Project Title:

Emotion and Gesture-Driven Smart Home System with Advanced Environmental Controls

Submitted by

Hala Jadallah / 20211502045

Supervised by

Dr. Nadia Al-Rousan

This report is submitted in fulfillment of the requirements for the B.Sc. in Computer Engineering

Department of Computer Engineering

School of Electrical Engineering and Information Technology

German Jordanian University

Second Semester 2023/2024

Abstract

In order to improve automation and user experience, this project proposes the creation of a complex smart home system that incorporates gesture control, emotion recognition, and environmental monitoring. Originally, emotion detection algorithms for faces and voice were developed and integrated to produce an emotion-based light control system that modifies LED colors according to the emotions observed. The project soon grew to include advanced fan control, which made use of a DHT11 sensor, an L293D motor driver, and PID control to regulate humidity and temperature. Users can set and preset preferred fan speeds based on current conditions using an IR receiver and remote controller, and an MPU6050 sensor-driven servo motor controls the fan's direction.

To further improve the system, a machine learning model that measures the angle and distance between two fingers was added to control the fan's speed and direction depending on hand movements. The fan speed and direction are dynamically altered by this model based on the user's hand movements, and it automatically returns to regular controls after a period of inactivity. To control the lighting and fan operation, a PIR motion sensor was also used. The light and emotion sensing systems turn on when motion is sensed, and the fan runs normally. The fan is turned up to speed 100, the light goes out, and all other systems are turned off when there is no movement.

The project also includes an automated water tank system that uses an ultrasonic sensor, a water pump, and a solenoid valve to monitor water levels. Five LEDs and a buzzer alert users to crucial low and high water levels. Additionally, a DS18B20 temperature sensor measures the water temperature and displays it on an LCD screen .A rain sensor deactivates the pump and valve to stop overflow while automatically opening the tank cover to collect rainfall. This system is a prime example of how various sensors and control systems may be integrated to produce a coherent and adaptable smart home environment.

Acknowledgments

With great appreciation, I would like to thank my supervisor, Dr. Nadia Al-Rousan, whose advice, encouragement, and support were crucial to the success of this project. Her knowledge and perceptive criticism were invaluable in helping to improve and make this effort successful.

I also want to thank my family and friends for their constant inspiration and support. Their endurance and support have consistently provided me with courage and motivation.

Table of Contents

Table of Figures

Table of Tables

Chapter One: Introduction

1.1. Background

The evolution of smart home systems has been driven by two key factors: advances in technology and a growing demand for enhanced user experiences. These systems integrate a variety of technologies to automate domestic functions, thereby enhancing comfort and efficiency. A significant aspect of smart home design is the integration of psychological well-being, which entails the adaptation of the environment to align with emotional and psychological requirements. Recent developments include the advent of emotion-based lighting systems and gesture-controlled devices, indicative of a paradigm shift towards more personalized and responsive home automation solutions.

1.2. Problem Statement

Despite the advent of new technologies, many smart home systems continue to face challenges in integrating multiple functionalities in a user-friendly and efficient manner. Existing systems frequently fail to adapt dynamically to emotional states and environmental conditions, which results in suboptimal user experiences. This project addresses these issues by developing a comprehensive smart home system that incorporates emotion recognition for lighting control, gesture-based fan control, and automated water management, with the objective of enhancing overall user satisfaction and sustainability.

1.3. Objectives

The objective of the project is to:

1. Enhance automation by developing sophisticated systems for home automation that enhance user convenience and comfort.
2. Integrate emotion recognition by creating and implementing algorithms for the detection of emotions expressed through facial and vocal cues will enable the adjustment of LED colors in accordance with the observed emotions.
3. optimize environmental control by deploying sensors and proportional-integral-derivative (PID) control mechanisms. This approach allows for the precise operation of fans, enhancing energy efficiency and improving environmental monitoring.
4. The implementation of gesture control is to be carried out. A machine learning model should be designed for controlling fan speed and direction based on hand gestures, thus providing an intuitive user interface.
5. Implementing gesture control by designing A machine learning model for controlling fan speed and direction based on hand gestures, thus providing an intuitive user interface.

1.4. Scope

The project focuses on the following key areas:

A. Inclusions: The integration of innovative control mechanisms, including emotion-based lighting control, advanced fan control using environmental sensors, gesture-based fan control, and an automated water tank system.

B. Exclusions: The project does not include the integration of third-party smart home platforms or external data sources beyond the specified sensors.

1.5 Report Structure

The report is structured as follows: Chapter 2 reviews relevant literature on smart home technologies and the systems integrated into this project. Chapter 3 outlines the system design, including both hardware and software components. Chapter 4 details the methodology, covering data collection, preprocessing, implementation, and evaluation. Chapter 5 presents the results with performance analysis and visualization of key components. Chapter 6 discusses the interpretation of results, along with challenges and limitations. Finally, Chapter 7 concludes with a summary of findings and suggestions for future work.

Chapter 2: Literature Review

2.1 Existing Smart Home Technologies

The rapid advancement of smart home technologies has revolutionized the way people manage and interact with their living spaces, transforming homes into interconnected, intelligent environments. These technologies encompass a wide array of devices and systems, including smart thermostats, lighting controls, security cameras, door locks, entertainment systems, and voice-activated assistants. By leveraging the Internet of Things (IoT), these devices communicate with each other and with centralized platforms, allowing for seamless automation and remote control.

One of the primary benefits of smart home technologies is the enhancement of energy efficiency. For example, smart thermostats can learn a homeowner's schedule and preferences, adjusting heating and cooling systems to optimize energy use. Similarly, smart lighting systems can be programmed to turn off when rooms are unoccupied, reducing electricity consumption. These technologies contribute to substantial cost savings and a reduced environmental footprint.

In addition to energy efficiency, smart home technologies significantly enhance security and safety. Advanced security systems integrate motion sensors, cameras, and smart locks, providing real-time alerts and remote access to monitor and control home security from anywhere. Smoke detectors, water leak sensors, and other safety devices are also integrated into smart home ecosystems, offering automated responses to potential hazards and minimizing the risk of damage or injury.

The user experience is another area where smart home technologies excel. Voice-activated assistants, such as Amazon Alexa, Google Assistant, and Apple Siri, have become central to smart home management, allowing users to control various devices with simple voice commands. Moreover, smart home platforms offer personalized experiences, adjusting environments based on user preferences, such as preferred lighting levels, music, and temperature settings.

However, despite these advancements, the integration of smart home technologies presents significant challenges. Interoperability between different devices and platforms remains a critical issue, as many manufacturers use proprietary systems that are not always compatible with one another. This can lead to fragmented user experiences and complicate the setup and management of smart home systems. Additionally, concerns around data privacy and security pose significant risks, as the increased connectivity of devices can make homes more vulnerable to cyber threats.

In conclusion, while existing smart home technologies offer tremendous benefits in terms of convenience, energy efficiency, security, and user experience, there are ongoing challenges in achieving seamless integration and ensuring robust security. As the smart home industry continues to evolve, addressing these challenges will be crucial for realizing the full potential of intelligent living environments.

2.2 Emotion-Based Light Control System

The Emotion-Based Light Control System is an innovative technology that enhances the living environment by adapting lighting conditions based on the user's emotional state. This system uses advanced emotion detection algorithms to analyze facial expressions and vocal cues, allowing it to interpret a wide range of emotions such as happiness, sadness, or anger. By integrating this real-time emotional data with smart lighting systems, the system can automatically adjust the color, intensity, and temperature of the lights to match the user's mood. For instance, calming blue tones might be used to soothe anger, while vibrant greens could alleviate feelings of fear. Soft pink hues could amplify happiness, while neutral grays might maintain a balanced, neutral state. Warm orange lighting might comfort sadness, and deeper pinks could enhance moments of surprise. If the system detects conflicting emotional signals, it can default to a neutral white to stabilize the atmosphere.

This dynamic lighting adjustment offers significant benefits for the user's psychological and emotional well-being. By creating a responsive environment that aligns with their emotional state, the system can help reduce stress, boost mood, and foster a sense of comfort and relaxation. Additionally, the personalized nature of the system means that it can learn and adapt to individual preferences over time, ensuring that the lighting is always optimized for the user's specific needs. This level of personalization not only enhances the user experience but also makes the home environment more supportive and nurturing.

However, the implementation of the Emotion-Based Light Control System also presents challenges, particularly in the areas of privacy and emotion recognition accuracy. The system relies on continuous monitoring of personal emotional data, which raises important privacy concerns. Ensuring that this data is securely handled and protected is crucial to maintaining user trust. Furthermore, accurately detecting emotions from diverse users with varying expressions and speech patterns is a complex task that requires sophisticated algorithms and extensive training data. Despite these challenges, the Emotion-Based Light Control System represents a significant advancement in smart home technology, with the potential to greatly improve the quality of life by making our living spaces more attuned to our emotional needs.

2.3 Emotion Detection Algorithms

Emotion detection algorithms play a pivotal role in the development of smart home environments that are attuned to the emotional states of their users, thereby enhancing the personalization and comfort of the living space. These algorithms are designed to interpret complex human emotions by analyzing both facial expressions and vocal cues, leveraging advancements in machine learning and artificial intelligence.

Facial recognition technology is one of the core components of these algorithms. It involves detecting and analyzing micro-expressions, such as subtle movements in facial muscles, to discern emotions like happiness, sadness, anger, or surprise. Advanced techniques such as convolutional neural networks (CNNs) are often employed to process facial images and extract key features that correlate with specific emotional states. These features are then mapped to predefined emotional categories using classifiers that have been trained on extensive datasets of facial expressions.

In addition to facial analysis, vocal cues are also crucial in emotion detection. The algorithms process various aspects of speech, including tone, pitch, rhythm, and intensity, to infer the

speaker's emotional state. For instance, a high-pitched, fast-paced tone might indicate excitement or happiness, while a lower, slower tone might be associated with sadness or fatigue. These vocal features are extracted using techniques like Mel-frequency cepstral coefficients (MFCCs) and analyzed using machine learning models such as support vector machines (SVMs) or recurrent neural networks (RNNs) to classify emotions accurately.

The integration of these facial and vocal cues allows the system to perform a more holistic and accurate assessment of the user's emotional state. Once the system determines the user's emotions, it can dynamically adjust the environment to suit their mood. For example, if the algorithms detect signs of or frustration, the system might shift the LED lighting to softer, cooler colors, which are known to have a calming effect. Conversely, if the user is detected to be in a positive or energetic mood, the lighting could change to more vibrant and warm hues, enhancing the overall ambiance and complementing the user’s emotional state.

Moreover, these emotion detection algorithms can adapt over time, learning from the user's responses and refining their accuracy. This adaptability is achieved through continuous feedback loops where the system monitors the user’s reaction to the adjustments it makes, gradually improving its ability to predict and respond to the user's emotional needs.

However, the deployment of emotion detection algorithms also presents significant challenges. Ensuring the accuracy of emotion detection across diverse users with varying expressions and vocal tones requires extensive training data and sophisticated models. Furthermore, privacy concerns are paramount, as the system must handle sensitive data with robust security measures to prevent misuse or unauthorized access.

Emotion detection algorithms are a critical element in creating responsive and empathetic smart home environments. By analyzing facial and vocal cues, these algorithms enable the system to tailor the home’s ambiance to the user's emotional state, enhancing comfort and well-being. As the technology evolves, it will continue to refine its accuracy and adaptability, further enriching the smart home experience.

2.4 Advanced Fan Control

Advanced Fan Control represents a sophisticated approach to managing and optimizing fan operations in smart home environments. Unlike traditional fan systems that operate on basic on/off mechanisms or simple speed settings, advanced fan control integrates a range of smart technologies to enhance comfort, energy efficiency, and system responsiveness. By leveraging sensors, automation algorithms, and user preferences, this system adjusts fan settings dynamically to meet varying environmental needs, offering a more tailored and efficient approach to climate control.

Central to advanced fan control is the integration of environmental sensors that monitor factors such as temperature, humidity, and air quality. These sensors provide real-time data, allowing the system to adjust fan speeds and modes based on current conditions. For instance, the fan may increase its speed to improve ventilation when humidity levels rise or decrease when the room reaches the desired temperature. This level of automation not only ensures optimal air circulation but also contributes to energy savings by avoiding unnecessary fan operation.

Additionally, advanced fan control systems often incorporate user interfaces and integration with home automation platforms, enabling remote management and personalized settings. Users

can customize fan behavior for specific temperature and humidity. This enhanced control and flexibility make it easier to maintain a comfortable living environment while reducing energy consumption and extending the lifespan of the fan equipment. As technology advances, the capabilities of advanced fan control systems are expected to grow, further improving their efficiency and adaptability in modern smart homes.

2.4.1 Environmental Monitoring

In order to optimize the indoor environment, the project incorporated a DHT11 sensor for the purpose of monitoring temperature and humidity levels. The data obtained from the sensor was employed to regulate the fan speed, thereby ensuring a comfortable and energy-efficient environment.

2.4.2 PID Control Mechanism

A proportional-integral-derivative (PID) control mechanism was utilized to achieve precise control over the fan speed. This methodology permits the implementation of precise adjustments based on real-time temperature and humidity readings, thereby enhancing user comfort.

2.4.3 IR Remote Control

Additionally, the fan speed may be modified via an IR receiver and remote control. This feature allows users to set and preset preferred fan speeds manually, thereby catering to different preferences and needs.

2.4.4 Direction Control with MPU6050 Sensor

The direction of the fan is controlled by a servo motor that is driven by an MPU6050 sensor. This enables the fan to be directed in accordance with the user's position, thereby enhancing the overall comfort and convenience.

2.5 Gesture-Based Control System

The Gesture-Based Control System is an emerging technology that enables intuitive and seamless interaction with smart home devices through simple hand gestures. As part of the broader trend toward touchless interfaces, this system leverages advanced sensors and machine learning algorithms to interpret physical movements, translating them into commands for controlling various aspects of the home environment, such as lighting, temperature, and entertainment systems. The appeal of gesture-based control lies in its ability to provide a natural and efficient way to manage smart home functions, eliminating the need for physical touch or voice commands, and offering an enhanced user experience that is both convenient and accessible.

Incorporating gesture-based control into smart home systems also aligns with the growing demand for contactless solutions, particularly in scenarios where hygiene and ease of use are critical. For example, in environments where touching surfaces might be undesirable or in situations where voice commands may not be practical, gesture control offers an alternative that is both effective and user-friendly. By recognizing and responding to specific hand movements, the system allows users to adjust settings, turn devices on or off, and navigate through smart

home menus with a simple wave or swipe, making it a versatile tool in modern home automation.

Despite its potential, the Gesture-Based Control System also faces challenges, particularly in ensuring accurate gesture recognition and minimizing false triggers. The effectiveness of the system relies heavily on the precision of the sensors and the sophistication of the underlying algorithms that interpret the gestures. As this technology continues to evolve, ongoing advancements in sensor accuracy and machine learning will be crucial in refining gesture recognition, expanding its capabilities, and ensuring that it becomes a reliable and integral part of the smart home ecosystem.

2.6 Machine Learning Model

A machine learning model was developed to interpret hand gestures, focusing on the angle and distance between the thumb and index finger. This model enables an intuitive and hands-free method for controlling fan speed and direction. By analyzing these specific gestures, the system can adjust the fan's operation in real-time, allowing users to modify settings effortlessly without physical contact. For instance, a certain hand gesture might signal an increase in fan speed or a change in direction, while another gesture could decrease speed or reverse the airflow. This gesture-based control not only enhances user convenience but also integrates seamlessly into smart home environments, offering a novel and efficient way to manage climate control.

2.6.1 Implementation Details

The system employs sensors to detect hand movements, which are then analyzed by a machine learning model to ascertain the optimal fan speed and direction. The speed of the fan is adjusted based on the distance between the fingers, while the direction is controlled by the angle.

2.7 PIR Motion Sensor Integration

PIR (Passive Infrared) Motion Sensor Integration is a fundamental component in enhancing the automation and security features of smart home systems. PIR sensors detect movement by measuring the infrared radiation emitted by objects, particularly humans, within their range. When integrated into a smart home environment, these sensors enable a wide array of applications, from automated lighting systems that activate when someone enters a room, to security systems that alert homeowners of potential intrusions. The simplicity and reliability of PIR sensors make them an essential tool for creating responsive and energy-efficient home automation solutions.

The integration of PIR motion sensors into smart home systems offers significant benefits in terms of convenience and energy conservation. For instance, by automating the activation and deactivation of lights based on room occupancy, PIR sensors help reduce unnecessary energy consumption, contributing to a more sustainable home environment. Additionally, these sensors can be paired with other smart devices to create complex automation scenarios, such as adjusting the thermostat when no movement is detected, or triggering security cameras to start recording when motion is sensed. This seamless integration enhances the overall functionality of the smart home, making it more intelligent and responsive to the needs of its occupants.

However, successful PIR motion sensor integration requires careful consideration of sensor placement and sensitivity settings to avoid false alarms and ensure accurate detection. Factors

such as the layout of the home, the presence of pets, and the typical movement patterns of occupants must be taken into account to optimize performance. As technology advances, future developments in PIR sensor design and integration strategies are expected to further improve their accuracy and expand their applications, solidifying their role as a cornerstone of modern smart home systems.

2.8 Illumination and Ventilation Control

A passive infrared (PIR) motion sensor was integrated into the system to enhance automation. Upon detecting motion, the illumination and emotional recognition systems are activated, and the fan operates at its standard speed. Conversely, in the absence of motion, the fan is set to a speed of 100, the light is deactivated, and all other systems are deactivated, thereby conserving energy.

2.9 Automated Water Tank System

The Automated Water Tank System is an innovative solution designed to manage water resources efficiently in residential and commercial settings. This system automates the monitoring and control of water levels in storage tanks, ensuring a consistent water supply while minimizing wastage. By integrating sensors and smart control mechanisms, the system can automatically fill the tank when water levels drop below a certain threshold and stop the flow once the tank is full. This not only simplifies water management but also conserves resources, making it an essential component of smart home and building automation systems.

The integration of the Automated Water Tank System into a smart home setup offers numerous advantages, including the reduction of manual intervention and the prevention of common issues such as tank overflow or dry run, which can lead to pump damage. The system can be customized to suit different water usage patterns, adjusting the refill cycles based on consumption data, and even integrating with weather forecasts to optimize water storage during rainy seasons. This level of automation ensures that water management becomes a hassle-free process, enhancing convenience for users while promoting sustainable water usage practices.

Moreover, the Automated Water Tank System can be integrated with mobile apps and home automation platforms, providing users with real-time notifications and control options remotely. This allows homeowners to monitor water levels, adjust settings, and receive alerts for maintenance needs or unusual activity, ensuring that the water supply is always managed efficiently. As smart home technology continues to evolve, the Automated Water Tank System stands out as a practical and environmentally friendly innovation that addresses one of the most critical aspects of modern living—efficient and sustainable water management.

2.9.1 Water Level Monitoring

An ultrasonic sensor was employed to monitor the water levels in a tank, thereby ensuring that they remained within the requisite safe ranges. The data is utilized to regulate the operation of a water pump and solenoid valve, thereby facilitating the automated filling and draining processes.

2.9.2 Temperature and Rain Detection

Additionally, the system incorporates a DS18B20 temperature sensor to quantify the temperature of the water, as well as a water level sensor to detect

instances of precipitation. Upon the detection of rainfall, the pump and valve are deactivated to prevent overflow, and the tank cover is opened to facilitate the collection of the rainwater.

2.9.3 Alert System

An alert system comprising five LEDs and a buzzer was introduced to inform users of critical low and high water levels. This system guarantees that users are informed of the tank's status and can take the requisite actions to maintain optimal water levels.

2.10 Architecture Overview

The smart home system designed in this project integrates multiple subsystems with the objective of achieving seamless automation and user interaction. The system's architecture comprises both hardware and software components that interact to control various aspects of the home environment, including lighting, fan speed, direction, and water management.

The system is centered upon the Arduino Mega, which serves as the primary controller for all hardware components. The machine learning algorithms for emotion detection and gesture recognition were developed and executed using the Spyder Integrated Development Environment (IDE), thereby providing intelligent decision-making capabilities. These subsystems facilitate communication through the use of sensors, actuators, and communication protocols, thereby creating a unified and responsive smart home environment.

2.11 Hardware Components

The smart home system employs a range of hardware components that are frequently utilized in automation projects. Each component is integral to the system's ability to achieve its objectives of enhanced automation, user interaction, and environmental control.

- The Arduino Mega is a microcontroller board that: A microcontroller with a wide range of capabilities, utilized for the management of inputs and outputs across the entire system. In this project, the Arduino Mega functions as the primary processing unit, orchestrating the operation of sensors, actuators, and communication interfaces.

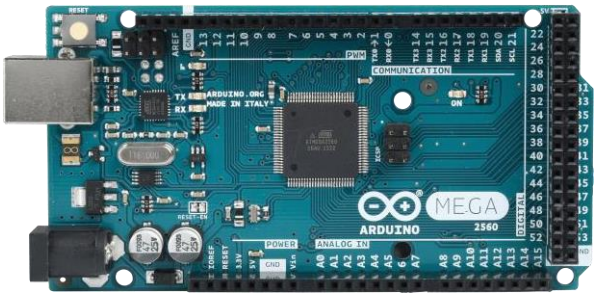


Figure 1: : Arduino Mega

- The DHT11 sensor is a device that measures temperature and humidity. A fundamental sensor that gauges temperature and humidity. These readings are crucial for maintaining an optimal indoor environment, and in this system, they are utilized to regulate the speedof the fan.

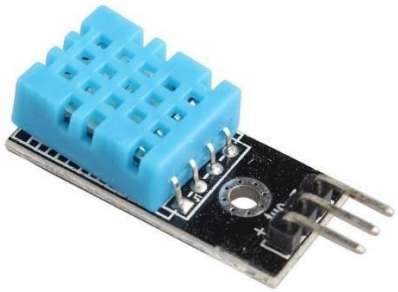


Figure 2: DHT11 sensor

- The MPU6050 sensor is a microelectromechanical (MEMS) device that integrates a microprocessor, a motion sensor, and an accelerometer. This sensor incorporates both a gyroscope and an accelerometer, which are typically employed for the purpose of motion tracking. In this context, the sensor is employed to discern alterations in the orientation of the device, thereby enabling the system to redirect the direction of the fan for enhanced air circulation.

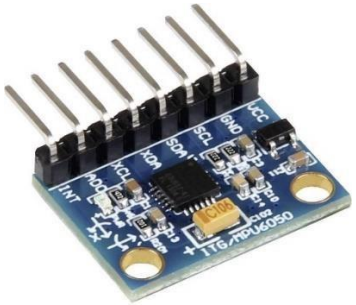


Figure 3: MPU6050 sensor

- Servo Motor (SG90): A small, precise motor that is often utilized in projects that necessitate rotational movement. In the context of the smart home system, the MPU6050 sensor is responsible for adjusting the direction of the fan in response to inputs received.

Figure 4: Servo Motor

- The L293D motor driver is a device utilized for the control of electric motors. A dual H- bridge motor driver is a device that is capable of controlling the direction and speed of DC motors. In this system, it is utilized to regulate the fan speed in response to environmental conditions.

Figure 5: L293D motor driver

- An infrared (IR) receiver is a device that detects and responds to infrared radiation. This component is commonly utilized in remote-controlled systems, enabling users to interact with the system via an IR remote, thereby providing an additional degree of control overthe fan's operation.

Figure 6: (IR) receiver

- An ultrasonic sensor is a device that emits an ultrasonic wave and then calculates the timeit takes for that wave to return to it after being reflected off an object in its path. An ultrasonic sensor is a device that measures distance by calculating the time it takes for an ultrasonic wave to return after being emitted. In this project, the device is utilized to monitor the water levels within a tank, thereby enabling the automated operation of a water pump and solenoid valve.

Figure 7: ultrasonic sensor

- The DS18B20 temperature sensor is a device utilized for the measurement of temperature. A digital sensor that is capable of accurately measuring temperature. The device is employed to oversee the temperature of the water within the tank, furnishing data for both the display of the water temperature and the water level sensor.

Figure 8: DS18B20 temperature sensor

- A water level sensor is typically employed for the purpose of measuring the depth of water within a container or tank. The device operates by detecting the presence and level of water, thereby providing valuable data for applications such as liquid level monitoring, overflow prevention, and automated water management. In this project, the water level sensor is repurposed to detect precipitation. Upon detecting rain, the system automatically opens the cover of the water tank to collect rainwater. This functionality enhances the system's sustainability by utilizing natural resources effectively.

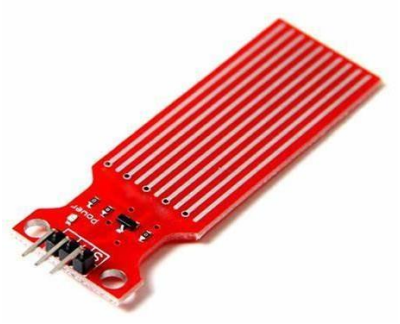


Figure 9: water level sensor

- PIR Motion Sensor: A sensor that detects motion by measuring changes in infrared radiation. This component is used to trigger the system's lighting and ventilation when movement is detected, optimizing energy usage.

Figure 10: PIR Motion Sensor

- LEDs and Buzzer: Visual and auditory output devices that are utilized to alert users to specific conditions within the system, such as the attainment of critical water levels.

Figure 11: LEDs & Buzzer

- A water pump is an electromechanical device that facilitates the movement of water from one location to another. It is commonly utilized in systems that necessitate water circulation, transfer, or pressure regulation. In this project, the water pump is controlled based on the water level detected by the sensor. The pump is responsible for transferring water when necessary and shutting off when the tank is full, thereby preventing overflow. The integration with the water level sensor ensures the efficient management of water resources.

Figure 12: water pump

- A solenoid valve is an electromechanically operated valve utilized for the regulation of liquid or gaseous flow. Such devices are frequently employed in automated systems where precise control of fluid flow is required. In this project, the solenoid valve functions in conjunction with the water level sensor and water pump to regulate the flow of water into and out of the tank. Upon detecting a critical water level, the solenoid valve initiates an adjustment in accordance with the necessary action: allowing an increased water intake or preventing an overflow by halting the flow, respectively.

Figure 13: solenoid valve

- An RGB LED (light-emitting diode) is a device that can produce a wide range of colors by combining red, green, and blue light at varying intensities. In the context of smart home systems, RGB LEDs are frequently employed for the purpose of implementing dynamic lighting effects and visual feedback. In this project, the RGB LED is employed for the purpose of emotion-based lighting control. The system modulates the color of the LED in accordance with the detected emotional state of the user, thereby contributing to apersonalized and responsive environment.

Figure 14: RGB LED

- A remote control is a wireless device that enables the operator to control electronic equipment from a distance. The typical function of a remote control is to transmit signals to a receiver, thereby enabling users to control various functions without direct interaction. In this project, the remote control is employed to regulate the fan speed via an IR receiver. This feature affords users the convenience of manual control over the fan's operation.

Figure 15: remote control

- Liquid Crystal Display (LCD): An LCD is a type of screen that is used to present information visually. It is a common feature of electronic devices, utilized for the display of data such as temperature, status messages, and other real-time information. In this project, the LCD display serves to present critical information related to water temperature, fan speed, and other system parameters, thereby enabling users to effectively monitor and interact with the smart home system.

Figure 16: LCD

- Stepper Motor with Driver: Utilized to open and close the water tank cover. The stepper motor offers precise control, ensuring that the cover opens fully to allow rainwater collection and closes securely when needed. The motor is controlled by a driver connected to the Arduino Mega, which responds to signals from the rain sensor.

Figure 17: stepper motor with driver

2.12 Software Components

The software architecture comprises several essential components that work together to control and manage the smart home system.

1. Control Algorithms

- **Emotion Detection Algorithm**:
  - Developed in Spyder using machine learning models, this component analyzes facial expressions and vocal tones to determine the user's emotional state. The detected emotions are then used to adjust the LED lighting, creating an environmentthat aligns with the user's mood.
- **Gesture Recognition Algorithm**:
  - Also developed in Spyder, this machine learning model interprets hand gestures tocontrol the fan's speed and direction. The model processes the angle and distance between the user's fingers, allowing for intuitive control of the fan through simplehand movements.
- **PID Control Algorithm**:
  - Implemented on the Arduino Mega, this algorithm regulates the fan speed based onreal-time temperature and humidity data from the DHT11 sensor. The PID controller ensures that the fan operates efficiently, maintaining a comfortable environment.

2. User Interface

- **Remote Control Integration**:
  - The software on the Arduino Mega includes code that listens for signals from an IR receiver, allowing users to manually adjust the fan speed using a remote control. This provides an additional layer of interaction, giving users direct control over the fan settings.
- **LCD Display**:
  - The Arduino Mega drives the LCD display, which shows essential information

such as water temperature, fan speed, and system status. This real-time feedback enhances user interaction by keeping them informed of the system's operation.

3. Sensor Data Processing

- **Real-time Data Processing**:
    - The Arduino Mega processes data from all connected sensors, including the DHT11 (temperature and humidity), water level sensor, and MPU6050 (gesture control). Based on predefined algorithms, the system makes decisions to control the various actuators, such as the fan, solenoid valve, and RGB LED.

2.13 Integration and Interoperability

The integration of the system components ensures that each subsystem functions in a harmoniousmanner within the broader context of the smart home environment.

- The communication between the Arduino Mega and the sensors/actuators is as follows: The Arduino Mega is responsible for processing input from all sensors and subsequently issuing control signals to actuators, including the fan motor, servo motor, and LEDs.
- The interplay between machine learning algorithms and the underlying hardware: The Spyder-based machine learning models for emotion detection and gesture control interact with the Arduino Mega via serial communication, thereby enabling real-time adjustments to the lighting and fan systems.
- Energy efficiency is achieved through the integration of passive infrared (PIR) sensors. The passive infrared (PIR) sensor optimizes energy usage by activating the system only when motion is detected, thereby reducing unnecessary power consumption.
- Coordination of the Water Management System: The ultrasonic sensor, temperature sensor, and rain sensor collectively facilitate optimal water level management. The Arduino Mega serves as the coordinating hub for the operation of the water pump, solenoid valve, and tank cover, leveraging sensor data to facilitate their seamless integration.

This architectural and integration strategy guarantees that the smart home system will be responsive, efficient, and adaptable to a variety of user requirements and environmental conditions.

Chapter Three: Methodology

3.1 Data Collection

3.1.1 Facial Emotion Recognition

This project employed the FER-2013 (Facial Expression Recognition 2013) dataset as the principalsource of data for training and testing the emotion detection model. The FER-2013 dataset is widely recognized in the field of facial expression recognition and comprises 35,887 grayscale images of faces, each with a resolution of 48x48 pixels. The images are classified into seven discrete emotional categories. The emotions included are: angry, disgusted, fearful, happy, neutral,sad, and surprised. The dataset is divided into three distinct sets: training, validation, and test. Thetraining set comprises 28,709 images, while the test set contains 7,178 images. This comprehensivedataset is well-suited for training deep learning models to recognize and classify human emotionsfrom facial expressions.

3.1.2 Speech Emotion Recognition

The data utilized for training and testing the speech emotion recognition model were derived fromfour primary datasets: The datasets utilized for training and testing the speech emotion recognitionmodel were RAVDESS, CREMA-D, TESS, and SAVEE. The datasets comprise audio recordingsof speech, each of which has been assigned a label indicating the emotion expressed, such as happiness, sadness, anger, fear, disgust, or neutrality.

- RAVDESS (The Ryerson Audio-Visual Database of Emotional Speech and Song) is a dataset.
- The Crowd-sourced Emotional Multimodal Actors Dataset (CREMA-D) constitutes another source of data for this study.
- TESS represents the Toronto Emotional Speech Set.
- SAVEE represents the Surrey Audio-Visual Expressed Emotion (SAVEE) dataset.

The datasets provided a diverse and comprehensive set of audio samples, which were instrumental in the development of a robust speech emotion recognition model.

3.1.3 Combined Modal Data Collection

The data collection process for the combined emotion recognition model was based on two primarysources:

1. Real-time data collection
    - Audio data were collected. Vocal inputs were captured in real time using a microphone for the purpose of conducting a speech emotion analysis.
    - Video data were also collected. Video footage was captured using a camera for the purpose of analyzing facial emotions in real time.
2. A pre-recorded dataset was also utilized.
    - RAVDESS Dataset: The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) was employed for the purposes of testing. The dataset comprises 7,356 files of emotional speech and song, providing a diverse range of

emotional expressions for comprehensive testing.

- The features dataset is as follows: A Comma-Separated Values file containing precomputed features for both speech and facial emotion detection was employed to fit the scaler and encoder models.

The data sources provided a comprehensive range of inputs for the training and testing of the integrated emotion detection system.

3.1.4 Hand Gesture Detection System

The data for the hand gesture detection system was collected via a real-time webcam feed, which was used to capture hand gestures. The following components were involved in the data collection process:

- The motion data are as follows: The primary data set comprised hand landmark coordinates obtained via webcam capture. The data was then processed using the MediaPipe Hand Landmark model, which tracks and records the positions of key points on the hand.
- Communication with the Arduino device was facilitated through the use of the following communication protocol: The data captured, specifically the distance and angle between
- the thumb and index finger, was transmitted to an Arduino device via serial communication. This enabled the provision of real-time feedback and interaction based on hand gestures.

The system was evaluated in real-time using live webcam input, thereby ensuring the relevance and applicability of the collected data to the intended use cases.

3.2 Data Preprocessing

3.2.1 Facial Emotion Recognition

Data preprocessing is a critical stage in the preparation of images from the FER-2013 dataset for input into the Convolutional Neural Network (CNN) model. The following preprocessing steps were undertaken:

1. The images were then subjected to a process of rescaling. All images were rescaled by a factor of 1/255 in order to normalize the pixel values. This step ensures that the pixel values, which originally ranged from 0 to 255, are scaled toa range of 0 to 1, which is more suitable for input into the neural network.

2. The images were converted to grayscale. As the images in the FER-2013 dataset are already in grayscale, no further conversion was required. The grayscale formatreduces the computational complexity of the images, thereby facilitating the training of the model in a more expedient and efficacious manner.

3. The process of data augmentation is as follows: To enhance the robustness of thetraining set, data augmentation techniques, including rotation, zooming, and horizontal flipping, were employed. These techniques serve to enhance the diversity of the training set, thereby reducing the risk of overfitting.

4. Image Batching: The images were grouped into batches of 64 for both the training and

testing phases. The process of batching improves computational efficiency andenables the model to learn more effectively by processing multiple images simultaneously.

The preprocessing steps were implemented using the ImageDataGenerator class from the TensorFlow Keras preprocessing image module, which enables the dynamic augmentationand preprocessing of data during model training

3.2.2 Speech Emotion Recognition

The data preprocessing stage comprised a series of procedures designed to prepare the audio files for feature extraction and model training.

1. Data labeling is the process of assigning labels to data points. The datasets were traversed in order to retrieve the file paths and corresponding emotion labels, which were then stored in dataframes.

2. The data underwent augmentation techniques, including the addition of noise, time-stretching, shifting, and pitch alteration. The audio files were subjected to a series of augmentation techniques, including the introduction of noise, time- stretching, shifting, and pitch alteration. These techniques were employed with theobjective of enhancing the diversity of the training data.

3. The process of feature extraction is as follows: A variety of audio features were extracted, including the zero-crossing rate, chroma STFT, mel-frequency cepstral coefficients (MFCC), and spectral roll-off, utilizing the librosa library. For each audio file, three sets of features were generated: the original, the file with added noise, and the file with pitch shifted and stretched.

4. The extracted features were stored for subsequent processing. The extracted features, along with their corresponding labels, were stored in a comma-separatedvalues (CSV) file for subsequent processing.

3.2.3 Combined Modal Data Preprocessing

The data preprocessing stage entailed a series of procedures to prepare both audio and video data for the emotion detection models.

1. Extraction of Audio Features

    - The audio data were subjected to feature extraction using the librosa library, which yielded the following features: zero-crossing rate (ZCR), chroma short-time Fourier transform (Chroma STFT), mel-frequency cepstral coefficients (MFCC), root mean square (RMS), and mel-spectrogram.

    - The aforementioned features were then aggregated into a feature vector, which was subsequently normalized using the StandardScaler algorithm to ensure consistent input ranges.

2. Video frame processing entailed the following steps:

    - Each video frame was converted to grayscale to reduce computational complexity

and resized to fit the input dimensions required by the facial emotion detection model.

- The Haar cascade classifier (haarcascade_frontalface_default.xml) was employed to identify faces within each frame, and the identified faces were then conveyed to the facial emotion model for prediction.

3. The next step is to encode and scale the data.
    - The OneHotEncoder was utilized to transform the emotion labels into a format that was compatible with the model's prediction process.
    - To ensure consistency in the predictions, both the scaler and encoder were trained on the same data set.

3.2.4 Hand Gesture Detection System

It was imperative that data preprocessing be conducted in order to guarantee that the raw data from the webcam would be suitable for gesture recognition and communication with the Arduino device. The following preprocessing steps were undertaken:

1. Landmark Detection:
    - The MediaPipe Hands solution was employed for the detection and tracking of 21 hand landmarks. The position of each landmark was recorded in pixel coordinates on the webcam feed.
2. The next stage of the process was feature extraction.
    - The positions of the thumb (landmark 4) and index finger (landmark 8) were isolated from the detected landmarks. The distance between the two points was calculated using the Euclidean distance formula, and the angle between them was computed using the arctangent function.
3. Data Smoothing:
    - In order to reduce noise and fluctuations in the measurements, a moving average technique was applied to both the distance and angle values, thereby producing a more stable and accurate representation. A window size of five was employed to compute the average, thereby ensuring more stable and accurate readings.

This preprocessing ensured the reliability and representativeness of the data transmitted to the Arduino device, reflecting the actual hand gestures performed.

3.3 System Implementation

3.3.1 Facial Emotion Recognition

The system for facial emotion recognition was implemented using the Python programming language, which was used to leverage the powerful libraries TensorFlow and Keras for deep

learning. The implementation can be divided into the following steps:

1. The model structure was designed as follows:
    - A sequential model was constructed using the Keras library.
    - The model comprises multiple layers, including convolutional layers (Conv2D), pooling layers (MaxPooling2D), dropout layers (Dropout), and fully connected layers (Dense).
    - The architecture commences with a series of convolutional layers, which are employed to extract features from the input images. This is followed by max-pooling layers, which serve to reduce the spatial dimensions. Dropout layers are employed to avert overfitting by randomly disabling a proportion of the neurons during the training phase. Subsequently, the flattened output is conveyed through fully connected layers, with the final layer exhibiting a softmax activation to output probabilities for the seven emotion classes.
2. The model was compiled as follows:
    - The model was compiled using the Adam optimizer with a learning rate of 0.0001. The loss function employed was categorical_crossentropy, which is well-suited to multi-class classification tasks. The metric utilized to assess the model's performance was accuracy.
3. Model Training:
    - The model was trained on the training set for a total of 150 epochs. The fit method was employed for the training process, and the model's performance was assessed on the validation set after each epoch.
    - The training process was conducted on batches of 64 images, with a total of 28,709 images in the training set and 7,178 images in the validation set.
4. Model Saving:
    - Upon completion of the training phase, the model's architecture was saved to a JSON file, and the trained weights were saved in a .h5 file. This facilitates the straightforward reloading of the model for subsequent inference or further training.

3.3.2 Speech Emotion Recognition

The system was developed in Python using TensorFlow and Keras for model construction.

Thesystem is comprised of several key components, including:

1. The data loading and preprocessing stage is concerned with the importation of data into asuitable format for further analysis. Functions were developed to facilitate the loading and preprocessing of data from CSV files, encompassing one-hot encoding of labels and normalization of features.
2. Model Design and Training: A convolutional neural network (CNN) comprising

residualblocks was devised for the purpose of emotion recognition. The model was trained usinga combination of augmented and original features, with early stopping and learning rate reduction applied for optimal training.

3. Real-time testing was conducted. A function was developed to capture audio via a microphone, extract features in real time, and predict the emotion using the trained model.

3.3.3 Combined Modal Implementation

The system was implemented using the Python programming language, which was used toleverage several libraries for different tasks.

1. Speech Emotion Detection:
    - The librosa library was employed for the purpose of extracting audio features.
    - A pre-trained Keras model was loaded and utilized to predict the emotion based on the extracted features.
2. Facial Emotion Detection:
    - The facial emotion detection system employed a Keras model that was loaded from a JSON file, with the objective of predicting emotions based on the processed video frames.
3. Serial Communication:
    - A serial communication link was established using the pyserial library to facilitate the transmission of the detected emotions to an Arduino board.
    - This enabled the provision of real-time feedback and interaction based on the detected emotions.
4. The aforementioned components were integrated in order to create a unified system.
    - The system was integrated to accommodate both real-time data from cameras and microphones, as well as pre-recorded videos from the RAVDESS dataset.
    - A significant aspect of the implementation was the capacity to compare and integrate predictions from both speech and facial emotion models, thereby identifying discrepancies when they arose.

3.3.4 Hand Gesture Detection System

The system was implemented using the Python programming language, with a variety of librariesfacilitating different aspects of the project.

1. The system was designed to detect hand gestures.
    - The MediaPipe library was employed for the real-time detection and tracking of hand landmarks. This approach yielded a robust method for identifying the

positions of the thumb and index finger, which are of paramount importance for gesture recognition.

2. The subsequent step involved the calculation of the distance and angle between the two points.

    - Utilizing the coordinates of the thumb and index finger, the system computed the distancebetween them and the angle formed by the line connecting the two. These metrics served as pivotal indicators of particular hand gestures.

3. Communication via Serial Port with Arduino:

    - A communication channel was established between the Python serial library and an Arduino device via a COM port. The smoothed distance and angle values were transmitted to the Arduino, which was thus enabled to respond to the detected gestures.

4. Webcam Interface:

    - The cv2 library from OpenCV was employed to facilitate the management of the webcam feed, display the processed video stream, and superimpose the calculated distance and angle on the video.

The integration of these components resulted in the creation of a fully functional system capableof detecting hand gestures and communicating the results in real time.

3.4 Algorithm Design

3.4.1 Facial Emotion Recognition

The algorithm utilized for facial emotion recognition is based on a Convolutional Neural Network (CNN), which is particularly well-suited for image classification tasks. The convolutional neural network (CNN) architecture designed for this project comprises the following key components:

1. The convolutional layers comprise:

    - These layers apply convolution operations to the input images, thereby effectively capturing local features such as edges, corners, and textures.

    - The initial convolutional layer comprises 32 filters, with subsequent layers incorporating 64, 128, and 128 filters, respectively. All convolutional layers employ a 3x3 kernel and the ReLU activation function.

2. Pooling layers are employed to

    - Subsequently, a max-pooling layer is applied following each set of convolutional layers, with the objective of reducing the spatial dimensions of the feature maps. This facilitates a reduction in the computational load and the probability of overfitting.

3. Dropout layers are employed to:
    - The dropout technique is employed following the pooling layers with the objective of preventing overfitting. This is achieved by randomly setting a proportion of the input units to zero during the training phase. This compels the network to discern more resilient features that are not contingent on the existence of particular neurons.
4. Fully Connected Layers:
    - Following the flattening of the output from the convolutional and pooling layers, the features are conveyed through a fully connected layer comprising 1024 units and ReLU activation. An additional dropout layer is introduced prior to the final output layer.
5. The output layer is comprised of the following components:
    - The final layer is a dense layer comprising seven units, which correspond to the seven discrete emotion classes. A softmax activation function is employed to output a probability distribution over the classes.

The overall design of the convolutional neural network (CNN) enables it to learn complex patterns and features from the input images, thereby facilitating accurate classification of emotions.

3.4.2 Speech Emotion Recognition

The emotion recognition model was designed as a convolutional neural network (CNN) comprising multiple residual blocks. The model architecture comprises the following elements:

1. The input layer is comprised of:
    - The model is configured to accept input features with dimensions that align with those of the extracted audio features.
2. The residual blocks are as follows:
    - These blocks comprise multiple layers of 1D convolutional layers, activation functions, and skip connections, thereby enabling the model to learn more complex representations.
3. The next stage of the model is the pooling and flattening layers.
    - The AveragePooling1D layer is employed for downsampling the feature maps, while the Flatten layer is utilized to convert the 2D outputs into 1D.
4. Fully Connected Layers:
    - Dense layers are employed for the purpose of learning higher-level features and generating predictions.
5. The final layer of the network is the output layer, which
    - A Dense layer with a softmax activation function is employed to predict

probabilities across the different emotion classes.

The model was compiled using the Adam optimizer with a categorical cross-entropy loss function, and its performance was monitored using accuracy metrics.

3.4.3 Combined Modal

The algorithm was designed to address the challenging task of emotion detection in a robust andefficient manner.

1. Speech Emotion Prediction:
    - The algorithm initially extracts features from the audio input through the utilisation of predefined feature extraction methods.
    - Subsequently, the extracted features undergo normalization and are conveyed to the pre- trained Keras model for the purpose of emotion prediction.
2. Facial Emotion Prediction:
    - Video frames are processed in real time, with faces being detected and isolated through the use of Haar cascades.
    - The isolated faces are then fed into a convolutional neural network (CNN)-based Keras model, which outputs the predicted emotion.
3. Emotion Matching:
    - The system compares the predicted speech and facial emotions. If the two predictions are in alignment, the combined emotion is deemed accurate. Conversely, a discrepancy between the predictions is identified as a mismatch.
4. Communication with Arduino:
    - The final combined emotion, whether matched or flagged as a mismatch, is transmitted to an Arduino device via serial communication for subsequent processing or interaction.

3.4.4 Hand Gesture Detection System

The algorithm was designed with the objective of efficiently processing hand gesture data and communicating the results to an external device. The principal stages in the design of the algorithm are as follows:

1. The process of identifying the distinct physical features of the human hand, known ashand landmark detection, is a crucial initial step in the algorithmic analysis of hand gestures.
    - The algorithm initiates the process by capturing a frame from the webcam and transmitting it to the MediaPipe Hands model. The model returns the coordinates of 21 hand landmarks.
2. Computations of distance and angle:
    - Subsequently, the algorithm identifies the positions of the thumb and index

finger, subsequently calculating the distance between them and the angle formed by their connecting line. These calculations serve as the foundation for the recognition of specific gestures.

3. Data Smoothing:

    - In order to mitigate fluctuations in the calculated distance and angle values, the algorithm employs a moving average filter. This process of data smoothing is essential for reducing any noise that may be present in the data set and ensuring that the data accurately reflects the intended gestures.

4. Serial Communication:

    - Subsequently, the smoothed values are formatted as a string and transmitted to the Arduino device via a serial connection. The Arduino utilizes the aforementioned values to perform actions or provide feedback based on the detected gesture.

5. A real-time display is also provided.

    - The processed video feed, along with the distance and angle measurements, is displayed in a dedicated window. This provides the user with visual feedback, which is beneficial for debugging and improving the system.

The algorithm was designed with the objective of ensuring efficiency and real-time processing, thereby guaranteeing the smooth operation of the system during live interactions.

3.5 System Evaluation

3.5.1 Facial Emotion Recognition

The trained facial emotion recognition model was evaluated on the test set to assess its performance. The evaluation process was conducted in accordance with the following steps:

1. The confusion matrix is presented below:

    - A confusion matrix was constructed to provide a visual representation of the model's performance across all emotion classes. The resulting matrix provides a visual representation of the model's performance, showcasing the number of true positives, false positives, false negatives, and true negatives for each emotion class.

    - The model demonstrated proficiency in recognizing emotions such as "happy" and "surprised," yet exhibited limitations in accurately identifying other emotions, including "fearful" and "sad." For example, the "Happy" emotion exhibited a high true positive rate, whereas the "Fearful" and "Sad" emotions demonstrated a greater degree of confusion with other classes.

confusion matrix:

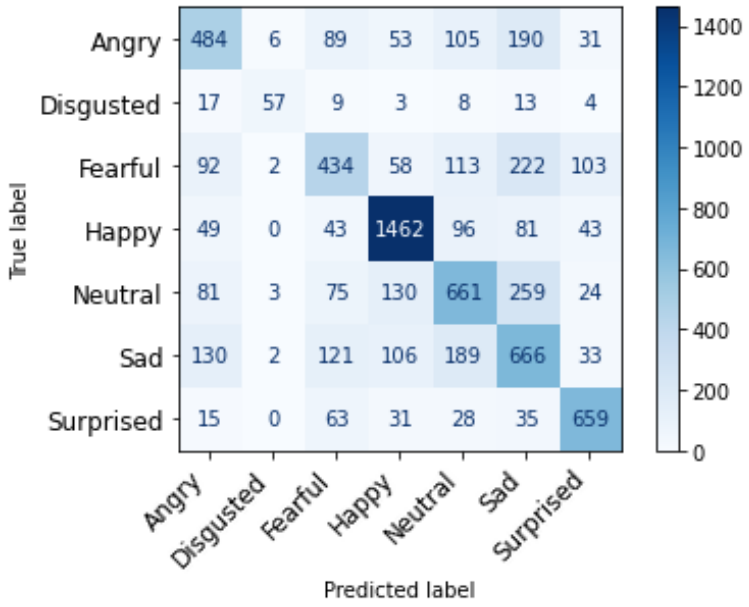


Figure 18: confusion matrix of facial emotion recognition

2. A classification report is provided below.

    - A comprehensive classification report was produced, comprising precision, recall, and F1-score data for each emotion class.
    - The overall accuracy of the model on the test set was 62%. The highest F1-score was observed for the "happy" emotion (0.81), while the "fearful" emotion exhibited a lower F1-score (0.47).

classification report:

Table 1: classification report of facial emotion recognition

Emotion	Precision	Recall	F1-Score	Support
Angry			3	958
Disgusted	0.81	0.51	0.63	111
Fearful	0.52	0.42	0.47	1024
Happy	0.79	0.82	0.81	1774
Neutral	0.55	0.54	0.54	1233
Sad	0.45	0.53	0.49	1247
Surprised	0.73	0.79	0.76	831
Overall Accuracy	\multicolumn4|c|0.62			

Figure 19: confusion matrix of facial emotion recognition

3. A visual evaluation was conducted to assess the system's performance.

- The system was evaluated in real time using a webcam feed. The model demonstrated efficacy in detecting faces within the video stream, in predicting the associated emotional state, and in displaying the corresponding emotion label on the video feed. This real-time application demonstrated the practical utility of the model in a live setting.

4. The following limitations were identified:

    - The evaluation revealed certain limitations, such as lower accuracy for emotions with subtle differences and the impact of lighting conditions on detection performance. The confusion matrix and classification report identified areas for further improvement, particularly in distinguishing between similar emotions such as "fearful" and "surprised."

3.5.2 Speech Emotion Recognition

The speech emotion recognition model was evaluated using a dataset that was divided into three distinct sets: training, testing, and validation. The performance of the model was evaluated using a number of metrics.

1. The accuracy of the training and testing phases is presented below.

    - The training accuracy of the model was 99.01%, indicating that the model demonstrated an excellent capacity to learn the patterns present in the training data.

    - However, the testing accuracy was 73.99%, and the validation accuracy was 73.17%, which suggests that the model may have exhibited some degree of overfitting or that its performance may be constrained when applied to data that it has not previously encountered.

2. The classification report is as follows:

    - The classification report presents the precision, recall, and F1-score for each emotion class, with an overall accuracy of 74% on the test set.

classification report:

Table 2: classification report of speech emotion recognition

Emotion	Precision	Recall	F1-Score	Support
Neutral	0.80	0.84	0.82	586
Calm	0.77	0.68	0.72	565
Happy	0.73	0.67	0.70	572
Sad	0.71	0.74	0.73	584
Angry	0.75	0.71	0.73	506
Fearful	0.69	0.80	0.74	582
Overall Accuracy	\multicolumn4|c|0.74			

3. A confusion matrix is a statistical tool used to evaluate the performance of a machine learning model. It is a matrix that presents the true positive (correct classification), false positive (misclassification), true negative (correct rejection), and false negative (misclassification) rates for each category in

    - The confusion matrix illustrates the efficacy of the model in differentiating between various emotional states. For example, the "neutral" emotion was correctly identifiedwith a high degree of consistency, whereas there were instances of confusion when attempting to distinguish between other emotions.

Confusion matrix:

$$\begin{bmatrix} 490 & 16 & 17 & 50 & 6 & 7 \\ 40 & 382 & 30 & 32 & 29 & 52 \\ 29 & 21 & 383 & 51 & 24 & 64 \\ 42 & 24 & 40 & 433 & 15 & 30 \\ 5 & 39 & 16 & 29 & 359 & 58 \\ 3 & 17 & 36 & 14 & 47 & 465 \end{bmatrix}$$

Figure 20: confusion matrix of speech emotion recognition

3.5.3 Combined Modal Evaluation

The system's performance was evaluated on both real-time data and the RAVDESS dataset, withthe following assessments:

1. An assessment of accuracy was conducted.
    - The accuracy of the speech and facial emotion detection models was evaluated through a comparative analysis of their predictions with the known labels present in the RAVDESS dataset.
2. Real-time testing was conducted.
    - The integrated system was evaluated in real-time with live inputs to assess its responsiveness and accuracy in a dynamic environment.
    - A total of 70 frames per video were processed to predict the most frequent facial emotion.
3. Mismatch Handling:
    - The system's capacity to identify and address discrepancies between speech and facial expressions was assessed.

4. Overall system performance was evaluated.
    - The combined predictions were evaluated with a particular emphasis on the system's capacity to make accurate predictions when both modalities concurred. The evaluation demonstrated that the system was operational and dependable in a range of scenarios.

3.5.4 Hand Gesture Detection System

The hand gesture detection system was evaluated based on its capacity to accurately detect gestures and convey the results to an Arduino device. The evaluation was conducted with a particular emphasis on several key aspects.

1. Real-time performance was evaluated in order to ascertain the system's capacity to process video frames in a timely manner.
    - The system's capacity to process video frames in real time was evaluated to ascertain the latency between the gesture being performed and the corresponding response from the Arduino device.
2. The accuracy of the gesture detection system was evaluated.
    - The precision of the distance and angle calculations was evaluated by comparing the system's outputs with known reference values. The moving average smoothing technique proved an effective means of reducing noise without introducing significant lag.
3. The reliability of the communication system was evaluated.
    - The reliability and consistency of the serial communication between the Python script and the Arduino device were evaluated to ascertain that the Arduino was able to correctly receive and interpret the data.
4. The system was evaluated in terms of its usability and responsiveness.
    - The system's usability was evaluated under a range of lighting conditions and with varying hand positions to ascertain the accuracy of gesture detection in real-world scenarios.

The results of the evaluation demonstrated that the system was capable of accurately detecting and responding to hand gestures in real time, thereby establishing its suitability for practical applications.

Chapter Four: Results

This chapter presents the findings of the project, with a particular focus on the performance analysis and visualization of the system's constituent components and their integration. The project encompassed a multitude of subsystems, each presenting its own distinctive challenges and outcomes. The results are presented in two main sections, as follows: A performance analysis and visualization of the system's various components and their integration.

4.1 Performance Analysis

4.1.1 PID Control of Fan Speed

A proportional-integral-derivative (PID) control system was implemented with the objective of regulating the speed of the fan using the L293D motor driver. Through a process of iterative testing and adjustment of the PID constants, the optimal values were determined to be:

The following values were determined through experimentation:

- Kp=40.0
- Ki=0.01
- Kd=1.0

The setpoints were defined as follows:

The temperature was set to 25.0°C, and the humidity was set to 30.0%. These parameters were then adjusted to achieve the desired response under a range of temperature conditions. The system exhibited effective control over the fan speed, maintaining the desired temperature and humidity levels. It was observed that these values could be modified according to user preferences, thereby allowing for flexibility in the system's operational parameters.

4.1.2 Integration of Servo Motor with MPU6050 Sensor

The servo motor was successfully controlled using the MPU6050 sensor, thereby enabling directional control based on sensor input. However, the integration of this system with the fan control initially resulted in the fan becoming non-functional, despite the system indicating that the fan speed was being set.

The issue was identified as a conflict between the libraries utilized for the servo motor and the fan control, specifically the Servo.h and Wire.h libraries, which both employed Timer1 on the Arduino.

The solution entailed a transition to the ServoTimer2.h library, which effectively resolved the conflict, enabling the simultaneous and optimal operation of both the fan and the servo motor when integrated.

4.1.3 IR Receiver and Remote Control Integration

The system was expanded to include an IR receiver and remote control, thereby enabling the storage of up to 10 user-defined temperature and humidity settings. Initially, the integration of the IR control with the servo and fan control resulted in a malfunction of the servo motor, caused by a conflict between the IRremote and ServoTimer2 libraries. The utilization of alternative libraries, such as IRLib2 and VarSpeedServo, led to the emergence of further

conflicts, particularly with regard to the fan control.

The final solution entailed the utilisation of the Servo library on a distinct timer and the repositioning of the fan control to an alternative pin (5 or 6), which employs Timer0. This approach effectively resolved the conflicts, enabling the servo motor, IR receiver, and fan to operate in a unified and streamlined manner.

4.1.4 LED Control with Emotion and Motion Detection

The LED system was designed to reflect the detected emotions, with colors selected based on studies that have demonstrated a correlation between specific colors and emotional states. The emotion detection was based on two models: one for facial emotion recognition and one for speech emotion recognition. Each model was trained separately, and the issue of overfitting was addressed by capping the training at 50 epochs for both models.

The combination of the models presented a considerable challenge due to the discrepancies in the output shapes, which resulted in a ValueError during the testing phase. The final solution entailed a logic-based combination, whereby the system would only recognize an emotion if both models reached a consensus.

The integration with the Arduino was achieved by transmitting the detected emotion as a numerical value, which was then used to control the LED color. Furthermore, a motion detection system was incorporated to ensure that the system would only be activated when motion was detected.

Integrating this subsystem with the fan and other components presented a challenge. Due to the limitations of the number of pins available on the Arduino, the LED control was initially relocated to the analog pins. However, this resulted in the inability to achieve the desired level of color control. Attempts to utilize the SoftPWM library were unsuccessful. The solution was to switch to an Arduino Mega, which provided an adequate number of pins and resources to manage all components effectively.

4.1.5 Water Tank Management System

The water tank management system was implemented with the integration of multiple components.

- An ultrasonic sensor was employed for the purpose of measuring water levels.
- LED indicators for monitoring water levels.
- A buzzer is also included for the purpose of alerting users to critical water levels.
- The system incorporates a water pump and solenoid valve for automated refilling.
- A water level sensor is employed for the purpose of rain detection and tank cover control.
- A DS18B20 sensor is employed for the measurement of water temperature.

The system demonstrated effective management of the water tank, with the cover opening to collect rainwater when necessary and maintaining a sufficient level of reserve capacity. Minor delays were observed in the response times of the system, particularly in the operation of the motion sensor and remote control, as well as in the opening and closing of the tank cover.

Nevertheless, these delays did not have a considerable impact on the overall performance of the system.

4.1.6 Hand Gesture Control for Fan

The functionality to control the fan speed manually via hand gestures has been incorporated into the system. The integration of this system with the existing emotion and motion detection subsystems was seamless. The primary challenge was to guarantee that the signals controlling the fan speed did not interfere with those detecting emotions. This issue was addressed by modifying the communication protocol to ensure the generation of distinct and non-conflicting signal values.

4.2 Visualization

4.2.1 Fan Speed and Temperature

A plot was generated to illustrate how the PID controller adjusts fan speed in response to changes in temperature and humidity. The plot, combined with a screenshot of the serial monitor, showcases the real-time performance of the system. It demonstrates how the final PID configuration achieves smooth fan speed adjustments, effectively maintaining the desired environmental conditions with minimal overshoot.

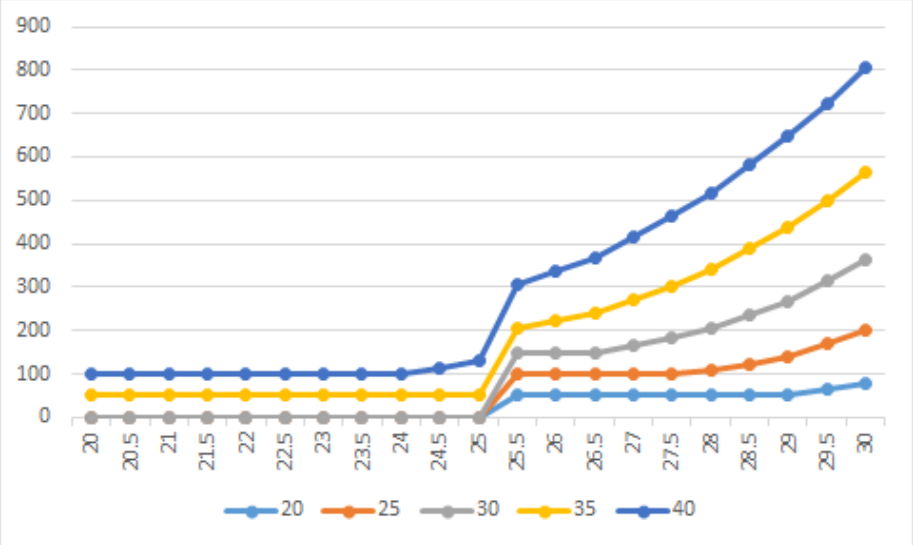


Figure 21: PID Control: Fan Speed Adjustment vs. Temperature and Humidity

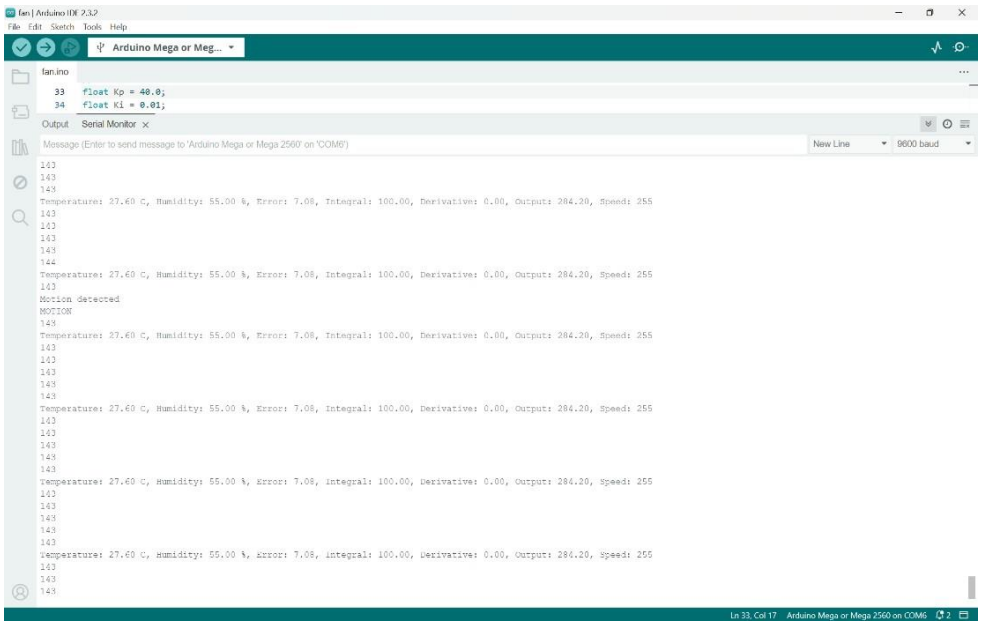


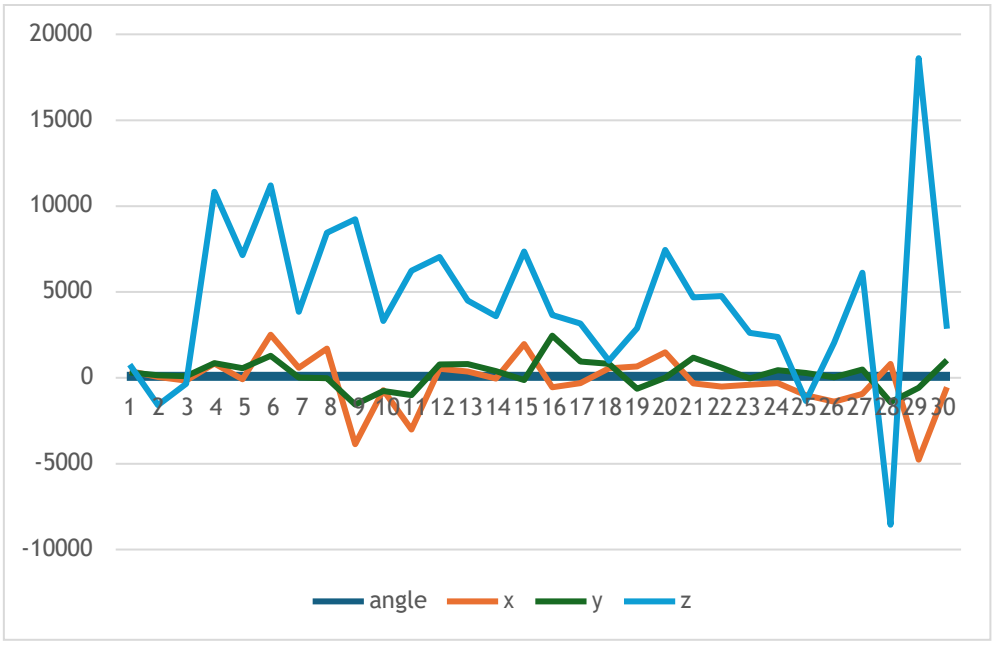
Figure 22:Example Serial Monitor Output of the system

4.2.2 Servo Motor Direction

The data visualization entailed the plotting of the servo motor's angle and the MPU6050 sensor's x, y, and z-axis readings over time. The resulting plots provide a comprehensive representation of the servo motor's response to changes detected by the sensor.

- The plot of Servo Angle versus Time illustrates the fluctuations in the servo motor's angle as it adjusts in response to sensor readings over time.
- Sensor x-Axis Readings vs. Time: This plot depicts the fluctuations in the x-axis readings, which reflect changes in horizontal tilt detected by the MPU6050 sensor.
- The plot of the sensor y-axis readings versus time illustrates the variations in the y-axis readings, which represent changes in vertical tilt.
- The plot of the sensor z-axis readings versus time demonstrates the changes in the z-axis readings, indicating rotational movements detected by the sensor.

The integrated analysis of the servo angle and sensor readings demonstrates that the servo motor effectively tracks the changes detected by the MPU6050 sensor. The plots demonstrate the servo's capacity for responsive adjustments and the sensor's capability to detect a comprehensive range of orientation alterations, encompassing horizontal tilt, vertical tilt, and rotational movement.



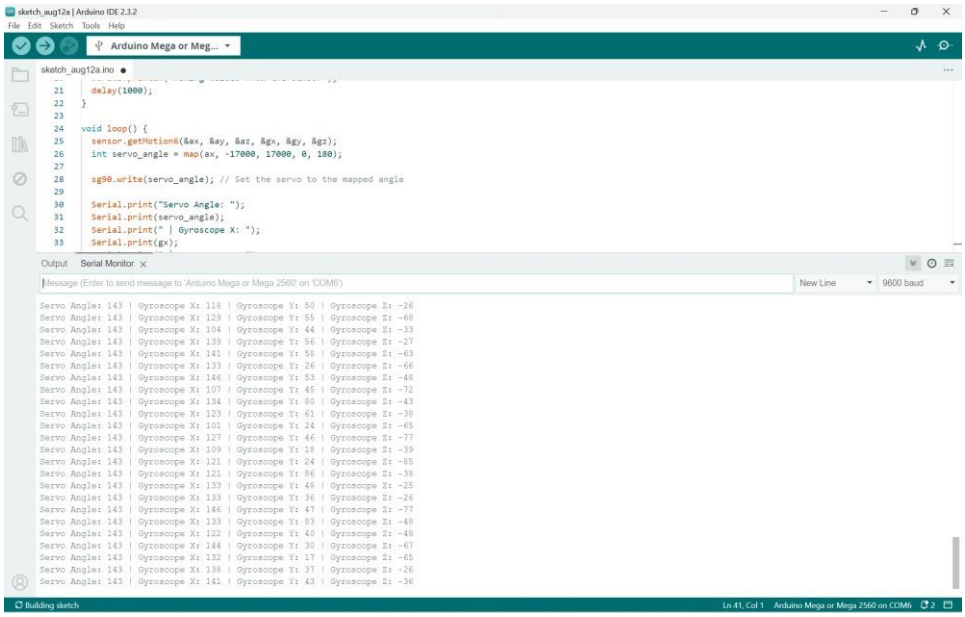


Figure 23: Servo Angle and MPU6050 Sensor Readings Over Time and Example serial monitor output

4.2.3 Emotion Detection

The specifics regarding the accuracy and performance of the system are discussed in Chapter 4. The following illustration depicts the manner in which the code is executed.

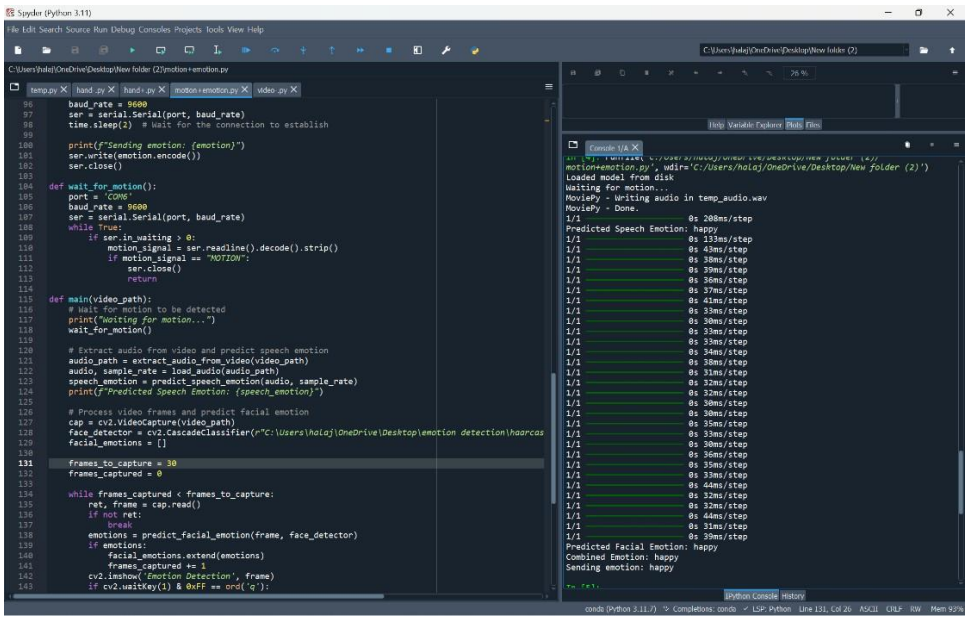


Figure 24: Example how the emotion recognition works

4.2.4 Water Tank Level

The performance of the water tank's level management system was evaluated through a series of time-based plots, which illustrated the water level, pump activity, and tank cover status. The system's capacity to maintain water levels within safe parameters while responding to rain detection is evident, with seamless transitions between different states.

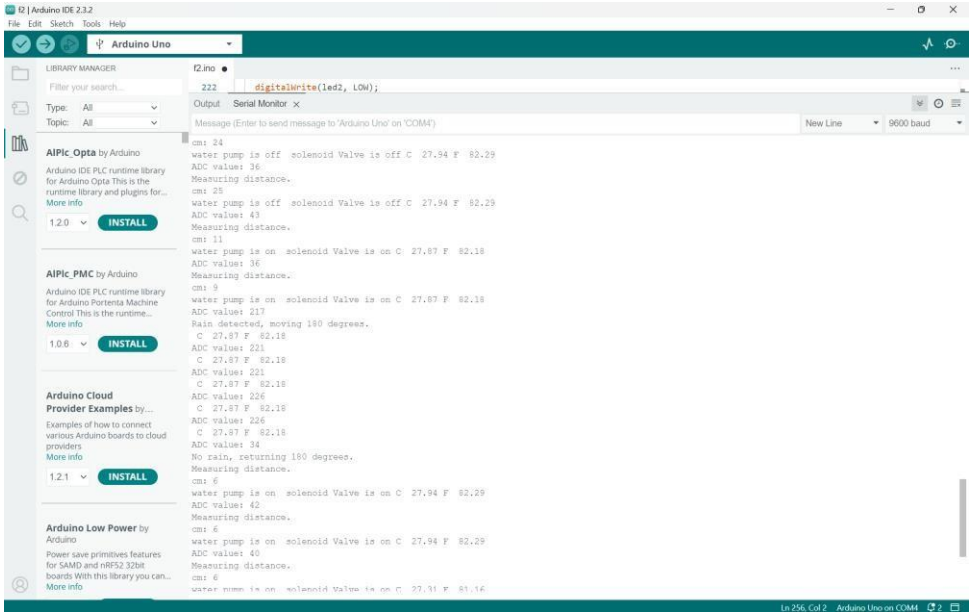


Figure 25: Example Serial Monitor Output of the water tank

Chapter Five: Discussion

5.1 Interpretation of Results

The results obtained from the system demonstrate notable advancements in the integration of multiple technologies to create a comprehensive smart home environment. The project was successfully completed, achieving its objectives of automating fan and lighting control through emotion recognition, gesture detection, and environmental monitoring.

- PID Control of Fan Speed: The effective tuning of the PID controller demonstrates the system's capacity to maintain desired temperature and humidity levels in a range of conditions. The capacity to customize these setpoints exemplifies the system's adaptability, enabling users to modify it to align with their particular requirements.
- Servo Motor and MPU6050 Integration: The successful resolution of the library conflict exemplifies the significance of meticulous resource management when integrating multiple subsystems. The result is a reliable system that is capable of simultaneously controlling both the direction and speed of the fan, thereby providing a responsive user experience.
- IR Receiver and Remote Control: The system's capacity to store and recall user-defined settings evinces its adaptability to user preferences. The eventual resolution of the integration issues highlights the significance of selecting appropriate libraries and optimizing hardware resource allocation.
- LED Control with Emotion and Motion Detection: The successful integration of facial and speech emotion recognition models with the LED system exemplifies the project's pioneering approach to enhancing user interaction. The system's capacity to alter lighting in accordance with discerned emotions, coupled with motion-activated initiation, effectively realizes the project's objective of developing a responsive and immersive environment.
- Water Tank Management System: The system's performance in managing the water tank is indicative of its capacity to efficiently undertake environmental monitoring tasks. Although minor delays were observed, they did not significantly impede functionality, thereby indicating that the system is sufficiently robust for practical applications.
- Hand Gesture Control for Fan: The incorporation of hand gesture control with the fan and other subsystems introduces an additional layer of intuitive control for the user. The seamless interaction between gesture detection and emotion recognition systems exemplifies the successful coordination of disparate input modalities, thereby enhancing the overall user experience.

Overall, the system met the project’s objectives, showing that it is capable of providing a comprehensive, user-friendly, and responsive smart home solution.

5.2 Challenges and Limitations

Despite the system's success, several challenges and limitations were encountered throughout the project:

- Hardware Conflicts: The integration of multiple components, particularly those

requiring specific timers and pins on the Arduino, presented considerable challenges. The necessity for careful management arose from conflicts between libraries and the limited resources of the Arduino Uno. Ultimately, the transition to an Arduino Mega was required to provide sufficient resources.

- Complexity of Emotion Recognition: Although the emotion recognition models demonstrated satisfactory performance, the accuracy, particularly in the recognition of subtle emotions or in varied lighting conditions, fell short of the desired level. This underscores the models' inherent limitation in generalizing across diverse environmental conditions and user expressions.
- Gesture Detection Accuracy: Although the hand gesture detection system demonstrated satisfactory performance, it was occasionally affected by variations in lighting and hand positioning. Further refinements to the accuracy of distance and angle calculations could be made to ensure even more reliable performance in diverse conditions.
- Real-Time Processing Delays: The incorporation of multiple subsystems resulted in minor delays in processing, particularly in real-time applications such as motion detection and the automated water tank system. Although these delays were minimal, they could prove problematic in scenarios requiring a more immediate response.
- User Experience and Feedback: Although the system was generally well-received by users, the complexity of managing multiple control inputs (e.g., remote, gestures, and
- automated controls) could prove overwhelming for some users. A reduction in the number of elements within the user interface and control mechanisms could result in an improvement in overall usability.
- Scalability: The existing system, while operational, is constrained in its capacity to accommodate expansion. The incorporation of additional sensors, actuators, or even the integration with other smart home ecosystems could present challenges that would require resolution in future iterations.

These challenges underscore potential avenues for enhancement in forthcoming iterations of the system. Mitigating these shortcomings would bolster the system's resilience, user experience, and scalability, thereby facilitating broader adoption and integration in smart home environments.

Chapter Six: Conclusion

6.1 Summary

The objective of this project was to develop an integrated smart home system that employs emotion recognition, gesture control, and environmental monitoring to enhance automation and user experience. The system was successfully integrated with various subsystems, including a fan control system based on proportional-integral-derivative (PID) theory, an emotion-based lighting system, and a water tank management system. The project yielded several noteworthy outcomes, including the successful integration of disparate components, the resolution of hardware conflicts, and the demonstration of the system's adaptability to user preferences. Despite some challenges, the project was ultimately successful in achieving its primary objectives, resulting in the creation of a functional, responsive, and user-friendly smart home environment.

6.2 Future Work

Future research and development could focus on several areas to further improve and expand the system's capabilities. These include the enhancement of the accuracy and robustness of emotion recognition models, particularly in varied lighting conditions, and the refinement of gesture detection for greater reliability. An expansion of the system's scalability to integrate with additional sensors and smart home platforms could facilitate a broader range of applications. Furthermore, simplifying the user interface and investigating alternative hardware solutions could enhance usability and facilitate broader accessibility.

References

- Ang, K. H., Chong, G., & Li, Y. (2005). PID control system analysis, design, and technology. IEEE Transactions on Control Systems Technology, 13(4), 559-576. https://doi.org/10.1109/TCST.2005.847331
- Barros, P., & Wermter, S. (2016). A deep architecture for emotional learning and recognition. Neurocomputing, 171, 32-38. https://doi.org/10.1016/j.neucom.2015.12.121
- Boyce, P. R., Veitch, J. A., & Newsham, G. R. (2003). Lighting quality and office work: A field simulation study. Lighting Research & Technology, 35(3), 161-176. https://doi.org/10.1191/1365782803li072oa
- Hori, M., Hori, M., & Nishida, K. (2007). Fan control system for air conditioning apparatus. US Patent No. 7,171,278. U.S. Patent and Trademark Office. https://patents.google.com/patent/US7171278B2
- Jin, Y. G., & Li, W. (2007). Influence of LED light color on physiological responses and subjective emotion. Journal of Environmental Psychology, 27(1), 77-85. https://doi.org/10.1016/j.jenvp.2007.01.009
- Knez, I. (2001). Effects of color of light on nonvisual psychological processes. Journal of Environmental Psychology, 21(2), 201-208. https://doi.org/10.1006/jevp.2000.0188
- Kwak, D., Kim, J., & Cho, D. (2015). A study on the environmental monitoring system using wireless sensor networks. Journal of Sensors, 2015. https://doi.org/10.1155/2015/580827
- Mitra, S., & Acharya, T. (2007). Gesture recognition: A survey. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 37(3), 311-324. https://doi.org/10.1109/TSMCC.2007.893280
- Partonen, T., & Lönnqvist, J. (2000). Bright light improves vitality and alleviates distress in healthy people. Journal of Affective Disorders, 57(1-3), 55-61. https://doi.org/10.1016/S0165-0327(99)00066-2
- Picard, R. W., Vyzas, E., & Healey, J. (2001). Toward machine emotional intelligence: Analysis of affective physiological state. IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(10), 1175-1191. https://doi.org/10.1109/34.954607
- Rajesh, M., & Soni, V. (2017). Integration of IR remote control systems with microcontroller-based devices. International Journal of Electrical and Computer Engineering

(IJECE), 7(3), 1299-1306. https://doi.org/10.11591/ijece.v7i3.pp1299-1306

- Wan, J., Zou, C., Ullah, S., & Li, D. (2014). Context-aware gesture recognition for smart homes. Computers & Electrical Engineering, 40(6), 1785-1798. https://doi.org/10.1016/j.compeleceng.2014.01.012
- Xiao, J., & Gertz, M. (2016). Implementing a position control system using the MPU6050 sensor and Arduino. IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), 690-695. https://doi.org/10.1109/AIM.2016.7576821
- Data Magic Lab. (n.d.). *Emotion Detection Using Convolutional Neural Network*. Retrieved from https://datamagiclab.com/emotion-detection-using-convolutional-neural- network/
- Kaggle. (n.d.). *Surrey Audio-Visual Expressed Emotion (SAVEE)*. Retrieved from https://www.kaggle.com/datasets/ejlok1/surrey-audiovisual-expressed-emotion-savee
- Kaggle. (n.d.). *CREMA-D Dataset*. Retrieved from https://www.kaggle.com/datasets/ejlok1/cremad
- Kaggle. (n.d.). *Toronto Emotional Speech Set (TESS)*. Retrieved from https://www.kaggle.com/datasets/ejlok1/toronto-emotional-speech-set-tess
- Kaggle. (n.d.). *RAVDESS Emotional Speech Audio Dataset*. Retrieved from https://www.kaggle.com/datasets/uwrfkaggler/ravdess-emotional-speech-audio
- Kaggle. (n.d.). *Speech Emotion Recognition by Shivam Burnwal*. Retrieved from https://www.kaggle.com/code/shivamburnwal/speech-emotion-recognition
- Kaggle. (n.d.). *FER 2013 Dataset*. Retrieved from https://www.kaggle.com/datasets/msambare/fer2013

Appendix

1. Arduino code

1.1. The Fan with Emotion code

```
#include <Wire.h>
#include <MPU6050.h>
#include <Servo.h>
#include <DHT.h>
#include <LiquidCrystal.h>
#include <IRremote.hpp>

#define RECEIVER_PIN 10 // Pin the receiver is plugged into
// Emotion-based light control and PIR setup
#define RED_PIN 5
#define GREEN_PIN 6
#define BLUE_PIN 7
#define PIR_PIN 38
#define DHTPIN A3
#define DHTTYPE DHT11
// Motor driver setup
#define EN1 2
#define IN1 34
#define IN2 36

// Servo and MPU6050 setup
Servo sg90;
int servo_pin = 3;
MPU6050 sensor;
int16_t ax, ay, az;
int16_t gx, gy, gz;

// LCD and DHT11 setup
LiquidCrystal lcd(32, 30, 28, 26, 24, 22);
DHT dht(DHTPIN, DHTTYPE);

// PID constants
float Kp = 40.0;
float Ki = 0.01;
float Kd = 1.0;

// PID variables
float setPointTemp = 25.0;
float setPointHumidity = 30.0;
float error, prevError = 0;
float integral = 0;
float derivative;
unsigned long lastTime = 0;
unsigned long sampleTime = 1000;  // Sample time in milliseconds
float integralLimit = 100;  // Reduced limit for integral wind-up

// User preference storage
struct Preference {
  float temperature;
  float humidity;
  int fanSpeed;
};

Preference preferences[10];  // Store up to 10 preferences
int preferenceCount = 0;

int speed = 0; // Declare speed as a global variable
unsigned long handDetectedTime = 0;
bool handControlMode = false; // State variable for hand control mode

int pirValue; // Place to store read PIR Value
unsigned long lastMotionTime = 0; // Time of the last detected motion
```

```
63. const unsigned long motionTimeout = 5000; // 5-second timeout
64. bool motionDetected = false;
65. char currentEmotion = '7'; // Default to 'Mismatch' or white color
66.
67. void setup() {
68.   Serial.begin(9600);
69.   pinMode(RED_PIN, OUTPUT);
70.   pinMode(GREEN_PIN, OUTPUT);
71.   pinMode(BLUE_PIN, OUTPUT);
72.   pinMode(PIR_PIN, INPUT);
73.   setColor(0, 0, 0); // Initialize with LED off
74.   Serial.println("Setup complete");
75.
76.   // Setup for Servo and MPU6050
77.   sg90.attach(servo_pin);
78.   Wire.begin();
79.   Serial.println("Initializing the sensor");
80.   sensor.initialize();
81.   Serial.println(sensor.testConnection() ? "Successfully Connected" : "Connection failed");
82.   delay(1000);
83.   Serial.println("Taking Values from the sensor");
84.   delay(1000);
85.
86.   // Setup for LCD and DHT11
87.   lcd.begin(16, 2);
88.   lcd.print("Initializing...");
89.   dht.begin();
90.
91.   // Setup for motor driver
92.   pinMode(EN1, OUTPUT);
93.   pinMode(IN1, OUTPUT);
94.   pinMode(IN2, OUTPUT);
95.
96.   // Setup for IR receiver
97.   IrReceiver.begin(RECEIVER_PIN, ENABLE_LED_FEEDBACK);
98.
99.   // Wait for the sensor to stabilize
100.  delay(2000);
101.  lcd.clear();
102. }
103.
104. void loop() {
105.   pirValue = digitalRead(PIR_PIN);
106.
107.   if (pirValue == HIGH) {
108.     lastMotionTime = millis();
109.     if (!motionDetected) {
110.       Serial.println("Motion detected");
111.       setColor(255, 255, 255); // Set LED to white when motion is first detected
112.       Serial.println("MOTION"); // Send motion signal to Python
113.       motionDetected = true;
114.     }
115.   }
116.
117.   if (Serial.available() > 0) {
118.     String receivedEmotion = Serial.readStringUntil('\n');
119.     receivedEmotion.trim(); // Trim any whitespace or newline characters
120.     Serial.print("Received: ");
121.     Serial.println(receivedEmotion);
122.     setColorBasedOnEmotion(receivedEmotion); // Set LED color based on the received emotion
123.   }
124.
125.   if (millis() - lastMotionTime > motionTimeout) {
126.     setColor(0, 0, 0); // Turn off the LED if no motion detected for 5 seconds
127.     motionDetected = false;
128.     // Set the fan speed to 100 if no motion is detected
129.     speed = 100;
130.     analogWrite(EN1, speed);
131.     digitalWrite(IN1, HIGH);
132.     digitalWrite(IN2, LOW);
```

```
    lcd.setCursor(9, 1);  // Update LCD with the adjusted speed
    lcd.print("Fan:");
    lcd.print(speed);
  }

  // Check for hand detection message
  if (Serial.available() > 0) {
    String data = Serial.readStringUntil('\n');
    int delimiterIndex = data.indexOf(',');
    if (delimiterIndex != -1) {
      int distance = data.substring(0, delimiterIndex).toInt();
      int angle = data.substring(delimiterIndex + 1).toInt();

      if (distance < 0) {
        handControlMode = false;
        handDetectedTime = millis();  // Record the time when the hand was last detected
      } else {
        handControlMode = true;
        handDetectedTime = millis();  // Reset the hand detection time

        // Control fan speed with L293D directly from distance
        int fanSpeed = (distance > 255) ? 255 : distance;
        analogWrite(EN1, fanSpeed);
        digitalWrite(IN1, HIGH);
        digitalWrite(IN2, LOW);

        // Control servo motor
        int servoAngle = map(angle, -180, 0, 0, 180); // Map -180 to 0 to 0 to 180
        sg90.write(servoAngle);

        // Update the LCD with the new speed
        lcd.setCursor(9, 1);  // Move the cursor to the appropriate position
        lcd.print("Fan:");
        lcd.print(fanSpeed);
      }
    }
  }

  // Switch to PID control if hand not detected for more than 5 seconds
  if (millis() - handDetectedTime > 5000) {
    handControlMode = false;
  }
    // Read humidity and temperature from DHT11
    float h = dht.readHumidity();
    float t = dht.readTemperature();
  // PID control mode
  if (!handControlMode && motionDetected) {
    // MPU6050 reading and servo control
    sensor.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
    int servoPosition = map(ax, -17000, 17000, 0, 180); // Adjusted for Servo angle
    sg90.write(servoPosition);
    Serial.println(servoPosition);
    delay(200);

    // Read humidity and temperature from DHT11
    float h = dht.readHumidity();
    float t = dht.readTemperature();

    // Check if any reads failed and exit early (to try again)
    if (isnan(h) || isnan(t)) {
      lcd.setCursor(0, 0);
      lcd.print("Failed to read");
      lcd.setCursor(0, 1);
      lcd.print("from sensor!");
      return;
    }

    // Get the current time
    unsigned long now = millis();
    if (now - lastTime >= sampleTime) {
```

```
203.       // Calculate temperature and humidity errors
204.       float tempError = t - setPointTemp;
205.       float humidityError = h - setPointHumidity;
206.       error = tempError * 0.8 + humidityError * 0.2;
207.
208.       // Calculate integral with wind-up guard
209.       integral += error * (now - lastTime);
210.       integral = constrain(integral, -integralLimit, integralLimit);
211.
212.       // Calculate derivative
213.       derivative = (error - prevError) / (now - lastTime);
214.
215.       // Compute PID output
216.       float output = Kp * error + Ki * integral + Kd * derivative;
217.
218.       // Constrain the PID output to PWM range (0-255)
219.       speed = constrain(output, 0, 255);
220.
221.       // Ensure minimum fan speed
222.       if (speed < 50 && (tempError > 0 || humidityError > 0)) {
223.         speed = 50;
224.       }
225.
226.       // Debug output
227.       Serial.print("Temperature: ");
228.       Serial.print(t);
229.       Serial.print(" C, Humidity: ");
230.       Serial.print(h);
231.       Serial.print(" %, Error: ");
232.       Serial.print(error);
233.       Serial.print(", Integral: ");
234.       Serial.print(integral);
235.       Serial.print(", Derivative: ");
236.       Serial.print(derivative);
237.       Serial.print(", Output: ");
238.       Serial.print(output);
239.       Serial.print(", Speed: ");
240.       Serial.println(speed);
241.
242.       // Update previous error and last time
243.       prevError = error;
244.       lastTime = now;
245.
246.       // Control the motor
247.       analogWrite(EN1, speed);
248.       digitalWrite(IN1, HIGH);
249.       digitalWrite(IN2, LOW);
250.
251.       // Print temperature, humidity, and fan speed to the LCD
252.       lcd.setCursor(0, 0);
253.       lcd.print("Temp: ");
254.       lcd.print(t);
255.       lcd.print(" C");
256.
257.       lcd.setCursor(0, 1);
258.       lcd.print("Hum: ");
259.       lcd.print((int)h);  // Cast humidity to an integer
260.       lcd.print("%");
261.
262.       lcd.setCursor(9, 1);  // Move the cursor to a new position to avoid overwriting
263.       lcd.print("Fan:");
264.       lcd.print(speed);
265.     }
266.
267.     // Check if current conditions match any stored preferences
268.     bool preferenceApplied = false;
269.     for (int i = 0; i < preferenceCount; i++) {
270.       if (abs(preferences[i].temperature - t) < 1 && abs(preferences[i].humidity - h) < 5) {
271.         speed = preferences[i].fanSpeed;
272.         analogWrite(EN1, speed);
```

```
273.         Serial.print("Applying saved preference: ");
274.         Serial.println(speed);
275.         lcd.setCursor(9, 1);  // Update LCD with the adjusted speed
276.         lcd.print("Fan:");
277.         lcd.print(speed);
278.         preferenceApplied = true;
279.         break;
280.       }
281.     }
282.
283.     if (!preferenceApplied) {
284.       // If no preference is applied, continue with the PID control
285.       analogWrite(EN1, speed);
286.       lcd.setCursor(9, 1);  // Update LCD with the PID controlled speed
287.       lcd.print("Fan:");
288.       lcd.print(speed);
289.     }
290.
291.     // IR remote handling
292.     if (IrReceiver.decode()) {
293.       int fanSpeedAdjustment = 0;
294.       uint16_t command = IrReceiver.decodedIRData.command;
295.       static uint16_t lastCommand = 0; // Store the last non-repeat command
296.
297.       if (command == 0xFFFFFFFF) { // Repeat code
298.         command = lastCommand; // Use the last non-repeat command
299.       } else {
300.         lastCommand = command; // Update the last non-repeat command
301.       }
302.
303.       switch (command) {
304.         case 69: // ON/OFF
305.           fanSpeedAdjustment = 0;
306.           break;
307.         case 70: // +VOL
308.           fanSpeedAdjustment = 10;
309.           break;
310.         case 21: // -VOL
311.           fanSpeedAdjustment = -10;
312.           break;
313.         case 9: // UP
314.           fanSpeedAdjustment = 20;
315.           break;
316.         case 7: // DOWN
317.           fanSpeedAdjustment = -20;
318.           break;
319.         case 13: // ST/REPT - Save current preferences
320.           preferenceCount = 0;
321.           Serial.println("User preferences reset.");
322.           break;
323.       }
324.
325.       if (fanSpeedAdjustment != 0) {
326.         speed = constrain(speed + fanSpeedAdjustment, 0, 255);
327.         analogWrite(EN1, speed);
328.         lcd.setCursor(9, 1);  // Update LCD with the adjusted speed
329.         lcd.print("Fan:");
330.         lcd.print(speed);
331.
332.         // Save the updated speed to preferences if within tolerance range
333.         bool preferenceFound = false;
334.         for (int i = 0; i < preferenceCount; i++) {
335.           if (abs(preferences[i].temperature - t) < 1 && abs(preferences[i].humidity - h) < 5) {
336.             preferences[i].fanSpeed = speed;
337.             preferenceFound = true;
338.             break;
339.           }
340.         }
341.
342.         if (!preferenceFound && preferenceCount < 10) {
```

```
343.           preferences[preferenceCount].temperature = t;
344.           preferences[preferenceCount].humidity = h;
345.           preferences[preferenceCount].fanSpeed = speed;
346.           preferenceCount++;
347.         }
348.       }
349.       IrReceiver.resume();
350.     }
351.   }
352. }
353.
354. void setColor(int red, int green, int blue) {
355.   analogWrite(RED_PIN, red);
356.   analogWrite(GREEN_PIN, green);
357.   analogWrite(BLUE_PIN, blue);
358. }
359.
360. void setColorBasedOnEmotion(String emotion) {
361.   if (emotion == "angry") {
362.     setColor(50, 50, 255); // Blue
363.   } else if (emotion == "disgusted") {
364.     setColor(64, 224, 208); // Turquoise
365.   } else if (emotion == "fear") {
366.     setColor(70, 255, 70); // Light green
367.   } else if (emotion == "happy") {
368.     setColor(255, 50, 50); // Pink
369.   } else if (emotion == "neutral") {
370.     setColor(192, 192, 192); // Light Gray
371.   } else if (emotion == "sad") {
372.     setColor(255, 255, 30); // Orange
373.   } else if (emotion == "surprised") {
374.     setColor(128, 40, 128); // Pink
375.   } else if (emotion == "Mismatch") {
376.     setColor(255, 255, 255); // White
377.   } else {
378.     setColor(255, 255, 255); // Default to White
379.   }
380. }
381.
```

1.2. Water Tank code

```
 1. #define STEPPER_PIN_1 2
 2. #define STEPPER_PIN_2 3
 3. #define STEPPER_PIN_3 4
 4. #define STEPPER_PIN_4 5
 5. #define RAIN_SENSOR_PIN A0
 6. #define THRESHOLD 120
 7.
 8. int step_number = 0;
 9. bool isRaining = false;
10.
11. // Define the pins for the second code
12. #define trigpin 7
13. #define echopin 6
14. #define buzzer 8  // Define the buzzer pin
15.
16. int led1 = A1;
17. int led2 = A2;
18. int led3 = A3;
19. int led4 = A4;
20. int led5 = A5;
21.
22. int solenoidValvePin = 9;  // Pin connected to the solenoid valve
23. int pumpPin = 10;          // Pin connected to the electric pump
24. #include <OneWire.h>
25. #include <DallasTemperature.h>
26.
```

```
#define ONE_WIRE_BUS 12

OneWire oneWire(ONE_WIRE_BUS);

DallasTemperature sensors(&oneWire);

 float Celcius=0;
 float Fahrenheit=0;

void setup() {
  // Setup for stepper motor and rain sensor
  pinMode(STEPPER_PIN_1, OUTPUT);
  pinMode(STEPPER_PIN_2, OUTPUT);
  pinMode(STEPPER_PIN_3, OUTPUT);
  pinMode(STEPPER_PIN_4, OUTPUT);

  // Setup for the second code
  Serial.begin(9600);
  pinMode(trigpin, OUTPUT);
  pinMode(echopin, INPUT);
  pinMode(buzzer, OUTPUT);  // Set the buzzer pin as an output
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);
  pinMode(led5, OUTPUT);
  pinMode(solenoidValvePin, OUTPUT);  // Set the solenoid valve pin as an output
  pinMode(pumpPin, OUTPUT);           // Set the pump pin as an output

  // Initialize all LEDs, the solenoid valve, the pump, and the buzzer to be off
  digitalWrite(led1, LOW);
  digitalWrite(led2, LOW);
  digitalWrite(led3, LOW);
  digitalWrite(led4, LOW);
  digitalWrite(led5, LOW);
  digitalWrite(buzzer, LOW);
  digitalWrite(solenoidValvePin, LOW);
  digitalWrite(pumpPin, LOW);
  sensors.begin();
}

void loop() {
  // Read the rain sensor value
  int rainValue = analogRead(RAIN_SENSOR_PIN);
  Serial.print("ADC value: ");
  Serial.println(rainValue);

  if (rainValue > THRESHOLD && !isRaining) {
    // Rain detected and motor is not yet moved
    moveStepper(1000, true); // Move forward 180 degrees
    isRaining = true;
    Serial.println("Rain detected, moving 180 degrees.");
    digitalWrite(solenoidValvePin, LOW);  // Turn off solenoid valve
    digitalWrite(pumpPin, LOW);           // Turn off pump
  } else if (rainValue <= THRESHOLD && isRaining) {
    // No rain detected and motor is in the moved position
    moveStepper(1000, false); // Move backward 180 degrees
    isRaining = false;
    Serial.println("No rain, returning 180 degrees.");
  }

  // Only run the second code if there is no rain
  if (!isRaining) {
    int duration, distance;
    Serial.println("Measuring distance.");
    // Send a pulse from the trigger pin
    digitalWrite(trigpin, HIGH);
    delayMicroseconds(10);  // Reduced to 10 microseconds for the pulse duration
    digitalWrite(trigpin, LOW);

```

```
    // Measure the duration of the echo pulse
    duration = pulseIn(echopin, HIGH);

    // Calculate the distance in centimeters
    distance = (duration / 2) / 29.1;

    // Print the distance to the serial monitor
    Serial.print("cm: ");
    Serial.println(distance);

    // Control LEDs based on the measured distance
    controlLEDs(distance);

    // Control the buzzer based on the measured distance
    controlBuzzer(distance);

    // Control the solenoid valve and pump based on the measured distance
    controlSolenoidAndPump(distance);
  }
  sensors.requestTemperatures();
  Celcius=sensors.getTempCByIndex(0);
  Fahrenheit=sensors.toFahrenheit(Celcius);
  Serial.print(" C  ");
  Serial.print(Celcius);
  Serial.print(" F  ");
  Serial.println(Fahrenheit);
  delay(1000); // Delay for a while before reading again
}

void moveStepper(int steps, bool dir) {
  for (int a = 0; a < steps; a++) {
    OneStep(dir);
    delay(2);
  }
}

void OneStep(bool dir) {
  if (dir) {
    switch (step_number) {
      case 0:
        digitalWrite(STEPPER_PIN_1, HIGH);
        digitalWrite(STEPPER_PIN_2, LOW);
        digitalWrite(STEPPER_PIN_3, LOW);
        digitalWrite(STEPPER_PIN_4, LOW);
        break;
      case 1:
        digitalWrite(STEPPER_PIN_1, LOW);
        digitalWrite(STEPPER_PIN_2, HIGH);
        digitalWrite(STEPPER_PIN_3, LOW);
        digitalWrite(STEPPER_PIN_4, LOW);
        break;
      case 2:
        digitalWrite(STEPPER_PIN_1, LOW);
        digitalWrite(STEPPER_PIN_2, LOW);
        digitalWrite(STEPPER_PIN_3, HIGH);
        digitalWrite(STEPPER_PIN_4, LOW);
        break;
      case 3:
        digitalWrite(STEPPER_PIN_1, LOW);
        digitalWrite(STEPPER_PIN_2, LOW);
        digitalWrite(STEPPER_PIN_3, LOW);
        digitalWrite(STEPPER_PIN_4, HIGH);
        break;
    }
  } else {
    switch (step_number) {
      case 0:
        digitalWrite(STEPPER_PIN_1, LOW);
        digitalWrite(STEPPER_PIN_2, LOW);
        digitalWrite(STEPPER_PIN_3, LOW);
```

```
        digitalWrite(STEPPER_PIN_4, HIGH);
        break;
      case 1:
        digitalWrite(STEPPER_PIN_1, LOW);
        digitalWrite(STEPPER_PIN_2, LOW);
        digitalWrite(STEPPER_PIN_3, HIGH);
        digitalWrite(STEPPER_PIN_4, LOW);
        break;
      case 2:
        digitalWrite(STEPPER_PIN_1, LOW);
        digitalWrite(STEPPER_PIN_2, HIGH);
        digitalWrite(STEPPER_PIN_3, LOW);
        digitalWrite(STEPPER_PIN_4, LOW);
        break;
      case 3:
        digitalWrite(STEPPER_PIN_1, HIGH);
        digitalWrite(STEPPER_PIN_2, LOW);
        digitalWrite(STEPPER_PIN_3, LOW);
        digitalWrite(STEPPER_PIN_4, LOW);
        break;
    }
  }
  step_number++;
  if (step_number > 3) {
    step_number = 0;
  }
}

void controlLEDs(int distance) {
  if (distance <= 3) {
    digitalWrite(led1, HIGH);
    digitalWrite(led2, HIGH);
    digitalWrite(led3, HIGH);
    digitalWrite(led4, HIGH);
    digitalWrite(led5, HIGH);
  } else if (distance > 3 && distance <= 6) {
    digitalWrite(led1, LOW);
    digitalWrite(led2, HIGH);
    digitalWrite(led3, HIGH);
    digitalWrite(led4, HIGH);
    digitalWrite(led5, HIGH);
  } else if (distance > 6 && distance <= 9) {
    digitalWrite(led1, LOW);
    digitalWrite(led2, LOW);
    digitalWrite(led3, HIGH);
    digitalWrite(led4, HIGH);
    digitalWrite(led5, HIGH);
  } else if (distance > 9 && distance <= 12) {
    digitalWrite(led1, LOW);
    digitalWrite(led2, LOW);
    digitalWrite(led3, LOW);
    digitalWrite(led4, HIGH);
    digitalWrite(led5, HIGH);
  } else if (distance > 12 && distance <= 14) {
    digitalWrite(led1, LOW);
    digitalWrite(led2, LOW);
    digitalWrite(led3, LOW);
    digitalWrite(led4, LOW);
    digitalWrite(led5, HIGH);
  } else if (distance >= 15) {
    digitalWrite(led1, LOW);
    digitalWrite(led2, LOW);
    digitalWrite(led3, LOW);
    digitalWrite(led4, LOW);
    digitalWrite(led5, LOW);
  }
}

void controlBuzzer(int distance) {
  if (distance <= 2  || distance >= 13) {
```

```
    digitalWrite(buzzer, HIGH);  // Turn on the buzzer
  } else {
    digitalWrite(buzzer, LOW);   // Turn off the buzzer
  }
}

void controlSolenoidAndPump(int distance) {
  if (distance > 3 && distance <= 14) {  // Tank is low
    digitalWrite(pumpPin, HIGH);
    Serial.print("water pump is on  ");
    delay(5000);
    digitalWrite(solenoidValvePin, HIGH);  // Open solenoid valve
    Serial.print("solenoid Valve is on");
  } else {  // Tank is within acceptable range
    digitalWrite(solenoidValvePin, LOW);  // Close solenoid valve
    digitalWrite(pumpPin, LOW);           // Turn off the pump
    Serial.print("water pump is off  ");
    Serial.print("solenoid Valve is off");
  }
}

```

2. Facial emotion recognition code

2.2 training code

```
import cv2
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Flatten
from keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_data_gen = ImageDataGenerator(rescale=1./255)
validation_data_gen = ImageDataGenerator(rescale=1./255)

train_generator = train_data_gen.flow_from_directory(
        'data/train',
        target_size=(48, 48),
        batch_size=64,
        color_mode="grayscale",
        class_mode='categorical')

validation_generator = validation_data_gen.flow_from_directory(
        'data/test',
        target_size=(48, 48),
        batch_size=64,
        color_mode="grayscale",
        class_mode='categorical')

emotion_model = Sequential()

emotion_model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(48, 48, 1)))
emotion_model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))
```

```
30.
31. emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
32. emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
33. emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
34. emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
35. emotion_model.add(Dropout(0.25))
36.
37. emotion_model.add(Flatten())
38. emotion_model.add(Dense(1024, activation='relu'))
39. emotion_model.add(Dropout(0.5))
40. emotion_model.add(Dense(7, activation='softmax'))
41.
42. cv2.ocl.setUseOpenCL(False)
43.
44.
45. emotion_model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.0001),
metrics=['accuracy'])
46.
47. emotion_model_info = emotion_model.fit(
48.         train_generator,
49.         steps_per_epoch=28709 // 64,
50.         epochs=150,
51.         validation_data=validation_generator,
52.         validation_steps=7178 // 64)
53.
54. model_json = emotion_model.to_json()
55. with open("emotion_model.json", "w") as json_file:
56.     json_file.write(model_json)
57.
58. emotion_model.save_weights('emotion_model.weights.h5')
59.
```

2.2 testing code

```
 1. import cv2
 2. import numpy as np
 3. from keras.models import model_from_json
 4.
 5. emotion_dict = {0: "Angry", 1: "Disgusted", 2: "Fearful", 3: "Happy", 4: "Neutral", 5: "Sad", 6:
"Surprised"}
 6. json_file = open('emotion_model.json', 'r')
 7. loaded_model_json = json_file.read()
 8. json_file.close()
 9. emotion_model = model_from_json(loaded_model_json)
10.
11. emotion_model.load_weights("emotion_model.weights.h5")
12. print("Loaded model from disk")
13.
14. cap = cv2.VideoCapture(0)
15. #cap = cv2.VideoCapture("C:\\Users\\halaj\\Downloads\\pexels-gabby-k-5273028 (2160p).mp4")
16.
17. while True:
18.     ret, frame = cap.read()
19.     frame = cv2.resize(frame, (1280, 720))
20.     if not ret:
21.         break
22.     face_detector = cv2.CascadeClassifier(r"C:\Users\halaj\OneDrive\Desktop\emotion
detection\haarcascades\haarcascade_frontalface_default.xml")
23.     gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
24.     num_faces = face_detector.detectMultiScale(gray_frame, scaleFactor=1.3, minNeighbors=5)
25.     for (x, y, w, h) in num_faces:
26.         cv2.rectangle(frame, (x, y-50), (x+w, y+h+10), (0, 255, 0), 4)
27.         roi_gray_frame = gray_frame[y:y + h, x:x + w]
```

```
28.         cropped_img = np.expand_dims(np.expand_dims(cv2.resize(roi_gray_frame, (48, 48)), -1),
0)
29.         emotion_prediction = emotion_model.predict(cropped_img)
30.         maxindex = int(np.argmax(emotion_prediction))
31.         cv2.putText(frame, emotion_dict[maxindex], (x+5, y-20), cv2.FONT_HERSHEY_SIMPLEX, 1,
(255, 0, 0), 2, cv2.LINE_AA)
32.
33.     cv2.imshow('Emotion Detection', frame)
34.     if cv2.waitKey(1) & 0xFF == ord('q'):
35.         break
36.
37. cap.release()
38. cv2.destroyAllWindows()
39.
```

2.3 evaluation code

```
 1. import cv2
 2. import numpy as np
 3. from tensorflow.keras.models import model_from_json
 4. import matplotlib.pyplot as plt
 5. from tensorflow.keras.preprocessing.image import ImageDataGenerator
 6. from sklearn.metrics import confusion_matrix, classification_report, ConfusionMatrixDisplay
 7.
 8. emotion_dict = {0: "Angry", 1: "Disgusted", 2: "Fearful", 3: "Happy", 4: "Neutral", 5: "Sad", 6:
"Surprised"}
 9. json_file = open('emotion_model.json', 'r')
10. loaded_model_json = json_file.read()
11. json_file.close()
12. emotion_model = model_from_json(loaded_model_json)
13. emotion_model.load_weights("emotion_model.weights.h5")
14. print("Loaded model from disk")
15. test_data_gen = ImageDataGenerator(rescale=1./255)
16.
17. test_generator = test_data_gen.flow_from_directory(
18.     'data/test',
19.     target_size=(48, 48),
20.     batch_size=64,
21.     color_mode="grayscale",
22.     class_mode='categorical',
23.     shuffle=False
24. )
25. predictions = emotion_model.predict(test_generator, steps=test_generator.samples //
test_generator.batch_size + 1)
26.
27. true_labels = test_generator.classes
28. predicted_labels = np.argmax(predictions, axis=1)
29. print("Confusion Matrix for Emotion Recognition")
30. c_matrix = confusion_matrix(true_labels, predicted_labels)
31. print(c_matrix)
32. cm_display = ConfusionMatrixDisplay(confusion_matrix=c_matrix, display_labels=[emotion_dict[i]
for i in range(7)])
33. cm_display.plot(cmap=plt.cm.Blues)
34. plt.show()
35. print("Classification Report for Emotion Recognition")
36. print(classification_report(true_labels, predicted_labels, target_names=[emotion_dict[i] for i
in range(7)]))
```

3. speech emotion recognition codes

```
import pandas as pd
import numpy as np
import tensorflow as tf
import os,time,librosa,warnings,glob
import regex as re
from sklearn.metrics import confusion_matrix,classification_report
import librosa.display
from sklearn.preprocessing import MinMaxScaler,OneHotEncoder
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.layers import Dense,Input,Add,Flatten,Dropout,Activation,AveragePooling1D,Conv1D
from keras.models import Model,Sequential,load_model
from keras.optimizers import Adam
from keras.callbacks import
LearningRateScheduler,EarlyStopping,ReduceLROnPlateau,ModelCheckpoint
from base64 import b64decode
from IPython.display import Audio,HTML
from scipy.io.wavfile import read as wav_read
import io
import ffmpeg
warnings.filterwarnings("ignore")

```

```
#decorator function for calculating the total time reqired to execute various function
def calc_time(func):
  def inner(*args, **kwargs):
    st = time.time()
    result = func(*args,**kwargs)
    end = time.time()-st
    print("Total time required: {:.3f} ms".format(end * 1000))
    return result
  return inner

#function for getting ravdess dataset details and labeling
def ravdess_data():
  #directory of the audio dataset
  ravdess = "C:/Users/halaj/OneDrive/Desktop/data/Ravdess/"
  #label ravdess data
  emotion_ravdess =
{'01':'neutral','02':'calm','03':'happy','04':'sad','05':'angry','06':'fearful','07':'disgust','08':
'surprised'}
  #list to store ravdess emotion
  ravdess_emotion = []
  #list to store ravdess audio path
  ravdess_path = []
  #get subfolders from the path
  ravdess_folder = os.listdir(ravdess)
  for i in ravdess_folder:
    inner_files = os.listdir(ravdess+i+'/')
    for j in inner_files:
      #get the split part which contains the emotion information then append it into lists
      emotion = j.split('-')[2]
      ravdess_path.append(ravdess+i+'/'+j)
      ravdess_emotion.append(emotion_ravdess[emotion])

  #convert to dataframe
  df_ravdess = pd.DataFrame([ravdess_path,ravdess_emotion]).T
  df_ravdess.columns = ["AudioPath","Label"]
  print("length of ravdess dataset",len(df_ravdess))

  return df_ravdess

```

```
#function for getting crema dataset details and labeling
def crema_data():
  #directory of the audio dataset
  crema = "C:/Users/halaj/OneDrive/Desktop/data/CREMA-D/AudioWAV/"
  #label ravdess data
  emotion_crema =
{'SAD':'sad','ANG':'angry','DIS':'disgust','FEA':'fear','HAP':'happy','NEU':'neutral'}
  #list to store crema emotion
  crema_emotion = []
  #list to store crema audio path
  crema_path = []
  #get crema files in directory
  crema_files = os.listdir(crema)
  for i in crema_files:
    emotion = i.split('_')[2]
    crema_emotion.append(emotion_crema[emotion])
    crema_path.append(crema+i)

  #convert to dataframe
  df_crema = pd.DataFrame([crema_path,crema_emotion]).T
  df_crema.columns = ["AudioPath","Label"]
  print("length of crema dataset",len(df_crema))

  return df_crema

#function for getting tess dataset and labeling
def tess_data():
    #directory of the audio dataset
    tess ="C:/Users/halaj/OneDrive/Desktop/data/TESS Toronto emotional speech set data/"
    tess_emotion = []
    tess_path = []
    tess_folder = os.listdir(tess)
    for i in tess_folder:
        emotion_parts = i.split('_', 1)
        if len(emotion_parts) < 2:
            print("Skipping folder:", i)
            continue
        emotion = emotion_parts[1]
        inner_files = os.listdir(tess + i + '/')
        for j in inner_files:
            tess_path.append(tess + i + '/' + j)
            tess_emotion.append(emotion)

    #convert to dataframe
    df_tess = pd.DataFrame([tess_path,tess_emotion]).T
    df_tess.columns = ["AudioPath","Label"]
    print("length of tess dataset",len(df_tess))
    return df_tess

#function to get savee dataset and labeling
def saveee_data():
  #directory of the audio dataset
  savee = "C:/Users/halaj/OneDrive/Desktop/data/Savee/ALL/"
  emotion_savee =
{'a':'anger','d':'disgust','f':'fear','h':'happiness','n':'neutral','sa':'sadness','su':'surprise'}
  savee_emotion = []
  savee_path = []
  savee_files = os.listdir(savee)
  for i in savee_files:
    emotion = i.split('_')[1]
    emotion = re.match(r"([a-z]+)([0-9]+)",emotion)[1]
    savee_emotion.append(emotion_savee[emotion])
    savee_path.append(savee+i)

  #convert to dataframe
```

```
    df_savee = pd.DataFrame([savee_path,savee_emotion]).T
    df_savee.columns = ["AudioPath","Label"]
    print("length of savee dataset",len(df_savee))

    return df_savee

@calc_time
def fetch_data():
    #get ravdess data
    df_ravdess = ravdess_data()
    #get crema data
    df_crema = crema_data()
    #get tess data
    df_tess = tess_data()
    #get savee data
    df_savee = saveee_data()
    #combine all four dataset into one single dataset and create a dataframe
    frames = [df_ravdess,df_crema,df_tess,df_savee]
    final_combined = pd.concat(frames)
    final_combined.reset_index(drop=True,inplace=True)
    #save the information of datasets with their path and labels into a csv file

final_combined.to_csv('C:/Users/halaj/OneDrive/Desktop/data/preprocesseddata.csv',index=False,header
=True)
    print("Total length of the dataset is {}".format(len(final_combined)))
    return final_combined

#below are four data agumentation functions for noise, stretch, shift, pitch
#function to add noise to audio
def noise(data):
    noise_amp = 0.035*np.random.uniform()*np.amax(data)
    data = data + noise_amp*np.random.normal(size=data.shape[0])
    return data

#fuction to strech audio
def stretch(data, rate=0.8):
    return librosa.effects.time_stretch(data, rate=rate)


#fucntion to shift audio range
def shift(data):
    shift_range = int(np.random.uniform(low=-5, high = 5)*1000)
    return np.roll(data, shift_range)

#function to change pitch
def pitch(data, sampling_rate, pitch_factor=0.7):
    return librosa.effects.pitch_shift(data, sr=sampling_rate, n_steps=pitch_factor)

returned as output
def extract_features(data,sample_rate):

    #zero crossing rate
    result = np.array([])
    zcr = np.mean(librosa.feature.zero_crossing_rate(y=data).T, axis=0)
    result = np.hstack((result, zcr))
    #print('zcr',result.shape)

    #chroma shift
    stft = np.abs(librosa.stft(data))
    chroma_stft = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T, axis=0)
    result = np.hstack((result, chroma_stft))
    #print('chroma',result.shape)

    #mfcc
    mfcc = np.mean(librosa.feature.mfcc(y=data, sr=sample_rate).T, axis=0)
```

```
164.    result = np.hstack((result, mfcc))
165.    #print('mfcc',result.shape)
166.
167.    #rmse
168.    rms = np.mean(librosa.feature.rms(y=data).T, axis=0)
169.    result = np.hstack((result, rms))
170.    #print('rmse',result.shape)
171.
172.    #melspectogram
173.    mel = np.mean(librosa.feature.melspectrogram(y=data, sr=sample_rate).T, axis=0)
174.    result = np.hstack((result, mel))
175.    #print('mel',result.shape)
176.
177.    #rollof
178.    rollof = np.mean(librosa.feature.spectral_rolloff(y=data, sr=sample_rate).T, axis=0)
179.    result = np.hstack((result, rollof))
180.    #print('rollof',result.shape)
181.
182.    #centroids
183.    centroid = np.mean(librosa.feature.spectral_centroid(y=data, sr=sample_rate).T, axis=0)
184.    result = np.hstack((result, centroid))
185.    #print('centroids',result.shape)
186.
187.    #contrast
188.    contrast = np.mean(librosa.feature.spectral_contrast(y=data, sr=sample_rate).T, axis=0)
189.    result = np.hstack((result, contrast))
190.    #print('contrast',result.shape)
191.
192.    #bandwidth
193.    bandwidth = np.mean(librosa.feature.spectral_bandwidth(y=data, sr=sample_rate).T, axis=0)
194.    result = np.hstack((result, bandwidth))
195.    #print('bandwidth',result.shape)
196.
197.    #tonnetz
198.    tonnetz = np.mean(librosa.feature.tonnetz(y=data, sr=sample_rate).T, axis=0)
199.    result = np.hstack((result, tonnetz))
200.    #print('tonnetz',result.shape)
201.
202.    return result
203.
204. #function is used to get all augmented plus original features for given audio file
205. def get_features(path):
206.    #set the duration and offset
207.    #librosa.load takes audio file converts to array and returns array of audio file with its
sampling rate
208.    data, sample_rate = librosa.load(path, duration=2.5, offset=0.6)
209.
210.    #get audio features without augmentation
211.    res1 = extract_features(data,sample_rate)
212.    result = np.array(res1)
213.
214.    #get audio features with noise
215.    noise_data = noise(data)
216.    res2 = extract_features(noise_data,sample_rate)
217.    result = np.vstack((result, res2))
218.
219.    #get audio features with stretching and pitching
220.    new_data = stretch(data)
221.    data_stretch_pitch = pitch(new_data, sample_rate)
222.    res3 = extract_features(data_stretch_pitch,sample_rate)
223.    result = np.vstack((result, res3))
224.
225.    return result
```

```python
@calc_time
def Audio_features_extract():
  #this function is used to fetch the data from all the four datasets
  df = fetch_data()
  #count is used to keep a check of number of files processed
  count = 0
  #list to store audio features and their label information
  X_data, Y_label = [], []
  #zip audio path and label information and then iterate over them
  for path, emotion in zip(df["AudioPath"], df["Label"]):
    print("Number of files processed ",count)
    #get the features
        feature = get_features(path)
    for ele in feature:
      X_data.append(ele)
      Y_label.append(emotion)
    count+=1
  #create a dataframe of aduio features
  Features = pd.DataFrame(X_data)
  #add label information
  Features['Label'] = Y_label
  #store the extracted features in a csv file
  Features.to_csv('C:/Users/halaj/OneDrive/Desktop/data/Audio_features_All_pr.csv',index=False)

Audio_features_extract()
```

```python
from tensorflow.keras.models import Sequential, Model, load_model
from tensorflow.keras.layers import Dense, Conv1D, Activation, Add, AveragePooling1D, Flatten, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import time

# Placeholder for calc_time decorator
def calc_time(func):
    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        print(f"{func.__name__} took {end - start} seconds")
        return result
    return wrapper

# Function to plot loss and accuracy curves on training set
def plotgraph(history):
    plt.figure(figsize=[8, 6])
    plt.plot(history.history['loss'], 'firebrick', linewidth=3.0)
    plt.plot(history.history['accuracy'], 'turquoise', linewidth=3.0)
    plt.legend(['Training loss', 'Training Accuracy'], fontsize=18)
    plt.xlabel('Epochs', fontsize=16)
    plt.ylabel('Loss and Accuracy', fontsize=16)
    plt.title('Loss Curves and Accuracy Curves', fontsize=16)
    plt.show()

# Function to carry out additional preprocessing on data
def additional_preprocess(filepath):
    df = pd.read_csv(filepath)
    print("\nLabels or emotions present in dataset\n", df["Label"].unique())
    df["Label"] = df["Label"].str.replace("sadness", "sad", case=True)
```

```
    df["Label"] = df["Label"].str.replace("happiness", "happy", case=True)
    df["Label"] = df["Label"].str.replace("Fear", "fear", case=True)
    df["Label"] = df["Label"].str.replace("Sad", "sad", case=True)
    df["Label"] = df["Label"].str.replace("Pleasant_surprise", "surprise", case=True)
    df["Label"] = df["Label"].str.replace("pleasant_surprised", "surprise", case=True)
    df["Label"] = df["Label"].str.replace("surprised", "surprise", case=True)
    df["Label"] = df["Label"].str.replace("fearful", "fear", case=True)
    df["Label"] = df["Label"].str.replace("anger", "angry", case=True)
    df.drop((np.where(df['Label'].isin(["surprise", "calm"]))[0]), inplace=True)
    print("\nUnique count of labels or emotions after dropping selected labels\n",
df["Label"].value_counts())
    print("\nLength of the total data is {}".format(len(df)))
    return df

# Function to get audio features, perform one-hot encoding, and split datasets into train,
test, and validation
@calc_time
def audio_features_final():
    df =
additional_preprocess("C:/Users/halaj/OneDrive/Desktop/data/Audio_features_All_pr.csv")
    data = df[df.columns[0:-1]].values
    encoder = OneHotEncoder()
    label = df["Label"].values
    label = encoder.fit_transform(np.array(label).reshape(-1, 1)).toarray()
    scaler = MinMaxScaler()
    data = scaler.fit_transform(data)
    x_train, x_test, y_train, y_test = train_test_split(data, label, test_size=0.20,
random_state=42, shuffle=True)
    x_test, x_val, y_test, y_val = train_test_split(x_test, y_test, test_size=0.50,
random_state=42, shuffle=True)
    print("\nLength of train data is {}, test data is {} and validation set is
{}".format(len(x_train), len(x_test), len(x_val)))
    print("\nShape of train features and label is {}".format(x_train.shape, y_train.shape))
    print("\nShape of test features and label is {}".format(x_test.shape, y_test.shape))
    print("\nShape of validation features and label is {}".format(x_val.shape, y_val.shape))
    return x_train, x_test, y_train, y_test, x_val, y_val, encoder

# Function to create and train the emotion recognition model
@calc_time
def emotion_recognition_model(x_train, y_train, x_val, y_val):
    reduce_lr = ReduceLROnPlateau(monitor='loss', factor=0.2, patience=5, min_lr=0.001)
    es = EarlyStopping(monitor='loss', patience=20)
    filepath = "C:/Users/halaj/OneDrive/Desktop/data/emotion-recognition.keras"
    checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1,
save_best_only=True, mode='max')
    callbacks_list = [reduce_lr, es, checkpoint]

    def residual_block(x, filters, conv_num=3, activation="relu"):
        s = Conv1D(filters, 1, padding="same")(x)
        for i in range(conv_num - 1):
            x = Conv1D(filters, 3, padding="same")(x)
            x = Activation(activation)(x)
        x = Conv1D(filters, 3, padding="same")(x)
        x = Add()([x, s])
        x = Activation(activation)(x)
        return x

    def build_model():
        inputs = Input(shape=(x_train.shape[1], 1))
        x = Dense(256, activation="relu")(inputs)
        x = residual_block(x, 16, 2)
        x = residual_block(x, 32, 2)
        x = residual_block(x, 32, 2)
        x = residual_block(x, 64, 3)
        x = residual_block(x, 64, 3)
```

```
        x = residual_block(x, 128, 3)
        x = residual_block(x, 128, 3)
        x = AveragePooling1D(pool_size=3, strides=3)(x)
        x = Flatten()(x)
        x = Dense(256, activation="relu")(x)
        x = Dense(128, activation="relu")(x)
        outputs = Dense(6, activation="softmax", name="output")(x)
        return Model(inputs=inputs, outputs=outputs)

    res_model = build_model()
    res_model.summary()
    res_model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=1e-4),
metrics=['accuracy'])

    history = res_model.fit(np.expand_dims(x_train, -1), y_train,
                            validation_data=(np.expand_dims(x_val, -1), y_val),
                            epochs=50,
                            batch_size=32,
                            shuffle=True,
                            verbose=1,
                            callbacks=callbacks_list)

    # Save model structure in JSON file
    model_json = res_model.to_json()
    with open("C:/Users/halaj/OneDrive/Desktop/data/emotion_model.json", "w") as json_file:
        json_file.write(model_json)

    # Save trained model weights in .h5 file
res_model.save_weights('C:/Users/halaj/OneDrive/Desktop/data/emotion_model_weights.weights.h5')

    # Save the entire model in .keras format
    res_model.save('C:/Users/halaj/OneDrive/Desktop/data/emotion_model_full.keras')

    plotgraph(history)

    return res_model
```

```
import sounddevice as sd

def get_audio(duration=3, sr=44100):
    print("Recording...")
    # Record audio for the specified duration and sampling rate
    audio = sd.rec(int(duration * sr), samplerate=sr, channels=1, dtype='float64')
    # Wait for recording to finish
    sd.wait()
    print("Recording finished.")
    return audio.flatten(), sr
```

```
#function is used to get the audio features recorded from the microphone
def get_features_recorded(data,sr):

  #get features for recorded audio using mircophone
  res1 = extract_features(data,sr)
  result = np.array(res1)

  #get audio features with noise
  noise_data = noise(data)
  res2 = extract_features(noise_data,sr)
  result = np.vstack((result, res2))

  #get audio features with stretching and pitching
  new_data = stretch(data)
  data_stretch_pitch = pitch(new_data, sr)
  res3 = extract_features(data_stretch_pitch,sr)
  result = np.vstack((result, res3))
```

```
 18.
 19.   return result
 20.
 21. import os
 22. import glob
 23.
 24. def test_realtime(encoder):
 25.     # Create the directory if it doesn't exist
 26.     directory = 'C:/Users/halaj/OneDrive/Desktop/data/realtimetested'
 27.     if not os.path.exists(directory):
 28.         os.makedirs(directory)
 29.
 30.     # Load the best model
 31.     res_model = load_model("C:/Users/halaj/OneDrive/Desktop/data/emotion-recognition.keras")
 32.
 33.     # Record the audio
 34.     audio, sr = get_audio()
 35.
 36.     # Save audio in a file
 37.     files = []
 38.     os.chdir(directory)
 39.     for file in glob.glob("*.npy"):
 40.         files.append(file)
 41.
 42.     # Save the audio file for reference
 43.     np.save('C:/Users/halaj/OneDrive/Desktop/data/audiorec{}.npy'.format(len(files)), audio)
 44.
 45.     # Plot the recorded audio
 46.     plt.figure(figsize=(5,5))
 47.     plt.plot(audio)
 48.     plt.show()
 49.
 50.     # Save the plot of audio file
 51.     plt.savefig("audiorec{}.png".format(len(files)))
 52.
 53.     # Convert int to float
 54.     audio = audio.astype('float')
 55.
 56.     # Get audio features from the recorded voice
 57.     feature = get_features_recorded(audio, sr)
 58.
 59.     # Apply min-max scaling
 60.     scaler = MinMaxScaler()
 61.     feature = scaler.fit_transform(feature)
 62.
 63.     # Get the predicted label
 64.     label = res_model.predict(feature)
 65.
 66.     # Get the label information by reversing one-hot encoded output
 67.     label_predicted = encoder.inverse_transform(label)
 68.
 69.     print("\nThe Emotion Predicted For Recorded Audio Using Microphone is
{}".format(label_predicted[0]))
 70.
 71.     # Create a dataframe for recorded audio features
 72.     df = pd.DataFrame(index=range(0,3),columns=['path','label','audio'])
 73.     for i in range(0,3):
 74.         df["path"][i] =
'C:/Users/halaj/OneDrive/Desktop/data/audiorec{}.npy'.format(len(files))
 75.         df["label"][i] = label_predicted[i]
 76.         df["audio"][i] = feature[i]
 77.
 78.     # Store the real-time predicted features in a CSV file
 79.     df.to_csv('C:/Users/halaj/OneDrive/Desktop/data/real_time_predicted_audio_features.csv',
mode='a', index=False)
```

```
 80.
 81. def evaluate_model(x_train, x_test, y_train, y_test, x_val, y_val):
 82.   #load the best model
 83.   model = load_model("C:/Users/halaj/OneDrive/Desktop/data/emotion-recognition.keras")
 84.   #evaluate training accuracy
 85.   _,train_acc = model.evaluate(np.expand_dims(x_train,-1),y_train, batch_size=1)
 86.   #evaluate testing acuracy
 87.   _,test_acc = model.evaluate(np.expand_dims(x_test,-1),y_test, batch_size=1)
 88.   #evaluate validation accuracy
 89.   _,val_acc = model.evaluate(np.expand_dims(x_val,-1),y_val, batch_size=1)
 90.   print("\n*********************************************")
 91.   print("\n Training accuracy of the model is {}".format(np.round(float(train_acc*100),2)))
 92.   print("\n Testing accuracy of the model is {}".format(np.round(float(test_acc*100),2)))
 93.   print("\n Validation accuracy of the model is {}".format(np.round(float(val_acc*100),2)))
 94.   print("*********************************************")
 95.   #predict the outcome of the model
 96.   y_pred = model.predict(x_test)
 97.   y_pred=np.argmax(y_pred, axis=1)
 98.   y_test=np.argmax(y_test, axis=1)
 99.   #View the classification report for test data and predictions
100.   print("\nClassification report for Emotion Recognition")
101.   print(classification_report(y_test, y_pred))
102.   #View confusion matrix for test data and predictions
103.   print("\nConfusion matrix for Emotion Recognition")
104.   print(confusion_matrix(y_test, y_pred))
105.   print("****************************")
```

```
 1. @calc_time
 2. def main():
 3.   #get train,test data and labels
 4.   x_train, x_test, y_train, y_test, x_val, y_val, encoder = audio_features_final()
 5.   #call the emotion recognition model
 6.   emotion_recognition_model(x_train,y_train,x_val,y_val)
 7.   #evaluate the model performance
 8.   evaluate_model(x_train, x_test, y_train, y_test, x_val, y_val)
 9.
10. if_name__:main()
11.
```

```
1. x_train, x_test, y_train, y_test, x_val, y_val, encoder = audio_features_final()
```

```
1. #mapping of the one hot ecoding with respect to their labels
2. print("\none hot encoding array\n",np.unique(y_train,axis=0))
3. print("\nOne hot encoding mapping to actual
label\n",encoder.inverse_transform(np.unique(y_train,axis=0)))
```

```
1. #this function call is used to record audio using microphone and test the model in real time on
speaker voice
2. test_realtime(encoder)
```

4. combined modal

4.1 combined modal code using webcam and microphone

```
 1. import librosa
 2. import sounddevice as sd
 3. import numpy as np
 4. import pandas as pd
 5. import keras
 6. from keras.models import load_model, model_from_json
 7. from sklearn.preprocessing import StandardScaler, OneHotEncoder
 8. import cv2
 9. from collections import Counter
10. import serial
11. import time
```

```
12.
13. # Load the trained speech model
14. speech_model = load_model('C:/Users/halaj/OneDrive/Desktop/speech/speech_model.keras')
15.
16. # Load the scaler and encoder
17. scaler = StandardScaler()
18. encoder = OneHotEncoder()
19.
20. # Fit the scaler and encoder with the same data used during training
21. features = pd.read_csv('C:/Users/halaj/OneDrive/Desktop/data/features.csv')
22. X = features.iloc[:, :-1].values
23. Y = features['labels'].values
24. scaler.fit(X)
25. encoder.fit(np.array(Y).reshape(-1, 1))
26.
27. def extract_features(data, sample_rate):
28.     result = np.array([])
29.
30.     # ZCR
31.     zcr = np.mean(librosa.feature.zero_crossing_rate(y=data).T, axis=0)
32.     result = np.hstack((result, zcr))
33.
34.     # Chroma_stft
35.     stft = np.abs(librosa.stft(data))
36.     chroma_stft = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T, axis=0)
37.     result = np.hstack((result, chroma_stft))
38.
39.     # MFCC
40.     mfcc = np.mean(librosa.feature.mfcc(y=data, sr=sample_rate).T, axis=0)
41.     result = np.hstack((result, mfcc))
42.
43.     # RMS
44.     rms = np.mean(librosa.feature.rms(y=data).T, axis=0)
45.     result = np.hstack((result, rms))
46.
47.     # MelSpectrogram
48.     mel = np.mean(librosa.feature.melspectrogram(y=data, sr=sample_rate).T, axis=0)
49.     result = np.hstack((result, mel))
50.
51.     return result
52.
53. def record_audio(duration=2.5, fs=22050):
54.     print("Recording...")
55.     audio = sd.rec(int(duration * fs), samplerate=fs, channels=1)
56.     sd.wait()
57.     print("Recording complete.")
58.     return np.squeeze(audio)
59.
60. def predict_speech_emotion(audio, sample_rate):
61.     features = extract_features(audio, sample_rate)
62.     features = scaler.transform([features])
63.     features = np.expand_dims(features, axis=2)
64.     prediction = speech_model.predict(features)
65.     predicted_emotion = encoder.inverse_transform(prediction)
66.     return predicted_emotion[0][0]
67.
68. json_file = open(r'C:\Users\halaj\OneDrive\Desktop\emotion detection\emotion_model.json', 'r')
69. loaded_model_json = json_file.read()
70. json_file.close()
71. emotion_model = model_from_json(loaded_model_json)
72. emotion_model.load_weights(r"C:\Users\halaj\OneDrive\Desktop\emotion
detection\emotion_model.weights.h5")
73. print("Loaded model from disk")
74.
75. # Facial emotion dictionary
```

```
emotion_dict = {0: "Angry", 1: "Disgusted", 2: "Fearful", 3: "Happy", 4: "Neutral", 5: "Sad", 6: "Surprised"}

def predict_facial_emotion(frame, face_detector):
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    num_faces = face_detector.detectMultiScale(gray_frame, scaleFactor=1.3, minNeighbors=5)
    emotions = []
    for (x, y, w, h) in num_faces:
        roi_gray_frame = gray_frame[y:y + h, x:x + w]
        cropped_img = np.expand_dims(np.expand_dims(cv2.resize(roi_gray_frame, (48, 48)), -1), 0)
        emotion_prediction = emotion_model.predict(cropped_img)
        maxindex = int(np.argmax(emotion_prediction))
        emotions.append(emotion_dict[maxindex])
    return emotions

def send_emotion_to_arduino(emotion):
    port = 'COM6'
    baud_rate = 9600
    ser = serial.Serial(port, baud_rate)
    time.sleep(2) # Wait for the connection to establish

    print(f"Sending emotion: {emotion}")
    ser.write(emotion.encode())
    ser.close()

def wait_for_motion():
    port = 'COM6'
    baud_rate = 9600
    ser = serial.Serial(port, baud_rate)
    while True:
        if ser.in_waiting > 0:
            motion_signal = ser.readline().decode().strip()
            if motion_signal == "MOTION":
                ser.close()
                return

def main():
    # Wait for motion to be detected
    print("Waiting for motion...")
    wait_for_motion()

    # Record audio and predict speech emotion
    duration = 2.5  # seconds
    sample_rate = 22050  # Hz
    audio = record_audio(duration=duration, fs=sample_rate)
    speech_emotion = predict_speech_emotion(audio, sample_rate)
    print(f"Predicted Speech Emotion: {speech_emotion}")

    # Start webcam feed and predict facial emotion
    cap = cv2.VideoCapture(0)
    face_detector = cv2.CascadeClassifier(r"C:\Users\halaj\OneDrive\Desktop\emotion detection\haarcascades\haarcascade_frontalface_default.xml")
    facial_emotions = []

    frames_to_capture = 10
    frames_captured = 0

    while frames_captured < frames_to_capture:
        ret, frame = cap.read()
        if not ret:
            break
        emotions = predict_facial_emotion(frame, face_detector)
        if emotions:
            facial_emotions.extend(emotions)
```

```
            frames_captured += 1
        cv2.imshow('Emotion Detection', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

    # Determine the most frequent facial emotion
    if facial_emotions:
        most_common_facial_emotion = Counter(facial_emotions).most_common(1)[0][0]
        print(f"Predicted Facial Emotion: {most_common_facial_emotion}")
    else:
        most_common_facial_emotion = None

    # Combine the results
    if most_common_facial_emotion and most_common_facial_emotion == speech_emotion:
        combined_emotion = most_common_facial_emotion
        print(f"Combined Emotion: {combined_emotion}")
    else:
        combined_emotion = "Mismatch"
        print("Emotion detection failed: The models do not agree.")

    # Send the combined emotion to Arduino
    send_emotion_to_arduino(combined_emotion)

if __name__ == "__main__":
    main()
```

4.2 combined modal using videos

```
import librosa
import numpy as np
import pandas as pd
import keras
from keras.models import load_model, model_from_json
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import cv2
from collections import Counter
import moviepy.editor as mp
import serial
import time

# Load the trained speech model
speech_model = load_model('C:/Users/halaj/OneDrive/Desktop/speech/speech_model.keras')

# Load the scaler and encoder
scaler = StandardScaler()
encoder = OneHotEncoder()

# Fit the scaler and encoder with the same data used during training
features = pd.read_csv('C:/Users/halaj/OneDrive/Desktop/data/features.csv')
X = features.iloc[:, :-1].values
Y = features['labels'].values
scaler.fit(X)
encoder.fit(np.array(Y).reshape(-1, 1))

def extract_features(data, sample_rate):
    result = np.array([])

    # ZCR
    zcr = np.mean(librosa.feature.zero_crossing_rate(y=data).T, axis=0)
    result = np.hstack((result, zcr))

```

```
    # Chroma_stft
    stft = np.abs(librosa.stft(data))
    chroma_stft = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T, axis=0)
    result = np.hstack((result, chroma_stft))

    # MFCC
    mfcc = np.mean(librosa.feature.mfcc(y=data, sr=sample_rate).T, axis=0)
    result = np.hstack((result, mfcc))

    # RMS
    rms = np.mean(librosa.feature.rms(y=data).T, axis=0)
    result = np.hstack((result, rms))

    # MelSpectrogram
    mel = np.mean(librosa.feature.melspectrogram(y=data, sr=sample_rate).T, axis=0)
    result = np.hstack((result, mel))

    return result

def extract_audio_from_video(video_path):
    video = mp.VideoFileClip(video_path)
    audio = video.audio
    audio_path = "temp_audio.wav"
    audio.write_audiofile(audio_path)
    return audio_path

def load_audio(audio_path, sample_rate=22050):
    data, sr = librosa.load(audio_path, sr=sample_rate)
    return data, sr

def predict_speech_emotion(audio, sample_rate):
    features = extract_features(audio, sample_rate)
    features = scaler.transform([features])
    features = np.expand_dims(features, axis=2)
    prediction = speech_model.predict(features)
    predicted_emotion = encoder.inverse_transform(prediction)
    return predicted_emotion[0][0]

json_file = open(r'C:\Users\halaj\OneDrive\Desktop\emotion detection\emotion_model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
emotion_model = model_from_json(loaded_model_json)
emotion_model.load_weights(r"C:\Users\halaj\OneDrive\Desktop\emotion
detection\emotion_model.weights.h5")
print("Loaded model from disk")

# Facial emotion dictionary
emotion_dict = {0: "angry", 1: "disgusted", 2: "fear", 3: "happy", 4: "neutral", 5: "sad", 6:
"surprised"}

def predict_facial_emotion(frame, face_detector):
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    num_faces = face_detector.detectMultiScale(gray_frame, scaleFactor=1.3, minNeighbors=5)
    emotions = []
    for (x, y, w, h) in num_faces:
        roi_gray_frame = gray_frame[y:y + h, x:x + w]
        cropped_img = np.expand_dims(np.expand_dims(cv2.resize(roi_gray_frame, (48, 48)), -1),
0)
        emotion_prediction = emotion_model.predict(cropped_img)
        maxindex = int(np.argmax(emotion_prediction))
        emotions.append(emotion_dict[maxindex])
    return emotions

def send_emotion_to_arduino(emotion):
    port = 'COM6'
```

```
    baud_rate = 9600
    ser = serial.Serial(port, baud_rate)
    time.sleep(2) # Wait for the connection to establish

    print(f"Sending emotion: {emotion}")
    ser.write(emotion.encode())
    ser.close()

def wait_for_motion():
    port = 'COM6'
    baud_rate = 9600
    ser = serial.Serial(port, baud_rate)
    while True:
        if ser.in_waiting > 0:
            motion_signal = ser.readline().decode().strip()
            if motion_signal == "MOTION":
                ser.close()
                return

def main(video_path):
    # Wait for motion to be detected
    print("Waiting for motion...")
    wait_for_motion()

    # Extract audio from video and predict speech emotion
    audio_path = extract_audio_from_video(video_path)
    audio, sample_rate = load_audio(audio_path)
    speech_emotion = predict_speech_emotion(audio, sample_rate)
    print(f"Predicted Speech Emotion: {speech_emotion}")

    # Process video frames and predict facial emotion
    cap = cv2.VideoCapture(video_path)
    face_detector = cv2.CascadeClassifier(r"C:\Users\halaj\OneDrive\Desktop\emotion
detection\haarcascades\haarcascade_frontalface_default.xml")
    facial_emotions = []

    frames_to_capture = 30
    frames_captured = 0

    while frames_captured < frames_to_capture:
        ret, frame = cap.read()
        if not ret:
            break
        emotions = predict_facial_emotion(frame, face_detector)
        if emotions:
            facial_emotions.extend(emotions)
            frames_captured += 1
        cv2.imshow('Emotion Detection', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

    # Determine the most frequent facial emotion
    if facial_emotions:
        most_common_facial_emotion = Counter(facial_emotions).most_common(1)[0][0]
        print(f"Predicted Facial Emotion: {most_common_facial_emotion}")
    else:
        most_common_facial_emotion = None

    # Combine the results
    if most_common_facial_emotion and most_common_facial_emotion == speech_emotion:
        combined_emotion = most_common_facial_emotion
        print(f"Combined Emotion: {combined_emotion}")
```

```
    else:
        combined_emotion = "Mismatch"
        print("Emotion detection failed: The models do not agree.")

    # Send the combined emotion to Arduino
    send_emotion_to_arduino(combined_emotion)

if __name__ == "__main__":
    video_path = r"C:\Users\halaj\OneDrive\Desktop\datare\Actor_02\01-01-03-02-01-01-02.mp4"
    main(video_path)
```

5. Gesture-Based Control code

```
import os
import cv2
import mediapipe as mp
import math
import numpy as np
import serial
import time

# Suppress TensorFlow logs
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

# Solution APIs
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands

# Webcam Setup
wCam, hCam = 640, 480
cam = cv2.VideoCapture(0)
cam.set(3, wCam)
cam.set(4, hCam)

# Set up the serial connection (adjust 'COM3' to your port)
arduino = serial.Serial(port='COM6', baudrate=9600, timeout=.1)
time.sleep(2) # Give some time for the serial connection to initialize

# Function to smooth values using a moving average
def moving_average(data, window_size):
    return np.convolve(data, np.ones(window_size)/window_size, mode='valid')

# Initialize lists for smoothing
distance_vals = []
angle_vals = []
window_size = 5

# Mediapipe Hand Landmark Model
with mp_hands.Hands(
    model_complexity=0,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as hands:

    while cam.isOpened():
        success, image = cam.read()
        if not success:
            print("Ignoring empty camera frame.")
            continue

        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        results = hands.process(image)
```

```
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # Multi-hand landmarks method for finding the position of hand landmarks
        lmList = []
        if results.multi_hand_landmarks:
            myHand = results.multi_hand_landmarks[0]
            for id, lm in enumerate(myHand.landmark):
                h, w, c = image.shape
                cx, cy = int(lm.x * w), int(lm.y * h)
                lmList.append([id, cx, cy])

        # Assigning variables for thumb and index finger position
        if len(lmList) != 0:
            x1, y1 = lmList[4][1], lmList[4][2]
            x2, y2 = lmList[8][1], lmList[8][2]

            # Marking thumb and index finger
            cv2.circle(image, (x1, y1), 15, (255, 255, 255), -1)
            cv2.circle(image, (x2, y2), 15, (255, 255, 255), -1)
            cv2.line(image, (x1, y1), (x2, y2), (0, 255, 0), 3)

            # Calculate distance and angle
            length = math.hypot(x2 - x1, y2 - y1)
            angle = math.degrees(math.atan2(y2 - y1, x2 - x1))

            # Smooth the values
            distance_vals.append(length)
            angle_vals.append(angle)

            if len(distance_vals) > window_size:
                distance_vals.pop(0)
            if len(angle_vals) > window_size:
                angle_vals.pop(0)

            smooth_distance = moving_average(distance_vals, window_size)[-1]
            smooth_angle = moving_average(angle_vals, window_size)[-1]

            # Ensure the angle is within 0 to -180 degrees
            if smooth_angle < -180:
                smooth_angle = -180
            elif smooth_angle > 0:
                smooth_angle = 0

            # Print smoothed distance and angle
            print(f'Smooth Distance: {smooth_distance}')
            print(f'Smooth Angle: {smooth_angle} degrees')

            # Send smoothed distance and angle to Arduino
            arduino.write(f"{int(smooth_distance)},{int(smooth_angle)}\n".encode())

            # Display smoothed distance and angle on the webcam feed
            cv2.putText(image, f'Distance: {int(smooth_distance)}', (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)
            cv2.putText(image, f'Angle: {int(smooth_angle)} deg', (10, 70),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)

        cv2.imshow('handDetector', image)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

# Release resources
cam.release()
cv2.destroyAllWindows()
```

6. Circuit diagram:

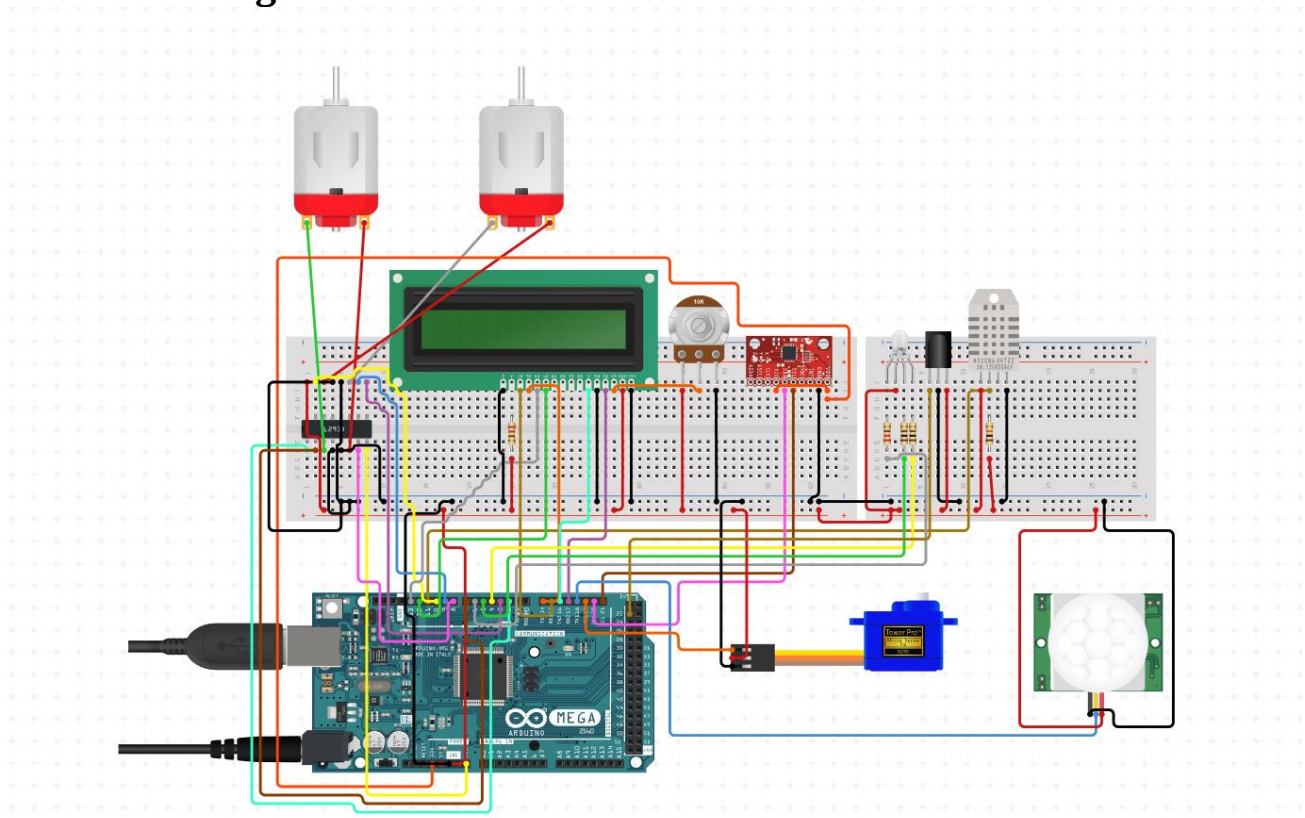


Figure 26: fan with motion circuit

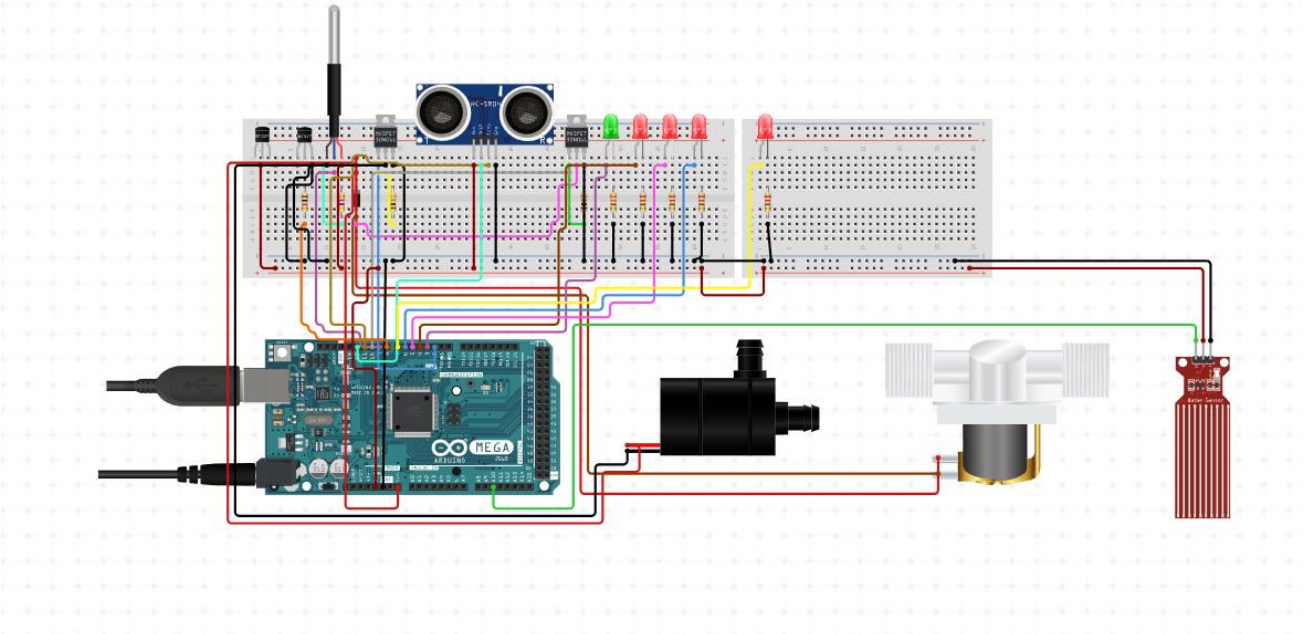


Figure 27: Water Tank circuit