system that utilizes real-time facial expression recognition and gaze tracking to enable more intuitive and accessible robot control. This aim was successfully achieved through the fulfilment of the following specific objectives:

1. **Design and Implement a Facial Expression Recognition Module**
   A robust FER model was successfully integrated and tested across a range of lighting conditions and facial structures, demonstrating consistent accuracy in detecting key emotions such as happiness and anger.
2. **Develop a Gaze Tracking Module**
   MediaPipe-based gaze tracking was implemented and validated, capable of accurately estimating gaze direction in real-time.
3. **Integrate Emotion and Gaze Data for Robot Control**
   A control mechanism was developed to merge FER and gaze data, enabling the Swift Bot to respond autonomously based on emotional and attentional cues.
4. **Establish Communication Between Processing Modules and the Robot**
   Real-time, bidirectional communication between the Python-based client and Java server was successfully established via socket programming.

5. **Evaluate the System's Performance and Usability**
   The system was rigorously tested in real-world settings, including varying lighting conditions and diverse users, and was evaluated through user surveys and standardized tools like the SUS.
6. **Explore the Potential for Enhanced Human-Robot Interaction**
   Practical applications in healthcare, elderly care, and education were identified and discussed, with promising outcomes observed in preliminary testing.

7.3. Chapter-by-Chapter Summary

1. Planning
Defined the core project goals—developing an AI-powered interface for human-robot interaction. Identified key technologies including FER, MediaPipe gaze tracking, and Haar Cascade face detection. Milestones and deliverables were clearly outlined.

2. Design
Created system architecture diagrams and flowcharts to map out real-time interaction. Designed user engagement models to support natural, intuitive communication with the Swift Bot.

3. Implementation
Developed and integrated deep learning models for facial expression recognition. Built a Python-based client for input processing and a Java-based server to control the Swift Bot, enabling real-time interaction through a custom communication protocol.

4. Testing
Conducted iterative testing across multiple scenarios—different lighting, facial structures, and user positions. Performance metrics (accuracy, latency, usability) guided improvements via feedback loops.

5. Deployment
Deployed the final integrated system on the Swift Bot. Real-world trials were conducted with test users to simulate target application scenarios.

6. Evaluation and Continuous Improvement
User feedback was gathered using both surveys and interviews. System performance was optimized based on this feedback, ensuring inclusivity and adaptability across different user demographics and environmental conditions.

7.4. Future work

I will focus on several key areas to enhance the system's capabilities and address current limitations. Firstly, we will refine gaze tracking accuracy by exploring advanced calibration techniques and incorporating contextual information to account for individual variations and environmental factors. Specifically, we will investigate the integration of 3D gaze estimation models to improve precision and robustness. Secondly, we will expand the range of recognized emotions by incorporating multimodal inputs, such as voice commands and gesture

recognition, to create a more comprehensive understanding of human intent. This will involve the development of a fusion algorithm that integrates data from multiple modalities in real-time. Thirdly, we will explore the integration of real-time adaptive learning to personalize human-robot interactions. This will involve developing a reinforcement learning framework that allows the robot to learn user preferences and adapt its behaviour accordingly. For example, the robot could learn to adjust its movement speed and trajectory based on user feedback. Fourthly, we will conduct extensive user studies with diverse populations, including individuals with disabilities and elderly users, to evaluate the system's usability and effectiveness in real-world scenarios. This will involve collecting both quantitative data, such as task completion times and error rates, and qualitative data, such as user satisfaction ratings and feedback. Finally, we will investigate the system's robustness in challenging environmental conditions, such as varying lighting conditions and occlusions, by implementing adaptive algorithms that can dynamically adjust to these changes.

This research represents a significant step towards creating robots that are not only functional but also deeply attuned to human emotional states and attentional focus. By addressing the limitations of traditional human-robot interaction and exploring innovative approaches, we contribute to the ongoing evolution of empathetic and intuitive robotic systems, ultimately fostering more harmonious and beneficial human-robot coexistence.

References

Personal Reflection

From a personal and academic standpoint, this project has been a deeply enriching experience. It allowed me to apply theoretical knowledge in artificial intelligence and computer vision to solve a real-world problem with measurable impact. The interdisciplinary nature of this project challenged me to think across domains—from technical algorithm design to human-centered usability and ethical considerations. One of the most rewarding aspects was seeing the system evolve through iterative feedback and testing, ultimately resulting in a more natural and intelligent robot. Professionally, this project strengthened my skills in full-stack system development, agile project management, and user-centered design—skills that I believe will serve as a strong foundation for future research and innovation in AI-powered human-robot interaction.

Python Code Development (Facial Emotion Detection)

Code:

```
import cv2

import numpy as np

from fer import FER

from PIL import Image

from io import BytesIO

import base64

import matplotlib.pyplot as plt

from keras.preprocessing import image

import mediapipe as mp

from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score

from facenet_pytorch import MTCNN, InceptionResnetV1

import torch

emotion_detector = FER()

face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

mp_face_mesh = mp.solutions.face_mesh

mp_drawing = mp.solutions.drawing_utils

mtcnn = MTCNN(keep_all=True)  # MTCNN for face detection
```

```
inception_resnet =
InceptionResnetV1(pretrained='vggface2').eval()  # Pretrained
FaceNet model

def adjust_gamma(image, gamma=1.0):

    invGamma = 1.0 / gamma

    table = np.array([((i / 255.0) ** invGamma) * 255 for i in
np.arange(0, 256)]).astype("uint8")

    return cv2.LUT(image, table)

def detect_faces_and_emotions(dataUri):

    img_data = base64.b64decode(dataUri.split(',')[1])

    image = Image.open(BytesIO(img_data))

    image = np.array(image)

    emotion, score = emotion_detector.top_emotion(image)

    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

    faces = face_cascade.detectMultiScale(gray, 1.1, 4)

    for (x, y, w, h) in faces:

        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

        cv2.putText(image, emotion, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 255, 0), 2)

    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

    plt.axis('off')  # Hide axes

    plt.show()

def get_face_embedding(img_path):

    img = Image.open(img_path)

    boxes, probs = mtcnn.detect(img)  # Detect faces with MTCNN

    if boxes is not None:

        faces = []

        for box in boxes:

            face = img.crop((box[0], box[1], box[2], box[3]))

            face_embedding = inception_resnet(face)  # Extract face
embedding using InceptionResnetV1

            faces.append(face_embedding)

        return faces

    else:
```

```
        return []
def get_landmarks(image):
    with mp_face_mesh.FaceMesh(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as face_mesh:
        rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  #
Convert to RGB as MediaPipe uses RGB
        results = face_mesh.process(rgb_image)
        landmarks_list = []
        if results.multi_face_landmarks:
            for face_landmarks in results.multi_face_landmarks:
                landmarks_list.append(face_landmarks)
        return landmarks_list
def start_camera():
    cap = cv2.VideoCapture(0)
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        frame = adjust_gamma(frame, gamma=1.2)
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        emotion, score = emotion_detector.top_emotion(rgb_frame)
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, 1.1, 4)
        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255,
0), 2)
            cv2.putText(frame, emotion, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 255, 0), 2)
            print(f"Detected emotion: {emotion}")
        landmarks = get_landmarks(frame)
        if landmarks:
            for face_landmarks in landmarks:
                for landmark in face_landmarks.landmark:
```

```
                    x, y = int(landmark.x * frame.shape[1]),
int(landmark.y * frame.shape[0])
                    cv2.circle(frame, (x, y), 1, (0, 255, 0), -1)
        cv2.imshow('Webcam', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
def evaluate_performance(y_true, y_pred):
    print(f"Accuracy: {accuracy_score(y_true, y_pred)}")
    print(f"Precision: {precision_score(y_true, y_pred,
average='weighted')}")
    print(f"Recall: {recall_score(y_true, y_pred,
average='weighted')}")
    print(f"F1 Score: {f1_score(y_true, y_pred,
average='weighted')}")
start_camera()
y_true = ["happy", "sad", "angry", "happy", "surprise"]
y_pred = ["happy", "sad", "angry", "surprise", "neutral"]
evaluate_performance(y_true, y_pred)
```

Java Code Development (SwiftBot Movement)

code:

```
import java.io.*;
import java.net.*;
import swiftbot.*;
public class ActionServer {
  public static void main(String[] args) {
  try {
  ServerSocket serverSocket = new ServerSocket(5000);
  System.out.println("Server listening on port 5000...");
  while (true) {
  Socket clientSocket = serverSocket.accept();
```

```
  System.out.println("Client connected: " +
clientSocket.getInetAddress());

  BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

  PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
true);

   String emotion = in.readLine();

  System.out.println("Received emotion: " + emotion);

  SwiftBotAPI api = new SwiftBotAPI();

  switch (emotion.toLowerCase()) {

  case "happy":

  api.startMove(100, 100);  // Move forward

  out.println("Moving forward (Happy emotion)");

  break;

  case "sad":

  api.startMove(-100, -100);  // Move backward

  out.println("Moving backward (Sad emotion)");

  break;

  case "neutral":

  api.stopMove();  // Stop moving

  out.println("Stopping (Neutral emotion)");

  break;

  case "angry":

  try {

  api.setUnderlight(Underlight.FRONT_LEFT, 255, 0, 0);

  api.stopMove();  // Stay still

  out.println("Stopping and turning on red light (Angry emotion)");

  } catch (IllegalArgumentException | IOException e) {

  e.printStackTrace();

  out.println("Error while setting red underlight for Angry
emotion.");

  }

  break;

  case "surprise":
```

```
  try {
  api.setUnderlight(Underlight.FRONT_LEFT, 255, 255, 0);
  api.stopMove();  // Stay still
  out.println("Stopping and turning on yellow light (Surprise
emotion)");
  } catch (IllegalArgumentException | IOException e) {
  e.printStackTrace();
  out.println("Error while setting yellow underlight for Surprise
emotion.");
  }
  break;
  default:
  out.println("Unknown emotion");
  break;
  }
   clientSocket.close();
  }
  } catch (IOException e) {
  e.printStackTrace();
  }
  }
}
```

Updating python code

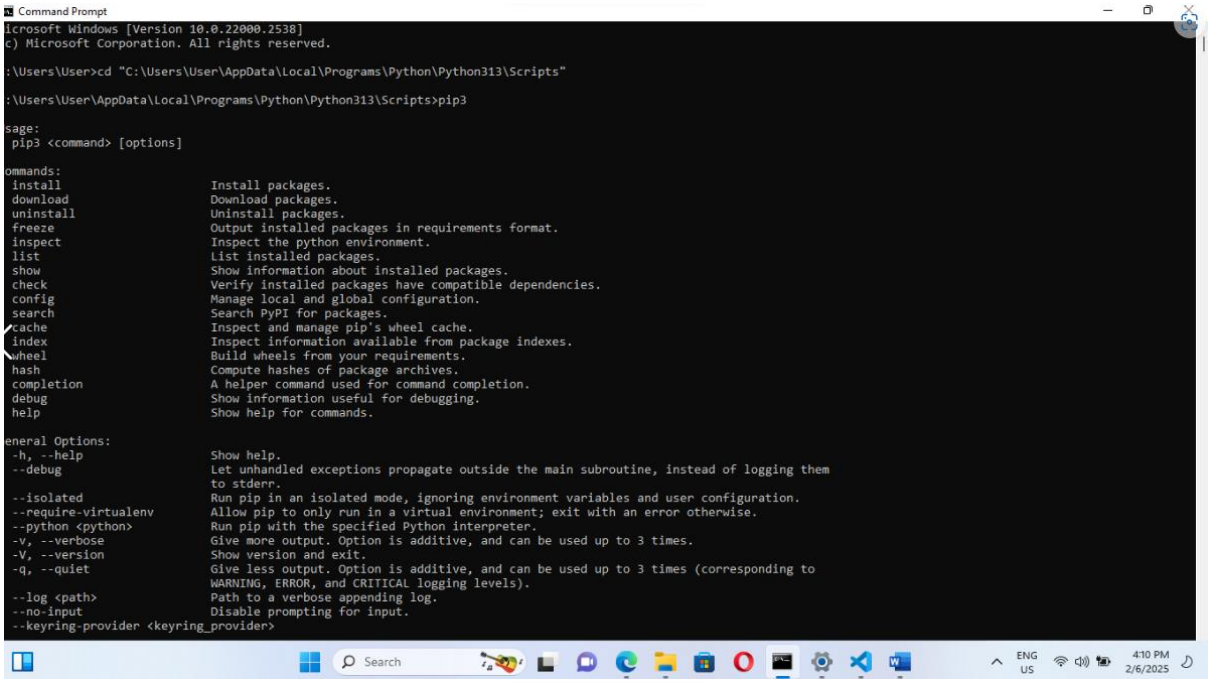
Code:

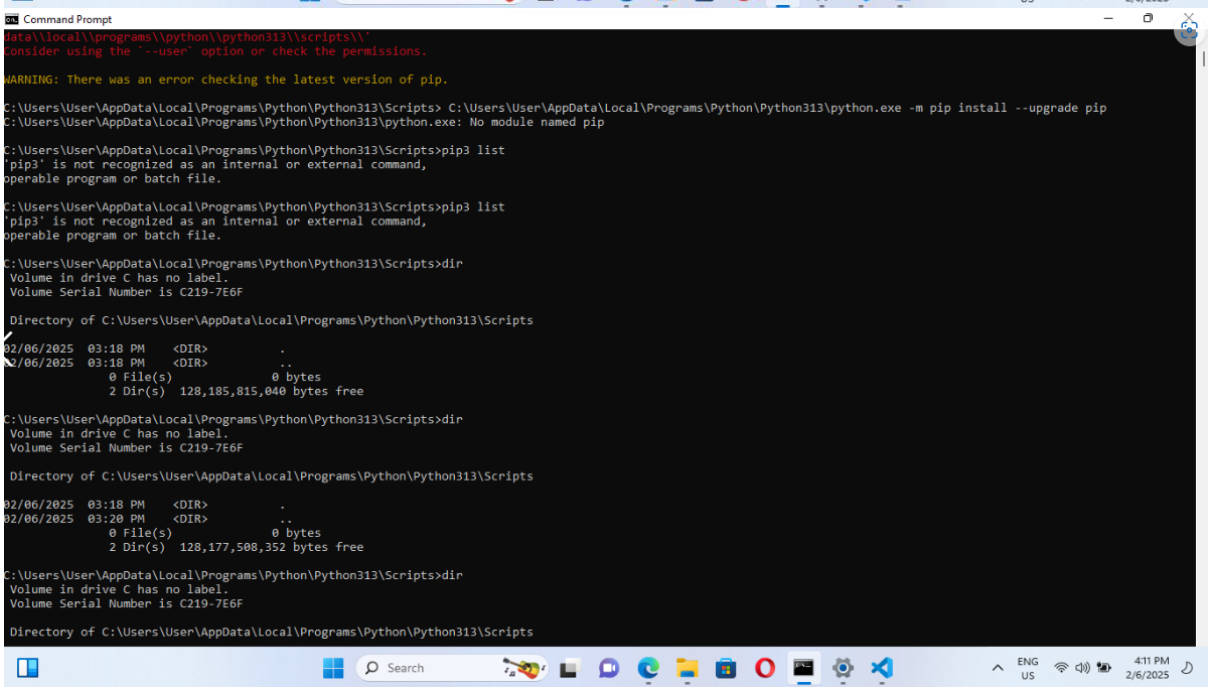
```
import socket
import cv2
import numpy as np
from fer import FER
from PIL import Image
from io import BytesIO
import base64
import matplotlib.pyplot as plt
from keras.preprocessing import image
import mediapipe as mp
from facenet_pytorch import MTCNN, InceptionResnetV1
```

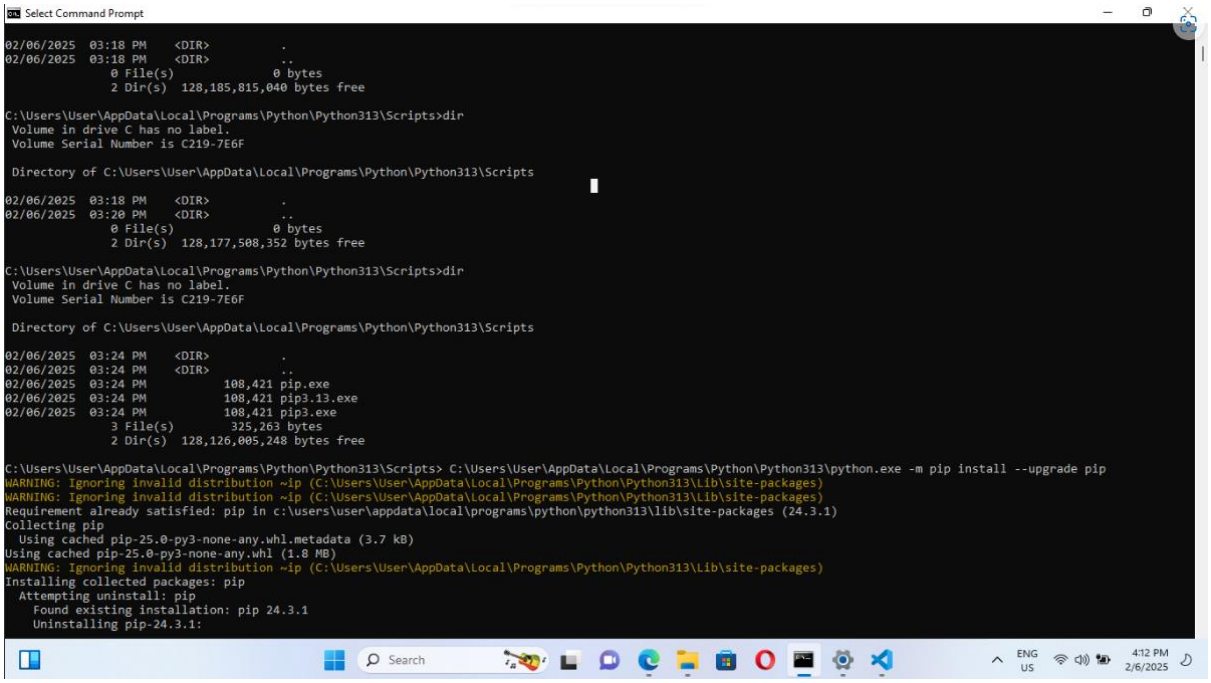
```
import torch
emotion_detector = FER()
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
mp_face_mesh = mp.solutions.face_mesh
mp_drawing = mp.solutions.drawing_utils
mtcnn = MTCNN(keep_all=True)  # MTCNN for face detection
inception_resnet =
InceptionResnetV1(pretrained='vggface2').eval()  # Pretrained
FaceNet model
def adjust_gamma(image, gamma=1.0):
    invGamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255 for i in
np.arange(0, 256)]).astype("uint8")
    return cv2.LUT(image, table)
def detect_faces_and_emotions(dataUri):
    img_data = base64.b64decode(dataUri.split(',')[1])
    image = Image.open(BytesIO(img_data))
    image = np.array(image)
    emotion, score = emotion_detector.top_emotion(image)
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)
    for (x, y, w, h) in faces:
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255,
0), 2)
        cv2.putText(image, emotion, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 255, 0), 2)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()
    return emotion
def send_emotion_to_server(emotion):
    server_host = '127.0.0.1'
    server_port = 5000
    client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    client_socket.connect((server_host, server_port))
    client_socket.sendall(emotion.encode())
    response = client_socket.recv(1024).decode()
    print("Server response:", response)
    client_socket.close()
emotion =
detect_faces_and_emotions('path_to_your_image_or_video_data')
```

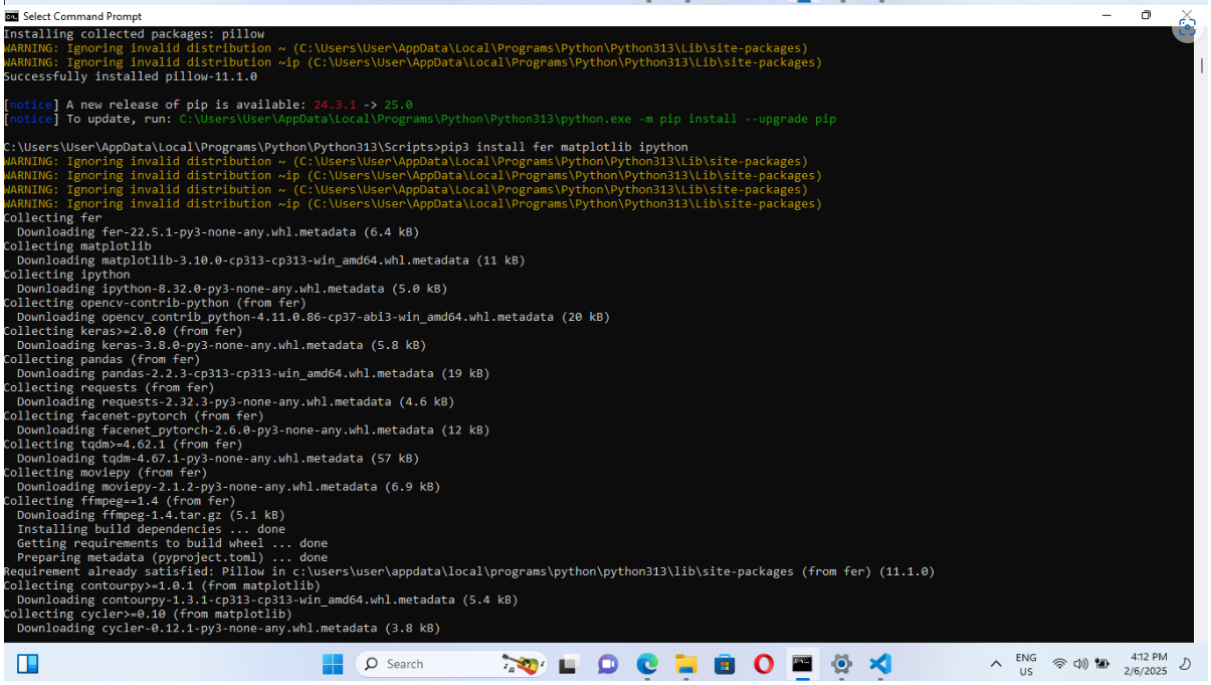
```
send_emotion_to_server(emotion)
```

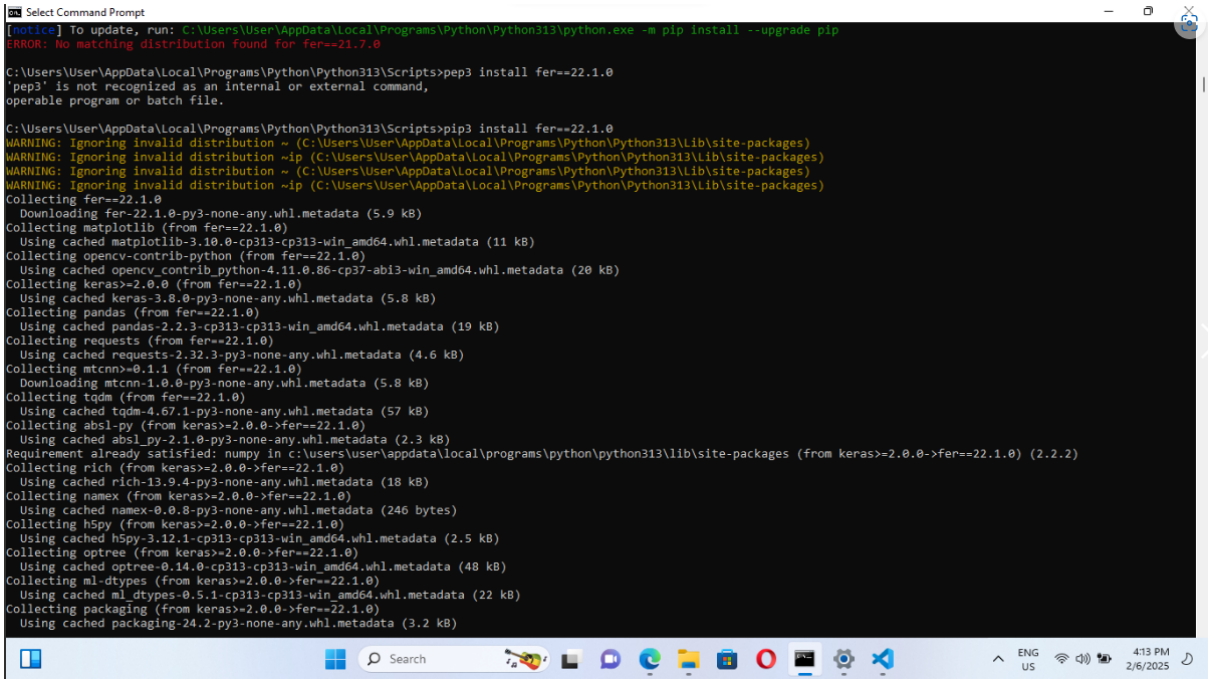
Screenshots

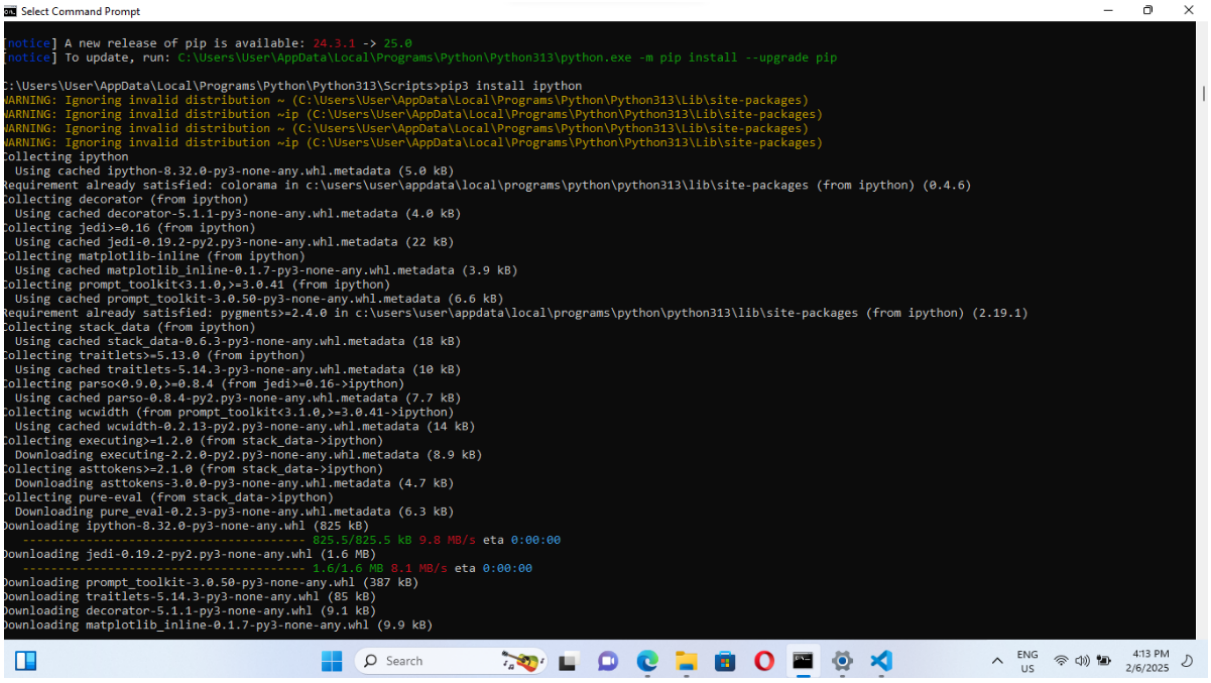


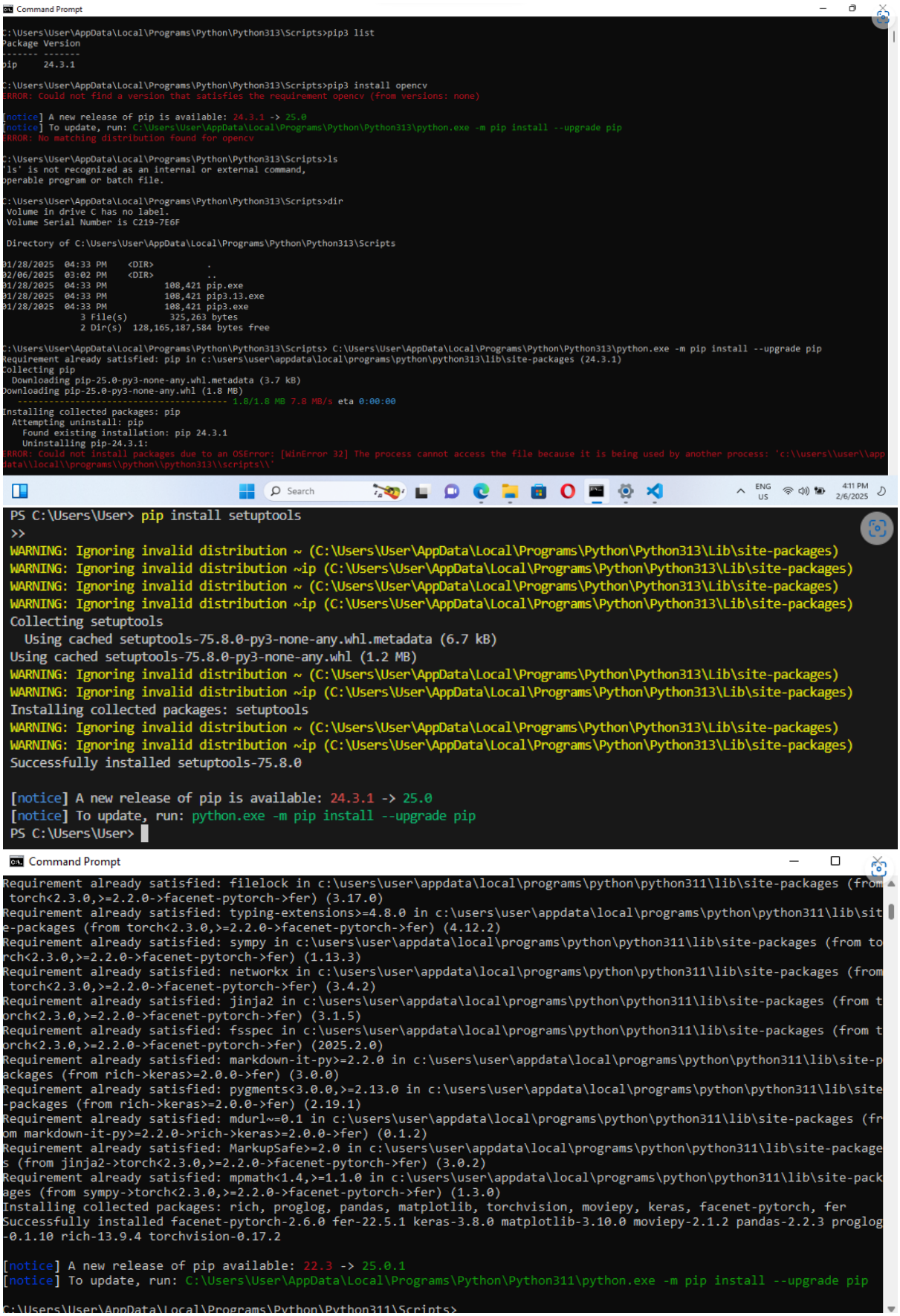


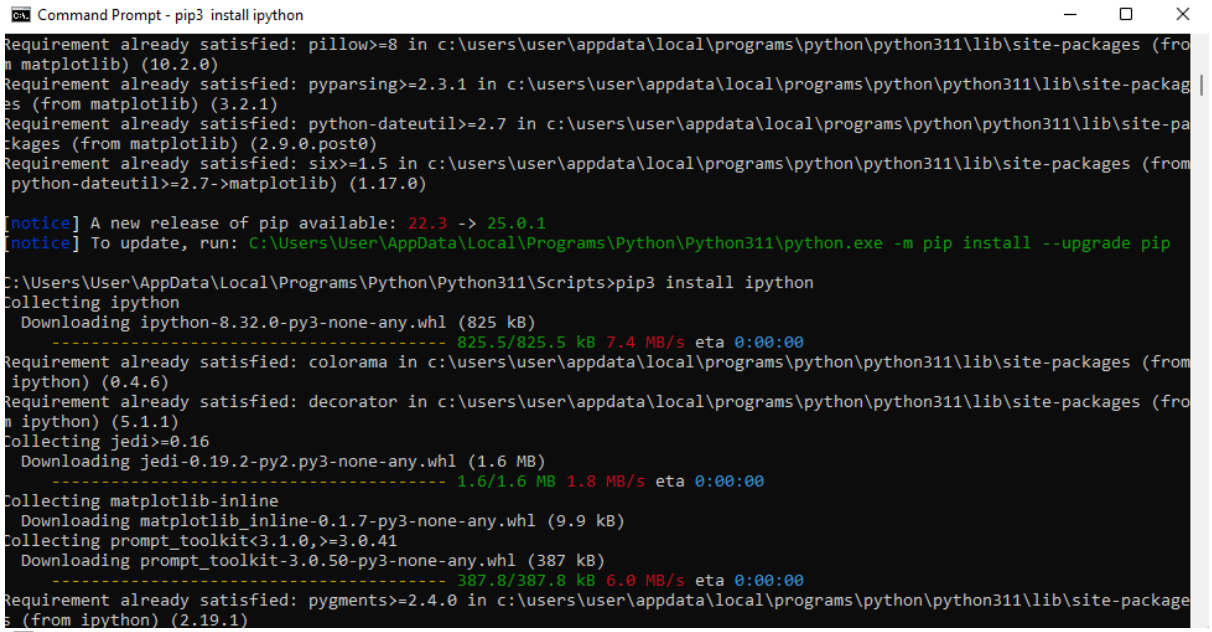

Select Command Prompt
02/06/2025 03:18 PM <DIR> .
02/06/2025 03:18 PM <DIR> ..
0 File(s) 0 bytes
2 Dir(s) 128,185,815,040 bytes free
C:\Users\User\AppData\Local\Programs\Python\Python313\Scripts>dir
Volume in drive C has no label.
Volume Serial Number is C219-7E6F
Directory of C:\Users\User\AppData\Local\Programs\Python\Python313\Scripts
02/06/2025 03:18 PM <DIR> .
02/06/2025 03:20 PM <DIR> ..
0 File(s) 0 bytes
2 Dir(s) 128,177,508,352 bytes free
C:\Users\User\AppData\Local\Programs\Python\Python313\Scripts>dir
Volume in drive C has no label.
Volume Serial Number is C219-7E6F
Directory of C:\Users\User\AppData\Local\Programs\Python\Python313\Scripts
02/06/2025 03:24 PM <DIR> .
02/06/2025 03:24 PM <DIR> ..
02/06/2025 03:24 PM 108,421 pip.exe
02/06/2025 03:24 PM 108,421 pip3.13.exe
02/06/2025 03:24 PM 108,421 pip3.exe
3 File(s) 325,263 bytes
2 Dir(s) 128,126,005,248 bytes free
C:\Users\User\AppData\Local\Programs\Python\Python313\Scripts> C:\Users\User\AppData\Local\Programs\Python\Python313\python.exe -m pip install --upgrade pip
WARNING: Ignoring invalid distribution ~ip (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~ip (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
Requirement already satisfied: pip in c:\users\user\appdata\local\programs\python\python313\lib\site-packages (24.3.1)
Collecting pip
Using cached pip-25.0-py3-none-any.whl.metadata (3.7 kB)
Using cached pip-25.0-py3-none-any.whl (1.8 MB)
WARNING: Ignoring invalid distribution ~ip (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
Installing collected packages: pip
Attempting uninstall: pip
Found existing installation: pip 24.3.1
Uninstalling pip-24.3.1:
Search
ENG US
4:12 PM 2/6/2025

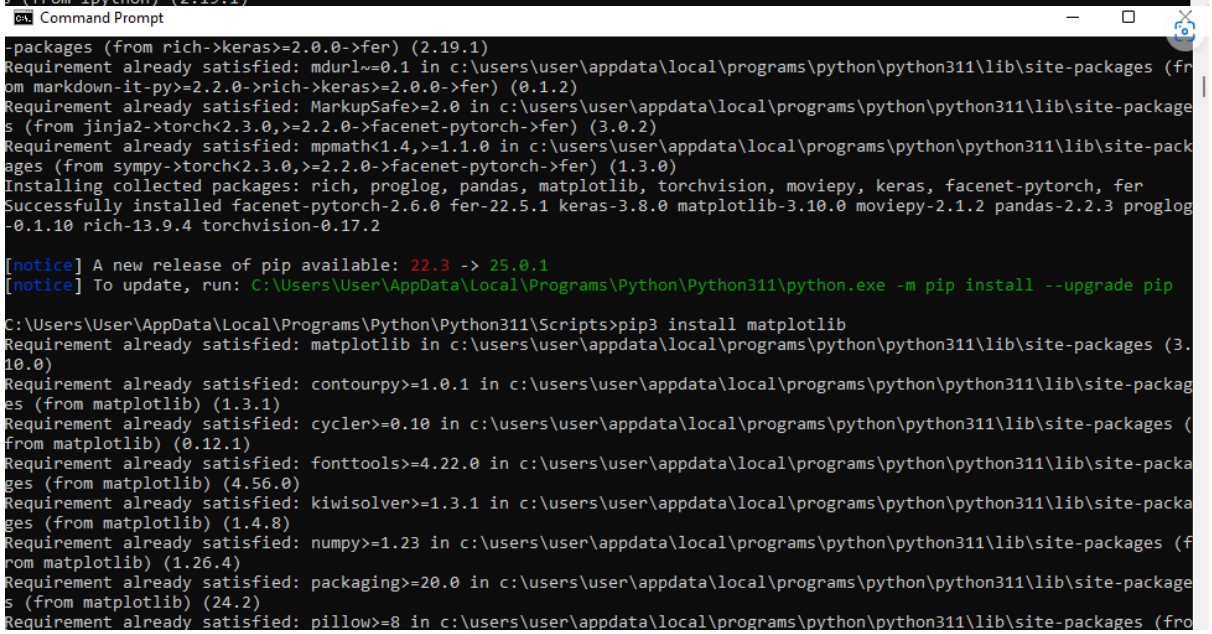

Select Command Prompt
Installing collected packages: pillow
WARNING: Ignoring invalid distribution ~ (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~ip (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
Successfully installed pillow-11.1.0
[notice] A new release of pip is available: 24.3.1 -> 25.0
[notice] To update, run: C:\Users\User\AppData\Local\Programs\Python\Python313\python.exe -m pip install --upgrade pip
C:\Users\User\AppData\Local\Programs\Python\Python313\Scripts>pip3 install fer matplotlib ipython
WARNING: Ignoring invalid distribution ~ (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~ip (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~ (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~ip (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
Collecting fer
Downloading fer-22.5.1-py3-none-any.whl.metadata (6.4 kB)
Collecting matplotlib
Downloading matplotlib-3.10.0-cp313-cp313-win_amd64.whl.metadata (11 kB)
Collecting ipython
Downloading ipython-8.32.0-py3-none-any.whl.metadata (5.0 kB)
Collecting opencv-contrib-python (from fer)
Downloading opencv_contrib_python-4.11.0.86-cp37-abi3-win_amd64.whl.metadata (20 kB)
Collecting keras>=2.0.0 (from fer)
Downloading keras-3.8.0-py3-none-any.whl.metadata (5.8 kB)
Collecting pandas (from fer)
Downloading pandas-2.2.3-cp313-cp313-win_amd64.whl.metadata (19 kB)
Collecting requests (from fer)
Downloading requests-2.32.3-py3-none-any.whl.metadata (4.6 kB)
Collecting facenet-pytorch (from fer)
Downloading facenet_pytorch-2.6.0-py3-none-any.whl.metadata (12 kB)
Collecting tqdm>=4.62.1 (from fer)
Downloading tqdm-4.67.1-py3-none-any.whl.metadata (57 kB)
Collecting moviepy (from fer)
Downloading moviepy-2.1.2-py3-none-any.whl.metadata (6.9 kB)
Collecting ffmpeg==1.4 (from fer)
Downloading ffmpeg-1.4.tar.gz (5.1 kB)
Installing build dependencies ... done
Getting requirements to build wheel ... done
Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: Pillow in c:\users\user\appdata\local\programs\python\python313\lib\site-packages (from fer) (11.1.0)
Collecting contourpy>=1.0.1 (from matplotlib)
Downloading contourpy-1.3.1-cp313-cp313-win_amd64.whl.metadata (5.4 kB)
Collecting cycler>=0.10 (from matplotlib)
Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Search
ENG US
4:12 PM 2/6/2025

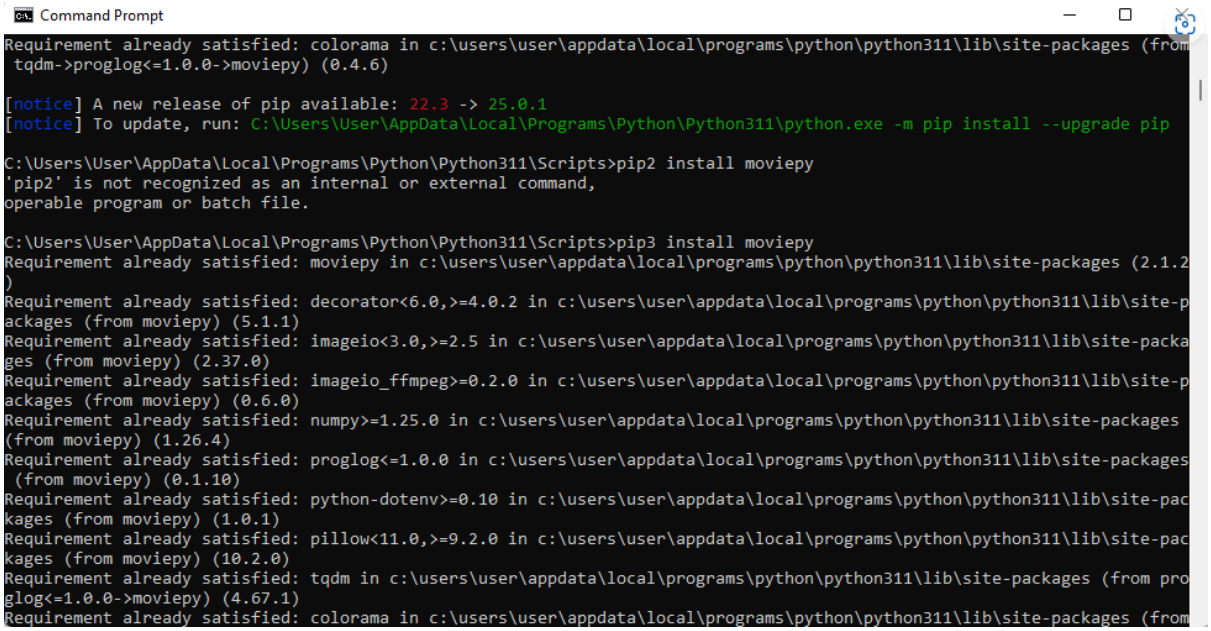

Select Command Prompt
[notice] To update, run: C:\Users\User\AppData\Local\Programs\Python\Python313\python.exe -m pip install --upgrade pip
ERROR: No matching distribution found for fer==21.7.0
C:\Users\User\AppData\Local\Programs\Python\Python313\Scripts>pep3 install fer==22.1.0
'pep3' is not recognized as an internal or external command,
operable program or batch file.
C:\Users\User\AppData\Local\Programs\Python\Python313\Scripts>pip3 install fer==22.1.0
WARNING: Ignoring invalid distribution ~ (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~ip (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~ (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~ip (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
Collecting fer==22.1.0
Downloading fer-22.1.0-py3-none-any.whl.metadata (5.9 kB)
Collecting matplotlib (from fer==22.1.0)
Using cached matplotlib-3.10.0-cp313-cp313-win_amd64.whl.metadata (11 kB)
Collecting opencv-contrib-python (from fer==22.1.0)
Using cached opencv_contrib_python-4.11.0.86-cp37-abi3-win_amd64.whl.metadata (20 kB)
Collecting keras>=2.0.0 (from fer==22.1.0)
Using cached keras-3.8.0-py3-none-any.whl.metadata (5.8 kB)
Collecting pandas (from fer==22.1.0)
Using cached pandas-2.2.3-cp313-cp313-win_amd64.whl.metadata (19 kB)
Collecting requests (from fer==22.1.0)
Using cached requests-2.32.3-py3-none-any.whl.metadata (4.6 kB)
Collecting mtcnn>=0.1.1 (from fer==22.1.0)
Downloading mtcnn-1.0.0-py3-none-any.whl.metadata (5.8 kB)
Collecting tqdm (from fer==22.1.0)
Using cached tqdm-4.67.1-py3-none-any.whl.metadata (57 kB)
Collecting absl-py (from keras>=2.0.0->fer==22.1.0)
Using cached absl_py-2.1.0-py3-none-any.whl.metadata (2.3 kB)
Requirement already satisfied: numpy in c:\users\user\appdata\local\programs\python\python313\lib\site-packages (from keras>=2.0.0->fer==22.1.0) (2.2.2)
Collecting rich (from keras>=2.0.0->fer==22.1.0)
Using cached rich-13.9.4-py3-none-any.whl.metadata (18 kB)
Collecting namex (from keras>=2.0.0->fer==22.1.0)
Using cached namex-0.0.8-py3-none-any.whl.metadata (246 bytes)
Collecting h5py (from keras>=2.0.0->fer==22.1.0)
Using cached h5py-3.12.1-cp313-cp313-win_amd64.whl.metadata (2.5 kB)
Collecting optree (from keras>=2.0.0->fer==22.1.0)
Using cached optree-0.14.0-cp313-cp313-win_amd64.whl.metadata (48 kB)
Collecting ml-dtypes (from keras>=2.0.0->fer==22.1.0)
Using cached ml_dtypes-0.5.1-cp313-cp313-win_amd64.whl.metadata (22 kB)
Collecting packaging (from keras>=2.0.0->fer==22.1.0)
Using cached packaging-24.2-py3-none-any.whl.metadata (3.2 kB)
Search
ENG
US
4:13 PM
2/6/2025

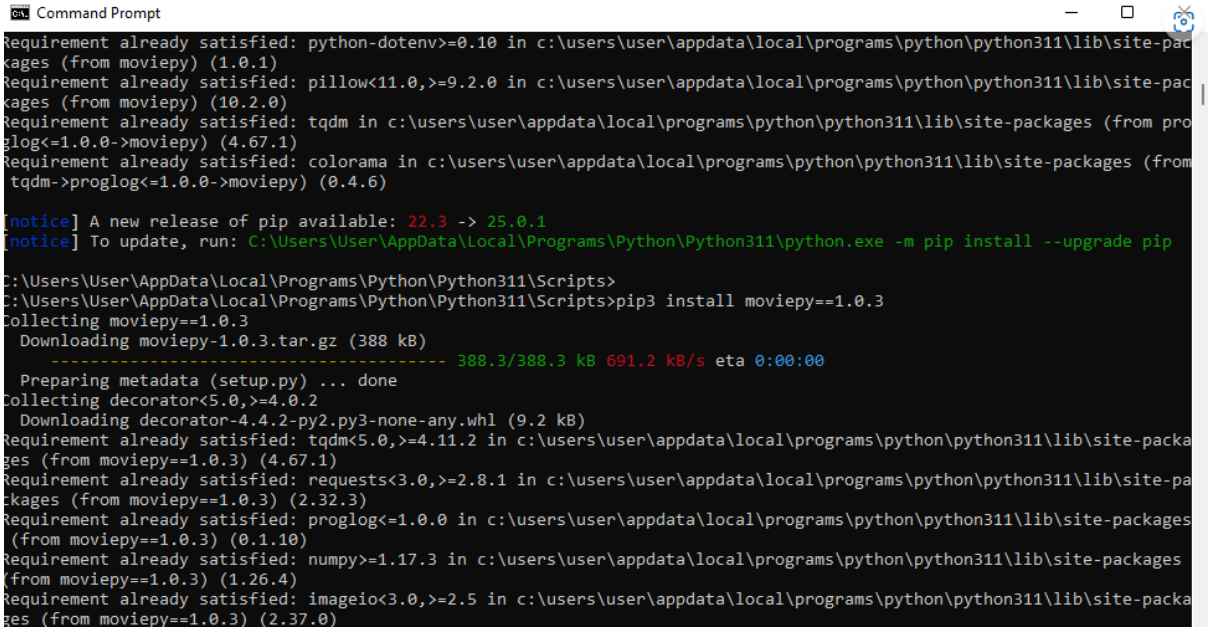

Select Command Prompt
[notice] A new release of pip is available: 24.3.1 -> 25.0
[notice] To update, run: C:\Users\User\AppData\Local\Programs\Python\Python313\python.exe -m pip install --upgrade pip
C:\Users\User\AppData\Local\Programs\Python\Python313\Scripts>pip3 install ipython
WARNING: Ignoring invalid distribution ~ (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~ip (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~ (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~ip (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
Collecting ipython
Using cached ipython-8.32.0-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: colorama in c:\users\user\appdata\local\programs\python\python313\lib\site-packages (from ipython) (0.4.6)
Collecting decorator (from ipython)
Using cached decorator-5.1.1-py3-none-any.whl.metadata (4.0 kB)
Collecting jedi>=0.16 (from ipython)
Using cached jedi-0.19.2-py2.py3-none-any.whl.metadata (22 kB)
Collecting matplotlib-inline (from ipython)
Using cached matplotlib_inline-0.1.7-py3-none-any.whl.metadata (3.9 kB)
Collecting prompt_toolkit<3.1.0,>=3.0.41 (from ipython)
Using cached prompt_toolkit-3.0.50-py3-none-any.whl.metadata (6.6 kB)
Requirement already satisfied: pygments>=2.4.0 in c:\users\user\appdata\local\programs\python\python313\lib\site-packages (from ipython) (2.19.1)
Collecting stack_data (from ipython)
Using cached stack_data-0.6.3-py3-none-any.whl.metadata (18 kB)
Collecting traitlets>=5.13.0 (from ipython)
Using cached traitlets-5.14.3-py3-none-any.whl.metadata (10 kB)
Collecting parso<0.9.0,>=0.8.4 (from jedi>=0.16->ipython)
Using cached parso-0.8.4-py2.py3-none-any.whl.metadata (7.7 kB)
Collecting wcwidth (from prompt_toolkit<3.1.0,>=3.0.41->ipython)
Using cached wcwidth-0.2.13-py2.py3-none-any.whl.metadata (14 kB)
Collecting executing>=1.2.0 (from stack_data->ipython)
Downloading executing-2.2.0-py2.py3-none-any.whl.metadata (8.9 kB)
Collecting asttokens>=2.1.0 (from stack_data->ipython)
Downloading asttokens-3.0.0-py3-none-any.whl.metadata (4.7 kB)
Collecting pure-eval (from stack_data->ipython)
Downloading pure_eval-0.2.3-py3-none-any.whl.metadata (6.3 kB)
Downloading ipython-8.32.0-py3-none-any.whl (825 kB)
825.5/825.5 kB 9.8 MB/s eta 0:00:00
Downloading jedi-0.19.2-py2.py3-none-any.whl (1.6 MB)
1.6/1.6 MB 8.1 MB/s eta 0:00:00
Downloading prompt_toolkit-3.0.50-py3-none-any.whl (387 kB)
Downloading traitlets-5.14.3-py3-none-any.whl (85 kB)
Downloading decorator-5.1.1-py3-none-any.whl (9.1 kB)
Downloading matplotlib_inline-0.1.7-py3-none-any.whl (9.9 kB)
Search
ENG
US
4:13 PM
2/6/2025

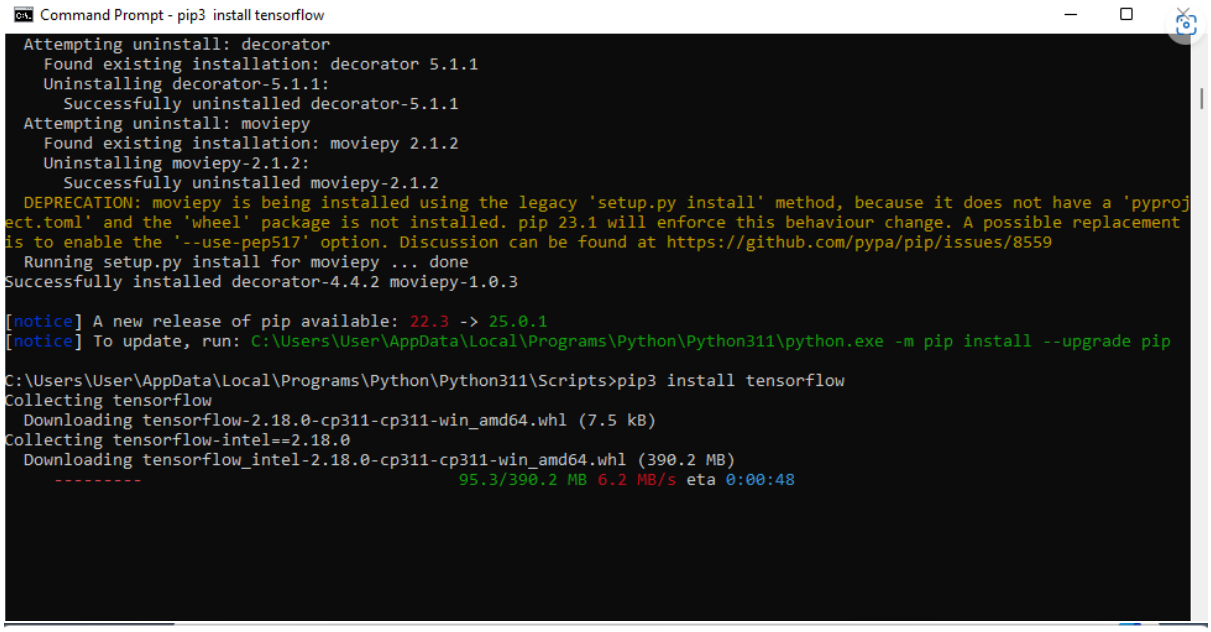

Command Prompt
C:\Users\User\AppData\Local\Programs\Python\Python313\Scripts>pip3 list
Package Version
pip 24.3.1
C:\Users\User\AppData\Local\Programs\Python\Python313\Scripts>pip3 install opencv
ERROR: Could not find a version that satisfies the requirement opencv (from versions: none)
[notice] A new release of pip is available: 24.3.1 -> 25.0
[notice] To update, run: C:\Users\User\AppData\Local\Programs\Python\Python313\python.exe -m pip install --upgrade pip
ERROR: No matching distribution found for opencv
C:\Users\User\AppData\Local\Programs\Python\Python313\Scripts>ls
'ls' is not recognized as an internal or external command,
operable program or batch file.
C:\Users\User\AppData\Local\Programs\Python\Python313\Scripts>dir
Volume in drive C has no label.
Volume Serial Number is C219-7E6F
Directory of C:\Users\User\AppData\Local\Programs\Python\Python313\Scripts
01/28/2025 04:33 PM <DIR> .
02/06/2025 03:02 PM <DIR> ..
01/28/2025 04:33 PM 108,421 pip.exe
01/28/2025 04:33 PM 108,421 pip3.13.exe
01/28/2025 04:33 PM 108,421 pip3.exe
3 File(s) 325,263 bytes
2 Dir(s) 128,165,187,584 bytes free
C:\Users\User\AppData\Local\Programs\Python\Python313\Scripts> C:\Users\User\AppData\Local\Programs\Python\Python313\python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\user\appdata\local\programs\python\python313\lib\site-packages (24.3.1)
Collecting pip
Downloading pip-25.0-py3-none-any.whl.metadata (3.7 kB)
Downloading pip-25.0-py3-none-any.whl (1.8 MB)
1.8/1.8 MB 7.8 MB/s eta 0:00:00
Installing collected packages: pip
Attempting uninstall: pip
Found existing installation: pip 24.3.1
Uninstalling pip-24.3.1:
ERROR: Could not install packages due to an OSError: [WinError 32] The process cannot access the file because it is being used by another process: 'c:\\users\\user\\appdata\\local\\programs\\python\\python313\\scripts\\'
Search
ENG US
4:11 PM 2/6/2025
PS C:\Users\User> pip install setuptools
>>
WARNING: Ignoring invalid distribution ~ (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~ip (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~ (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~ip (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
Collecting setuptools
Using cached setuptools-75.8.0-py3-none-any.whl.metadata (6.7 kB)
Using cached setuptools-75.8.0-py3-none-any.whl (1.2 MB)
WARNING: Ignoring invalid distribution ~ (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~ip (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
Installing collected packages: setuptools
WARNING: Ignoring invalid distribution ~ (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
WARNING: Ignoring invalid distribution ~ip (C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages)
Successfully installed setuptools-75.8.0
[notice] A new release of pip is available: 24.3.1 -> 25.0
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\Users\User>
Command Prompt
Requirement already satisfied: filelock in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from torch<2.3.0,>=2.2.0->facenet-pytorch->fer) (3.17.0)
Requirement already satisfied: typing-extensions>=4.8.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from torch<2.3.0,>=2.2.0->facenet-pytorch->fer) (4.12.2)
Requirement already satisfied: sympy in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from torch<2.3.0,>=2.2.0->facenet-pytorch->fer) (1.13.3)
Requirement already satisfied: networkx in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from torch<2.3.0,>=2.2.0->facenet-pytorch->fer) (3.4.2)
Requirement already satisfied: jinja2 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from torch<2.3.0,>=2.2.0->facenet-pytorch->fer) (3.1.5)
Requirement already satisfied: fsspec in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from torch<2.3.0,>=2.2.0->facenet-pytorch->fer) (2025.2.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from rich->keras>=2.0.0->fer) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from rich->keras>=2.0.0->fer) (2.19.1)
Requirement already satisfied: mdurl~=0.1 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from markdown-it-py>=2.2.0->rich->keras>=2.0.0->fer) (0.1.2)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from jinja2->torch<2.3.0,>=2.2.0->facenet-pytorch->fer) (3.0.2)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from sympy->torch<2.3.0,>=2.2.0->facenet-pytorch->fer) (1.3.0)
Installing collected packages: rich, proglog, pandas, matplotlib, torchvision, moviepy, keras, facenet-pytorch, fer
Successfully installed facenet-pytorch-2.6.0 fer-22.5.1 keras-3.8.0 matplotlib-3.10.0 moviepy-2.1.2 pandas-2.2.3 proglog-0.1.10 rich-13.9.4 torchvision-0.17.2
[notice] A new release of pip available: 22.3 -> 25.0.1
[notice] To update, run: C:\Users\User\AppData\Local\Programs\Python\Python311\python.exe -m pip install --upgrade pip
C:\Users\User\AppData\Local\Programs\Python\Python311\Scripts>

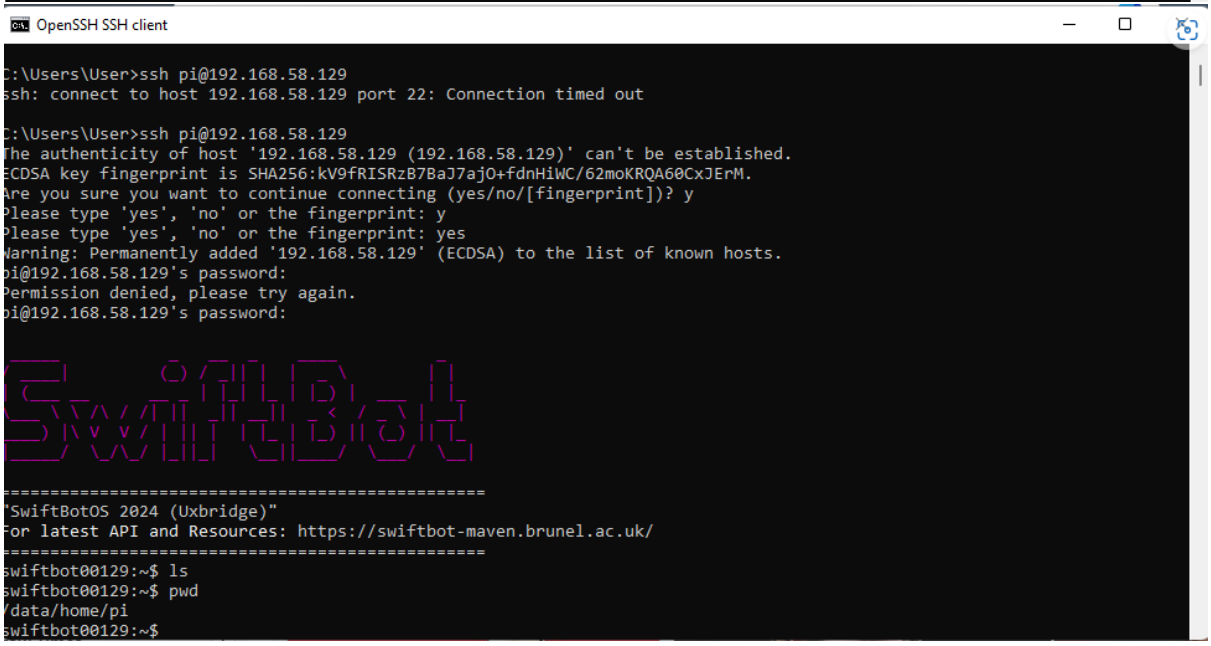

Command Prompt - pip3 install ipython
Requirement already satisfied: pillow>=8 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (fro
m matplotlib) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\user\appdata\local\programs\python\python311\lib\site-packag
es (from matplotlib) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\user\appdata\local\programs\python\python311\lib\site-pa
ckages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from
python-dateutil>=2.7->matplotlib) (1.17.0)
[notice] A new release of pip available: 22.3 -> 25.0.1
[notice] To update, run: C:\Users\User\AppData\Local\Programs\Python\Python311\python.exe -m pip install --upgrade pip
C:\Users\User\AppData\Local\Programs\Python\Python311\Scripts>pip3 install ipython
Collecting ipython
Downloading ipython-8.32.0-py3-none-any.whl (825 kB)
825.5/825.5 kB 7.4 MB/s eta 0:00:00
Requirement already satisfied: colorama in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from
ipython) (0.4.6)
Requirement already satisfied: decorator in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (fro
m ipython) (5.1.1)
Collecting jedi>=0.16
Downloading jedi-0.19.2-py2.py3-none-any.whl (1.6 MB)
1.6/1.6 MB 1.8 MB/s eta 0:00:00
Collecting matplotlib-inline
Downloading matplotlib_inline-0.1.7-py3-none-any.whl (9.9 kB)
Collecting prompt_toolkit<3.1.0,>=3.0.41
Downloading prompt_toolkit-3.0.50-py3-none-any.whl (387 kB)
387.8/387.8 kB 6.0 MB/s eta 0:00:00
Requirement already satisfied: pygments>=2.4.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-package
s (from ipython) (2.19.1)

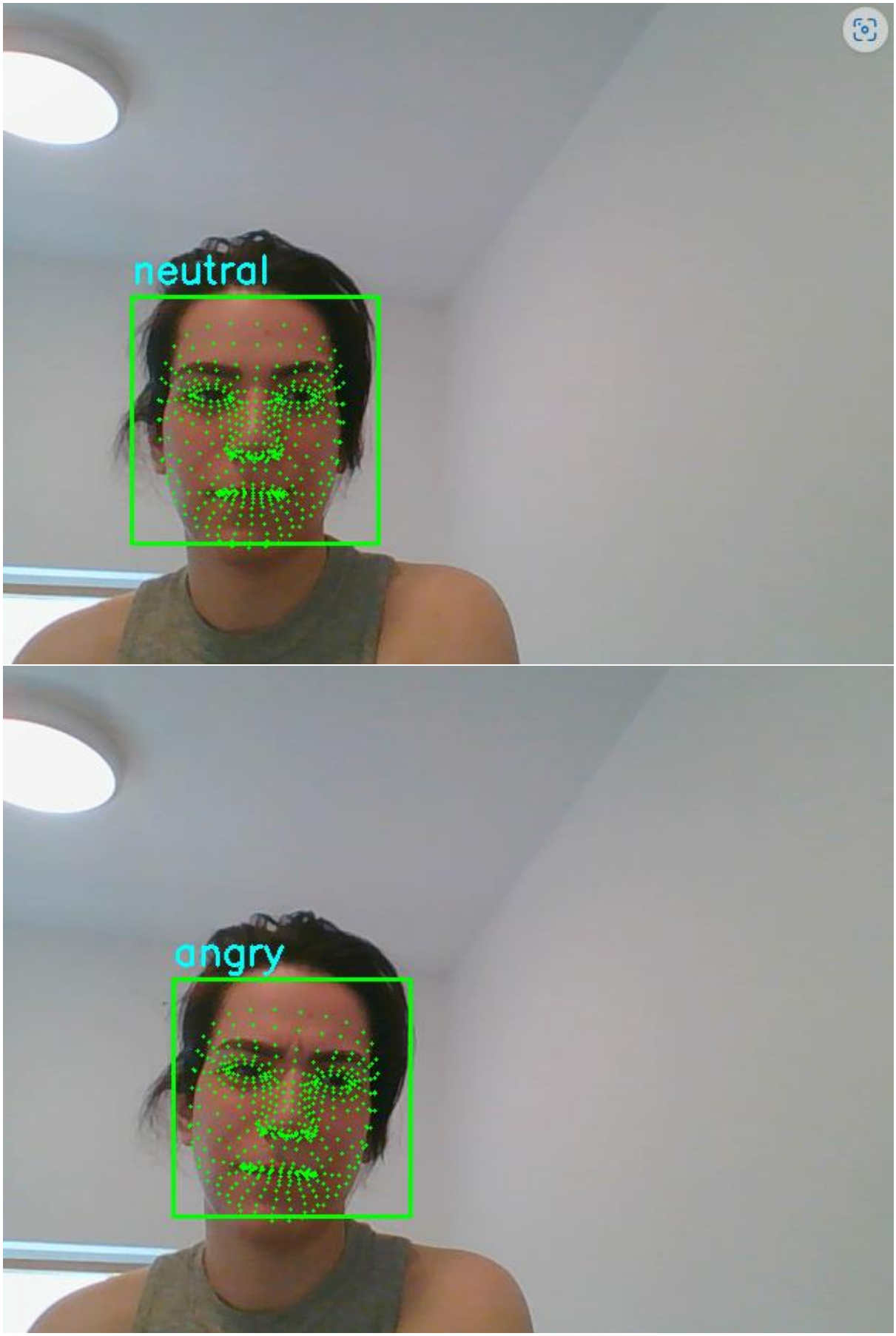

Command Prompt
-packages (from rich->keras>=2.0.0->fer) (2.19.1)
Requirement already satisfied: mdurl~=0.1 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (fr
om markdown-it-py>=2.2.0->rich->keras>=2.0.0->fer) (0.1.2)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-package
s (from jinja2->torch<2.3.0,>=2.2.0->facenet-pytorch->fer) (3.0.2)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-pack
ages (from sympy->torch<2.3.0,>=2.2.0->facenet-pytorch->fer) (1.3.0)
Installing collected packages: rich, proglog, pandas, matplotlib, torchvision, moviepy, keras, facenet-pytorch, fer
Successfully installed facenet-pytorch-2.6.0 fer-22.5.1 keras-3.8.0 matplotlib-3.10.0 moviepy-2.1.2 pandas-2.2.3 proglog
-0.1.10 rich-13.9.4 torchvision-0.17.2
[notice] A new release of pip available: 22.3 -> 25.0.1
[notice] To update, run: C:\Users\User\AppData\Local\Programs\Python\Python311\python.exe -m pip install --upgrade pip
C:\Users\User\AppData\Local\Programs\Python\Python311\Scripts>pip3 install matplotlib
Requirement already satisfied: matplotlib in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (3.
10.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\user\appdata\local\programs\python\python311\lib\site-packag
es (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (
from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-packa
ges (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\user\appdata\local\programs\python\python311\lib\site-packa
ges (from matplotlib) (1.4.8)
Requirement already satisfied: numpy>=1.23 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (f
rom matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-package
s (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (fro

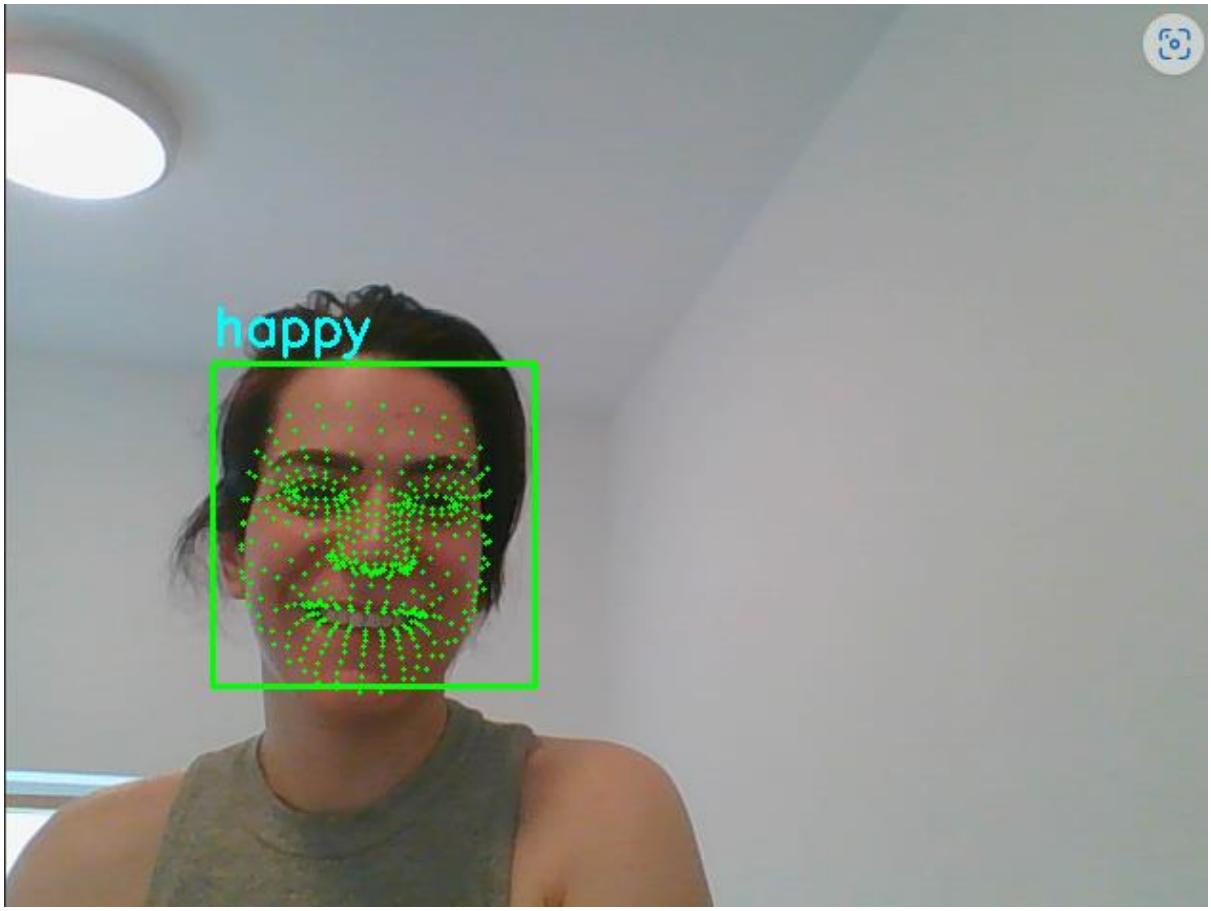

Command Prompt
Requirement already satisfied: colorama in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from
tqdm->proglog<=1.0.0->moviepy) (0.4.6)
[notice] A new release of pip available: 22.3 -> 25.0.1
[notice] To update, run: C:\Users\User\AppData\Local\Programs\Python\Python311\python.exe -m pip install --upgrade pip
C:\Users\User\AppData\Local\Programs\Python\Python311\Scripts>pip2 install moviepy
'pip2' is not recognized as an internal or external command,
operable program or batch file.
C:\Users\User\AppData\Local\Programs\Python\Python311\Scripts>pip3 install moviepy
Requirement already satisfied: moviepy in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (2.1.2
)
Requirement already satisfied: decorator<6.0,>=4.0.2 in c:\users\user\appdata\local\programs\python\python311\lib\site-p
ackages (from moviepy) (5.1.1)
Requirement already satisfied: imageio<3.0,>=2.5 in c:\users\user\appdata\local\programs\python\python311\lib\site-packa
ges (from moviepy) (2.37.0)
Requirement already satisfied: imageio_ffmpeg>=0.2.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-p
ackages (from moviepy) (0.6.0)
Requirement already satisfied: numpy>=1.25.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages
(from moviepy) (1.26.4)
Requirement already satisfied: proglog<=1.0.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages
(from moviepy) (0.1.10)
Requirement already satisfied: python-dotenv>=0.10 in c:\users\user\appdata\local\programs\python\python311\lib\site-pac
kages (from moviepy) (1.0.1)
Requirement already satisfied: pillow<11.0,>=9.2.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-pac
kages (from moviepy) (10.2.0)
Requirement already satisfied: tqdm in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from pro
glog<=1.0.0->moviepy) (4.67.1)
Requirement already satisfied: colorama in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from

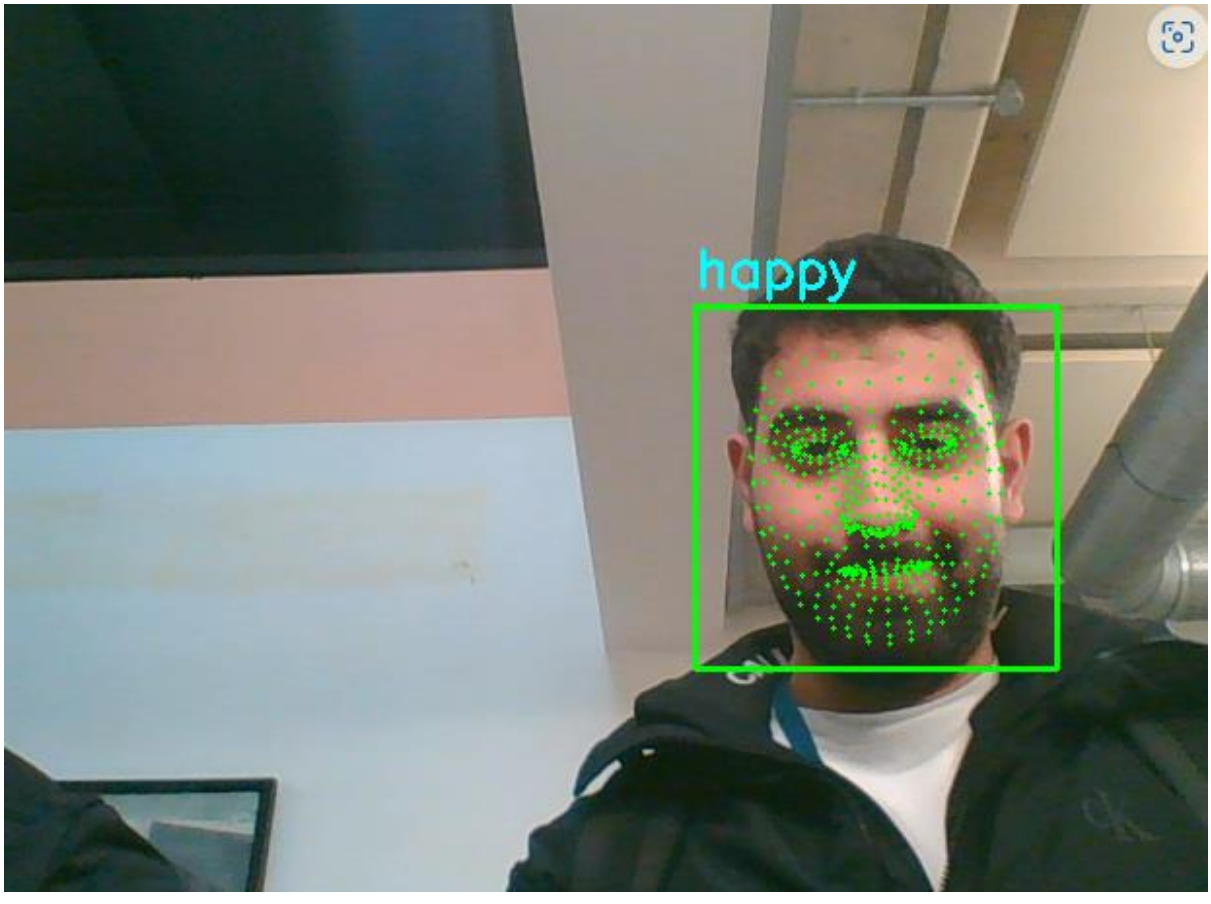

Command Prompt
Requirement already satisfied: python-dotenv>=0.10 in c:\users\user\appdata\local\programs\python\python311\lib\site-pac
kages (from moviepy) (1.0.1)
Requirement already satisfied: pillow<11.0,>=9.2.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-pac
kages (from moviepy) (10.2.0)
Requirement already satisfied: tqdm in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from pro
glog<=1.0.0->moviepy) (4.67.1)
Requirement already satisfied: colorama in c:\users\user\appdata\local\programs\python\python311\lib\site-packages (from
tqdm->proglog<=1.0.0->moviepy) (0.4.6)
[notice] A new release of pip available: 22.3 -> 25.0.1
[notice] To update, run: C:\Users\User\AppData\Local\Programs\Python\Python311\python.exe -m pip install --upgrade pip
C:\Users\User\AppData\Local\Programs\Python\Python311\Scripts>
C:\Users\User\AppData\Local\Programs\Python\Python311\Scripts>pip3 install moviepy==1.0.3
Collecting moviepy==1.0.3
Downloading moviepy-1.0.3.tar.gz (388 kB)
388.3/388.3 kB 691.2 kB/s eta 0:00:00
Preparing metadata (setup.py) ... done
Collecting decorator<5.0,>=4.0.2
Downloading decorator-4.4.2-py2.py3-none-any.whl (9.2 kB)
Requirement already satisfied: tqdm<5.0,>=4.11.2 in c:\users\user\appdata\local\programs\python\python311\lib\site-packa
ges (from moviepy==1.0.3) (4.67.1)
Requirement already satisfied: requests<3.0,>=2.8.1 in c:\users\user\appdata\local\programs\python\python311\lib\site-pa
ckages (from moviepy==1.0.3) (2.32.3)
Requirement already satisfied: proglog<=1.0.0 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages
(from moviepy==1.0.3) (0.1.10)
Requirement already satisfied: numpy>=1.17.3 in c:\users\user\appdata\local\programs\python\python311\lib\site-packages
(from moviepy==1.0.3) (1.26.4)
Requirement already satisfied: imageio<3.0,>=2.5 in c:\users\user\appdata\local\programs\python\python311\lib\site-packa
ges (from moviepy==1.0.3) (2.37.0)

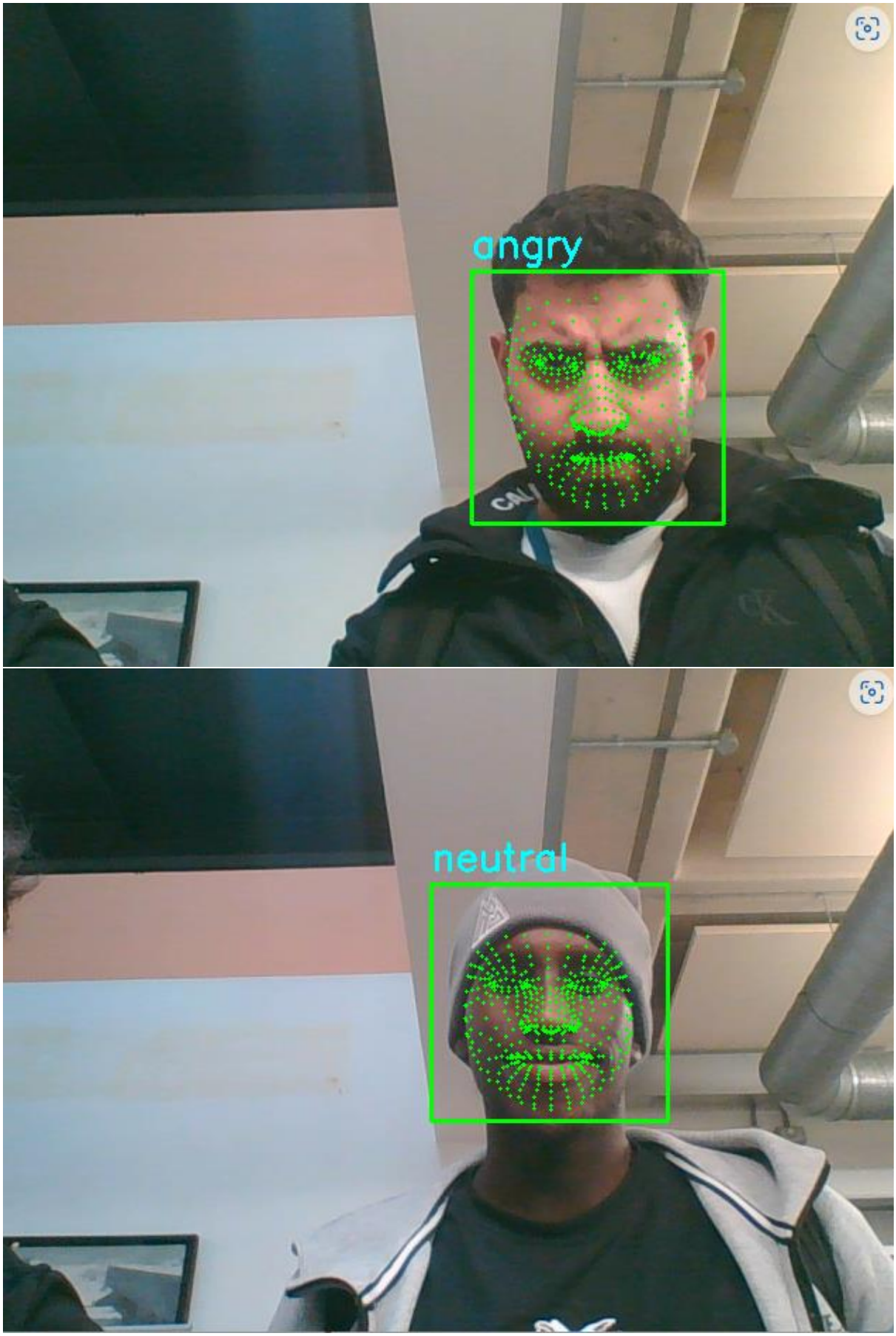

Command Prompt - pip3 install tensorflow
Attempting uninstall: decorator
Found existing installation: decorator 5.1.1
Uninstalling decorator-5.1.1:
Successfully uninstalled decorator-5.1.1
Attempting uninstall: moviepy
Found existing installation: moviepy 2.1.2
Uninstalling moviepy-2.1.2:
Successfully uninstalled moviepy-2.1.2
DEPRECATION: moviepy is being installed using the legacy 'setup.py install' method, because it does not have a 'pyproj
ect.toml' and the 'wheel' package is not installed. pip 23.1 will enforce this behaviour change. A possible replacement
is to enable the '--use-pep517' option. Discussion can be found at https://github.com/pypa/pip/issues/8559
Running setup.py install for moviepy ... done
Successfully installed decorator-4.4.2 moviepy-1.0.3
[notice] A new release of pip available: 22.3 -> 25.0.1
[notice] To update, run: C:\Users\User\AppData\Local\Programs\Python\Python311\python.exe -m pip install --upgrade pip
C:\Users\User\AppData\Local\Programs\Python\Python311\Scripts>pip3 install tensorflow
Collecting tensorflow
Downloading tensorflow-2.18.0-cp311-cp311-win_amd64.whl (7.5 kB)
Collecting tensorflow-intel==2.18.0
Downloading tensorflow_intel-2.18.0-cp311-cp311-win_amd64.whl (390.2 MB)
95.3/390.2 MB 6.2 MB/s eta 0:00:48

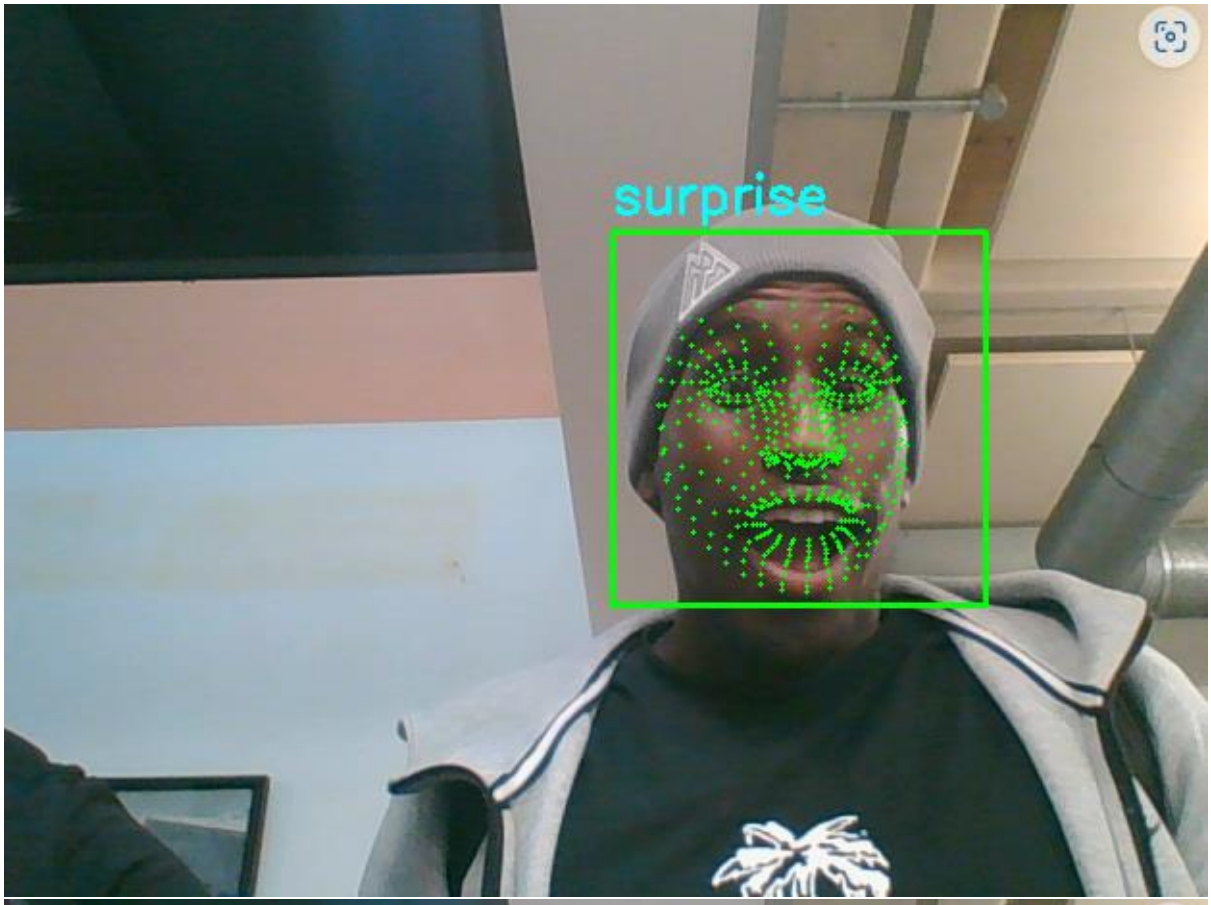

OpenSSH SSH client
C:\Users\User>ssh pi@192.168.58.129
ssh: connect to host 192.168.58.129 port 22: Connection timed out
C:\Users\User>ssh pi@192.168.58.129
The authenticity of host '192.168.58.129 (192.168.58.129)' can't be established.
ECDSA key fingerprint is SHA256:kV9fRISRzB7BaJ7ajO+fdnHiWC/62moKRQA60CxJErM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added '192.168.58.129' (ECDSA) to the list of known hosts.
pi@192.168.58.129's password:
Permission denied, please try again.
pi@192.168.58.129's password:
"SwiftBotOS 2024 (Uxbridge)"
For latest API and Resources: https://swiftbot-maven.brunel.ac.uk/
swiftbot00129:~$ ls
swiftbot00129:~$ pwd
/data/home/pi
swiftbot00129:~$

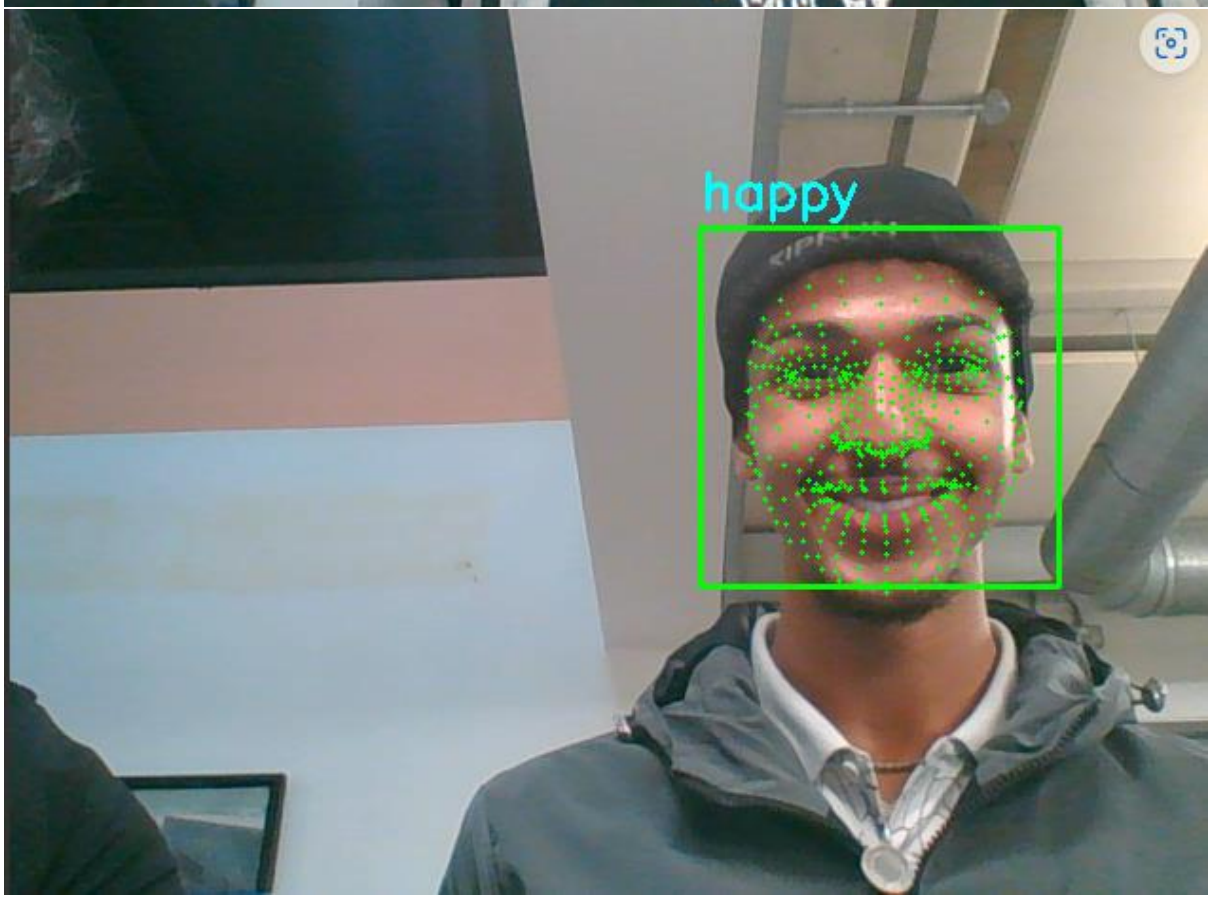

neutral
angry

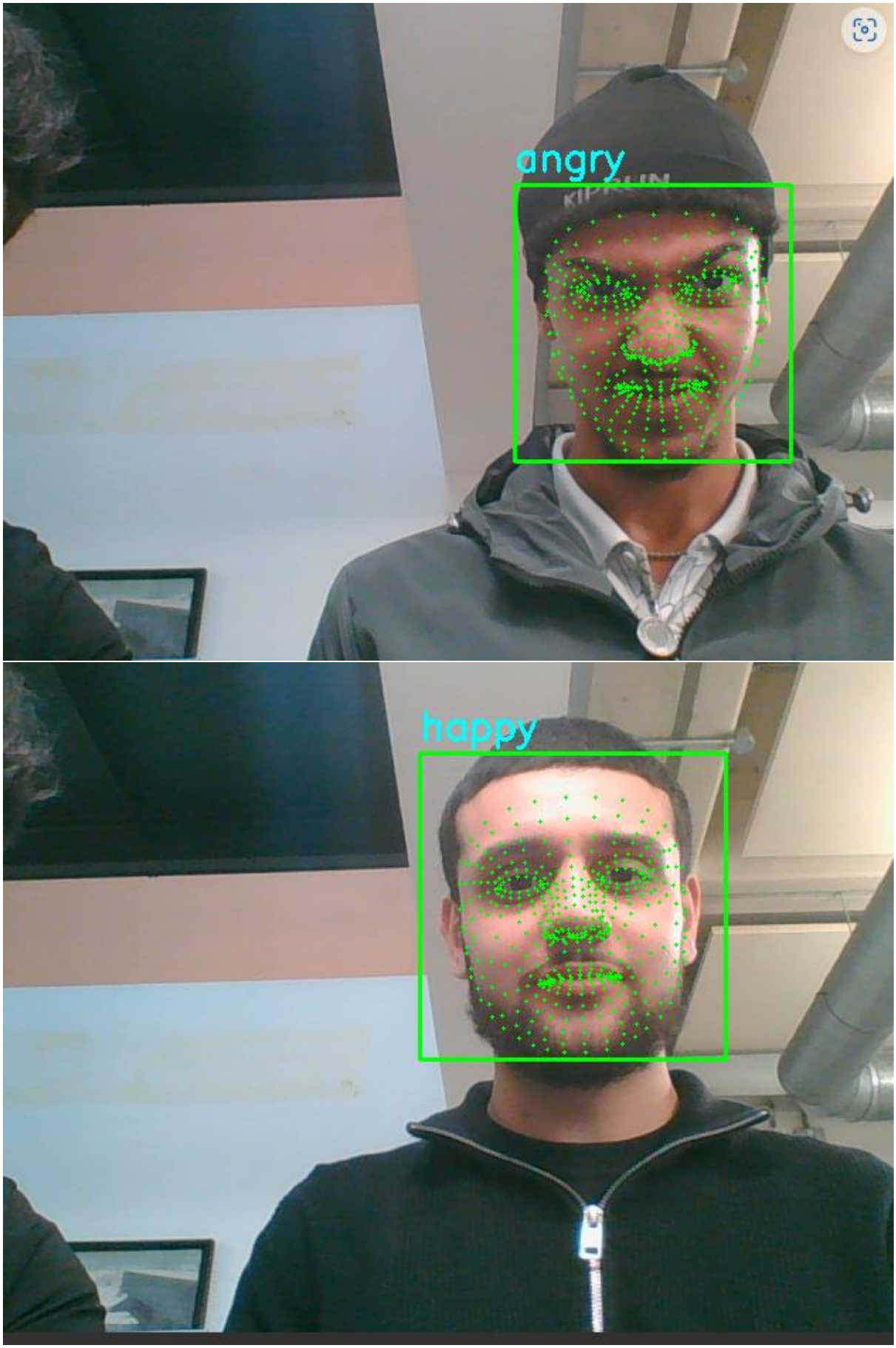

happy

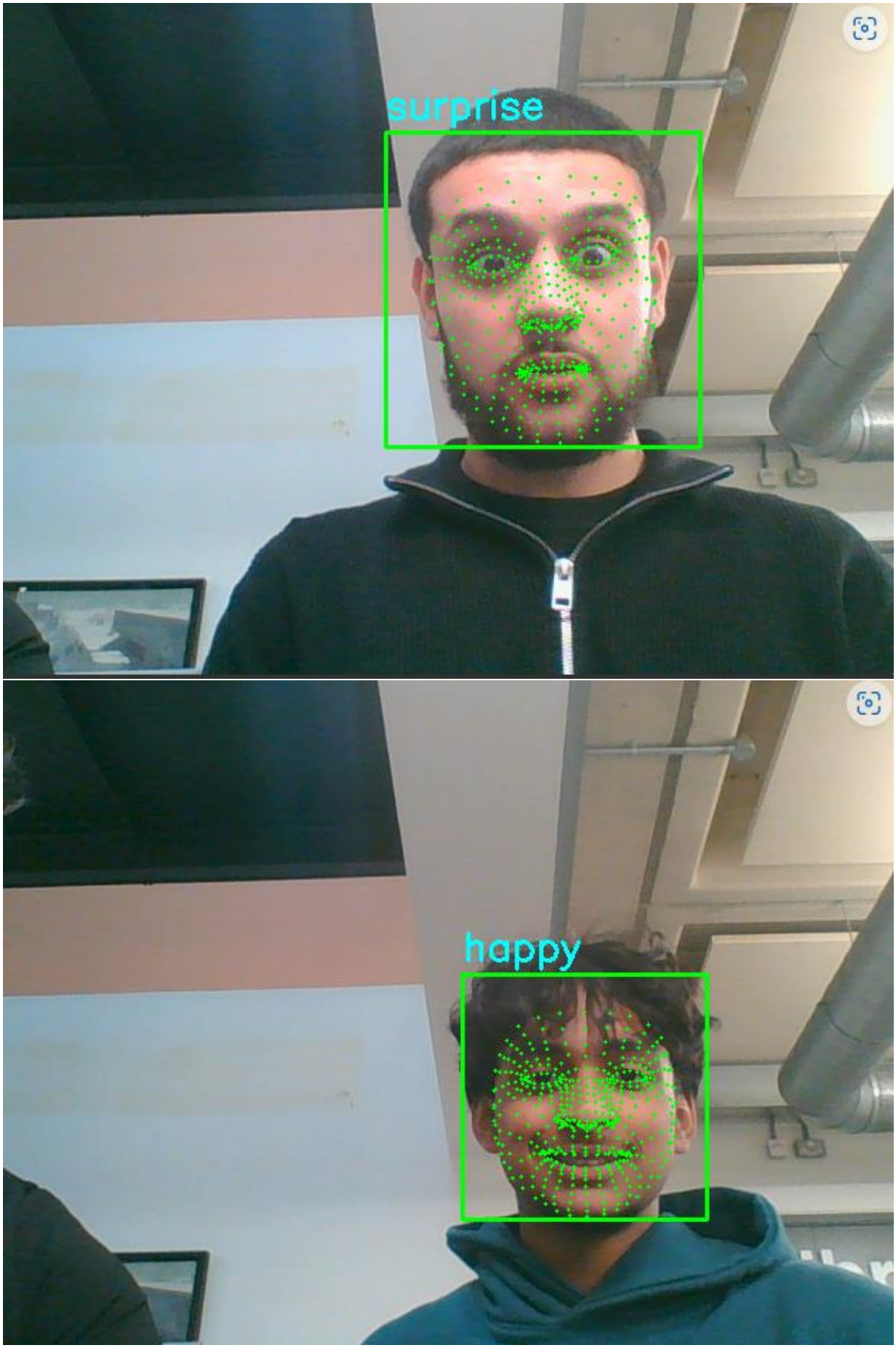

happy

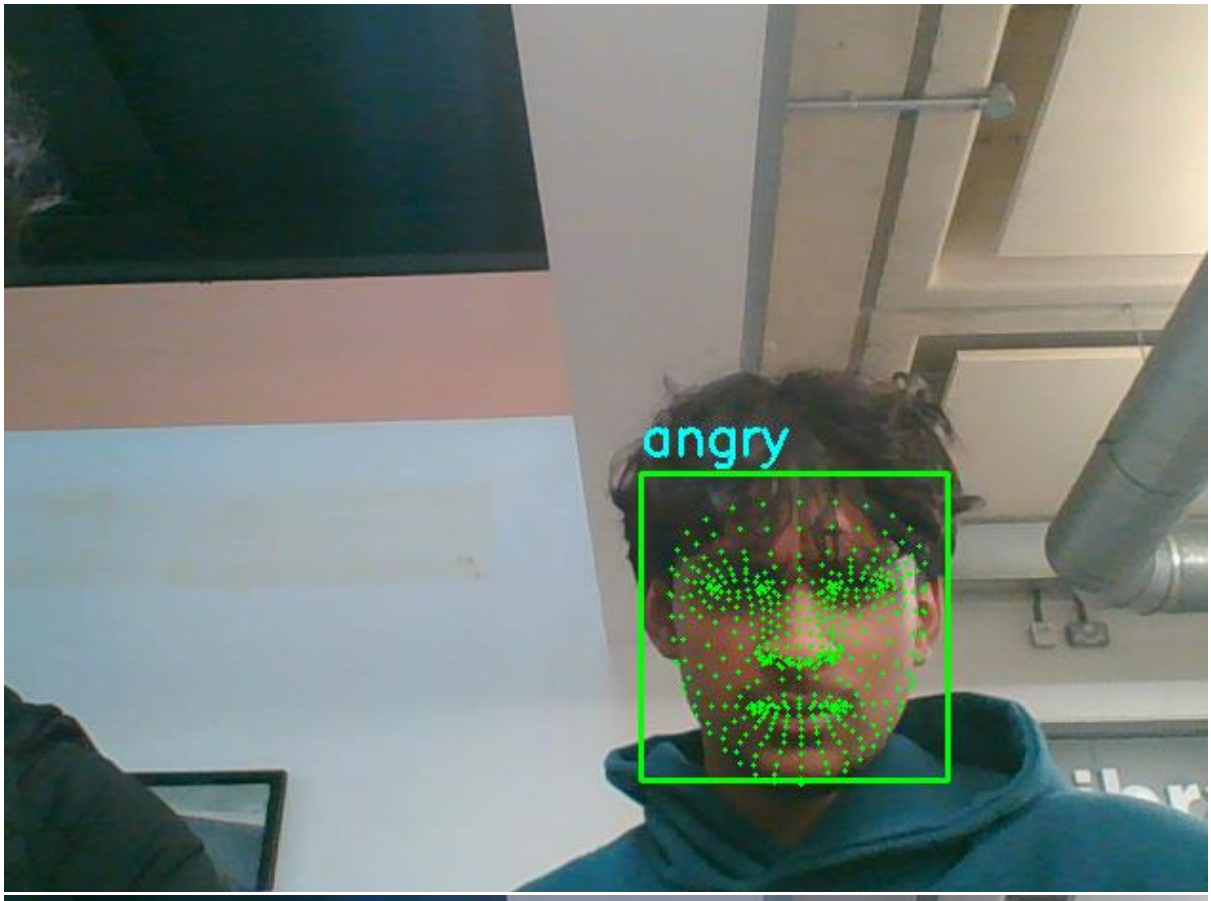

angry
neutral

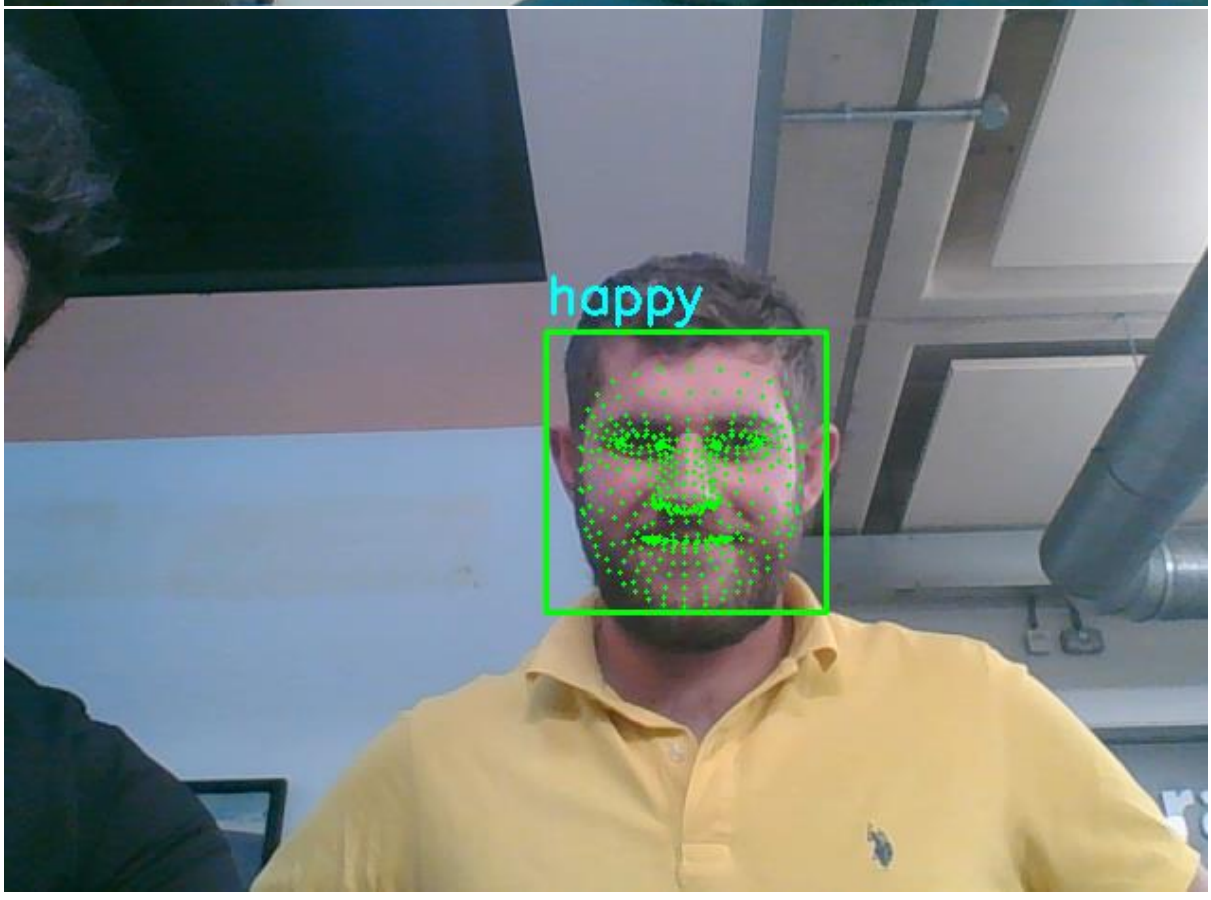

surprise

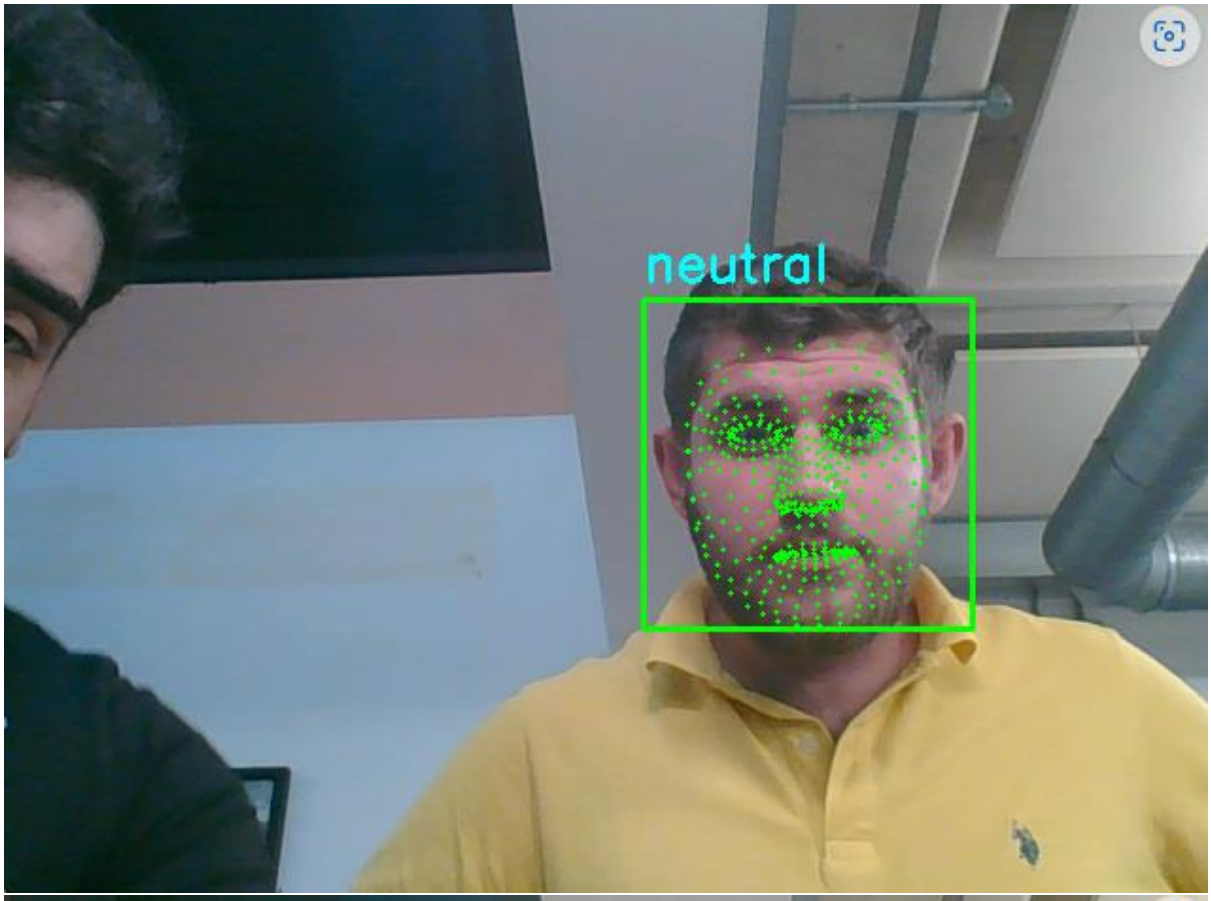

happy


angry
happy

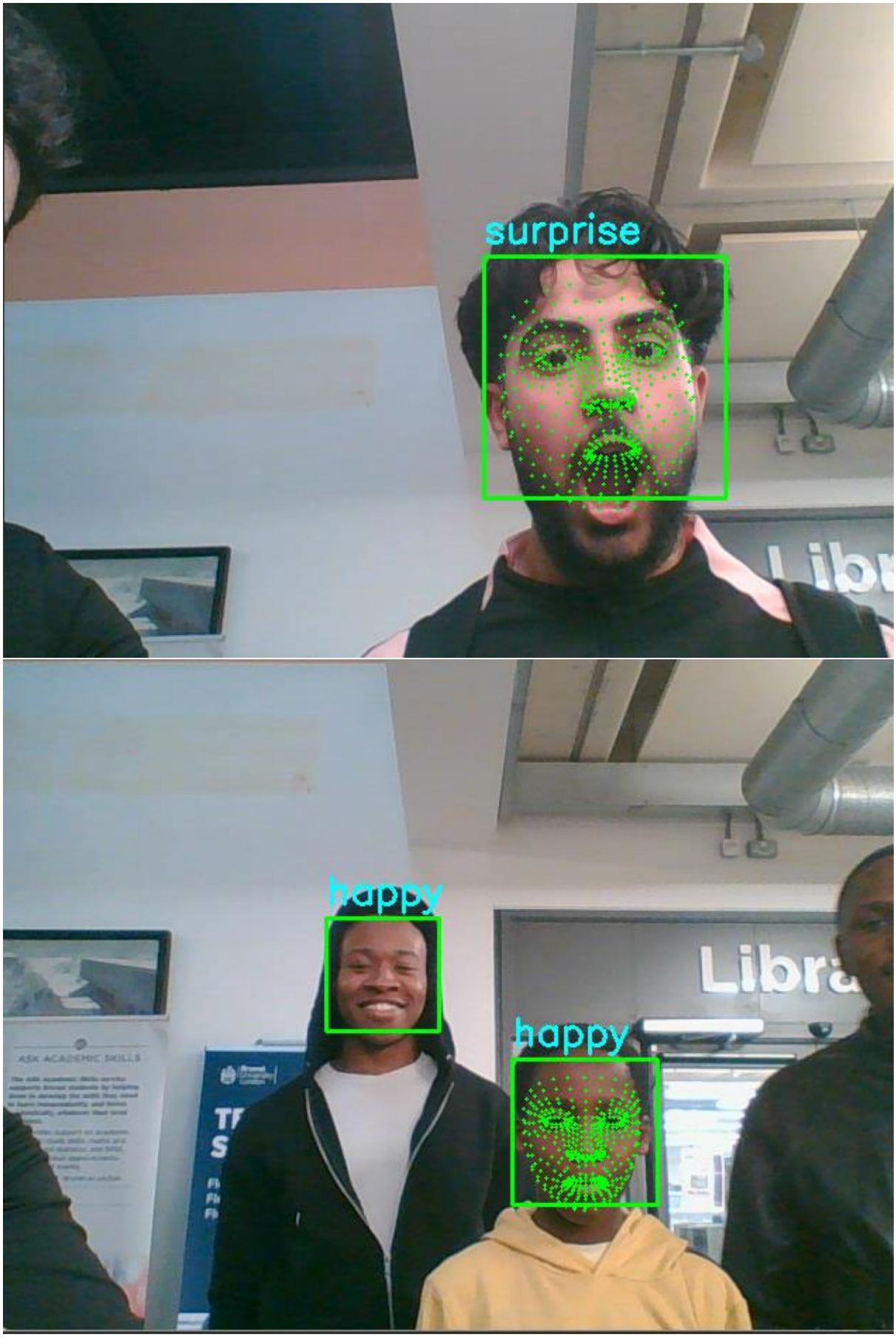

surprise
happy


angry


happy


neutral


happy


surprise
happy
happy