Coventry University Group

Module: 603EN

Project Title: Face tracking phone camera stand

Author: Oluwaseun Odusanya

SID: 1016 3553

Supervisor: Shah Karim

Word count: 6579

Academic year: 2024/2025

Coventry University Group
Coventry
London
Scarborough

Contents

Coventry University Group
Coventry
London
Scarborough

List of Figures

Abstract

This report details the development and implementation of a face and gesture tracking system, which incorporates advanced embedded systems, machine learning, and robotics to achieve real-time tracking capabilities. The project aimed to create a versatile tracking system capable of three distinct modes: hand gesture tracking, face tracking, and manual control. These functionalities were integrated into an interactive and user-friendly Graphical User Interface (GUI) using Python's Tkinter library. While the system initially functioned effectively, delivering successful tracking outcomes, it experienced hardware failure due to structural bending forces, rendering it inoperable.

The project utilised an Arduino microcontroller and Python programming to interface with servo motors and sensors. For face and gesture detection, machine learning algorithms were implemented, leveraging libraries such as OpenCV to process visual input from a camera module. The design involved precise servo motor control to enable smooth and accurate tracking movements. Hand gesture tracking relied on pre-defined gestures to adjust the system's position dynamically, while face tracking used Haar Cascade classifiers for consistent recognition and tracking. The manual mode provided users with direct control via a Tkinter GUI.

Throughout the testing phase, the tracking system demonstrated effective performance in all modes. Hand gestures were recognised accurately, and face tracking was responsive to movement and lighting variations. The integration of hardware and software was successful in providing a reliable tracking solution during initial demonstrations.

However, despite the system's promising performance, it encountered structural challenges that ultimately compromised its operation. The hardware's physical framework was subjected to bending forces during extended testing and usage, which exceeded the material limits of the supporting components. This mechanical failure disrupted the alignment of sensors and motors, preventing further operation of the system. Although the software components, including the GUI and tracking algorithms, remained functional, the hardware issues underscored the importance of robust mechanical design in embedded systems.

In conclusion, the face and gesture tracking project showcased significant achievements in embedded systems and GUI design, delivering a functional and versatile tracking solution. While the hardware failure limited its operational longevity, the project serves as a valuable case study in the integration of software and hardware for innovative engineering applications. Future improvements could focus on enhancing the durability of mechanical components to complement the robust software framework.

Introduction

In recent years, video content has become a central element of social media, fitness instruction, and virtual communication. Whether capturing dynamic performances, conducting presentations, or recording exercise routines, many users rely on traditional camera stands and tripods. However, these conventional solutions present a fundamental limitation: they require manual adjustment to keep the subject in frame. This project proposes a face-tracking camera stand that eliminates the need for manual adjustments by using a mobile phone camera that autonomously follows the movement of a person. With such a system, users can focus entirely on their activities, knowing that the camera will maintain optimal positioning.

Tracking Software analysis

- The face tracking and gesture tracking software operate as intended by successfully utilising the haar cascade available with computer vision.

Motor control responsiveness analysis

- The conversion of the tracked coordinates to angles correctly corresponds with the angles used on the servo motors.

Hardware implementation analysis

- Whilst the hardware and software worked cohesively initially, due to the load and bending involved the joint for the vertical motor became unstable and is no longer operational and one of the projects failures.

The primary goal of this project is to design and develop a prototype of an automatic face-tracking tripod that uses Arduino-based embedded systems and OpenCV computer vision. The camera stand, controlled by servo motors, will detect and track a subject's face in real time. This combination of software and hardware enables automatic tracking, allowing the camera to pan and tilt to keep the subject consistently centred in the frame. By leveraging computer vision and embedded systems, this project addresses the need for an autonomous camera solution that can benefit a variety of users, including content creators, fitness instructors, professionals conducting virtual presentations, and general consumers aiming to enhance the quality of their video calls.

Problem Statement

For many potential users, recording content while staying in frame is challenging, especially when moving around or focusing on other tasks. The conventional tripods used in filming, video calls, and live streams often require frequent manual adjustments, which not only interrupts the flow of activities but also results in footage that is less engaging. Content creators, for instance, who rely on seamless video quality, face particular challenges when recording themselves while performing various actions, often having to interrupt filming to adjust the camera manually.

Aim

Design and build a prototype of an automatic face tracking tripod for a phone camera that will be capable of being controlled as a response of tracking a person's movement.

Objectives

•Track a person's movement on camera using Object oriented programming (Python) and the OpenCV library (OpenCV is a machine learning library).

• Assemble a motor controlled phone camera tripod to hold the mobile phone so that manoeuvring of the camera is possible vertically and horizontally.

• Seamlessly integrate the tracking system (software code) with the camera hardware to ensure smooth operation.

• Test the prototype to evaluate the performance of the tracking and optimise the system. Provide a summary of the research, outlining the aims and objectives and/or research questions and the proposed research design and methods

Project completion plan

The aim of this project is to design and prototype an automatic face-tracking tripod for mobile phones, allowing the camera to autonomously follow a person's movement, thus delivering a streamlined filming experience. The key objectives of the project are as follows:

1. **Detection and Tracking**: The system will feature 3 detection modes: face tracking, gesture tracking and a manual control mode to be able to orientate the camera stand. Using Python and OpenCV, the project aims to develop a face-tracking algorithm that reliably detects and tracks a person's face in real time. This will serve as the foundation of the system, ensuring accurate detection and tracking under various conditions. The success of this objective will be assessed based on the system's capability to consistently and smoothly track a person's face, even when environmental factors, such as lighting, change.

2. **Motor-Controlled Camera Stand Assembly**: The prototype will include servo motors to control vertical and horizontal camera movements. These motors will respond to the tracking system, enabling smooth camera adjustments. The performance of this component will be measured by the tripod's range of movement and responsiveness, ensuring that the motor adjustments are timely and correspond accurately to the subject's movements.
3. **Hardware and Software Integration**: For a seamless user experience, the camera tracking system needs to synchronise hardware and software effectively. The Arduino and motor control system will work in tandem with the OpenCV-based tracking software, allowing the camera to follow the subject without noticeable lag or jittering. The success of this integration will be evaluated by assessing the fluidity of the camera's movements, aiming to create a smooth filming experience without delays or interruptions.
4. **Prototype Testing and Optimisation**: Finally, extensive testing will assess the accuracy and smoothness of the face-tracking system. This will include refining the code and motor controls for enhanced performance, aiming to produce consistent tracking accuracy and minimal reaction time.

Model Requirements

The proposed face-tracking camera stand prototype will need to meet specific requirements to ensure reliable and smooth operation. These requirements include:

- **Tracking Accuracy**: The system must be capable of consistently identifying and following the subject's face in real time, adjusting for environmental conditions such as lighting changes.
- **Motor Responsiveness**: Servo motors should provide rapid and smooth adjustments to keep the camera aligned with the subject's movements, with minimal lag or jittering.
- **Software-Hardware Synchronisation**: Real-time integration between the OpenCV tracking software and the motor-controlled hardware is essential for smooth, uninterrupted operation.
- **Portability and Ease of Use**: Given the range of potential users, the camera stand should be designed to be compact, portable, and user-friendly, requiring minimal setup.

This face-tracking camera stand project addresses the growing demand for hands-free, autonomous camera systems in content creation, fitness recording, and professional video presentations. By focusing on developing a prototype that integrates advanced computer vision techniques with embedded system control, this project offers an innovative solution that reduces the need for manual camera adjustments and enables users to produce high-quality, consistent video content effortlessly.

Literature review

2.01 Themes of the literature review

The project's themes are organised as follows:

- **Computer Vision Software**
  - Discusses the project's face-tracking process.
  - Details the machine learning software employed.
- **Servo Motors in Embedded Systems**
  - Explains embedded systems and their usefulness.
  - Identifies the microcontroller selected.
- **Object-Oriented Programming Techniques**
  - Explores basic programming concepts.
  - Demonstrates their application to the project.

2.1 Computer Vision

2.11 Background Information

Computer vision, an artificial intelligence subfield, has rapidly progressed in recent years, focusing on enabling machines to interpret and act on visual data, typically from images or video. Object detection, which identifies items within an image, is foundational to computer vision. Convolutional neural networks (CNNs) are commonly used in this context due to their accuracy (Krizhevsky et al., 2012).

Real-time applications, where quick processing is essential, are another active research area, employing algorithms like the Kalman Filter, Mean Shift, and DeepSORT to balance precision with efficiency (Luo et al., 2024). The YOLO (You Only Look Once) algorithm, known for its speed and detection accuracy, is particularly useful in dynamic environments (Redmon et al., 2016). In facial recognition, Multi-task Cascaded Convolutional Networks (MTCNN) are effective in face detection and alignment (Zhang et al., 2016). Newer methods, such as the Discriminative Correlation Filter (DCF), have demonstrated reliable object tracking even during movement, achieving high precision (Danelljan et al., 2016). These advancements apply broadly across fields like surveillance and autonomous systems. For this project, facial detection is essential, as it provides the data needed for camera orientation.

2.12 Relevant Existing Projects

Two key projects highlight the use of computer vision for similar objectives. Viraktamath et al. (2013) explore face detection and tracking using OpenCV, a central aspect of this project's computer vision requirements. Their work details OpenCV's pre-built libraries, which detect facial features in real-time using video input. OpenCV algorithms, including Haar cascades, examine pixel intensities in facial regions like the eyes, nose, and mouth to distinguish faces from other objects in the camera's view.

This principle is applicable to the face-tracking phone stand by incorporating a camera that captures images continuously. OpenCV's algorithms can identify and locate faces within each frame, enabling smooth, frame-by-frame detection. Once the system recognises a face, it calculates its position relative to the frame's centre, which then guides the phone stand's motor adjustments to keep the subject's face in focus as they move.

Goyal et al. (2017) extend this understanding with more advanced detection and tracking techniques, optimised for real-time applications. Their work underscores the importance of balancing accuracy with processing speed, essential in a face-tracking phone stand. They describe improved methods, such as Local Binary Patterns (LBP), that enhance detection rates while lowering computational demand.

Additionally, Goyal et al. (2017) highlight OpenCV's tracking algorithms, such as MedianFlow and KCF, for reliably following detected faces across frames. These trackers identify key facial features initially and predict their movement in subsequent frames, enabling the camera to track a face smoothly even with sudden movements. Combining face detection and tracking algorithms, the proposed system can maintain real-time focus on a subject, ensuring a seamless experience.

Together, these studies provide valuable insights: Viraktamath et al. (2013) cover the basics of detection, while Goyal et al. (2017) offer enhanced tracking techniques. By combining these elements, a face-tracking phone stand can achieve continuous, accurate tracking, seamlessly integrating vision software with motor hardware for reliable performance.

Haar Cascade classifiers are an effective way for object detection. This method was proposed by Paul Viola and Michael Jones in their paper Rapid Object Detection using a Boosted Cascade of Simple Features (Viola, Paul & Jones, Michael. 2001).

2.13 Application to a Face-Tracking Camera Stand

To develop a face-tracking camera stand, the integration of real-time face detection and tracking systems is necessary. By using CNNs for initial face detection and DCF-based tracking for continuous focus, the camera can adjust its position dynamically. YOLO or MTCNN can facilitate initial detection, while DCF-based tracking maintains smooth tracking as the subject moves.

For this project, a microcontroller with a camera module will interact with machine learning models for face detection and a servo motor system for real-time orientation adjustments. Here, a laptop camera and USB connection will serve as the interface for embedded instructions.

2.2 Servo Motors in Embedded Systems

Arduino, created by Massimo Banzi and his team in 2005, offers an accessible, affordable platform for electronics and embedded systems (Banzi & Shiloh, 2014). Developed for open-source use, Arduino boards are based on AVR microcontrollers, supporting both digital and analogue I/O pins, which allows for a variety of applications, from automation to IoT projects (Monk, 2013).

The flexibility of Arduino, especially in controlling sensors, motors, and actuators, has made it indispensable in educational and prototype projects. Its capability to integrate with Wi-Fi or Bluetooth also makes it ideal for remote monitoring systems. Many DIY enthusiasts use Arduino for projects such as weather stations, wearables, and automated vehicles (Norrie, 2020). Its popularity arises from its open-source design, community support, and compatibility with a variety of sensors and modules.

In embedded systems, servo motors are crucial in applications needing precise control, like robotic arms or surveillance systems. Khalil et al. (2021) highlight how servo motors, regulated by PWM signals, allow control over angular position, speed, and torque. Sharma et al. (2020) emphasises the importance of real-time performance in these systems, typically achieved with microcontrollers like Arduino. Yadav and Patel (2019) underscore the role of feedback loops in maintaining accurate motor positioning, a common requirement in tracking systems.

This knowledge is applicable to the face-tracking camera stand, where a microcontroller with servo motors receives real-time data from a camera. Machine learning algorithms process this data, guiding the motors to adjust the camera's orientation.

2.21 Relevant Existing Projects

In 2017, Ayi et al. discussed a face-tracking system using MATLAB and Arduino, providing insights into motor control for real-time face alignment. In a similar phone stand setup, the embedded system interprets camera input to track a face, then directs servo motors to adjust positioning, keeping the face centred.

Small, precise servo motors are vital for this purpose, rotating the stand to ensure continuous facial alignment within the frame. Ayi et al. (2017) demonstrate how control signals enable the motors to respond to facial movements, making the system responsive and efficient for dynamic tracking.

2.22 Application to a Face-Tracking Camera Stand

This project will use an Arduino setup operating MG996R servo motors, as provided in the Adeept 5-DOF Robotic Arm Kit. Given time constraints, existing components will be modified, prioritising software development for face tracking to prevent scope expansion.

2.3 Object-Oriented Programming Techniques

Languages like Python and C++ enable object-oriented programming (OOP), an approach that facilitates code modularity and reusability by organising software into objects representing real-world elements. Python's simplicity makes it ideal for rapid prototyping with libraries such as OpenCV, which detects and tracks faces in real time (Raschka, 2015). For hardware tasks, C++ provides efficient motor control, essential for the phone stand's adjustments (Stroustrup, 2013).

OOP principles, including inheritance, help design motor control classes with shared properties, ensuring consistent, efficient code structure (Li & Bahsoon, 2018). Python will manage higher-level functions and face tracking, while C++ handles hardware interaction for optimised performance.

Different microcontroller systems, like the LPC1768 and Arduino, may require language adjustments based on compatibility, as the LPC1768 primarily uses C++ while Arduino employs a variant (Pan & Zhu, 2018).

2.31 Application in a Face-Tracking Camera Stand

The project's Arduino system from the Adeept Robotic Arm Kit requires instructions in Arduino code, while face tracking will use Python and OpenCV. OOP allows modular design, with separate objects for the camera, motors, and tracking algorithm, each encapsulating its respective functionality. For instance, a camera object may inherit properties from a general sensor class, while polymorphism allows dynamic positioning adjustments in response to real-time face detection.

Methodology

Preparations

The initial stage of the project involved procuring the necessary equipment to ensure the development could proceed seamlessly. The following components were acquired: an Adeept Embed board, three MG996R servo motors, Adeept robotics kit components for the frame, a tripod stand attachment, and a phone holder camera attachment. The AD002 motors were chosen for their high torque, with one motor dedicated to horizontal movement and two motors installed vertically to support the system's load.

To facilitate the coding and implementation phases, relevant software applications and libraries for both Python and Arduino were installed. For Python, the primary libraries included NumPy, essential for numerical computations, and PySerial for serial communication with the Adeept Embed. For Arduino, the Adafruit_SSD1306.h library was installed to manage the OLED display. These libraries were selected based on their compatibility with the project's hardware and the specific functionality they provided.

Additionally, the necessary development environments were set up. Python scripts were executed in an IDE capable of handling library dependencies, while the Arduino IDE was configured for programming the Adeept Embed board. Care was taken to ensure compatibility across platforms by testing library installation and resolving dependency conflicts in advance.

This preparation stage laid a robust foundation for the project, enabling efficient transitions into hardware assembly and software development.

To prepare for the software development of the project the relevant libraries needed to be installed for the necessary features to be available in both the python and the Arduino programs. There are essential python libraries that must be installed before programming:

- Pyserial
- Cv2
- Numpy
- Tkinter
- Threading

The installation process has been done in the command prompt with examples seen in figures 1 and 2.

```
Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\soduu>pip install pyserial
Collecting pyserial
  Downloading pyserial-3.5-py2.py3-none-any.whl.metadata (1.6 kB)
Downloading pyserial-3.5-py2.py3-none-any.whl (90 kB)
Installing collected packages: pyserial
Successfully installed pyserial-3.5

[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\soduu>
```

Figure 1, pyserial installation

```
Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\soduu>pip -V
pip 24.2 from C:\Users\soduu\AppData\Local\Programs\Python\Python312\Lib\site-packages\pip (python 3.12)

C:\Users\soduu>pip install opencv-python
Collecting opencv-python
  Downloading opencv_python-4.10.0.84-cp37-abi3-win_amd64.whl.metadata (20 kB)
Collecting numpy>=1.21.2 (from opencv-python)
  Downloading numpy-2.1.3-cp312-cp312-win_amd64.whl.metadata (60 kB)
Downloading opencv_python-4.10.0.84-cp37-abi3-win_amd64.whl (38.8 MB)
   ---------------------------------------- 38.8/38.8 MB 15.6 MB/s eta 0:00:00
Downloading numpy-2.1.3-cp312-cp312-win_amd64.whl (12.6 MB)
   ---------------------------------------- 12.6/12.6 MB 16.8 MB/s eta 0:00:00
Installing collected packages: numpy, opencv-python
Successfully installed numpy-2.1.3 opencv-python-4.10.0.84

[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\soduu>
```

Figure 2, OpenCV installation

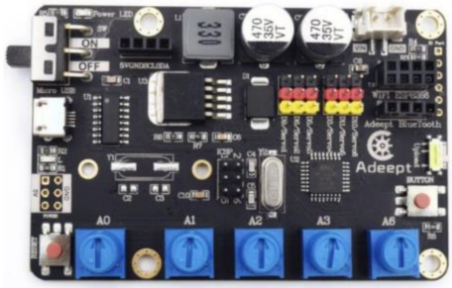
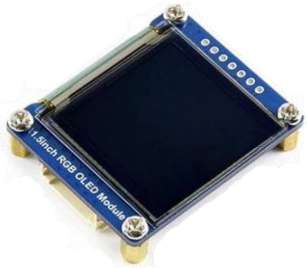
Before the embed could be programmed, essential Arduino libraries for this project must be installed as shown in figures 1 and 2 The libraries required have been selected based on the suggested libraries from Adeept. The software has been made to work cohesively with the embed hardware using the equipment listed in table 1.

Adafruit_BusIO	07/11/2024 16:32	File folder
Adafruit_GFX_Library	07/11/2024 16:32	File folder
Adafruit_SSD1306	07/11/2024 16:35	File folder
ArduinoHttpClient	11/11/2024 10:10	File folder
Robot_Control	11/11/2024 11:49	File folder
U8glib	11/11/2024 11:52	File folder
LiquidCrystal_I2C	11/11/2024 11:52	File folder
Keypad	11/11/2024 11:52	File folder
IRremote	11/11/2024 11:52	File folder
Dht11	11/11/2024 11:52	File folder
ArduinoJson	11/11/2024 11:52	File folder

Figure 3, Arduino libraries

Equipment:

Table 1, Equipment list

Equipment Acquired	
Component name	Component image
Adeept embed	 *Figure 4, (Arm, R., 2017)*
AD002 Micro Servo	 *Figure 5, AD002 servo motor (Arm, 2017)*
Adeept 4pcs 0.96 Inch OLED Module	 *Figure 6, OLED screen (Arm, R., 2017).*
USB connector	 *Figure 7, USB connector (Arm, R., 2017)*

Face Tracking Software Development

Software development began with pseudocode to outline the logical flow of the project, detailing how face tracking, gesture recognition, and manual control would interact with the hardware. The pseudocode was instrumental in breaking down the complex system into manageable modules, identifying variables, control structures, and potential points of integration (pseudocode can be found in the appendix).

Learning resources from platforms such as GeeksforGeeks, specifically the "guidance for Python | Haar Cascades for Object Detection" webpage has been used to further understand the OpenCV software.

Pseudocode is a high-level, informal description of code logic and structure, facilitating planning without strict syntax (Chand, 2023). Pseudocode was developed to plan the foundational logic and workflow which can be found in the appendix.

Additionally, the project has advanced with the integration of OpenCV's Haar cascades for face detection, successfully detecting faces in video frames and relaying coordinates to the Arduino system to adjust camera orientation.

Face Tracking Program

The face tracking software was the first module developed, leveraging OpenCV's Haar cascades for facial detection (Face tracking code can be found in the appendix). Frames captured from the webcam were converted into greyscale to enhance detection accuracy. A mask was applied over the image to highlight regions of interest, and the Haar cascade identified the user's face, drawing a rectangle around it. At this point, the face-tracking system is functional as evident in figure 4.

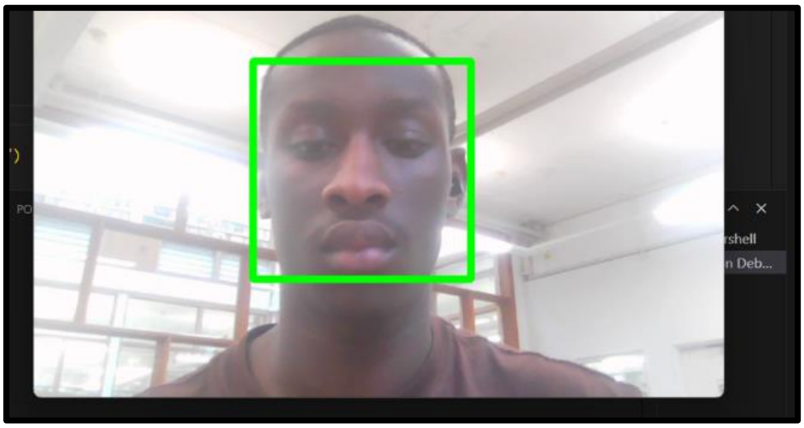


Figure 8, Face tracking

A graphical user interface (GUI) was also designed to enhance usability, though further refinements are expected based on user feedback.

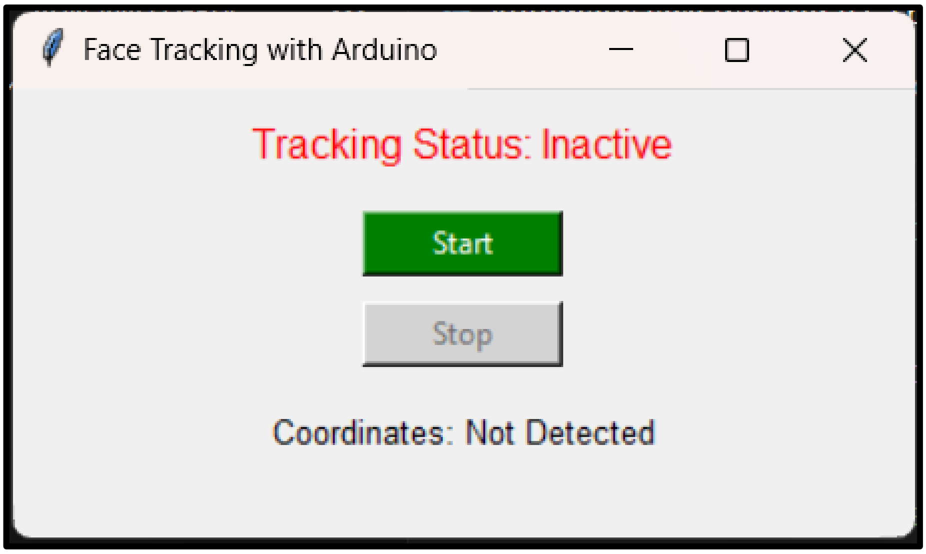


Figure 9, Inactive GUI

Once tracking takes place, the coordinates sent to the terminal are relayed to the interface in real-time.

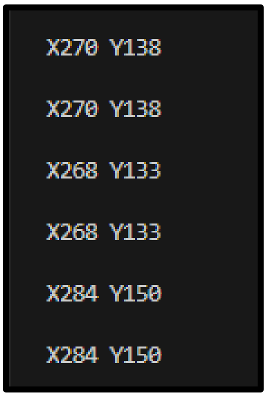


Figure 10, Tracking Coordinates

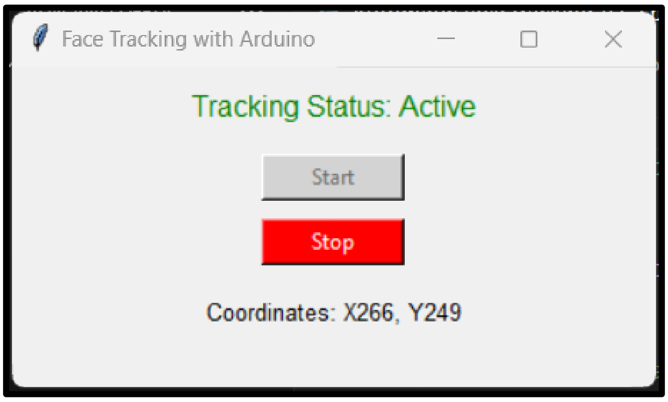


Figure 11, Active GUI

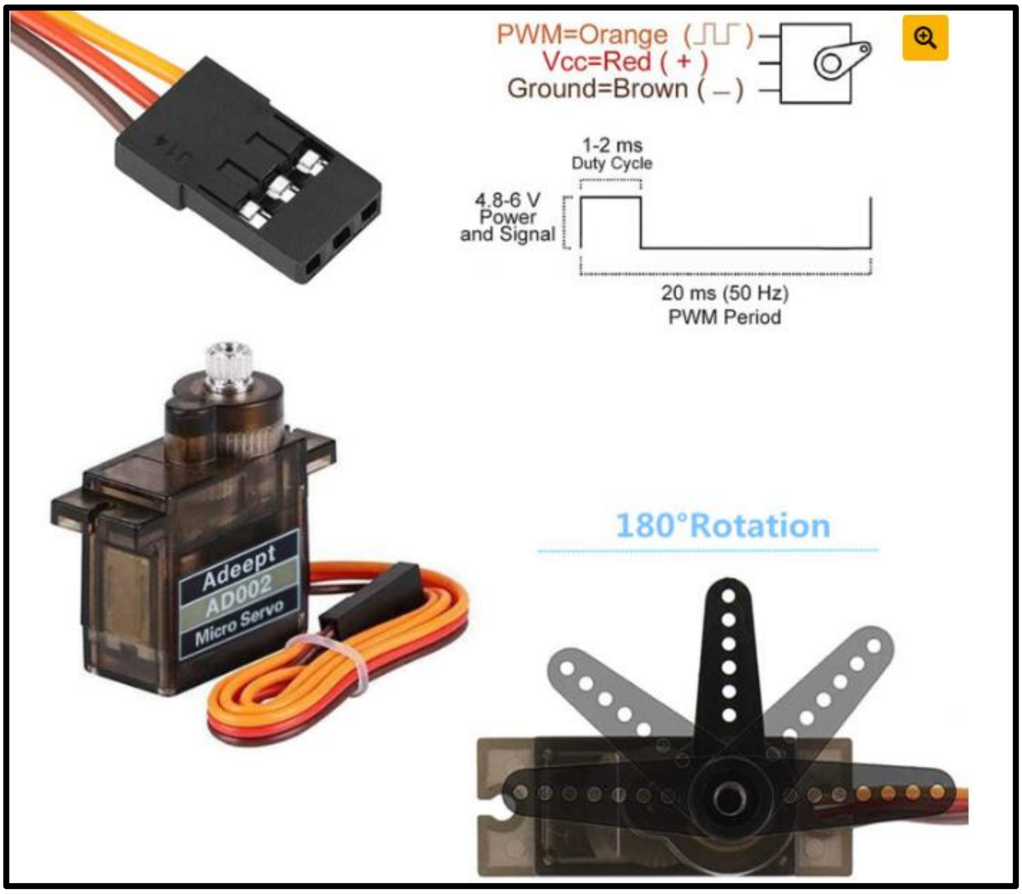


Figure 12, AD002 Servo motor (Arm, 2017)

Figure 8 shows the AD002 micro servo motor which will be used throughout the project with an 180 degree range of motion this range has been used to convert the coordinates to angles where pixels positioned at the centre of the Lense would have a horizontal angle of 90 degrees and a vertical angle of 90 degrees.

The face tracking python code and the face tracking Arduino code located in the appendix implements a face-tracking system that translates facial coordinates detected in a video feed into corresponding servo motor angles. This transformation ensures that the hardware, such as a camera or a robotic head, aligns its position to keep the face centred in its field of view. The process involves close interaction between the Python script running on a computer and the Arduino program controlling the servos. Here is a detailed explanation of how the transformation occurs:

Python Script (Computer Vision and Coordinate Transmission)

1. **Face Detection**:
   The Python script uses OpenCV, a library for computer vision tasks, to detect faces in real time. The CascadeClassifier loads a pre-trained Haar cascade file for frontal face detection. When a face is detected, the algorithm calculates its bounding box, defined by the coordinates of the top-left corner (x, y) and its dimensions (width, height).

2. **Coordinate Preparation**:
   The script formats the detected x and y coordinates into a string, e.g., "x_pos,y_pos\r". This string format is crucial for the Arduino, which parses these values for further processing. The coordinates represent the position of the face within the resolution of the video feed, which is 1920x1080 pixels.

3. **Serial Communication**:
   Using the PySerial library, the formatted coordinates are transmitted over a serial connection to the Arduino. The script also displays these coordinates in a graphical user interface (GUI) created with Tkinter, providing real-time feedback to the user about the detected face's position.

Arduino Program (Coordinate-to-Angle Transformation)

Step 1) Receiving Coordinates:
The Arduino continuously listens for incoming data on the serial port. When data is received, it reads the string until a carriage return (\r) and parses it to extract the x and y values. For example, the string "1000,540\r" would yield x_axis = 1000 and y_axis = 540.

Step 2) Mapping Coordinates to Servo Angles:
The map() function on the Arduino transforms the pixel-based coordinates to angles suitable for servo motors. The horizontal range of the video feed, typically from 0 to 1920, is mapped to the servo's horizontal motion range of 0° to 180°. Similarly, the vertical range, usually 0 to 1080, is mapped to 180° to 0° to ensure the servo moves correctly in the vertical plane. This inversion is necessary because increasing y-coordinates in the video feed represent downward motion, which corresponds to upward movement of the servo.

Step 3) Servo Control:
The Arduino sends the calculated angles to the respective servos using the write() function. For example, if the face is detected at the centre of the frame (960, 540), both servos would align to approximately 90°, which is their neutral or centre position.

Step 4) Feedback on OLED Display:
The Arduino displays the received coordinates and calculated angles on a connected OLED screen. This feedback is particularly useful for debugging or monitoring the system's performance in real time.

This process ensures that the system tracks the face dynamically, maintaining its position at the centre of the camera's field of view. The transformation from pixel coordinates to servo angles allows for seamless integration between the computer vision system and the hardware motion control.

The rectangle's coordinates were used to determine the x and y positions, which were then translated into angular commands for the servos, a schematic has been developed in EasyEDA.

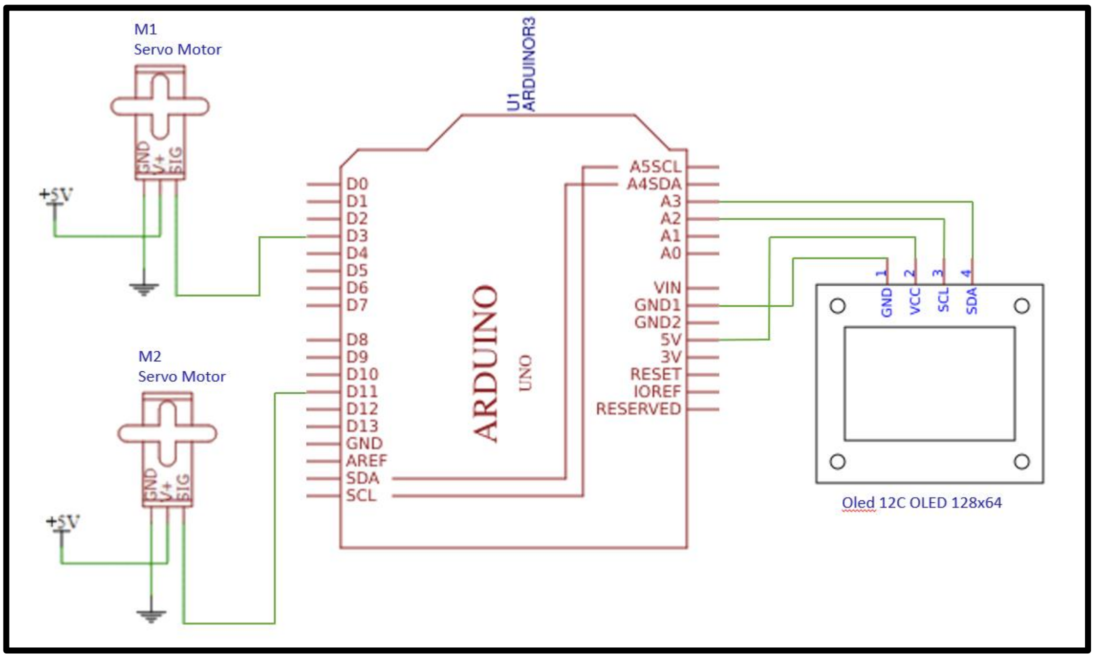


Figure 13, Hardware schematic

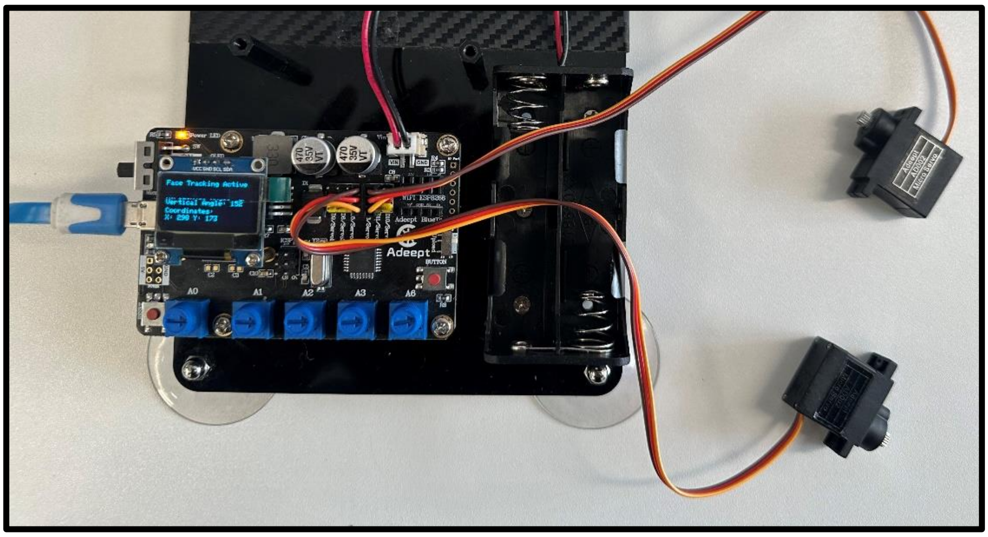


Figure 14, Hardware initialiastion

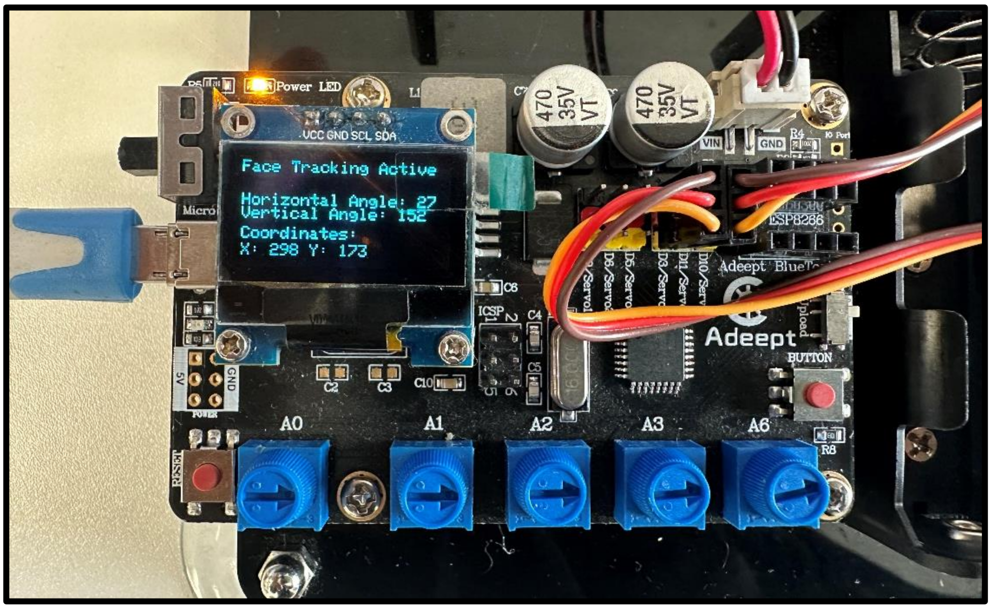


Figure 15, OLED reponse to coordinates

A progress report was periodically prepared to document the development stage, highlighting milestones, issues encountered, and subsequent solutions. This iterative approach ensured that software modules were thoroughly tested and debugged before integration.

The face tracking system is fully functional and the gesture tracking system is next to be developed; While NumPy was initially used for gesture detection, inconsistencies in tracking necessitated a switch to MediaPipe. This library provided a more reliable framework for detecting hand gestures and processing their movements. MediaPipe's efficiency ensured that gestures such as pointing fingers up or down could accurately control the horizontal and vertical servos.

The first step of developing the program was to install the media pipe which is a necessary python library for tracking reference points. This process can only be completed by manually uninstalling the numpy library otherwise the following error occurs:

```
Installing collected packages: numpy, CFFI, sounddevice, opencv-contrib-python, ml-dtypes, contourpy,
ERROR: Could not install packages due to an OSError: [WinError 5] Access is denied: 'C:\\Users\\sodu
e-packages\\cv2\\cv2.pyd'
Consider using the `--user` option or check the permissions.


C:\Users\soduu>$ python -m pip install mediapipe
'$' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\soduu>pip install mediapipe--user
ERROR: Could not find a version that satisfies the requirement mediapipe--user (from versions: none)
ERROR: No matching distribution found for mediapipe--user
```

Figure 16, Media pipe installation

With numpy uninstalled and the system restarted the mediapipe library may now be installed and imported to the python program. The mediapipe library is used from this point on instead of numpy.

Gesture tracking and manual control code breakdown

In parallel, the gesture tracking software was developed using the MediaPipe library (face and gesture tracking code can be found in the appendix). A Tkinter GUI was developed to provide users with an intuitive interface for switching between tracking modes—face tracking, gesture tracking, and manual control. The GUI also displayed system status messages and allowed manual adjustment of the servo motors through on-screen buttons.

Reading material, such as *Embedded Systems, Microcontrollers, and ARM* by Wilmshurst (2017), supported the learning and development of this phase. Online resources such as the official documentation for OpenCV and Arduino provided additional insights and troubleshooting guidance.

The initial Python code, which focused solely on face detection and communication with Arduino for servo motor control, was integrated with gesture recognition capabilities using the cvzone.HandTrackingModule.

The original face tracking algorithm employed a Haar cascade classifier to identify and track faces in a video feed. This was enhanced by adding the ability to track hand gestures, thus creating a dual-mode system. The new code introduced a tracking mode selection mechanism using a Tkinter GUI, allowing users to switch between face tracking, gesture tracking, or manual control.

To incorporate gesture tracking, the cvzone library was used. This library provided a convenient module for detecting hands and identifying which fingers were raised. Specific finger combinations were mapped to predefined actions for the servo motors, such as adjusting horizontal and vertical angles. A mechanism was implemented to verify gestures over consecutive frames to prevent erratic motor movements due to brief or false detections.

Tracking software testing

The software was subjected to rigorous testing to validate its functionality. During the initial testing of the face tracking module, the rectangle intended to outline the user's face did not appear. Debugging revealed logical errors in the Haar cascade implementation, particularly in setting the detection parameters. Adjustments were made to refine the scale factor and minNeighbours parameters, ensuring the face detection algorithm could reliably identify facial features in varying lighting conditions.

The gesture tracking software was tested by simulating different hand gestures within the webcam's field of view. Despite MediaPipe's reliability, minor inconsistencies in gesture recognition were identified and resolved by increasing the detection confidence threshold. This adjustment reduced false positives while maintaining responsiveness.

Hardware development

The hardware assembly utilised components from the Adeept robotics kit, including plastic frames and servo motor plates. Modifications were made to accommodate the phone holder attachment, ensuring stable and adjustable support for the camera.

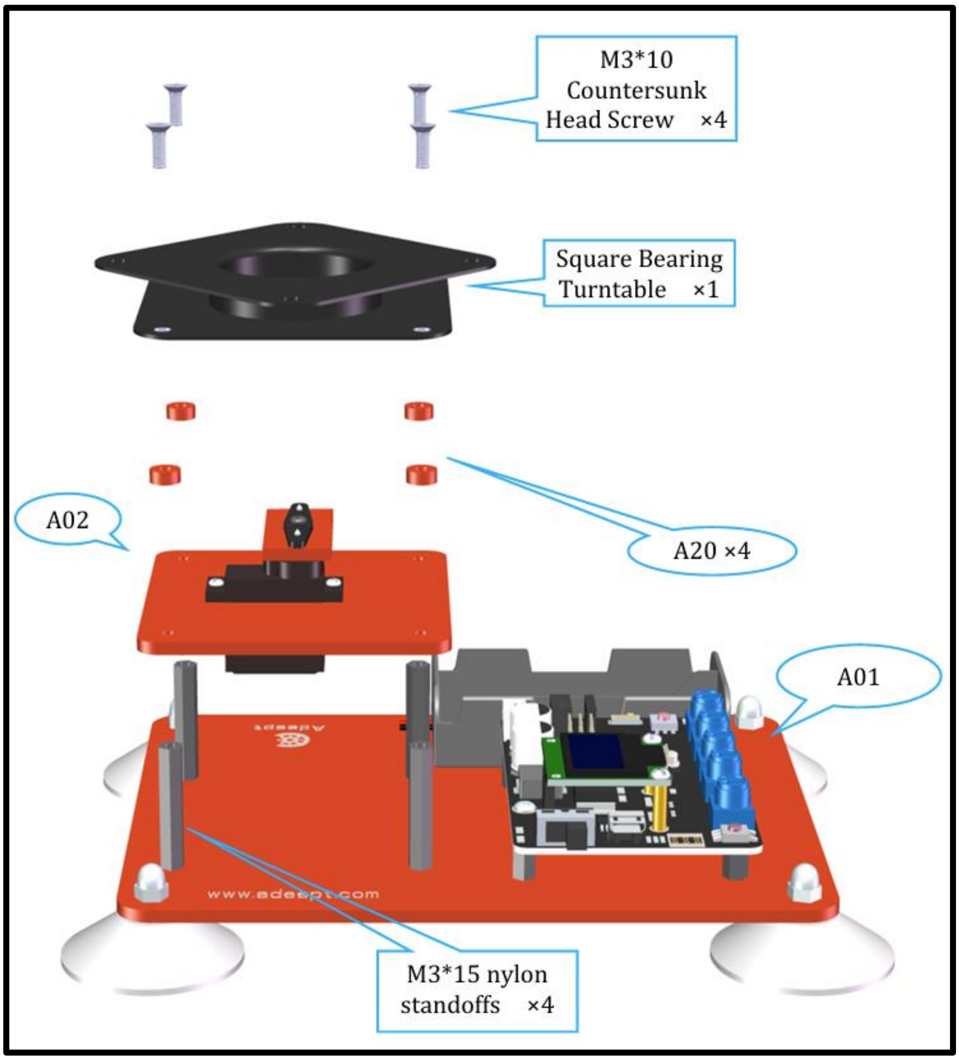


Figure 17, Hardware instructions (Adeept, 2022)

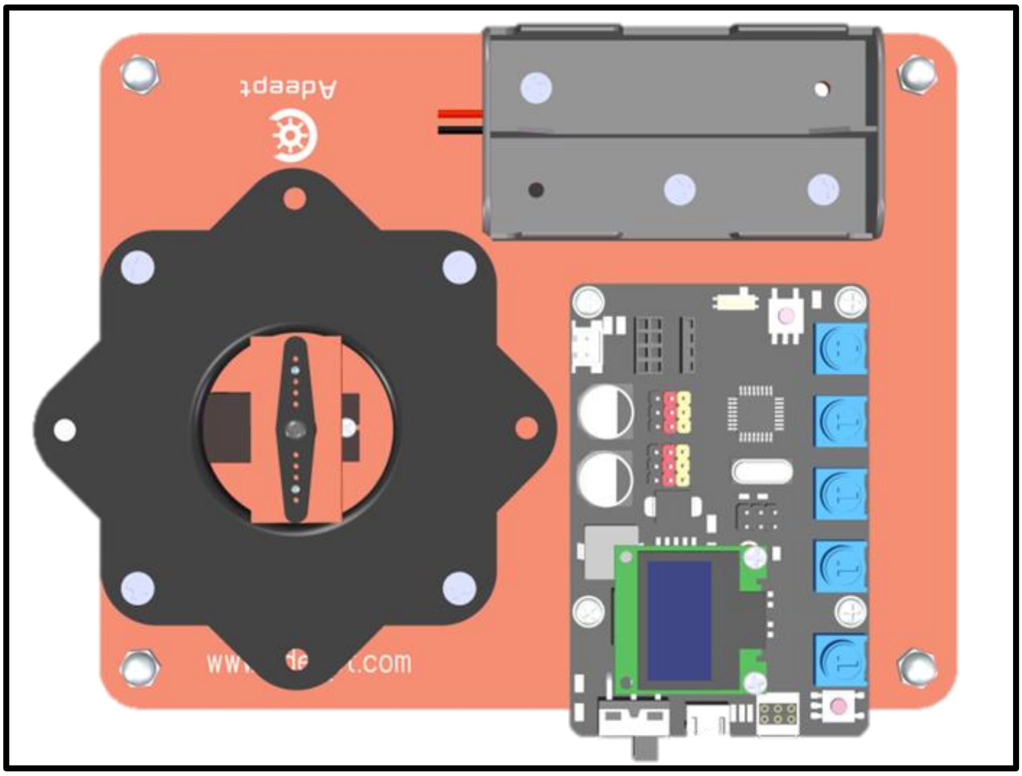


Figure 18, Hardware assembly (Adeept, 2022)

Adhesive provided in the kit was used to secure connections, particularly for components under significant load. The horizontal servo motor was mounted onto the frame, while the two vertical servo motors were positioned to share the load of the camera's vertical movements. Wiring was carefully organised to prevent entanglement during operation.

The Adeept Embed board was integrated with the servos and mounted securely onto the frame. Ensuring reliable communication between the board and the servo motors was critical, requiring the use of precise connections and thorough testing of electrical pathways.

Hardware testing

Hardware testing focused on verifying the mechanical functionality of the servo motors and the structural stability of the frame. Initial tests revealed that the vertical motors occasionally struggled under load due to inadequate distribution. Reinforcements were made by adjusting the servo placements and adding additional adhesive to stabilise high-stress points.

Manual Control and GUI Refinement

The Tkinter GUI was refined to improve usability, adding features such as real-time motor position updates and error notifications. Manual tracking capabilities were enhanced through additional buttons on the GUI, allowing users to adjust the camera's orientation manually. This was particularly useful for demonstrations, where precise control was often necessary.

A significant challenge arose when preparing for the final demonstration. One day prior, the system's structural integrity was compromised due to bending and excessive load on the frame. Emergency repairs were conducted, but these were insufficient to restore full functionality. Documentation of this failure was included in the methodology to highlight the importance of stress testing and the limitations of the selected materials.

Results and Discussion

Tracking Software Performance

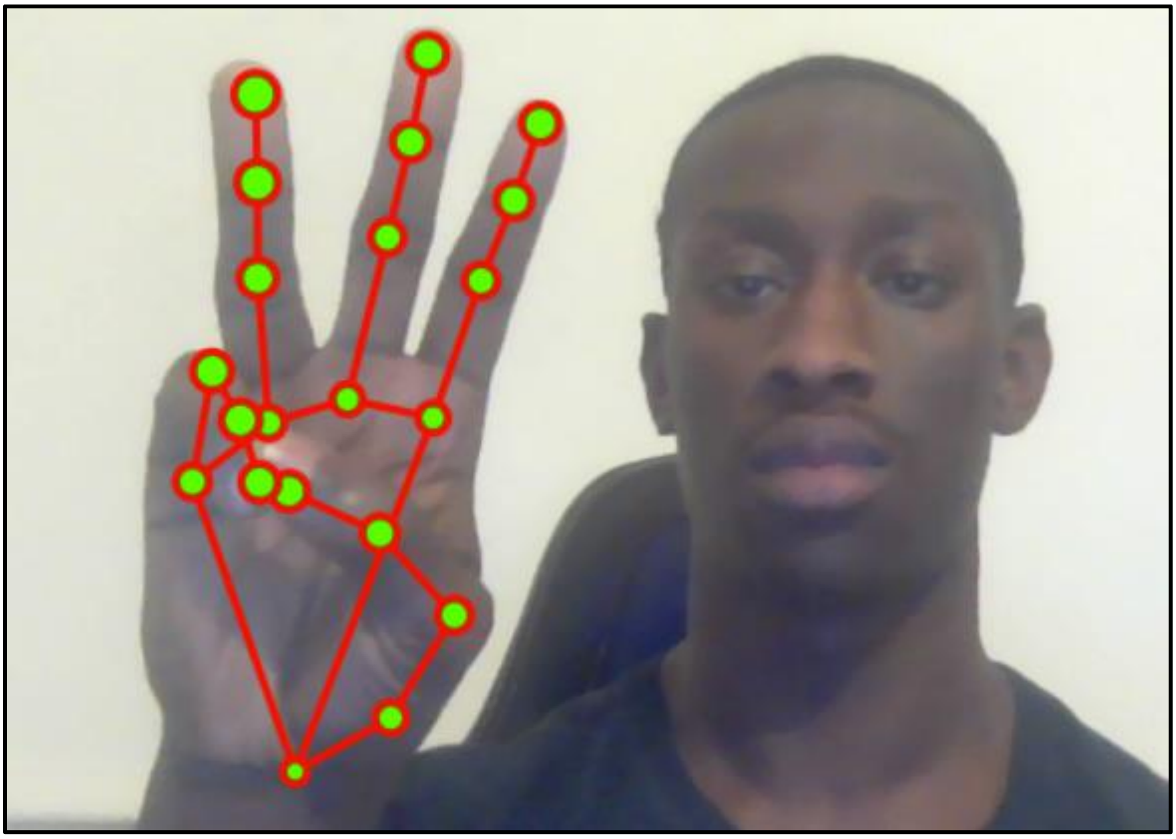


Figure 19, Gesture tracking

The face and gesture tracking software, implemented using Python and the OpenCV library, performed effectively. The software utilised Haar cascades for object detection, a machine learning-based approach that allowed the system to identify and track faces and gestures in real time.

```
2024-11-22 14:26:15,131 - INFO - Serial connection established.
2024-11-22 14:26:26,486 - INFO - Motor Command Sent: Right,90
2024-11-22 14:28:21,052 - INFO - Serial connection established.
2024-11-22 14:28:31,657 - INFO - Motor Command Sent: 90,Down
2024-11-22 14:28:32,977 - INFO - Motor Command Sent: Left,90
2024-11-22 14:28:33,025 - INFO - Motor Command Sent: Left,90
2024-11-22 14:28:33,070 - INFO - Motor Command Sent: Left,90
2024-11-22 14:28:33,120 - INFO - Motor Command Sent: Left,90
2024-11-22 14:28:33,169 - INFO - Motor Command Sent: Left,90
2024-11-22 14:28:33,231 - INFO - Motor Command Sent: Left,90
```

Figure 20, Motor command log

Once a target was detected, the software converted the tracked coordinates into angles, enabling the motors to adjust the camera's orientation accordingly.

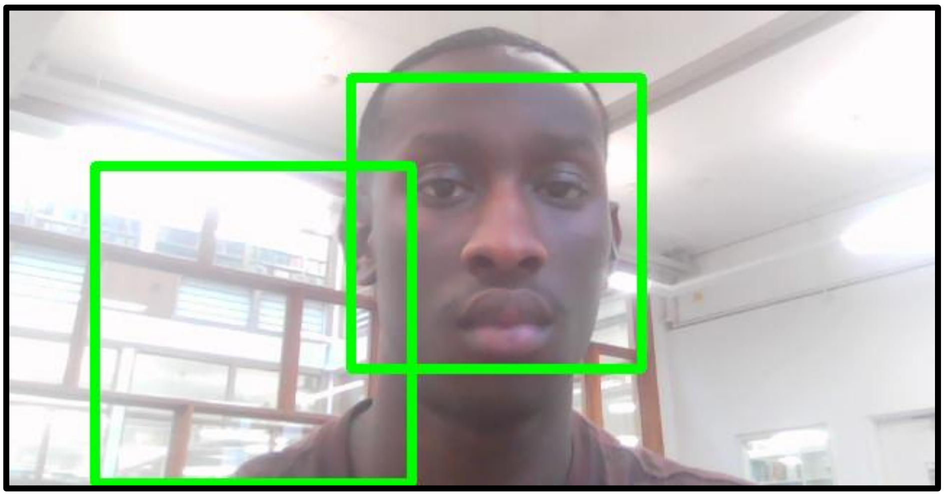


Figure 21, Inconsistent Tracking

improvements are necessary to address certain inconsistencies, such as variable lighting that can cause incorrect detections as seen in figure 17. While switching to using the mediapipe library instead of using the numpy library has improved the tracing of faces there are still inconsistencies in the tracking to be resolved.

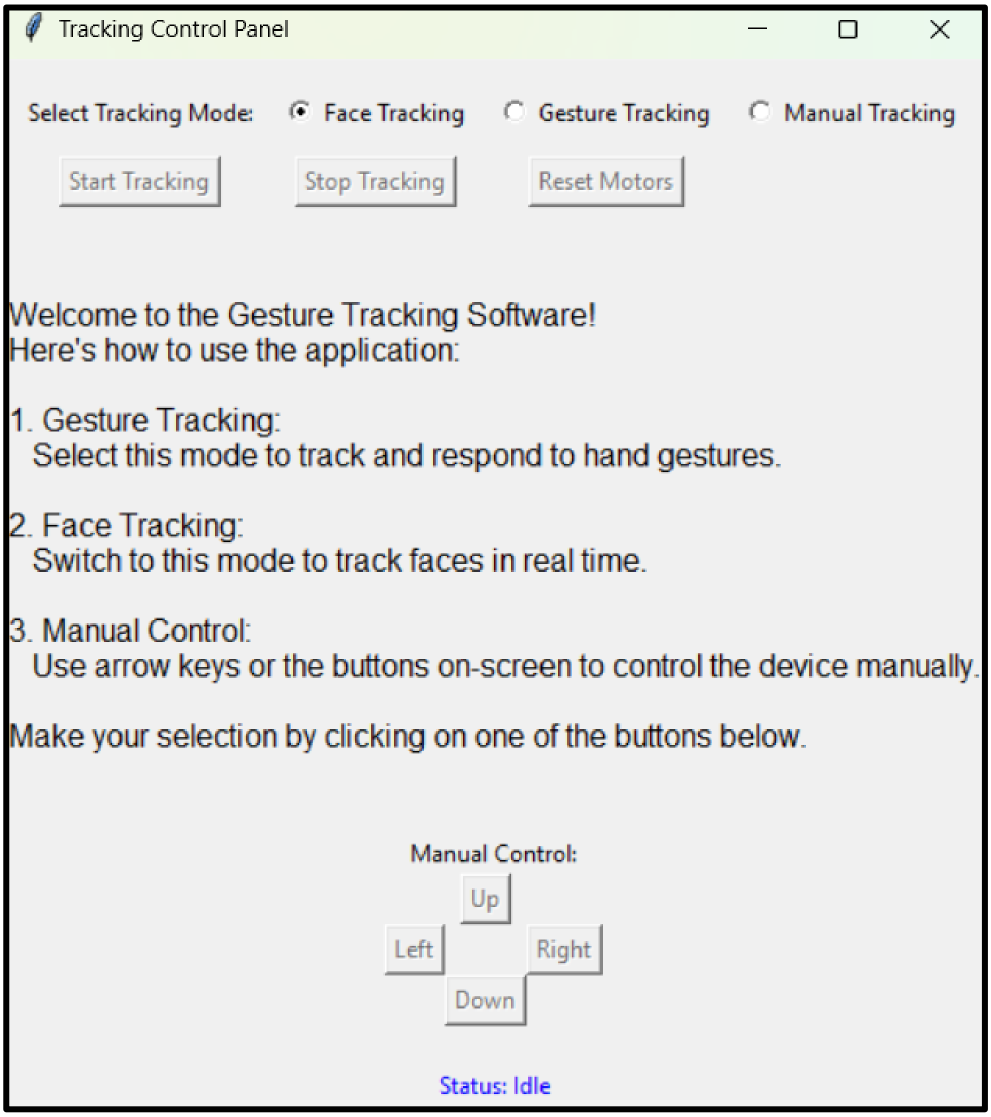


Figure 22, Final Project GUI

The robustness of the tracking software was apparent during testing, where it consistently detected and followed objects with minimal lag. The use of object-oriented programming facilitated modular and efficient code development, ensuring scalability and ease of debugging. Additionally, the inclusion of multiple tracking modes, accessible through a Tkinter-based graphical user interface (GUI), enhanced the system’s usability. Users could seamlessly switch between face tracking, hand gesture tracking, and manual control, showcasing the flexibility of the software design.

However, some limitations were observed. The accuracy of the Haar cascades diminished under certain conditions, such as poor lighting or occlusions, which led to occasional tracking errors. These issues are consistent with known limitations of Haar cascades (Viola & Jones, 2001). Future iterations could address these challenges by incorporating more advanced tracking algorithms, such as convolutional neural networks (CNNs), which have demonstrated superior performance in complex environments (Simonyan & Zisserman, 2015).

Motor Control Responsiveness

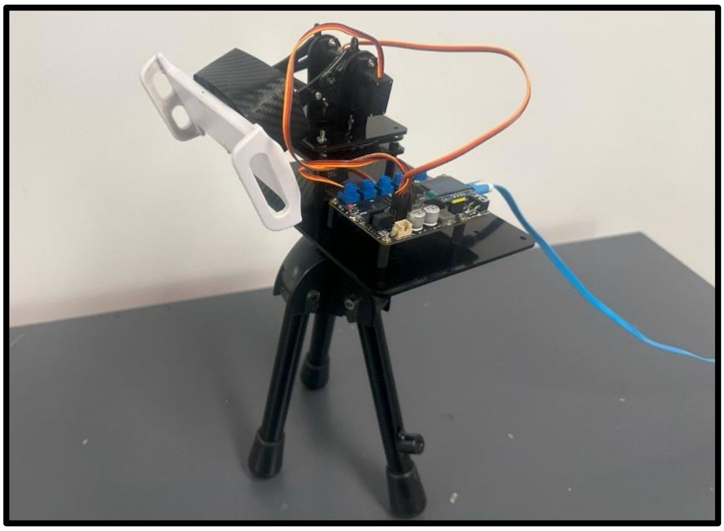


Figure 23, Final Project

The motor control system, responsible for adjusting the camera's orientation, exhibited a degree of responsiveness that aligned with the project's objectives. Servo motors were employed to facilitate vertical and horizontal camera manoeuvring, with the angles calculated by the tracking software dictating the motor movements. During testing, the system successfully adjusted the camera's position in response to real-time tracking data, confirming the effectiveness of the integration between software and hardware.

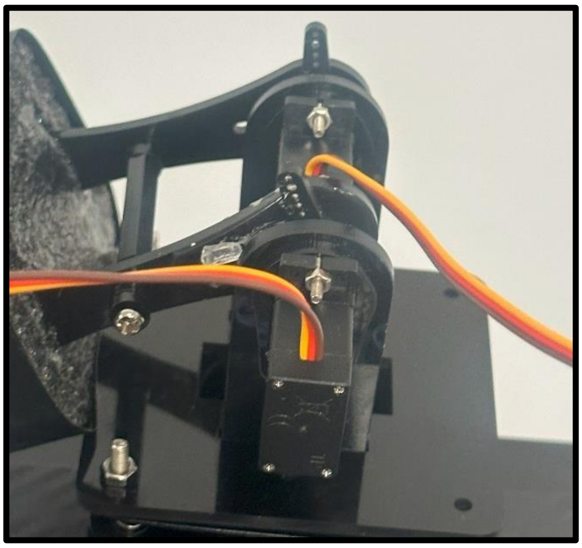


Figure 24, Broken vertical motor connection

Despite these successes, significant limitations were identified in the hardware's performance. The vertical rotation mechanism experienced instability under load, leading to inconsistent movement. This issue appeared to stem from a combination of factors, including the arm length of the vertical mount, the power supply limitations, and insufficient torque provided by the servo motors. Although the addition of a secondary servo motor improved performance slightly, it was insufficient to fully address the instability.

This highlights the importance of selecting hardware components that match the system's mechanical demands. For future improvements, upgrading to higher-torque servo motors or implementing a counterweight system could enhance stability. Therefore, adopting a more robust power supply with a stable current output could mitigate fluctuations that affect motor performance. The findings align with literature emphasising the significance of torque and power supply in robotic arm design (Kim et al., 2018).

Integration and Usability

A key achievement of this project was the seamless integration of tracking software with hardware components, fulfilling one of the primary objectives. The software's ability to communicate real-time tracking data to the servo motors was facilitated through Python libraries, including PySerial and GPIO. This integration enabled smooth camera adjustments and demonstrated the feasibility of combining embedded systems with machine learning applications.

The user interface also contributed to the system's overall usability. Designed using Tkinter, the GUI provided an intuitive platform for users to select tracking modes and manually control the camera. The GUI design adhered to principles of simplicity and accessibility, ensuring that even users with minimal technical expertise could operate the system effectively.

While the integration was largely successful, challenges arose during calibration. Ensuring that the angles calculated by the tracking software translated accurately into motor movements required extensive trial and error. This process highlighted the complexity of synchronising software outputs with hardware inputs, particularly when dealing with multiple axes of movement. Incorporating real-time feedback mechanisms, such as encoders, could improve calibration and enhance precision in future designs.

Testing and Optimisation

Testing played a crucial role in evaluating the system's performance and identifying areas for optimisation. The prototype was subjected to various scenarios, including dynamic movements and varying lighting conditions, to assess its tracking accuracy and responsiveness. The results indicated that the system performed best in well-lit environments with clear line-of-sight to the target. Tracking accuracy decreased when the target moved rapidly or when environmental conditions were suboptimal.

Efforts to optimise the system included fine-tuning the Haar cascade parameters and adjusting the motor speed to balance responsiveness with stability. These adjustments improved overall performance but did not entirely eliminate the observed limitations. For instance, the reliance on Haar cascades made the system less adaptable to diverse conditions, suggesting the need for more advanced algorithms in future iterations.

Limitations and Future Work

Software limitations included the reliance on Haar cascades, which, while effective in controlled conditions, struggled in complex environments. Transitioning to more robust tracking methods, such as deep learning-based object detection, could significantly enhance performance. Additionally, implementing predictive algorithms could improve the system's ability to track fast-moving objects, addressing one of the key challenges identified during testing.

Future work could also explore the integration of additional sensors, such as accelerometers or gyroscopes, to provide real-time feedback on the system's orientation. This would enable more precise adjustments and reduce the reliance on visual data alone. Furthermore, expanding the system’s capabilities to include multiple object tracking or recognition could broaden its applications, making it suitable for more complex tasks, such as crowd monitoring or interactive robotics.

Conclusion

Summary of Findings

This project successfully achieved its aim of designing and building a prototype for an automatic face-tracking tripod for a phone camera, capable of autonomously orienting the camera in response to a person's movements. The software was developed using Python, object-oriented programming principles, and the OpenCV machine learning library, effectively integrating face and gesture tracking functionalities with motor control systems.

The prototype's hardware assembly included a motorised tripod capable of vertical and horizontal camera manoeuvring. Servo motors controlled by Python scripts facilitated camera orientation, enabling smooth and responsive tracking. Additionally, the project featured a Tkinter-based graphical user interface (GUI) that allowed users to select tracking modes, including face tracking, hand gesture tracking, and manual control using arrow keys and on-screen buttons.

Testing revealed that the system operated effectively in tracking a person's movements, demonstrating the practical application of the integrated hardware and software. However, limitations were observed in the hardware, particularly regarding joint stability under load. These constraints highlight areas for future improvement, such as enhancing the mechanical design to support more robust operation.

The findings underscore the project's alignment with its objectives:

1. Developing tracking software using Python and OpenCV.
2. Designing and assembling a functional motor-controlled tripod.
3. Achieving seamless software-hardware integration.
4. Evaluating the system's performance through iterative testing and optimisation.

By meeting these objectives, the prototype demonstrated the feasibility of integrating embedded systems with machine learning techniques for practical applications. The project contributes to the field of automated camera systems, providing a foundation for further advancements in stability and multi-modal tracking functionalities.

Limitations

Several limitations were encountered during the development of the automatic face-tracking tripod prototype, which impacted the system's performance and design efficiency. One significant constraint was the lack of experience in hardware testing, which led to overlooked load-bearing issues in the design. These challenges became evident during operation, where the MG996R servo motors proved insufficient for the system's weight requirements. The limited torque capacity of 11 kg/cm hindered the ability to achieve smooth and consistent movements, emphasising the need for more robust motors in future iterations.

Additionally, the prototype's tracking system, while functional, faced challenges under certain conditions. For example, clothing accessories such as hats or environmental factors like poor lighting disrupted the accuracy of the face-tracking feature. These limitations are inherent to machine learning-based computer vision systems, such as OpenCV, and highlight the importance of refining algorithm robustness to account for variable real-world conditions.

Recommendations

Figure 25, the ANNIMOS Coreless Digital Motor (ANNIMOS, 2020)

To enhance the performance and durability of the automatic face-tracking tripod prototype, several recommendations are proposed. First, upgrading the motor from the current MG 996R, with a torque of 11 kg/cm, to a higher-capacity motor such as the ANNIMOS Coreless Digital Motor, which offers a torque of 35 kg/cm, is advised. This upgrade would improve the system's ability to handle heavier loads and operate more efficiently. However, this improvement would increase the total project cost by approximately £32, necessitating careful cost-benefit analysis (American Psychological Association, 2020).

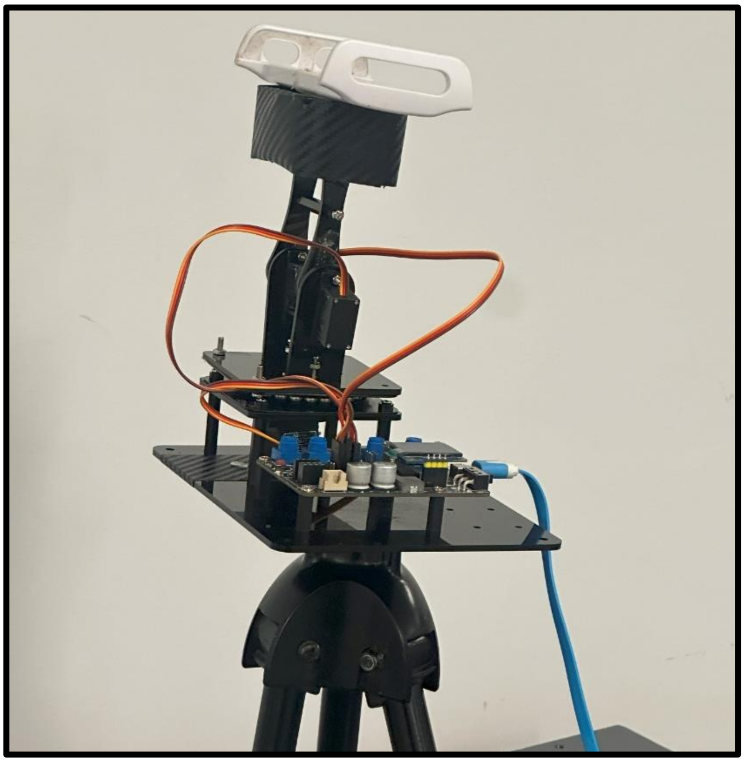


Figure 26, Improved default position system

Second, modifying the system's default resting position to face upwards is suggested. This adjustment would alleviate unnecessary strain on the joints during inactivity, prolonging the lifespan of the components and reducing wear over time.

Third, future researchers should investigate alternative materials and joint mechanisms to enhance the structural robustness of the tripod. This approach would mitigate issues related to bending and instability, ensuring better performance under dynamic conditions.

Lastly, testing alternative computer vision software to improve tracking capabilities is recommended. The current prototype utilises OpenCV, which has demonstrated effective functionality in this application. However, exploring other machine learning-based libraries or optimisation techniques may yield improved tracking accuracy and processing efficiency.

Research Summary

The primary aim of this project was to design and develop a prototype of an automatic face-tracking tripod for mobile phone cameras, capable of responding to a person's movements. The objectives included: using object-oriented programming with Python and the OpenCV library to implement a tracking algorithm; assembling a motor-controlled phone tripod for vertical and horizontal camera manoeuvres; integrating the tracking software with the camera hardware for seamless operation; and evaluating the prototype's performance to optimise the system.

The research adopted an iterative design approach, combining hardware and software development to achieve a functional prototype. Object-oriented programming facilitated modular and scalable code development, while OpenCV enabled robust face-detection and tracking features. The integration of servo motors with a Tkinter-based GUI provided a user-friendly interface for controlling the system and switching between tracking modes, including face tracking, gesture recognition, and manual control.

Through systematic testing and evaluation, the project demonstrated the potential of combining embedded systems, machine learning, and mechanical design to create an effective face-tracking device. These recommendations build on the findings and challenges identified, paving the way for future developments in this domain.

Appendix

Declaration of Authenticity

Name	Oluwaseun Odusanya
Identification	1016 3553
Course	Electro-Mechanical Engineering BEng
Project title	Face tracking phone camera stand

Checklist: I confirm that this assessment is my own work, and that I have…

Read and understood the regulations on academic conduct, contained within Student Contract and Student Handbook. The 2020/2021 contract can be found here: https://www.coventry.ac.uk/globalassets/media/documents/registry/20-21-combined-coventry-university-ug-and-pg-student-contract.pdf	Y/N
VClearly acknowledged, and appropriately applied, all work taken from sources in accordance with Coventry University Group's Harvard Referencing system. The full guide can be found here: https://www.coventry.ac.uk/study-at-coventry/student-support/academic-support/centre-for-academic-writing/support-for-students/academic-writing-resources/cu-harvard-reference-style-guide/	Y/N
I understand that any false claim, in respect of this work, will result in disciplinary action in accordance with University regulations	Y/N

Declaration: I am aware of, and understand, the University's assessment policy. I certify that this assessment is my own work (with exception formally cited works). I have followed good academic practice and have read and conformed to the 'Statement of Ethical Principles' as detailed by the Engineering Council and the Royal Academy of Engineering.

Disclaimer: Undergraduates at enrollment and registration are required to agree that Coventry University has ownership of Intellectual Property they create during the period of, and relating to, their studies and/or research in return for exploitation rewards, as if they were a member of staff.

Signed: Oluwaseun Odusanya

OneDrive link to all Project files:

First_Face_Tracking_Python_Code.py

Log Book

Table , Logbook

Date	Activity	Details	Outcome
09/05/24	Contacted a superior regarding project	Project requirements and structure have been given.	Supervisor has been assigned.
29/05/24	Meeting with supervisor	Discuss current project proposals of multiple project ideas to determine an aim and purpose.	Feasibility has been assessed by supervisor and readjustments are to be made based on feedback.
27/06/24	Propose project to supervisor	Explain to supervisor how the feedback has been implemented. Outline the project purpose	Supervise has approved of the project idea and given further recommendations.
07/07/24	Email supervisor with update	Documentation of the project aims and objectives was emailed to supervisor	Supervisor has acknowledged project focus.
11/07/24	Meeting with supervisor	Research of project has been communicated with supervisor.	Alterations/improvements have been suggested as feedback
29/07/24	Meeting with supervisor	Present improvements with supervisor.	Supervisor has provided additional feedback and insight into potential solutions.
24/08/24	Ethics application is submitted	An ethics application has been documented and submitted	The application is awaiting review.
13/09/24	Supervisor has reviewed ethics application	Feedback has been given regarding the documentation structure being inadequate.	The application has been returned for improvements to be made
16/09/24	Meeting with supervisor	The supervisor has given suggestions as to how the ethics application can be improved.	The application has had the feedback implemented.

Gantt Chart

The following figures will show the project Gantt chart however, chart details (specific dates, durations, completion rate, employee delegation etc.) are presented more accurately in the appropriate software. To directly access the Gantt chart file, follow these steps:

1 - Download the file from the OneDrive link below

First_Face_Tracking_Python_Code.py

2 - Open the Free Online Gantt platform link below

Free Online Gantt platform: https://www.onlinegantt.com/#/gantt

3 – Open the downloaded file

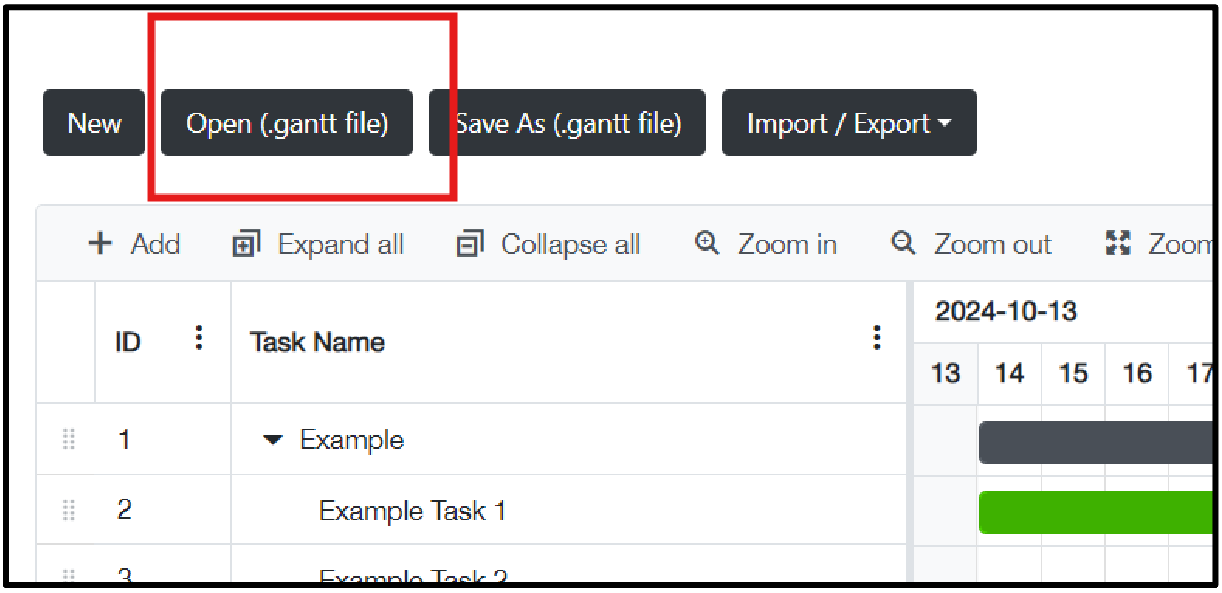


Figure 27, Gantt chart access guide

Shaded bars represent completion (open file to view percentages)

Milestone	
Sub-task	
Tolerance	

Figure 28

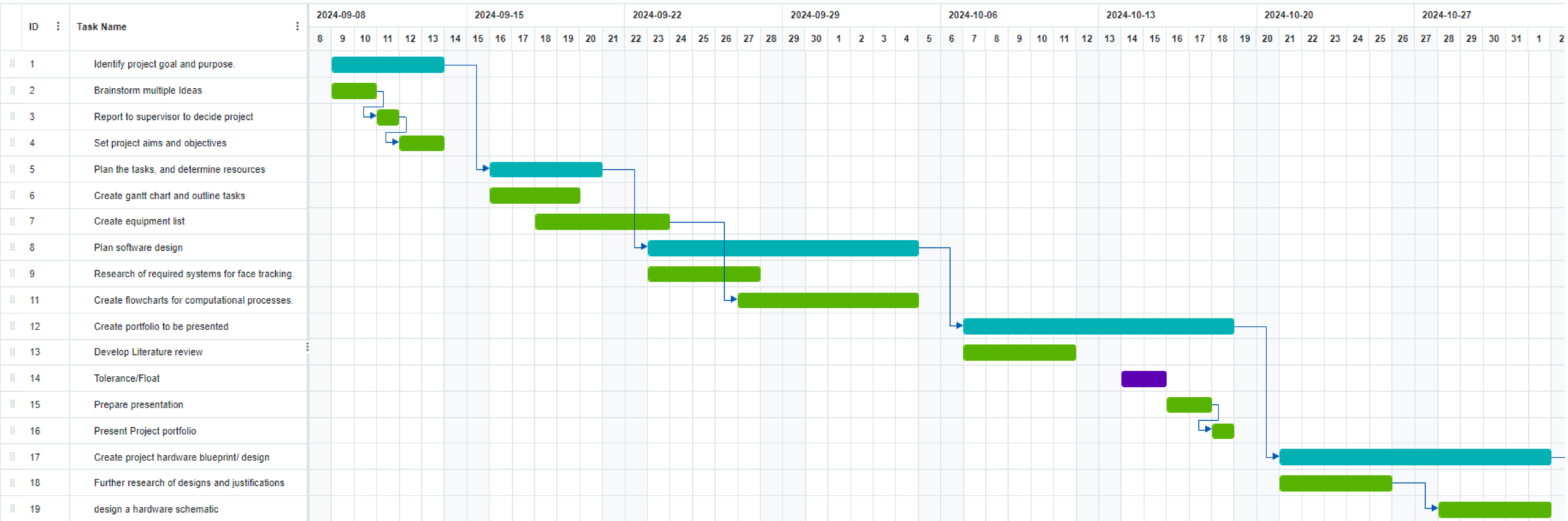


Figure 29, Gantt pt.1

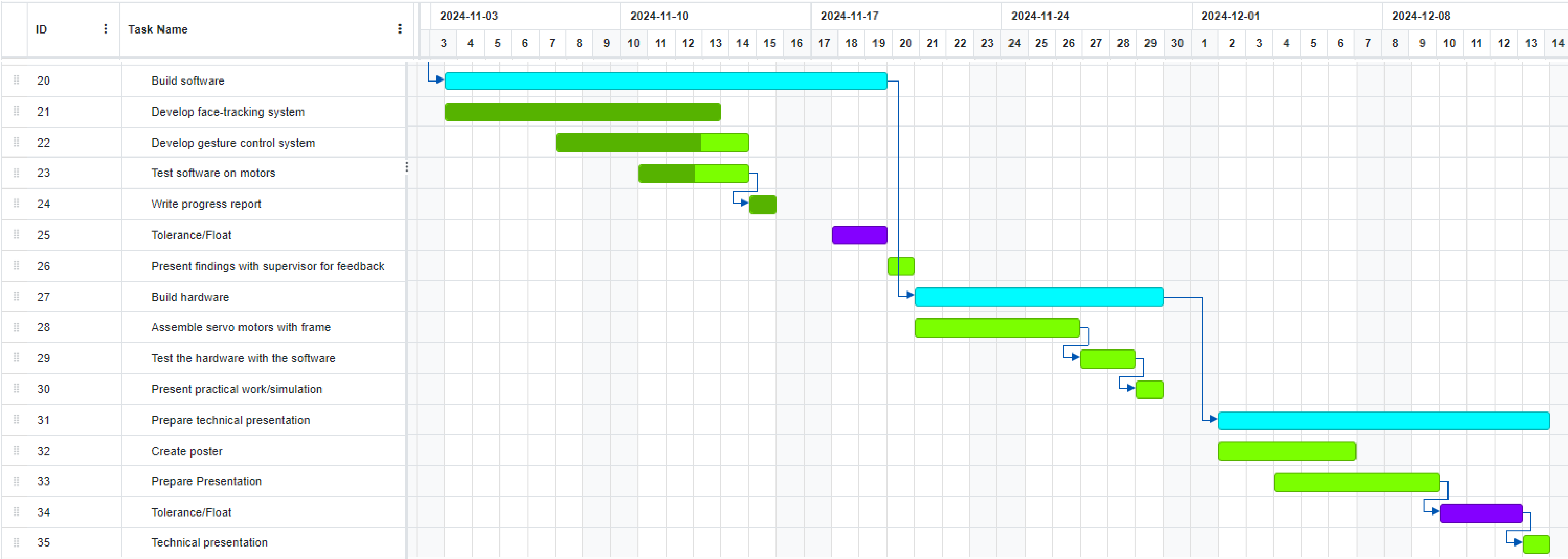


Figure 30, Gantt pt

Coventry
London
Scarborough
University Group

ID	Task Name	Start	End	Duration	Progress...	Dependency	Resources	Color
1	Identify project goal and purpose.	2024-09-09	2024-09-13	5 days	100		Project manager,Data analyst	
2	Brainstorm multiple Ideas	2024-09-09	2024-09-10	2 days	100		Project manager,Data analyst,Systems architect	
3	Report to supervisor to decide project	2024-09-11	2024-09-11	1 day	100	2FS	Project manager	
4	Set project aims and objectives	2024-09-12	2024-09-13	2 days	100	3FS	Project manager,Systems architect	
5	Plan the tasks, and determine resources	2024-09-16	2024-09-20	5 days	100	1FS	Project manager,Systems architect	
6	Create gantt chart and outline tasks	2024-09-16	2024-09-19	4 days	100		Project manager	
7	Create equipment list	2024-09-18	2024-09-23	4 days	100		Systems architect,Project manager	
8	Plan software design	2024-09-23	2024-10-04	10 days	100	5FS	Systems architect,Data analyst,Project manager	
9	Research of required systems for face tracking.	2024-09-23	2024-09-27	5 days	100		Data analyst,Systems architect,Project manager	
11	Create flowcharts for computational processes.	2024-09-27	2024-10-04	6 days	100	7FS+3 days	Systems architect	
12	Create portfolio to be presented	2024-10-07	2024-10-18	10 days	100	8FS	Project manager	
13	Develop Literature review	2024-10-07	2024-10-11	5 days	100		Project manager	
14	Tolerance/Float	2024-10-14	2024-10-15	2 days	100		Project manager	
15	Prepare presentation	2024-10-16	2024-10-17	2 days	100		Project manager	
16	Present Project portfolio	2024-10-18	2024-10-18	1 day	100	15FS	Project manager	
17	Create project hardware blueprint/ design	2024-10-21	2024-11-01	10 days	100	12FS	Systems architect,Electronic engineering team	
18	Further research of designs and justifications	2024-10-21	2024-10-25	5 days	100		Data analyst,Systems architect,Electronic engineering team	
19	design a hardware schematic	2024-10-28	2024-11-01	5 days	100	18FS	Systems architect	
20	Build software	2024-11-04	2024-11-19	12 days	0	17FS	Software development team	
21	Develop face-tracking system	2024-11-04	2024-11-13	8 days	100		Software development team	
22	Develop gesture control system	2024-11-08	2024-11-14	5 days	75		Software development team	
23	Test software on motors	2024-11-11	2024-11-14	4 days	50		Software development team,Quality assurance team	
24	Write progress report	2024-11-15	2024-11-15	1 day	100	23FS	Project manager	

Figure 31

Figure 32

25	Tolerance/Float	2024-11-18	2024-11-19	2 days	0		Software development team	
26	Present findings with supervisor for feedback	2024-11-20	2024-11-20	1 day	0		Software development team	
27	Build hardware	2024-11-21	2024-11-29	7 days	0	20FS+1 day	Electronic engineering team	
28	Assemble servo motors with frame	2024-11-21	2024-11-26	4 days	0		Electronic engineering team	
29	Test the hardware with the software	2024-11-27	2024-11-28	2 days	0	28FS	Electronic engineering team,Quality assurance team	
30	Present practical work/simulation	2024-11-29	2024-11-29	1 day	0	29FS	Electronic engineering team,Software development team,Project manager,Systems architect	
31	Prepare technical presentation	2024-12-02	2024-12-13	10 days	0	27FS	Project manager,Systems architect,Software development team,Quality assurance team	
32	Create poster	2024-12-02	2024-12-06	5 days	0		Project manager,Systems architect	
33	Prepare Presentation	2024-12-04	2024-12-09	4 days	0		Project manager	
34	Tolerance/Float	2024-12-10	2024-12-12	3 days	0	33FS		
35	Technical presentation	2024-12-13	2024-12-13	1 day	0	34FS		

Figure 33

Face Tracking Python pseudocode:

```
Python Code Pseudocode:

1. Import required libraries: serial, time, numpy, cv2, tkinter, and threading.

2. Set up serial communication with Arduino on COM3 at 9600 baud rate.

3. Define global variable `is_tracking` to manage the status of the video capture loop.

4. transmit coordinates to arduino:
    - Format `x_pos` and `y_pos` as a string, then send to Arduino via serial.
    - Print coordinates to console for debugging.
    - Update the Tkinter label with the coordinates.

5. Define function `begin_video_feed()`:
    - Load Haar cascade classifier for face detection.
    - Read a frame from the video feed.
    - Display the video frame with face rectangles.
    - If 'd' key is pressed, exit the loop.
    - Release video capture resources and close any OpenCV windows.
    - Update GUI status and buttons to indicate tracking is inactive.

6. Define function `end_video_feed()`:
    - Set `is_tracking` to False to stop video capture loop.

7. Define function `on_start_press()`:
    - Start a new thread for `begin_video_feed()` to keep GUI responsive.

8. Set up the Tkinter GUI:
    - Display tracking status.
    - Display coordinates.

9. Run Tkinter's main loop to keep the GUI active.
```

Figure 34

Face Tracking Arduino Pseudocode:

```
Arduino Code Pseudocode:

1. Import necessary libraries: Servo, Wire, Adafruit_GFX, and Adafruit_SSD1306 for OLED display.

2. Define OLED display with reset pin.

3. Create servo objects `servo_horizontal` and `servo_vertical` for horizontal and vertical movement.

4. Define global variables:
    - `received_data` to store incoming data.
    - `horizontal_angle` and `vertical_angle` initialized to 90 degrees for the initial position.

5. Setup function:
    - Attach `servo_horizontal` to pin 11 and `servo_vertical` to pin 3.
    - Begin serial communication at 9600 baud.
    - Initialize OLED display and show "Face Tracking Active" for 2 seconds.

6. Main loop:
    - Check if data is available from the serial port.
    - If data is available:
        - Read data until newline character and split it into `face_x` and `face_y`.
        - Map horizontal angle between 0 and 180 degrees.
        - Map vertical angle between 0 and 180 degrees.
        - Move horizontal servo and vertical servo to mapped angles.
        - Clear and update the OLED display with the current angles and coordinates.
```

Figure 35

Face tracking Python code pt.1:

```python
import serial  # Library for serial communication
import time  # Library for time-related functions
import numpy as np  # Library for numerical operations
import cv2  # Library for computer vision tasks
import tkinter as tk  # Library for GUI creation
from threading import Thread  # Library to handle threads

# Set up serial communication with Arduino on COM3 at 9600 baud rate
arduino_connection = serial.Serial('COM3', 9600)  # Modify COM port if needed, e.g., COM3 or COM6

# Global variable to manage the status of the video capture loop
is_tracking = False

# Function to send face position coordinates to Arduino and update Tkinter label
def transmit_coordinates_to_arduino(x_pos, y_pos, width, height):
    # Format coordinates as a comma-separated string with a newline at the end
    coordinates = f"{x_pos},{y_pos}\r"
    # Send the encoded string to Arduino
    arduino_connection.write(coordinates.encode())
    # Log coordinates to console for debugging purposes
    print(f"X{x_pos} Y{y_pos}\n")
    # Update Tkinter label with current coordinates
    coordinates_label.config(text=f"Coordinates: X{x_pos}, Y{y_pos}")

# Function to initiate video feed and detect faces
def begin_video_feed():
    global is_tracking
    is_tracking = True
    # Update tracking status and button states in the GUI
    status_display.config(text="Tracking Status: Active", fg="green")
    start_button.config(state="disabled", bg="lightgrey")
    stop_button.config(state="normal", bg="red")

    # Initialize video capture from the default camera (usually 0 for an internal camera)
    video_capture = cv2.VideoCapture(0)

    # Load pre-trained Haar cascade classifier for face detection
    face_detector = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

    # Verify if the Haar cascade file loaded successfully
    if face_detector.empty():
        print("Error: Haar cascade XML file not found.")
        video_capture.release()
        cv2.destroyAllWindows()
        return

    print("Haar cascade XML file loaded successfully.")

    # Loop for capturing frames from the camera
    while is_tracking:
        # Read a frame from the video feed
        frame_available, video_frame = video_capture.read()

        if not frame_available:
            break

        # Convert frame to grayscale for the face detection algorithm
        grayscale_frame = cv2.cvtColor(video_frame, cv2.COLOR_BGR2GRAY)

        # Apply the face detection algorithm
        detected_faces = face_detector.detectMultiScale(grayscale_frame, 1.05, 8, minSize=(120,120))

```

Figure 36

Face tracking Python code pt.2:

```python

        # Process each detected face
        for (x_pos, y_pos, width, height) in detected_faces:
            # Draw a green rectangle around each detected face
            cv2.rectangle(video_frame, (x_pos, y_pos), (x_pos + width, y_pos + height), (0, 255, 0), 5)
            # Send detected face coordinates to Arduino and update the GUI label
            transmit_coordinates_to_arduino(x_pos, y_pos, width, height)

        # Display the video feed with rectangles around detected faces
        cv2.imshow('Video', video_frame)

        # Check for 'd' key press to manually stop video feed
        if cv2.waitKey(20) & 0xFF == ord('d'):
            break

    # Release resources when tracking stops
    video_capture.release()
    cv2.destroyAllWindows()

    # Update tracking status and button states in the GUI
    status_display.config(text="Tracking Status: Inactive", fg="red")
    start_button.config(state="normal", bg="green")
    stop_button.config(state="disabled", bg="lightgrey")

# Function to stop video capture
def end_video_feed():
    global is_tracking
    is_tracking = False

# Function to handle the start button event in the GUI
def on_start_press():
    # Run the video capture in a separate thread to keep the GUI responsive
    video_thread = Thread(target=begin_video_feed)
    video_thread.start()

# Tkinter GUI setup
app = tk.Tk()
app.title("Arduino Face Tracking")

# Label to display tracking status
status_display = tk.Label(app, text="Tracking Status: Inactive", fg="red", font=("Helvetica", 12))
status_display.pack(pady=10)

# Start and Stop buttons with initial colors
start_button = tk.Button(app, text="Start", command=on_start_press, bg="green", fg="white", width=10)
start_button.pack(pady=5)

stop_button = tk.Button(app, text="Stop", command=end_video_feed, bg="lightgrey", fg="white", width=10, state="disabled")
stop_button.pack(pady=5)

# Label to display face coordinates
coordinates_label = tk.Label(app, text="Coordinates: Not Detected", font=("Helvetica", 10))
coordinates_label.pack(pady=10)

# Run Tkinter's main event loop
app.mainloop()

```

Figure 37

Face tracking Arduino code pt.1:

```
#include <Servo.h> // Library for controlling servos
#include <Wire.h>  // Library for I2C communication
#include <Adafruit_GFX.h> // Library for graphics on OLED
#include <Adafruit_SSD1306.h> // OLED display library

#define OLED_RESET 4
Adafruit_SSD1306 oled_display(128, 64, &Wire, OLED_RESET); // Initialize OLED with resolution

Servo servo_horizontal; // Servo for horizontal movement
Servo servo_vertical; // Servo for vertical movement

String received_data; // String to store incoming data
int horizontal_angle = 90; // Initial horizontal angle
int vertical_angle = 90; // Initial vertical angle

void setup() {
  // Attach servos to specified pins
  servo_horizontal.attach(11); // Horizontal servo on pin 11
  servo_vertical.attach(3);    // Vertical servo on pin 3
  Serial.begin(9600); // Begin serial communication

  // Initialize OLED display
  oled_display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  oled_display.setTextColor(WHITE);
  oled_display.clearDisplay();
  oled_display.setTextSize(1);
  oled_display.setCursor(0, 0);
  oled_display.println("Face Tracking Active");
  oled_display.display();
  delay(2000); // Display message for 2 seconds
}

void loop() {
  // Check if data is available on serial port
  while (Serial.available()) {
    received_data = Serial.readStringUntil('\r'); // Read data until newline
    int face_x = received_data.substring(0, received_data.indexOf(',')).toInt(); // Extract X
    int face_y = received_data.substring(received_data.indexOf(',') + 1).toInt(); // Extract Y

    // Map coordinates to servo angles
    horizontal_angle = map(face_x, 0, 1920, 0, 180);
    vertical_angle = map(face_y, 0, 1080, 180, 0);

    // Move servos to corresponding angles
    servo_horizontal.write(horizontal_angle);
    servo_vertical.write(vertical_angle);

    // Display coordinates and angles on OLED
    oled_display.clearDisplay();
    oled_display.setCursor(0, 0);
    oled_display.setTextSize(1);
    oled_display.println("Face Tracking Active");
    oled_display.setCursor(0, 20);
    oled_display.print("Horizontal Angle: ");
    oled_display.println(horizontal_angle);
    oled_display.print("Vertical Angle: ");
    oled_display.println(vertical_angle);
    oled_display.setCursor(0, 40);
    oled_display.print("Coordinates:");
    oled_display.setCursor(0, 50);
    oled_display.print("X: ");
    oled_display.print(face_x);
    oled_display.print(" Y: ");
```

Figure 38

Face tracking Arduino code pt.2

```
    // Display coordinates and angles on OLED
    oled_display.clearDisplay();
    oled_display.setCursor(0, 0);
    oled_display.setTextSize(1);
    oled_display.println("Face Tracking Active");
    oled_display.setCursor(0, 20);
    oled_display.print("Horizontal Angle: ");
    oled_display.println(horizontal_angle);
    oled_display.print("Vertical Angle: ");
    oled_display.println(vertical_angle);
    oled_display.setCursor(0, 40);
    oled_display.print("Coordinates:");
    oled_display.setCursor(0, 50);
    oled_display.print("X: ");
    oled_display.print(face_x);
    oled_display.print(" Y: ");
    oled_display.println(face_y);
    oled_display.display();

    // Print for debugging
    Serial.print("Horizontal Angle: ");
    Serial.println(horizontal_angle);
    Serial.print("Vertical Angle: ");
    Serial.println(vertical_angle);
    Serial.print("Coordinates: X");
    Serial.print(face_x);
    Serial.print(", Y");
    Serial.println(face_y);
  }
}
```

Figure 39, Face tracking Arduino code pt.2

Face and gesture tracking python code pt.1

```python
import serial
import cv2
import tkinter as tk
from threading import Thread
from cvzone.HandTrackingModule import HandDetector

# Initialize serial communication with Arduino
arduino_connection = serial.Serial('COM6', 9600, timeout=1)

# Global variables
is_tracking = False
tracking_mode = "Face"
detector = HandDetector(detectionCon=0.8, maxHands=1)
horizontal_position = 90  # Default position
vertical_position = 90  # Default position
consistent_detect_count = 0
last_detected_gesture = None

# Function to send motor control commands to Arduino
def send_motor_command(horizontal_adjust, vertical_adjust):
    global horizontal_position, vertical_position
    horizontal_position = max(0, min(180, horizontal_position + horizontal_adjust))
    vertical_position = max(45, min(162, vertical_position + vertical_adjust))
    command = f"{horizontal_adjust},{vertical_adjust}\r"
    arduino_connection.write(command.encode())
    print(f"Motor Command Sent: {command}")

# Function to reset all motors to initial positions
def reset_motors():
    global horizontal_position, vertical_position
    arduino_connection.write(b"RESET\r")  # Send reset command
    response = arduino_connection.readline().decode().strip()  # Read response
    if response.startswith("INIT"):
        try:
            positions = response.split(":")[1].split(",")
            horizontal_position = int(positions[0])
            vertical_position = int(positions[1])
```

Figure 40, Face and gesture tracking python code pt.1

Face and gesture tracking python code pt.2

```python
            vertical_position = int(positions[1])
            update_status("Motors Reset to Initial Positions", "blue")
        except (ValueError, IndexError):
            update_status("Error in Reset Response", "red")
    else:
        update_status("Reset Failed", "red")

# Function to update the status label in the GUI
def update_status(message, color):
    status_label.config(text=message, fg=color)

# Function to enable/disable start/stop buttons
def toggle_buttons(start_enabled):
    start_button.config(state=tk.NORMAL if start_enabled else tk.DISABLED)
    stop_button.config(state=tk.DISABLED if start_enabled else tk.NORMAL)

# Function to handle manual control
def manual_control(horizontal_adjust, vertical_adjust):
    send_motor_command(horizontal_adjust, vertical_adjust)
    update_status(f"Manual Control: H={horizontal_position}°, V={vertical_position}°", "green")

# Function to handle gesture tracking
def gesture_tracking():
    global is_tracking, consistent_detect_count, last_detected_gesture
    is_tracking = True
    update_status("Gesture Tracking: Active", "green")
    toggle_buttons(start_enabled=False)
    send_mode_to_arduino("Gesture Tracking Mode")  # Send mode to Arduino

    video_capture = cv2.VideoCapture(0)

    while is_tracking:
        ret, frame = video_capture.read()
        if not ret:
            break

```

Figure 41, Face and gesture tracking python code pt.2

Face and gesture tracking python code pt.3

```python
        frame = cv2.flip(frame, 1)
        hands, img = detector.findHands(frame)

        if hands:
            lm_list = hands[0]
            fingers_up = detector.fingersUp(lm_list)

            if fingers_up == last_detected_gesture:
                consistent_detect_count += 1
            else:
                consistent_detect_count = 1
                last_detected_gesture = fingers_up

            if consistent_detect_count == 8:
                consistent_detect_count = 0
                if fingers_up == [0, 1, 0, 0, 0]:
                    send_motor_command(-9, 0)  # Rotate horizontally left
                elif fingers_up == [0, 1, 1, 0, 0]:
                    send_motor_command(9, 0)   # Rotate horizontally right
                elif fingers_up == [0, 1, 1, 1, 0]:
                    send_motor_command(0, -9)  # Rotate vertically down
                elif fingers_up == [0, 1, 1, 1, 1]:
                    send_motor_command(0, 9)   # Rotate vertically up

        cv2.imshow("Gesture Tracking", img)
        if cv2.waitKey(20) & 0xFF == ord('q'):
            break

    video_capture.release()
    cv2.destroyAllWindows()
    update_status("Gesture Tracking: Inactive", "red")
    send_mode_to_arduino("Idle")  # Send mode to Arduino
    toggle_buttons(start_enabled=True)

# Function to handle face tracking
def face_tracking():
```

Figure 42, Face and gesture tracking python code pt.3

Face and gesture tracking python code pt.4

```python
def face_tracking():
    global is_tracking, horizontal_position, vertical_position
    is_tracking = True
    update_status("Face Tracking: Active", "green")
    toggle_buttons(start_enabled=False)
    send_mode_to_arduino("Face Tracking Mode")  # Send mode to Arduino

    video_capture = cv2.VideoCapture(0)
    face_detector = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

    screen_width = 640  # Default video capture width
    screen_height = 480  # Default video capture height

    while is_tracking:
        ret, frame = video_capture.read()
        if not ret:
            break

        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_detector.detectMultiScale(gray_frame, 1.1, 4)

        if len(faces) > 0:
            x, y, w, h = faces[0]
            face_center_x = x + w // 2
            face_center_y = y + h // 2

            horizontal_adjust = int((face_center_x - screen_width // 2) / 20)
            vertical_adjust = int((screen_height // 2 - face_center_y) / 20)

            send_motor_command(horizontal_adjust, vertical_adjust)

            # Draw a rectangle around the face
            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)

        cv2.imshow("Face Tracking", frame)
        if cv2.waitKey(20) & 0xFF == ord('q'):
            break
```

Figure 43, Face and gesture tracking python code pt.4

Face and gesture tracking python code pt.5

```python
    video_capture.release()
    cv2.destroyAllWindows()
    update_status("Face Tracking: Inactive", "red")
    send_mode_to_arduino("Idle")  # Send mode to Arduino
    toggle_buttons(start_enabled=True)

# Function to handle manual tracking
def manual_tracking():
    send_mode_to_arduino("Manual Tracking Mode")  # Send mode to Arduino
    update_status("Manual Tracking Mode: Active", "green")
    toggle_buttons(start_enabled=False)
    # Here we can add logic for manual control or other functionality as needed

# Function to send the current mode to Arduino
def send_mode_to_arduino(mode):
    arduino_connection.write(f"MODE:{mode}\r".encode())
    print(f"Mode Sent to Arduino: {mode}")

# Function to start tracking based on the selected mode
def start_tracking():
    global tracking_mode, is_tracking
    if is_tracking:
        return  # Avoid starting another tracking if one is already running
    tracking_mode = mode_var.get()
    if tracking_mode == "Gesture":
        Thread(target=gesture_tracking).start()
    elif tracking_mode == "Face":
        Thread(target=face_tracking).start()
    elif tracking_mode == "Manual":
        Thread(target=manual_tracking).start()

# Function to stop tracking
def stop_tracking():
    global is_tracking
    is_tracking = False
    update_status("Tracking Stopped", "red")
```

Figure 44, Face and gesture tracking python code pt.5

Face and gesture tracking python code pt.6

```python
    is_tracking = False
    update_status("Tracking Stopped", "red")
    toggle_buttons(start_enabled=True)

# Tkinter GUI setup
root = tk.Tk()
root.title("Tracking Control Panel")

mode_var = tk.StringVar(value="Face")

frame = tk.Frame(root)
frame.pack(pady=10)

tk.Label(frame, text="Select Tracking Mode:").grid(row=0, column=0, padx=5, pady=5)
tk.Radiobutton(frame, text="Face Tracking", variable=mode_var, value="Face").grid(row=0, column=1, padx=5, pady=5)
tk.Radiobutton(frame, text="Gesture Tracking", variable=mode_var, value="Gesture").grid(row=0, column=2, padx=5, pady=5)
tk.Radiobutton(frame, text="Manual Tracking", variable=mode_var, value="Manual").grid(row=0, column=3, padx=5, pady=5)

# Add instructions text
instructions = """
Welcome to the Gesture Tracking Software!
Here's how to use the application:

1. Gesture Tracking:
   Select this mode to track and respond to hand gestures. \n - Index finger up to rotate left \n - index and middle fingers up to rotate right \n

2. Face Tracking:
   Switch to this mode to track faces in real time.

3. Manual Control:
   Use arrow keys or the buttons on-screen to control the device manually.

Make your selection by clicking on one of the buttons below.
"""
instructions_label = tk.Label(root, text=instructions, font=("Helvetica", 12), justify="left", wraplength=550)
instructions_label.pack(pady=10)
```

Figure 45, Face and gesture tracking python code pt.6

Face and gesture tracking python code pt.7

```python
instructions_label = tk.Label(root, text=instructions, font=("Helvetica", 12), justify="left", wraplength=550)
instructions_label.pack(pady=10)

start_button = tk.Button(frame, text="Start Tracking", command=start_tracking)
start_button.grid(row=1, column=0, padx=5, pady=5)

stop_button = tk.Button(frame, text="Stop Tracking", command=stop_tracking, state=tk.DISABLED)
stop_button.grid(row=1, column=1, padx=5, pady=5)

reset_button = tk.Button(frame, text="Reset Motors", command=reset_motors)
reset_button.grid(row=1, column=2, padx=5, pady=5)

manual_frame = tk.Frame(root)
manual_frame.pack(pady=10)

tk.Label(manual_frame, text="Manual Control:").grid(row=0, column=0, columnspan=3)

tk.Button(manual_frame, text="Up", command=lambda: manual_control(0, 9)).grid(row=1, column=1)
tk.Button(manual_frame, text="Left", command=lambda: manual_control(-9, 0)).grid(row=2, column=0)
tk.Button(manual_frame, text="Right", command=lambda: manual_control(9, 0)).grid(row=2, column=2)
tk.Button(manual_frame, text="Down", command=lambda: manual_control(0, -9)).grid(row=3, column=1)

status_label = tk.Label(root, text="Status: Idle", fg="blue")
status_label.pack(pady=10)

# Start the Tkinter main loop
root.mainloop()
```

Figure 46, Face and gesture tracking python code pt.7

Face and gesture tracking Arduino code pt.1

```
#include <Servo.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define OLED_RESET 4
Adafruit_SSD1306 display(128, 64, &Wire, OLED_RESET);

Servo servo_horizontal;
Servo servo_vertical;
Servo servo_vertical2;

// Initial servo positions
int horizontal_angle = 90;
int vertical_angle = 90;
int vertical_angle2 = 90;

void setup() {
  // Attach servos
  servo_horizontal.attach(11);
  servo_vertical.attach(3);
  servo_vertical2.attach(5);


  // Set initial positions
  servo_horizontal.write(horizontal_angle);
  servo_vertical.write(vertical_angle);
  servo_vertical2.write(vertical_angle);

  // Initialize serial communication
  Serial.begin(9600);

  // Initialize the OLED display
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.clearDisplay();
  display.setTextColor(WHITE);
```

Figure 47, Face and gesture tracking Arduino code pt.1

Face and gesture tracking Arduino code pt.2

```
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.println("System Ready");
  display.display();
  delay(2000); // Display initial message for 2 seconds
}

void loop() {
  // Check for incoming serial data
  if (Serial.available()) {
    String received_data = Serial.readStringUntil('\r');

    // Handle mode change
    if (received_data.startsWith("MODE:")) {
      String mode = received_data.substring(5);
      displayMode(mode);
    }
    // Handle reset command
    else if (received_data == "RESET") {
      resetMotors();
    }
    // Handle motor control commands
    else {
      handleMotorControl(received_data);
    }
  }
}

// Function to display mode and angles on the OLED
void displayMode(String mode) {
  display.clearDisplay();
  display.setCursor(0, 0);
  display.setTextSize(1);
  display.println("Mode:");
  display.setCursor(0, 10);
  display.println(mode);
```

Figure 48, Face and gesture tracking Arduino code pt.2

Face and gesture tracking Arduino code pt.3

```
  display.println(vertical_angle);
  display.display();
}

// Function to reset servos to initial positions
void resetMotors() {
  horizontal_angle = 90;
  vertical_angle = 90;
  vertical_angle2 = 90;

  servo_horizontal.write(horizontal_angle);
  servo_vertical.write(vertical_angle);
  servo_vertical2.write(vertical_angle);

  Serial.print("INIT:");
  Serial.print(horizontal_angle);
  Serial.print(",");
  Serial.println(vertical_angle);

  displayMode("System Reset");
}

// Function to handle motor control commands
void handleMotorControl(String data) {
  if (data.indexOf(',') == -1) {
    Serial.println("Invalid Data Format");
    return;
  }

  int horizontal_adjust = data.substring(0, data.indexOf(',')).toInt();
  int vertical_adjust = data.substring(data.indexOf(',') + 1).toInt();
  int vertical_adjust2 = vertical_adjust;

  horizontal_angle = constrain(horizontal_angle + horizontal_adjust, 0, 180);
  vertical_angle = constrain(vertical_angle + vertical_adjust, 45, 162);
  vertical_angle2 = vertical_angle;
```

Figure 49, Face and gesture tracking Arduino code pt.3

Face and gesture tracking Arduino code pt.4

```

  servo_horizontal.write(horizontal_angle);
  servo_vertical.write(vertical_angle);
  servo_vertical2.write(vertical_angle);

  Serial.print("Horizontal Angle: ");
  Serial.println(horizontal_angle);
  Serial.print("Vertical Angle: ");
  Serial.println(vertical_angle);

  displayMode("Tracking Active");
}

```

Figure 50, Face and gesture tracking Arduino code pt.4

References

American Psychological Association. (2020). *Publication manual of the American Psychological Association* (7th ed.). Washington, DC: Author.

Arm, R. (2017). *Adeept Robotic Arm Drive Board V3.0 - Compatible with Arduino UNO R3 MEGA328P*. Adeept. https://ozrobotics.com/shop/adeept-robotic-arm-drive-board-v3-0-compatible-with-arduino-uno-r3-mega328p/

Ayi, M., Ganti, A. K., Adimulam, M., Karthik, B., Banam, M., & Kumari, G. V. (2017). Face Tracking and Recognition Using MATLAB and Arduino. International Journal for Research in Applied Science & Engineering Technology (IJRASET), 5.

Banzi, M., & Shiloh, M. (2014). *Getting started with Arduino*. Maker Media, Inc.

Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.

Danelljan, M., Bhat, G., Shahbaz Khan, F., & Felsberg, M. (2016). Eco: Efficient convolution operators for tracking*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 6638-6646.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Goyal, K., Agarwal, K., & Kumar, R. (2017, April). Face detection and tracking: Using OpenCV. In *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)* (Vol. 1, pp. 474-478). IEEE.

Khalil, A., Zhang, Y., & Wei, W. (2021). Servo motor control in embedded systems: Applications and challenges. *Journal of Robotics and Automation*, 34(2), 45-60.

Kim, J., Park, S., & Kim, H. (2018). Torque control in robotic arm systems: Challenges and solutions. *International Journal of Robotics Research, 37*(5), 623-641.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet classification with deep convolutional neural networks*. Advances in Neural Information Processing Systems, 25, 1097-1105.

Larman, C. (2002). *Applying UML and patterns: An introduction to object-oriented analysis and design and the unified process*. Prentice Hall.

Lehtola, V., Huttunen, H., Christophe, F., & Mikkonen, T. (2017). Evaluation of visual tracking algorithms for embedded devices. In *Image Analysis: 20th Scandinavian Conference, SCIA 2017, Tromsø, Norway, June 12–14, 2017, Proceedings, Part I 20* (pp. 88-97). Springer International Publishing.

Li, F., & Bahsoon, R. (2018). *Software Architecture* (2nd ed.). Springer.

Luo, Z., Bi, Y., Yang, X., Li, Y., Yu, S., Wu, M., & Ye, Q. (2024). Enhanced YOLOv5s+ DeepSORT method for highway vehicle speed detection and multi-sensor verification. Frontiers in Physics, 12, 1371320.

Meyer, B. (1997). *Object-oriented software construction*. Prentice Hall.

Mitchell, J. C. (2002). *Concepts in programming languages*. Cambridge University Press.

Monk, S. (2013). *Programming Arduino: Getting started with sketches*. McGraw-Hill Education.

Norrie, R. (2020). *Arduino project handbook: 25 practical projects to get you started*.

No Starch Press.
Pan, T., & Zhu, Y. (2018). Designing Embedded Systems with Arduino. *Designing Embedded Systems with Arduino*.
Raschka, S. (2015). Python machine learning. Packt Publishing Ltd.
Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, real-time object detection*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 779-788.
Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. (1991). *Object-oriented modeling and design*. Prentice Hall.
Sharma, R., & Verma, S. (2020). Real-time performance of embedded systems in robotics applications. *International Journal of Embedded Systems*, 12(3), 129-138.
Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
servo. (2019). *Động cơ servo MG996 360 độ*. Nshopvn.com.
https://nshopvn.com/product/dong
-co-servo-mg996-360-do/
Sipe, T. A., & Stallings, W. M. (1996). Cooper's Taxonomy of Literature Reviews Applied to Meta-Analyses in Educational Achievement.
Torraco, R. J. (2005). Writing integrative literature reviews: Guidelines and examples. Human Resource Development Review, 4(3), 356-367. https://doi.org/10.1177/1534484305278283
Viola, Paul & Jones, Michael. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. IEEE Conf Comput Vis Pattern Recognit. 1. I-511. 10.1109/CVPR.2001.990517.
Viraktamath, S. V., Katti, M., Khatawkar, A., & Kulkarni, P. (2013). Face detection and tracking using OpenCV. *The SIJ Transactions on Computer Networks & Communication Engineering (CNCE)*, 1(3), 45-50.
Vogels, E. A. (2020). The State of Online Video 2020: Content Creators and Audiences. Pew Research Center.
Wilmshurst, T., & Toulson, R. (2016). *Fast and effective embedded systems design: Applying the ARM mbed*. Newnes.
Yadav, R., & Patel, V. (2019). Feedback mechanisms in embedded control systems with servo
motors. *Embedded Systems Journal*, 7(1), 15-23.
Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. (2016). *Joint face detection and alignment using Multi-task Cascaded Convolutional Networks*. IEEE Signal Processing Letters, 23(10), 1499-1503.