Zagazig University

Faculty Of Engineering

Computer and Systems Department


# Graduation Project:

# Vision Guard: Monitoring and Surveillance Robot

Supervised by:

DR Sanaa Fekry

Team members:

Omar Abdelaziz Mohamed

Amr Essam Goda

Mohamed Mahmoud Ali

Mostafa Mohamed Mostafa

Medhat Mohamed Sobhy

Omar Mohamed Elsayed

Mahmoud Abdelrahman Elsayed

Osama Shehta Fathy

Omar Abdelhamid Mohamed

Mahmoud Hesham Mohamed

Norhan Ahmed Mohamed

Ahmed Elsaied Ali

Sherine Tarek Barges

Mohamed Abdelnaser Elsayed

Ahmed Kamal Hosny

# ABSTRACT:

Enhancing security and safety through cutting-edge surveillance systems is paramount in contemporary technological development. This project introduces an advanced solution with the creation of a Surveillance Robot: Monitoring and Tracking, utilizing state-of-the-art AI models to detect, track, and respond in real-time. The system is designed not only to identify and monitor individuals but also to search for specific users, providing highly targeted surveillance capabilities. In its AI control mode, the robot autonomously navigates towards any detected individual, ensuring proactive and dynamic monitoring.

Organizations today face numerous security challenges, such as unauthorized access, theft, and workplace safety threats. Traditional security systems often rely on human monitoring, which can be prone to errors and inefficiencies. This project addresses these issues by employing a sophisticated surveillance robot that offers continuous and accurate monitoring without the need for constant human oversight.

Firstly, the robot's AI-driven facial recognition system enhances security by identifying and tracking unauthorized personnel. This capability is crucial for preventing unauthorized access to restricted areas, ensuring that only authorized individuals can enter secure zones. Secondly, the robot can be programmed to search for specific users, such as VIPs or known security threats, allowing for swift and appropriate responses in various situations.

The robot's integration of multiple manual control methods, including intuitive glove control and a versatile mobile app interface, ensures seamless user interaction and adaptability. This feature enables security personnel to control and monitor the robot remotely, enhancing their ability to respond to incidents promptly. Additionally, the robot's real-time video streaming and AI-controlled camera system provide live updates and continuous monitoring, which is essential for maintaining high-security standards in dynamic environments.

By leveraging advanced control algorithms, precise sensor data processing, and robust AI technology, the project ensures unparalleled reliability and accuracy in surveillance. The implementation of ultrasonic sensors further enhances the robot's ability to navigate and measure distances accurately, contributing to its overall operational efficiency.

This thesis provides a comprehensive analysis of the hardware and software integration required to develop a highly effective surveillance robot. The project aims to offer an innovative and dependable solution for security and safety applications, particularly within organizational environments. By significantly reducing the need for manual intervention, the robot enhances overall operational efficiency and responsiveness, addressing critical security challenges faced by modern organizations and contributing to a safer, more secure environment.

# List Of Contents

# List Of Figures:

# Chapter1: Introduction

## Overview

Millions of organizations around the world face significant security challenges in ensuring the safety and protection of their assets and personnel. Traditional surveillance systems often rely on human monitoring, which can be prone to errors, inefficiencies, and limited coverage. Security breaches, unauthorized access, and theft are common concerns that necessitate the development of more advanced and reliable surveillance solutions.

Unauthorized access to restricted areas poses a substantial risk to sensitive information and critical infrastructure. Intruders and unauthorized personnel can compromise the integrity of an organization's operations. Traditional methods of monitoring and controlling access can fall short, particularly in large or complex environments. An advanced surveillance robot can mitigate these risks by providing continuous, real-time monitoring and alerting security personnel of any unauthorized entry.

Theft of valuable assets, both physical and intellectual, remains a significant threat to organizations. Conventional surveillance systems may not provide the necessary level of deterrence or the ability to track suspicious activities effectively. By utilizing AI-driven facial recognition and real-time video streaming, the surveillance robot can detect and respond to suspicious behavior immediately, enhancing the organization's ability to prevent theft.

Workplace safety is another critical area where advanced surveillance systems can play a pivotal role. In hazardous environments, unauthorized presence can lead to accidents and injuries. The surveillance robot can patrol such areas, detect unauthorized personnel, and alert management to potential safety violations, thereby preventing accidents and ensuring a safer workplace.

Monitoring high-risk areas such as data centers, research labs, and financial institutions requires heightened security measures. Continuous and accurate surveillance is essential to protect these critical assets from both internal and external threats. The surveillance robot, equipped with AI and advanced sensor technology, can provide uninterrupted monitoring and quick response capabilities, ensuring that these high-risk areas remain secure.

Advancements in robotics, sensor technology, and artificial intelligence offer enormous potential for developing sophisticated surveillance systems. The Surveillance Robot: Monitoring and Tracking project leverages these technologies to address critical security challenges faced by modern organizations, providing a robust and reliable solution that enhances overall operational efficiency and security.

## History:

### Early Concepts and Precursors

### 1950s-1960s: Early Ideas and Theoretical Foundations

- **1956:** The concept of robotic surveillance began to take shape with early theoretical discussions in robotics and automation. Visionaries like Isaac Asimov explored ideas of intelligent machines in science fiction, laying the groundwork for future developments in robotics.
- **1960s:** The field of robotics was in its infancy, with basic experimental robots like the **Unimate** being developed for industrial applications. Although not designed for surveillance, these early robots demonstrated the potential for automation and remote operation.

### Emergence of Surveillance Technologies

### 1970s-1980s: Development of Early Surveillance Systems

- **1972:** The **Stanford Cart** was one of the first mobile robots equipped with cameras for remote viewing and control. Although not a dedicated surveillance robot, it demonstrated the feasibility of using robots for visual monitoring tasks.
- **1980s:** The development of **closed-circuit television (CCTV)** systems became widespread for security and surveillance. While CCTV was not robotic, it set the stage for integrating video technology into automated systems.

### First Generation of Surveillance Robots

### 1990s: Introduction of Robotic Surveillance Systems

- **1991:** The **Robot Systems** company developed the **RoboGuard**, one of the first commercially available surveillance robots. It was equipped with cameras and sensors for security applications, marking the beginning of dedicated surveillance robots in the market.
- **1997: Adept Technology** introduced the **Adept Robot**, which featured early forms of automated inspection and surveillance capabilities. This period saw significant advancements in mobile robotics and sensor technologies.

### Advancements and Commercialization

### 2000s: Growth in Capabilities and Applications

- **2002: iRobot** launched the **iRobot PackBot**, a robot initially designed for military applications but later adapted for civilian uses, including surveillance and reconnaissance. Its capabilities included remote-controlled operation and real-time video streaming.

- **2005:** The **TROOPER** surveillance robot, developed by **SRI International**, featured advanced sensors and remote operation capabilities for security and surveillance missions. It was used in various applications, including law enforcement and public safety.
- **2008: ASIMOV Robotics** introduced the **ASIMOV Security Robot**, which integrated advanced sensors, cameras, and autonomous navigation features for security and monitoring applications.

**Modern Innovations and AI Integration**

**2010s-Present: Advanced Technologies and AI Integration**

- **2010: Knightscope** introduced the **K5 Security Robot**, a highly advanced surveillance robot equipped with cameras, sensors, and AI algorithms for public safety and security applications. The K5 was designed for autonomous patrols and real-time threat detection.
- **2014:** The **ROAMEO** robot, developed by **Robotics Lab at the University of Naples**, showcased advanced capabilities in autonomous navigation and surveillance. It integrated sophisticated AI models for recognizing and tracking individuals.
- **2016:** The **Unitree Go1** robot featured advanced mobility, camera systems, and AI algorithms for surveillance and monitoring tasks. Its design included improvements in autonomous navigation and environmental interaction.
- **2020: DJI** released the **RoboMaster S1**, a versatile educational robot with surveillance capabilities. It incorporated advanced sensors and AI algorithms for learning and experimentation in robotic surveillance.
- **2022: HAL Robotics** introduced the **HAL Security Bot**, a state-of-the-art surveillance robot featuring real-time video analytics, advanced facial recognition, and autonomous patrolling capabilities. It represented the latest advancements in robotic security solutions.

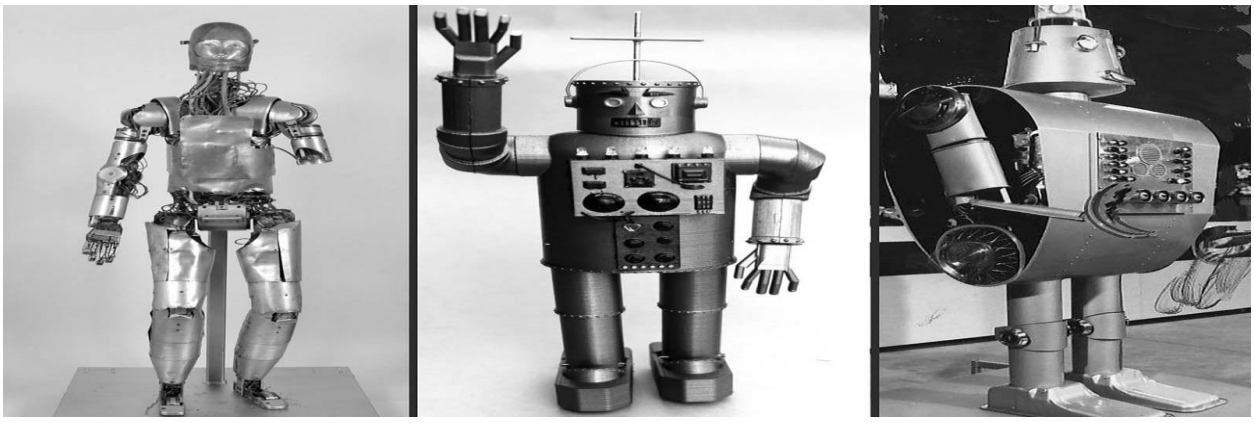


*Figure 1*

## Main Features:

### 1. AI-Driven Facial Recognition System

**Description:** The robot utilizes advanced AI models to perform facial recognition tasks, allowing it to identify and track individuals based on their facial features.

**Key Technologies:**

- **Deep Learning Algorithms:** Employs convolutional neural networks (CNNs) for accurate face detection and recognition.
- **Face Database:** Maintains a database of known faces for comparison and identification.
- **Real-Time Processing:** Processes video data in real-time to detect, recognize, and track faces.

**Benefits:**

- **Enhanced Security:** Identifies and monitors individuals, distinguishing between authorized and unauthorized personnel.
- **Dynamic Tracking:** Automatically follows detected individuals, ensuring continuous surveillance.

### 2. AI Control Mode for Autonomous Navigation

**Description:** In AI control mode, the robot uses AI algorithms to navigate towards detected individuals and maintain surveillance.

**Key Technologies:**

- **AI Navigation Algorithms:** Uses pathfinding algorithms to move towards detected faces.
- **Sensor Fusion:** Integrates data from cameras, ultrasonic sensors, and other sensors for navigation.
- **Autonomous Decision Making:** Makes real-time decisions about movement and tracking based on AI analysis.

**Benefits:**

- **Proactive Monitoring:** Automatically approaches and tracks individuals, enhancing the effectiveness of surveillance.
- **Autonomous Operation:** Reduces the need for manual control and supervision.

### 3. Real-Time Video Streaming and Data Transmission

**Description:** The robot streams live video footage and transmits data to a remote control station or mobile app.

**Key Technologies:**

- **High-Definition Cameras:** Captures clear and detailed video footage.
- **Streaming Protocols:** Uses protocols like RTSP (Real-Time Streaming Protocol) for video transmission.
- **Secure Communication:** Ensures secure data transfer over Wi-Fi or cellular networks.

**Benefits:**

- **Immediate Feedback:** Provides security personnel with real-time visual updates and data.
- **Remote Monitoring:** Enables remote observation and control from any location.

### 4. Ultrasonic Sensors for Obstacle Detection and Distance Measurement

**Description:** Ultrasonic sensors measure distances and detect obstacles in the robot's path.

**Key Technologies:**

- **Ultrasonic Rangefinders:** Measures the time it takes for ultrasonic waves to bounce back from obstacles.
- **Obstacle Avoidance Algorithms:** Uses sensor data to navigate around obstacles.

**Benefits:**

- **Safe Navigation:** Prevents collisions with objects and obstacles.
- **Accurate Distance Measurement:** Provides precise distance information for navigation and tracking.

### 5. Multi-Mode Control Interface

**Description:** The robot features multiple control interfaces for user interaction, including a mobile app and a glove-based control system.

**Key Technologies:**

- **Mobile App Interface:** Offers joystick controls, video streaming, and status updates.
- **Glove Control System:** Uses flex sensors and Bluetooth communication for manual control.

**Benefits:**

- **Versatile Control Options:** Provides users with different methods for controlling and interacting with the robot.

### 6. Glove-Based Control System

**Description:** A specialized glove with flex sensors and Bluetooth communication allows users to manually control the robot's movements.

**Key Technologies:**

- **Flex Sensors:** Detect hand movements to control robot actions.
- **Bluetooth Communication:** Connects the glove to the robot for command transmission.
- **AT Commands:** Used for pairing and configuring Bluetooth modules (HC-05).

**Benefits:**

- **Manual Control:** Offers a tactile and intuitive way to control the robot.
- **Flexible Operation:** Allows users to interact with the robot using hand gestures.

- troubleshooting and monitoring the robot's functions.

### 7. Robust Power Management System

**Description:** The robot is equipped with a reliable power management system to ensure consistent operation.

**Key Technologies:**

- **Battery Pack:** Powers all electronic components and sensors.
- **Power Distribution:** Manages power supply to different parts of the robot.

**Benefits:**

- **Extended Operation:** Ensures the robot can operate for extended periods.
- **Efficient Power Use:** Optimizes power distribution for all components.

### 8. Advanced Software Framework

**Description:** The robot's software integrates various components, including AI algorithms, control interfaces, and sensor management.

**Key Technologies:**

- **Software Architecture:** Modular design for ease of development and maintenance.
- **Integration Framework:** Combines AI, sensor data, and user inputs into a cohesive system.

**Benefits:**

- **Scalable and Maintainable:** Facilitates updates and improvements.
- **Efficient Operation:** Ensures smooth interaction between hardware and software

# Chapter2: Design

## User-Machine Interaction

Our project offers advanced user-machine interaction, allowing users to directly control the robot and send specific user data for targeted searches and tracking. The robot's AI capabilities enable it to autonomously seek out and follow individuals based on the provided data. Additionally, the robot streams high-definition video feedback to users, facilitating effective monitoring of organizations or institutions where the robot is deployed. This combination of AI-driven tracking and real-time video surveillance ensures comprehensive security solutions for a wide range of applications.

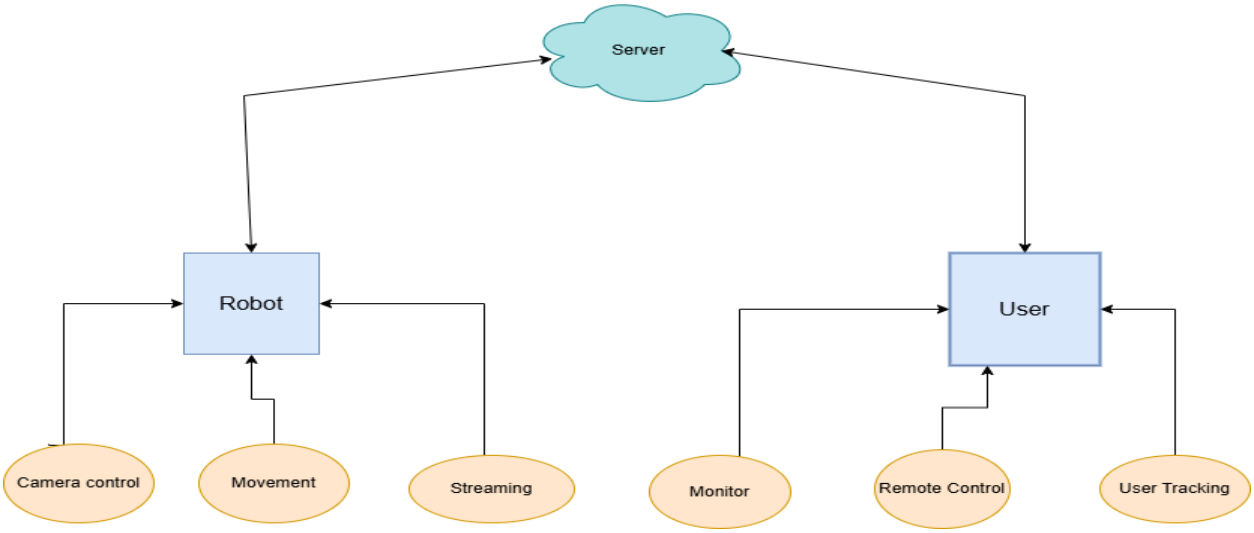



*Figure 2*

The next figure explains the flowchart of how this robot sends the video stream to the user in order of surveillance and monitoring.

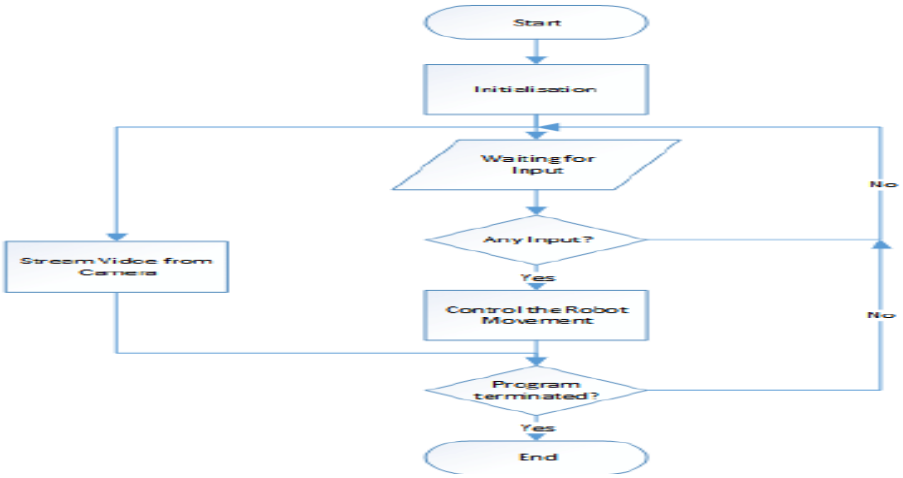



*Figure 3*

# Embedded System Concepts

An embedded system is a specialized computing system designed to perform dedicated functions or tasks as part of a larger system. Unlike general-purpose computers, embedded systems are built to operate with specific requirements and constraints, providing functionality through a combination of hardware and software.

## Characteristics of Embedded Systems

**a. Dedicated Functionality:**
Embedded systems are designed to perform a specific function or set of functions. For example, a washing machine controller or a microwave oven's timer.

**b. Real-Time Operation:**
Many embedded systems operate in real-time, meaning they must process data and respond to inputs within strict time constraints. Real-time systems can be categorized as **hard real-time** (where deadlines must be met) or **soft real-time** (where deadlines are preferred but not critical).

**c. Limited Resources:**
Embedded systems often operate with limited processing power, memory, and storage. This is due to cost, space, and power consumption constraints.

**d. Integration with Hardware:**
Embedded systems are tightly integrated with hardware components, including sensors, actuators, and communication interfaces. They often control or monitor physical processes.

**e. Low Power Consumption:**
Power efficiency is crucial, especially in portable or battery-operated devices. Embedded systems are designed to be power-efficient, extending battery life and reducing energy costs.

## Components of Embedded Systems

**a. Microcontroller / Microprocessor:**
The central processing unit of an embedded system, responsible for executing software instructions. Microcontrollers integrate the CPU, memory, and I/O interfaces on a single chip, while microprocessors may be part of a more complex system.

- **Microcontroller Example:** Atmega328 (used in Arduino boards)
- **Microprocessor Example:** ARM Cortex-A series (used in high-end devices)

**b. Memory:**
Memory is used to store instructions, data, and states. Embedded systems typically use **RAM** for temporary data storage and **ROM** or **Flash** for permanent storage of software and firmware.

- **RAM:** Temporary data storage for runtime operations.
- **ROM/Flash:** Stores the embedded software or firmware.

**c. Input/Output Interfaces:**
Interfaces for communicating with the external environment. Common interfaces include GPIO (General-Purpose Input/Output), ADC (Analog-to-Digital Converter), DAC (Digital-to-Analog Converter), UART (Universal Asynchronous Receiver/Transmitter), SPI (Serial Peripheral Interface), and I2C (Inter-Integrated Circuit).

**d. Sensors and Actuators:**
Sensors collect data from the environment (e.g., temperature, light levels), while actuators perform actions based on the system's decisions (e.g., motors, relays).

- **Sensors:** Temperature sensor, Accelerometer
- **Actuators:** DC motor, Servo motor

**e. Power Supply:**
Provides the necessary electrical power for the system's operation. It must be stable and meet the specific voltage and current requirements of the components.

**f. Communication Modules:**
Used for data exchange between the embedded system and external devices or networks. Examples include Wi-Fi, Bluetooth, and GSM modules.

**g. Software/Firmware:**
The embedded software or firmware runs on the microcontroller or microprocessor, implementing the system's functionality. It includes operating systems (if applicable) and application-specific code.

Embedded systems are integral to modern technology, providing specialized functionalities in various applications. They are characterized by their dedicated purpose, real-time capabilities, and integration with both hardware and software. Understanding these concepts is essential for designing, developing, and deploying effective embedded solutions across multiple domains, from consumer electronics to industrial automation and beyond.

## Layered Architecture

Each layer in the architecture serves a unique role and interacts with adjacent layers through well-defined interfaces. Here's a breakdown of each layer:

Our project satisfies this approach, as shown in the next figure, the implementation of our embedded subsystem layers in the project:

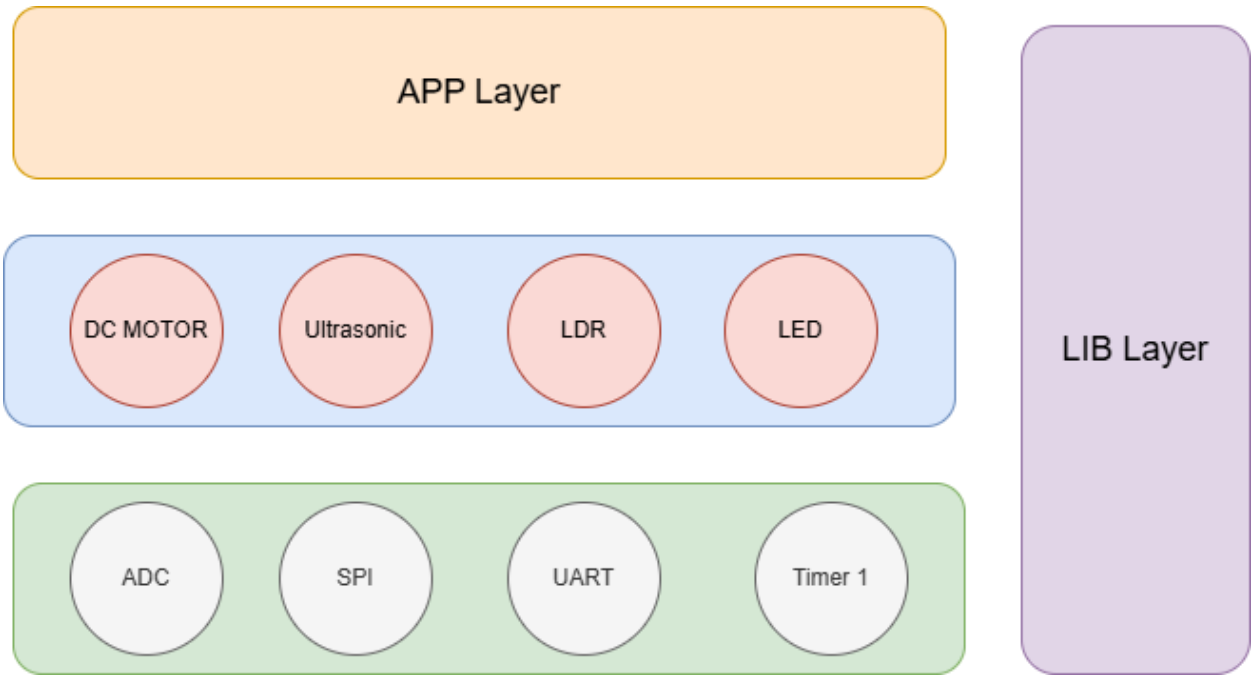



*Figure 4*

Each peripheral will be illustrated clearly in the upcoming chapters.

As shown the HAL Layer contains DC Motor, Ultrasonic, LDR and LED Drivers, each is used for a specific purpose.

MCAL Layer contains ADC, SPI, UART and Timer1 including ICU, PWM Mode.

# Hardware Design

In order to implement the Embedded System Part:

A Microcontroller is essentially used in order to give control and provide the following peripherals.

Each peripheral provides a specific operation, each will be explained individually.

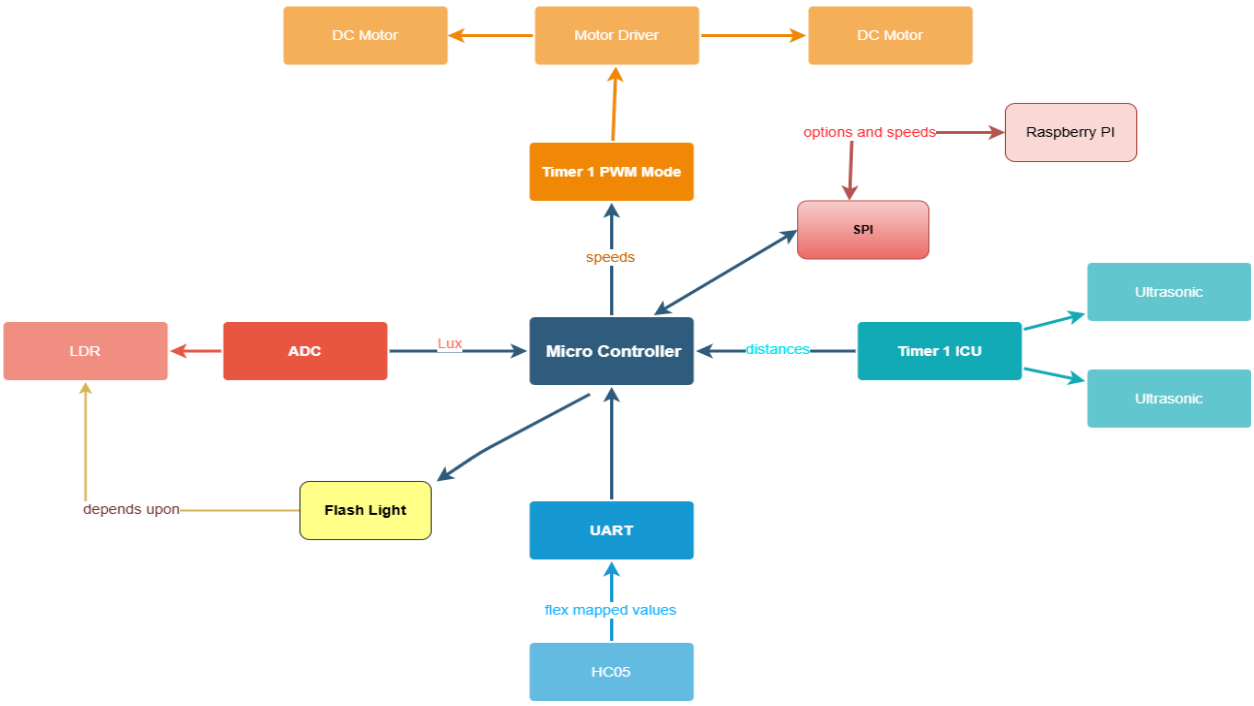



*Figure 5*

## The Hardwired Circuits Simulation

Microcontroller Circuit and Connections:

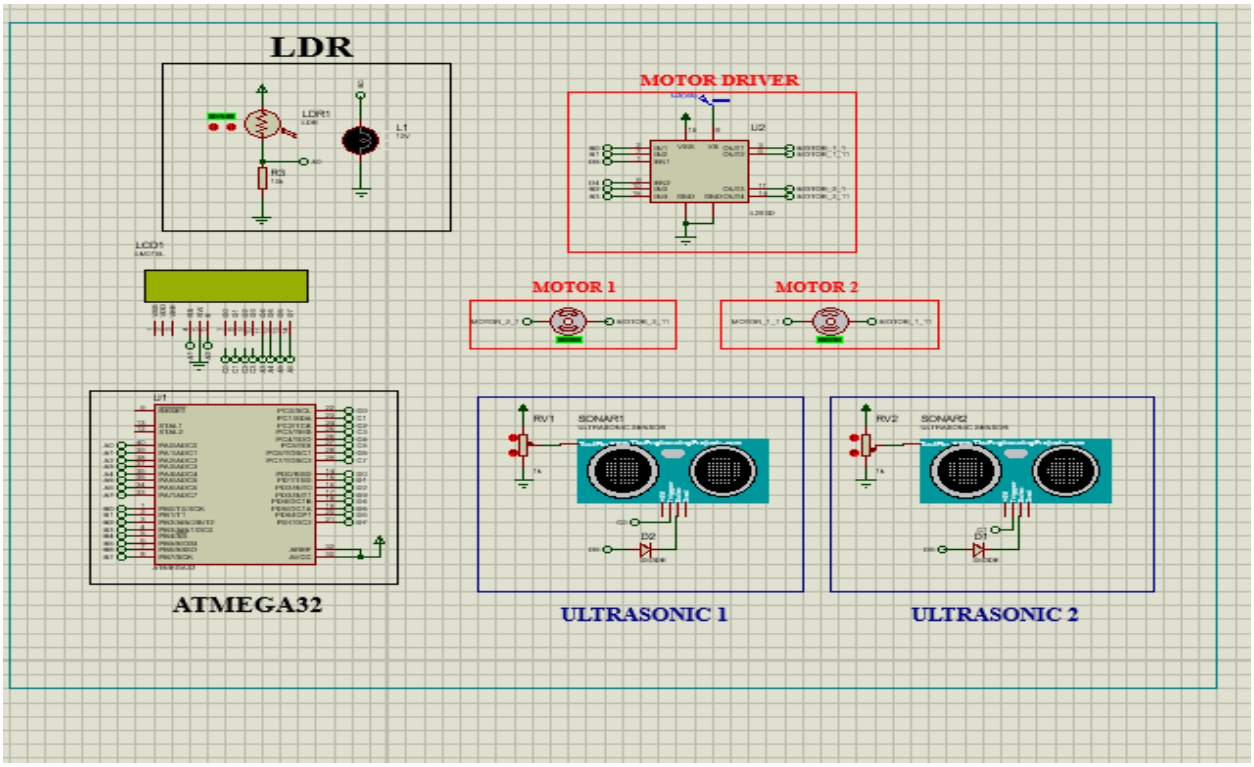



*Figure 6*

Glove Control Circuit:

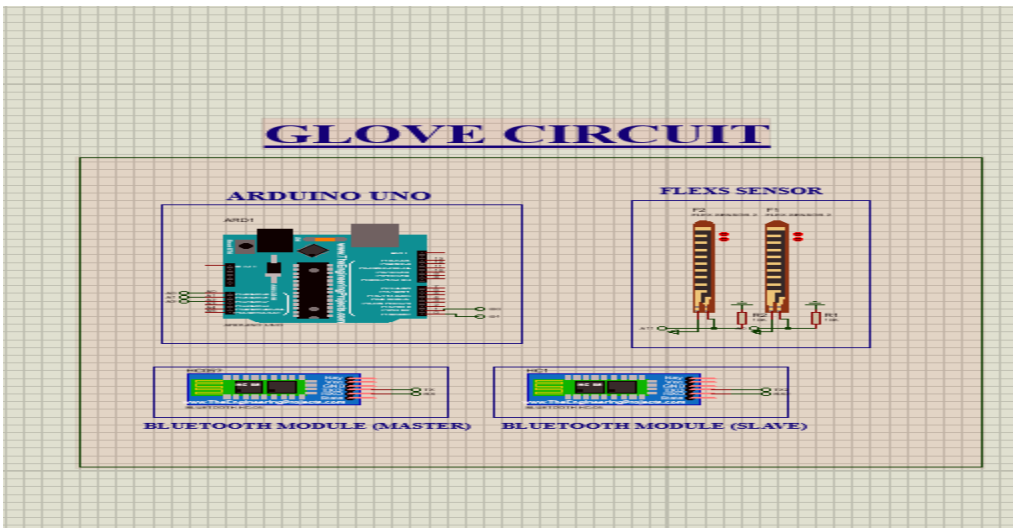



*Figure 7*

Raspberry Pi connections:

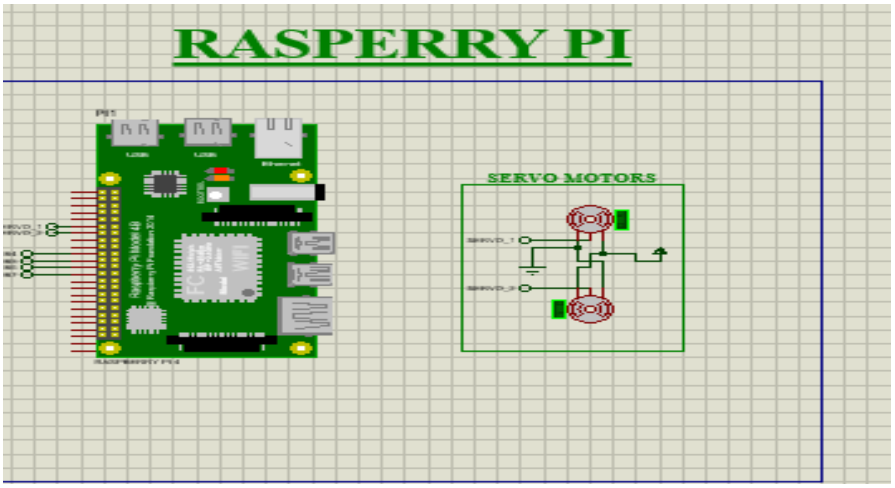



*Figure 8*

## HIGH-LEVEL Design

The high-level design of the surveillance robot involves the strategic arrangement of key components and subsystems to achieve the primary objective: real-time facial recognition, tracking, and security monitoring. This design outlines the main architectural elements and interactions among different components, ensuring that the system effectively meets its goals

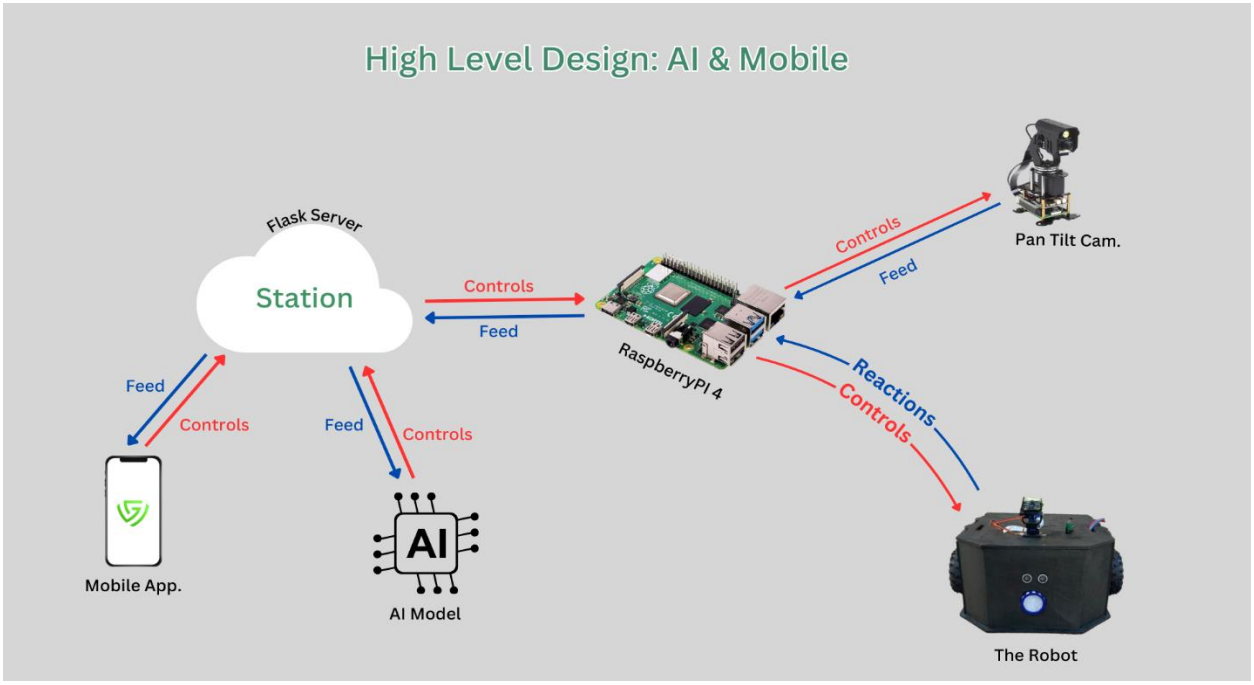
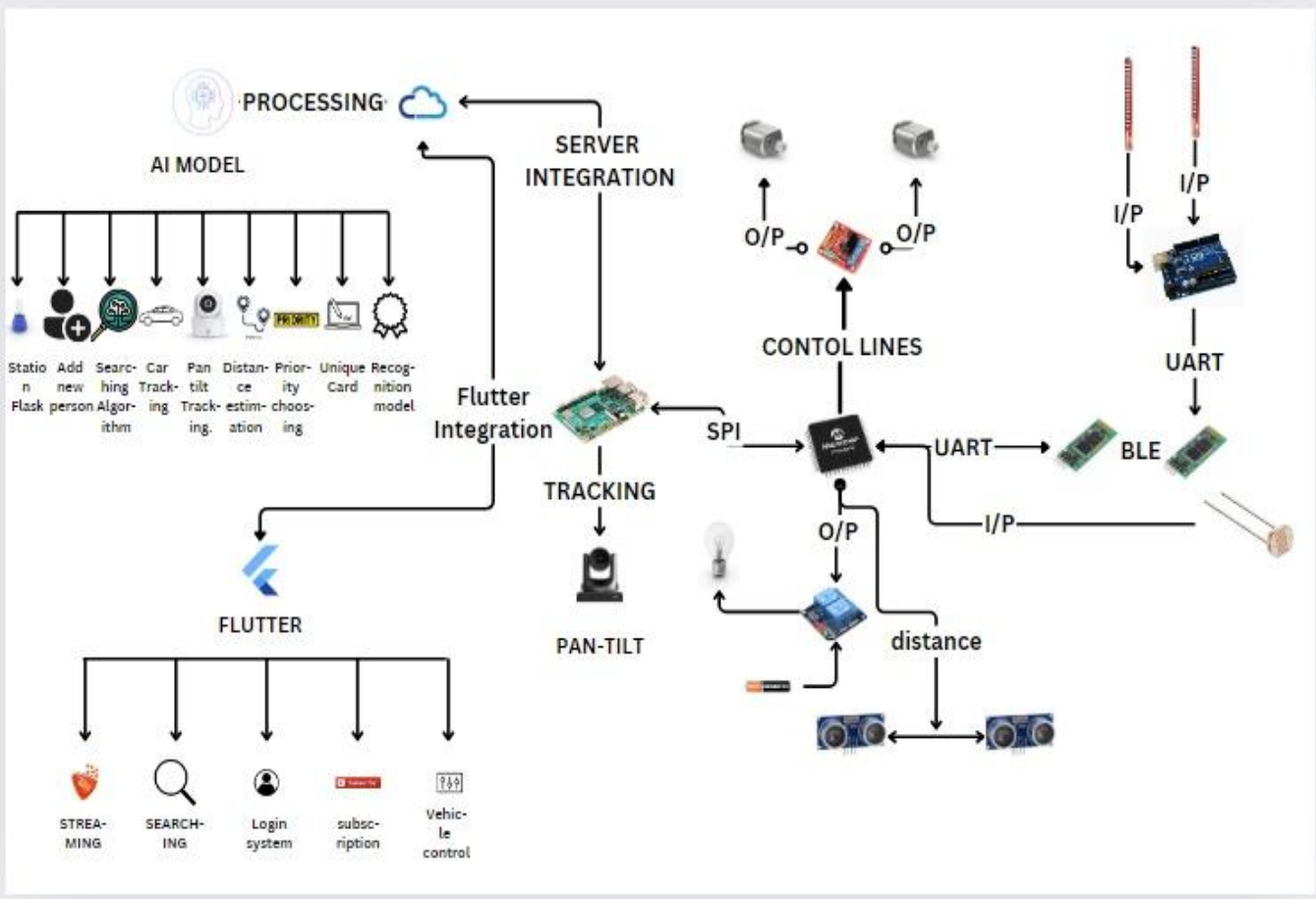



*Figure 9*

## Project Design

This version emphasizes that the following sections will cover the complete design of the robot through the specific technologies and methodologies used in the project.

*Figure 10*



## Hardware Components:

*Figure 11*

### Car Chassis

### Ultrasonic

*Figure 12*

### Motor Driver

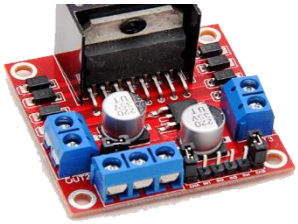


*Figure 13*

### DC Motor

*Figure 14*

### Microcontroller

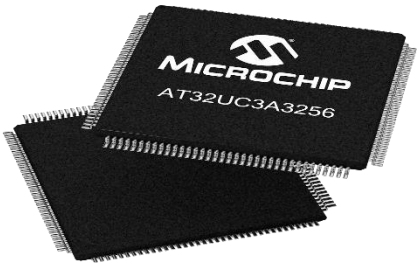



*Figure 15*

Arduino

*Figure 16*

Raspberry Pi 4

*Figure 17*

Pan-Tilt (2 servo motors)


*Figure 18*

LDR

*Figure 19*

Flash Light

*Figure 20*

Batteries

*Figure 21*

## Flex sensor

**2.21 Figure**

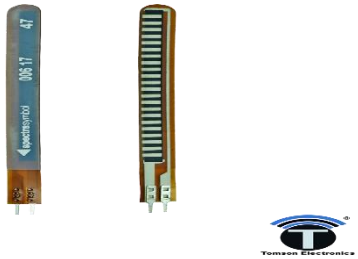


*Figure 22*

## Bluetooth Module

**2.22 Figure**

*Figure 23*

## General purpose Diode

**2.23 Figure**

*Figure 24*

## Switch

**2.24 Figure**

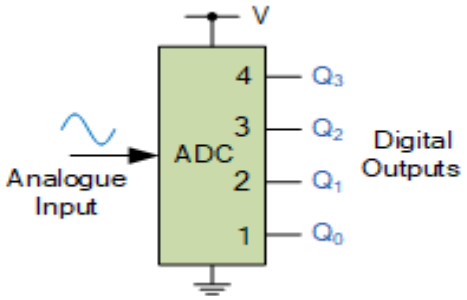
*Figure 25*

# Chapter3: Adaptive light control

## What is ADC?

An Analog-to-Digital Converter (ADC) is an electronic device that converts an input analog signal into a corresponding digital signal. The analog signal is continuous in time and amplitude, while the digital signal is discrete.

*Figure 26*




## Importance of ADC

ADCs are essential because most real-world signals (like sound, light, and temperature) are analog. However, digital systems (computers, microcontrollers) require digital inputs. ADCs bridge this gap, enabling the processing of real-world signals in digital form.

## Characteristics of ADC

- **ADC Resolution:** It is the number of bits used to represent the digital value after conversion.
- **Reference Voltage:** It is the max analog voltage that can be converted by the ADC.
- **ADC step:** It is the analog value required to increment digital value by 1.
- **ADC clock:** clock is used by ADC to synchronize conversion.
- **ADC conversion Time:** It is time taken by ADC to convert from analog to digital

## How do ADCs, operate?

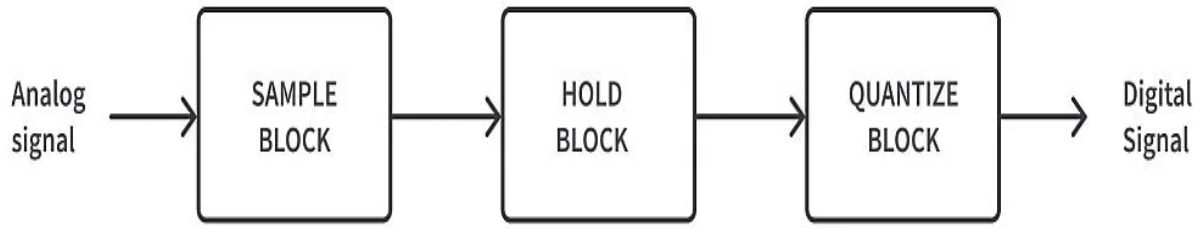
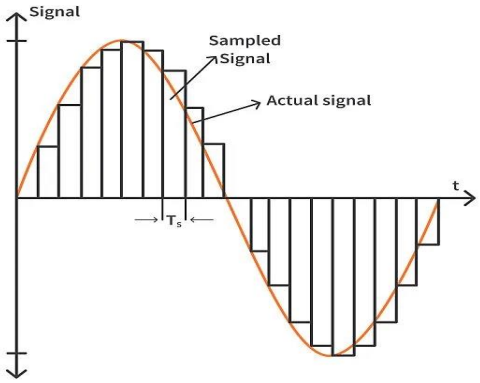



*Figure 27*

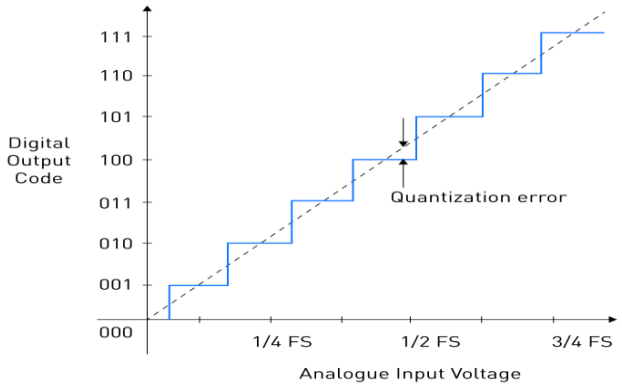
- Sample: The analog signal can be sampled at an exact time interval in the sample block. The samples are discrete in terms of time but are employed in continuous amplitude and have real value. The sampling frequency is crucial in the signal conversion process. for it to be kept at a precise pace. The sample rate can be fixed based on the requirements of the system.

*Figure 28*



➢ Quantization: Quantization comes after sampling as a crucial step in converting continuous analog signals to digital signals. A continuous set of values (like voltage levels) is quantized into a discrete set of values. During the analog-to-digital conversion process, each sampled value is matched with the closest value among a limited number of discrete levels.

*Figure 29*




➢ Encoding: Quantized values are then encoded into binary form, representing the digital signal. The resolution of the ADC, determined by the number of bits, dictates the precision of this encoding.

- Types of ADCs:

## Successive Approximation Register (SAR) ADC:

**Working Principle:** The SAR ADC uses a binary search algorithm to convert the analog signal to a digital output. It is known for its balance between speed and accuracy.

**Applications:** General-purpose applications, including data acquisition and instrumentation.

## Delta-Sigma ADC:

**Working Principle:** This type uses oversampling and noise shaping to achieve high resolution. It is slower compared to other types but offers excellent precision.
**Applications**: Audio processing, precision measurement instruments.

## Flash ADC:

**Working Principle:** Uses a parallel comparator circuit, providing the fastest conversion times. However, it is less power-efficient and more expensive.
**Applications:** High-speed applications like digital oscilloscopes and radar systems.

## Pipeline ADC:

**Working Principle:** Combines multiple stages of lower-resolution ADCs to achieve high-speed conversions with moderate resolution.
**Applications:** Digital video processing, communication systems.

## ADC in AVR

- **10-bit resolution.**
- **8 Multiplexed Single Ended Input Channels**
- **7 Differential Input Channels**
- **2 Differential Input Channels with Optional Gain of 10x and 200x**
- **Free Running or Single Conversion Mode**
- **ADC Start Conversion by Auto Triggering on Interrupt Sources**
- **Interrupt on ADC Conversion Complete**

## LDR-Sensor

## What is an LDR Sensor?

A Light Dependent Resistor (LDR), also known as a photoresistor or photoconductor, is a type of resistor whose resistance decreases as the intensity of light falling on it increases. This makes LDRs useful for detecting light levels and creating light-sensitive devices.

## Basic Structure:

LDRs are made from semiconductor materials such as cadmium sulfide (CdS). They have a high resistance in the dark and low resistance in the light.

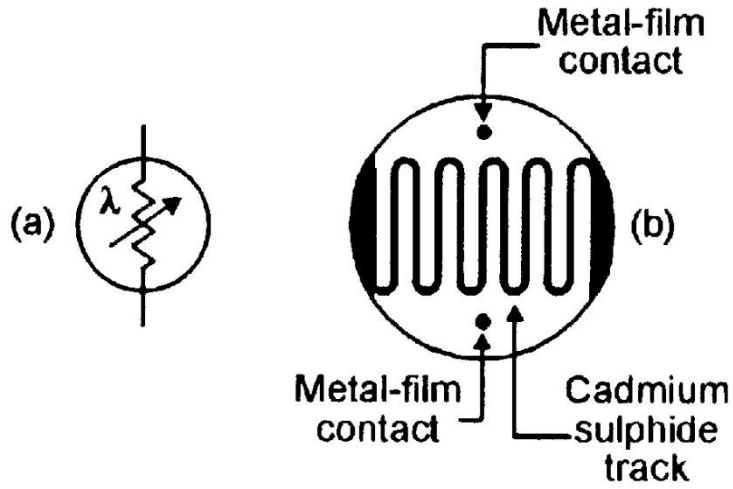



*Figure 30*

## Working Principle:

It is the optical phenomena known as photoconductivity. The conductivity of the substance increases when light is absorbed by it. The electrons in the material's valence band are eager to move to the conduction band when light strikes the LDR. But for the electrons to move from one band to another (valance to conduction), the photons in the incident light must have energy greater than the material's bandgap.

Hence, more electrons are excited to the conduction band in the presence of abundant light energy, grading in a large number of charge carriers. The resistance of the device reduces when the effect of this process and the current flow increase.

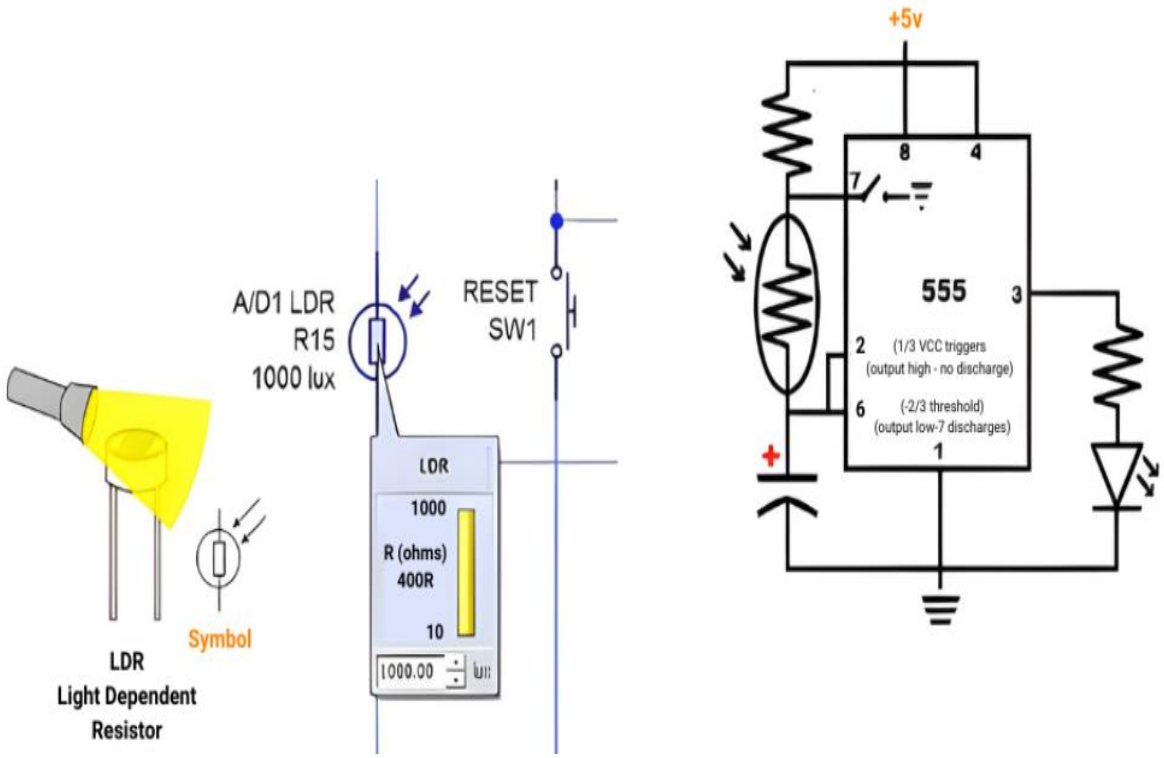



*Figure 31*

## LDR with ADC in AVR

The high resistance semiconductor material is used in the light dependent resistor. Such a semiconductor absorbs light energy from the incident pictures through bound electrons, also known as valence electrons. With an LDR and a fixed resistor, we can build a voltage divider circuit. Then, we can use the microcontroller's ADC to measure the voltage, which represents the resistance or the intensity of the light. An LDR's resistance can be as high as 2MΩ in complete darkness, 8kΩ in room light, and 5kΩ in daylight. To create an extra loop over the crucial components and ensure that the voltage is well-defined, a pull-up resistor is placed between the supply voltage and the LDR sensor's output.

Equation for the LDR circuit's output voltage is:

**VA0 = Vsource (RLDR / (RLDR + Rfixed))**

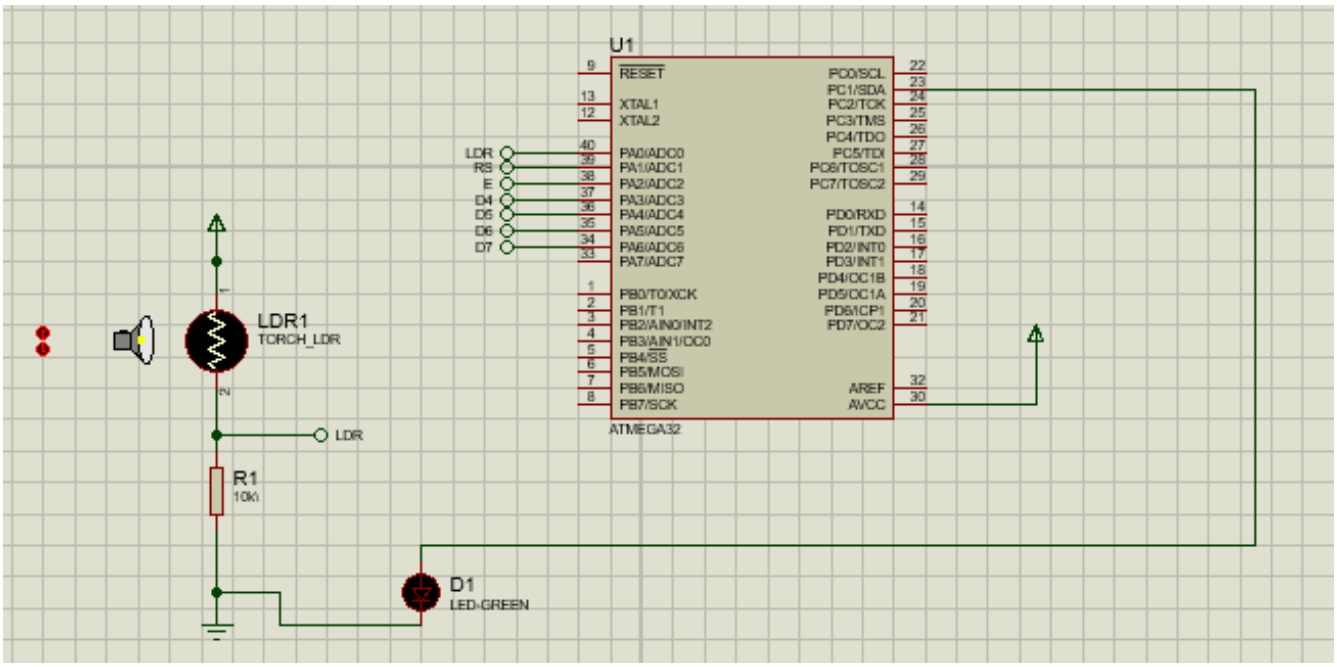



*Figure 32*

## LDR Characteristics:

**Sensitivity:** Changes resistance based on light intensity.

**Response Time:** Takes a short time to adjust to changes in light levels.

**Spectral Response:** Sensitive to visible light, especially in the wavelength range of 500-700 nm.

## Applications of LDR Sensors:

- **Street Lighting:**
  **Automatic Street Lights:** LDRs are used in street lighting systems to automatically turn lights on at dusk and off at dawn. The high resistance in darkness triggers the circuit to turn the lights on, while the low resistance in daylight turns them off.

- **Light Meters:**
  **Photography:** LDRs are used in light meters to measure light intensity. This helps photographers set the correct exposure for capturing images.

- **Burglar Alarms:**
  **Security Systems:** LDRs are used in burglar alarms to detect changes in light levels. When an intruder disrupts a light beam, the change in resistance triggers the alarm.

- **Solar Tracking Systems:**
  **Solar Panels:** LDRs are used in solar tracking systems to keep solar panels aligned with the sun. The LDRs detect the sun's position, and the system adjusts the panel's angle for maximum efficiency.

# Chapter4: Communication

## Background

When you connect a microcontroller to a sensor, display, or other module, do you ever think about how the two devices talk to each other? What exactly are they saying? How are they able to understand each other?

Communication between electronic devices is like communication between humans. Both sides need to speak the same language. In electronics, these languages are called communication protocols

**ECU is Electronic Control Unit**; it is a MC unit and external peripherals on the same PCB responsible for a certain task.

•Most of Complex Systems consists of several ECUs, each responsible of one task, they communicate with each other using communication protocols

### What is a Protocol:

•A protocol is a set of rules that governs data communication.

•It defines **what** is communicating, **how** it is communicating, and **when** it is communicating.

•Any protocol must contain the following:
1.Syntax: structure or format or order of data represented.

2.Semantics: meaning of each section of bits, how a particular pattern is interpreted and what should be the response.

3.Timing: when data should be sent and how fast can they be sent.

4.Hardware Interface: defines the hardware connection between the two communicating devices

### Standard Protocol:

It is a fixed method of communication that is globally defined and documented. Any different systems that support this protocol shall easily communicate to each other without any constraints.

**Communication protocols examples in embedded systems**

- **UART:** Universal Asynchronous Receiver Transmitter
- **SPI:** Serial Peripheral Interface
- **I2C:** Inter-Integrated Circuit
- **CAN:** Controller Area Network
- **LIN:** Local Interconnect Network

## Communication Protocol Specifications

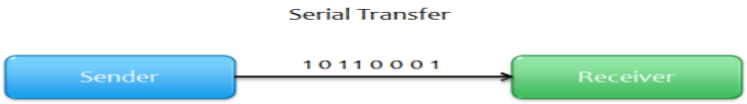



Figure 33

**Serial Communication**

1.is the process of sending data one bit at
bus.

2.Slower.

3.Less number of cables required to transmit data (low cost).

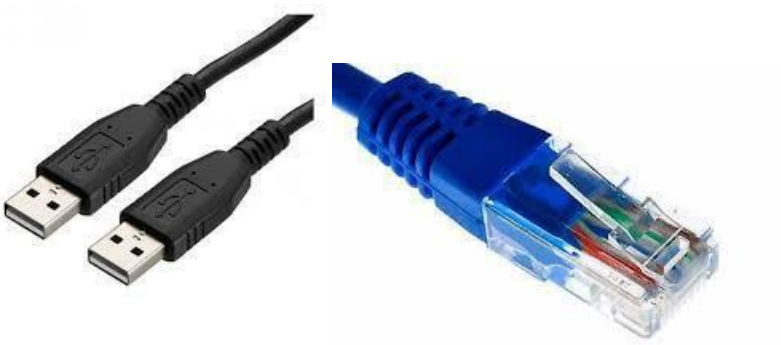


Figure 34


Figure 35

**Parallel Communication:**

1.where several bits are sent as a whole at the same time, on a link with several parallel channels.

2.Faster.

3.Higher number of cables required (high cost).

***Parallel Communication Problems***

1- **Complex connections**
2- **Cross talk**

Cross Talk is the phenomenon where a wire that having electrical signal passing through it generates a magnetic field that affects other wires on the same cable!

***3- Timing Skew (Data Skew)*** is a phenomenon where the data transmitted on the parallel cables arrives at different time which led to wrong data sampling. The skew is directly proportional with the data transmission speed, which means increasing data rate increases the skew!

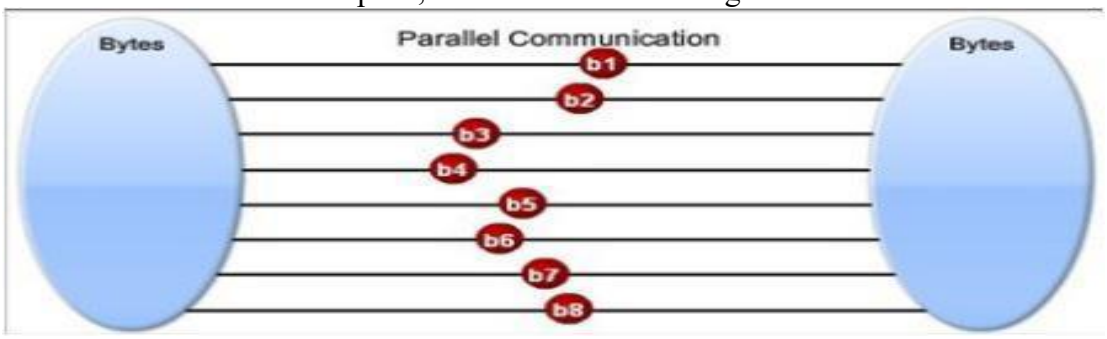



*Figure 36*

## Synchronous Vs Asynchronous

***Synchronous Communication*** It is a type of communication the nodes of the network share the same clock.

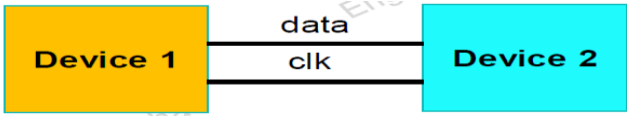
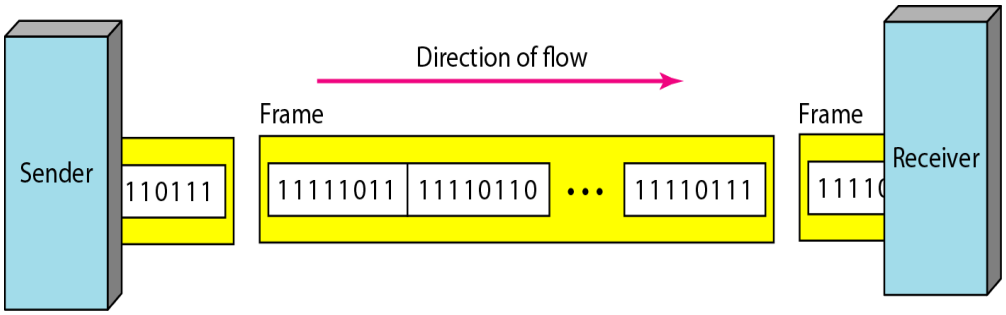



*Figure 37*

**In synchronous transmission, we send bits one after another without start or stop bits or gaps. It is the responsibility of the receiver to group the bits. The bits are usually sent as bytes and many bytes are grouped in a frame. A frame is identified with a start and an end byte.**

*Figure 38*




***Asynchronous Communication***

It is a type of communication that doesn't have a shared clock between the communicating nodes, instead the nodes are configured with the communicating rate and each node is having its own clock generator system that generates this clock.

**In asynchronous transmission, we send 1 start bit (0) at the beginning and 1 or more stop bits (1s) at the end of each byte. There may be a gap between each byte.**
**•Asynchronous here means "asynchronous at the byte level," but the bits are still synchronized; their durations are the same.**

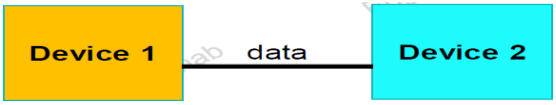


Figure 39

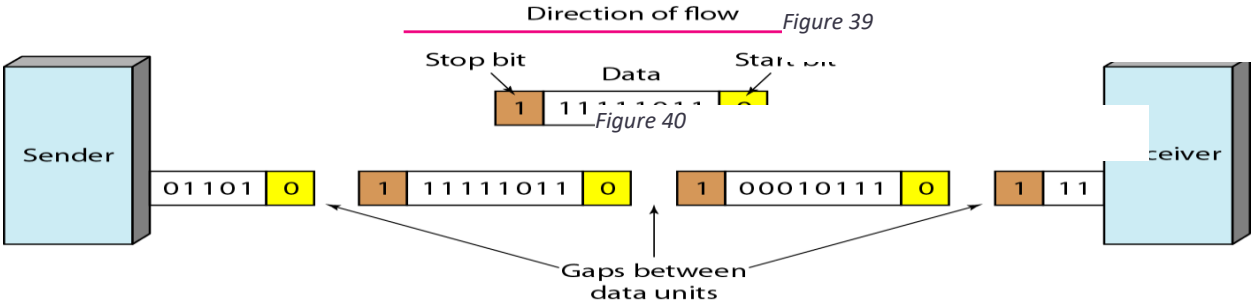
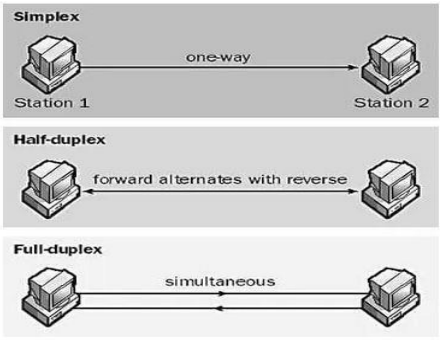


Figure 40

## Simplex Vs Half Duplex Vs Full duplex

***Simplex Channel***: The data is in one direction, from a transmitter to receiver.
***Half duplex channel***: The data is bidirectional, each node can transmit and receive but not in the same time!
***Full duplex channel***: The data is bidirectional and each node can transmit and receive at the same time.



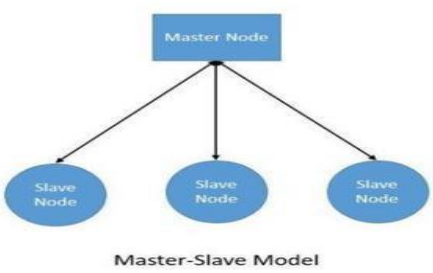
Full-duplex

Figure 41

### *Peer to Peer* Vs *Master Slave*

***Peer to Peer communication***: In this type of communication the com
to each other any time with no privileges. ***Master Slave communicati***
communication there is a master node that can send data to any other
is the only node that can initiate the communication, the slave can never initiate the communication. The slave can send data to master only when the master permits the slave to send. **The Master / Slave network can be divided to:**

- Single Master Single Slave (SMSS)
- Single Master Multi Slave (SMMS)
- Multi Master Multi Slave (MMMS)



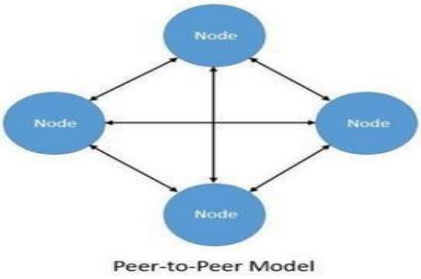




Figure 42

# SPI Communication

## Overview

SPI is a synchronous, Full duplex communication protocol, which support SMMS Architecture.

The standard 4-wire SPI consists of 4 external pins, typically called Master Out Slave In (MOSI), Master in Slave out (MISO), Serial Clock (SCLK), and Slave Select (SS).

| PIN Name | Functional Description |
|---|---|
| SCLK | -It is used to output a clock signal generated by master to all the slave(s).<br>-It is used for synchronizing the data transfer taking place across different devices. |
| SS | -It is an active low pin used by a master to select which slave.<br>-It must go low before a data transfer begins and must stay low during the process. Otherwise, the data transfer will be aborted. |
| MOSI | -It is a unidirectional pin used to transfer serial data from the master to the slave<br>-When a device is configured as a master, serial data is sent through this pin.<br>-When a device is configured as a slave, serial data is received through this pin |
| MISO | -It is a unidirectional pin used to transfer serial data from the slave to the master.<br>-When a device is configured as a slave and it is selected (the slave's SS pin goes low), serial data is sent through this pin.<br>-When a device is configured as a slave and it is not selected, the slave will drive this pin to high impedance<br>-When a device is configured as a master, serial data is received through this pin. |

## SPI Concept

SPI is a full-duplex communication protocol, which in our project is used as the connection between the Raspberry Pi and the Microcontroller.

It is Single Master Multi slave protocol (SMMS), in our project, the Raspberry pi is the master and the microcontroller is the slave.

The SPI operation is a shift register operation. The master swaps a bit with the slave every clock cycle.

## Why We need SPI?

- A full duplex communication protocol was needed as the robot requires sending and receiving data.
- deterministic timing, as it is a synchronous communication protocol.

## SPI Operation

The Master sends zero on the SS pin, in order to initiate transmission, the Microcontroller frequently check the SPI FLAG, if it is raised the MC read the sent data from the Raspberry and simultaneously send the ultrasonic distance values to the raspberry, the Raspberry send at first the control option then the speeds are sent.

## SPI Pros and Cons

### Pros

**High-Speed Communication**: SPI can operate at much higher data rates compared to other serial communication protocols like I2C, making it suitable for applications requiring fast data transfer.

**Simultaneous Data Transfer:** SPI supports full-duplex communication, allowing data to be sent and received simultaneously. This can improve efficiency in data transfer.

### Cons

**High Pin Count:** SPI requires 4 primary signals, which seems more complex than protocols like UART and I2C.

## Struggling with SPI

**Synchronization Issues between Raspberry Pi 4 and AVR MC:**

- **Clock Speed Mismatch:** The Raspberry Pi 4 and AVR microcontrollers have different clock speeds and timing requirements, leading to synchronization problems. The Raspberry Pi 4 operates at much higher frequencies compared to AVR microcontrollers, which caused timing mismatches and data corruption during communication.

We have struggled with SPI protocol between AVR and Raspberry pi, as the Raspberry operates at higher data rate than the AVR MC, 1 MHZ SPI speed in Raspberry Pi is used to operate, the CHOL and CPOL have been set in both Raspberry Pi and AVR MC, **so how did we solve this issue?**

This issue was solved by **software handling**, as 3 values were sent by raspberry pi (Control option, Motor 1 Speed, Motor 2 Speed), and on the other hand, AVR MC is sending ‘k’ in order to confirm transmission and forward ultrasonic flag and backward ultrasonic flag, the problem was the mismatching in data sent order, so we made sure that the two controllers will ignore the incoming data if it is not in that specific order, as we expect that the data sent to raspberry pi would be ‘k’ and ‘0’ or ‘1’ and ‘2’ or ‘3’ → 0, 1 Forward Ultrasonic flag, 2,3 are Backward Ultrasonic flag.

```
if(SPI_FLAG_Check()==1){
SPI_CHK_1=SPI_transceive('k');  // Option
SPI_CHK_2=SPI_transceive(ultra1_flag); // '0' or '1'
SPI_CHK_3=SPI_transceive(ultra2_flag);// '2' or '3'

}


if((SPI_CHK_1 ==GLOVE_CONTROL||SPI_CHK_1 ==AI_CONTROL||SPI_CHK_1 ==MOBILE_CONTROL) //
&& (SPI_CHK_2 >=Speed1_MIN && SPI_CHK_2 <=Speed1_MAX )          // if 1st motor speed
&& (SPI_CHK_3 >=Speed2_MIN && SPI_CHK_3 <=Speed2_MAX)){     // if 2nd motor speed rar
```

*Figure 43*

## SPI Mechanism

SPI (Serial Peripheral Interface) communication involves the use of shift registers for sending and receiving data between master and slave devices. Shift registers are crucial in SPI as they allow the synchronous serial data transfer, which is the essence of the protocol.

**Master Device:** Initiates and controls the data transfer by generating the clock signal.

**Slave Device(s):** Respond to the master's clock and data signals.

Shift registers in SPI

Shift registers play a key role in SPI data transmission by serially shifting data in and out bit by bit in sync with the clock signal.

At the start of a communication cycle, both the master and the slave load their data to be transmitted into their respective shift registers.

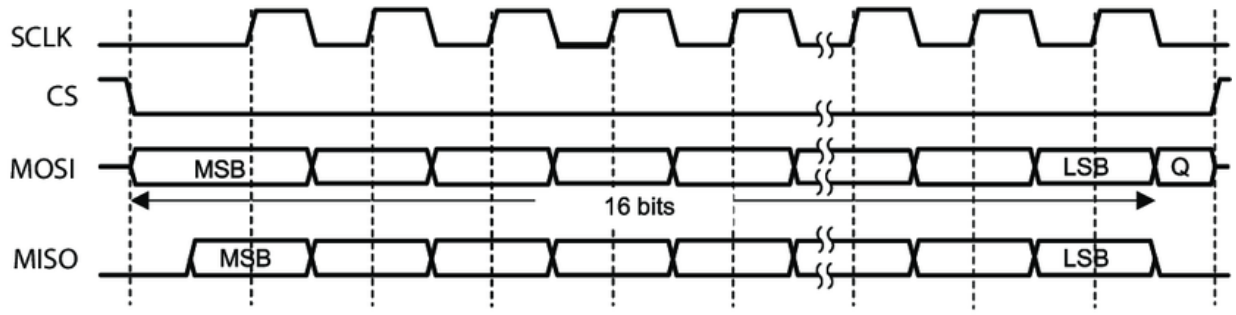
The master generates the clock signal (SCLK). Each clock pulse causes both the master and the slave shift registers to shift their data by one bit.

On each clock pulse:

The master sends a bit from its shift register to the MOSI line.

Simultaneously, the slave sends a bit from its shift register to the MISO line.

The data on the MOSI and MISO lines is read and shifted into the receiving shift registers of the slave and master, respectively.




Note: Q is undefined.

*Figure 44*

SPI Connection Diagram

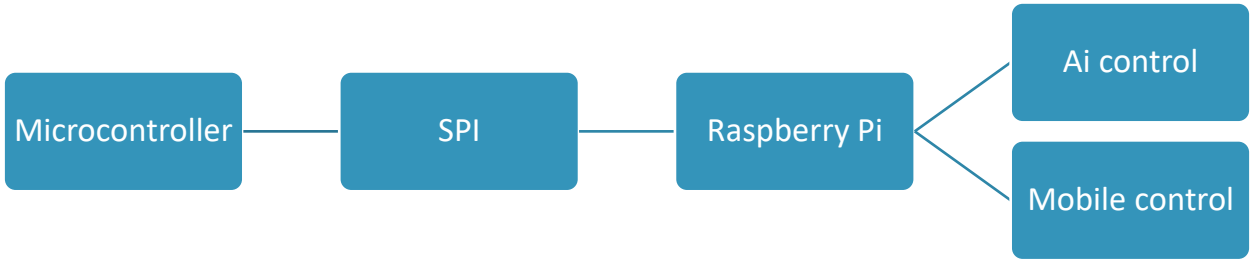



*Figure 45*

SPI connection provides communication between Raspberry Pi and the AVR MC, Raspberry is used at this point to receive data from a server, shown in the 2.1 Figure.

A mobile application is provided as an interface to the robot, the app contains a UI page showing the control modes, when the user press a specific option, it is sent through the server to the Raspberry and the SPI transmission is established, the AVR now receives the option, and is waiting for the two motors speeds (Corner Case: If glove option is used, two default speeds are sent and then are ignored in the MC), the user can send the option just for once, and then can send speeds all the time, as the option is reserved as a global variable in Raspberry, and won't be changed unless the user changes option.

Now after the control options UI page is shown, the user can switch to the Joystick page, additionally showing video feedback from the Raspberry Pi camera, so the user now can control the robot movement smoothly (Mobile control option), if Ai control mode is shown, the Ai models are inserted and starting to apply the technique onto the Raspberry Pi camera, providing detection, recognition and tracking technologies.

So, what is going to happen if the user chose the Glove control mode?

Bluetooth and UART Protocols take over…

# UART Communication Protocol:

## AVR USART vs. AVR UART

The USART is fully compatible with the AVR UART regarding:
• Bit locations inside all USART Registers
• Baud Rate Generation
• Transmitter Operation
• Transmit Buffer Functionality
• Receiver Operation

## UART Specifications

- **Serial**
- **Full Duplex**
- **Asynchronous**
- **Peer to Peer**

## UART HW connection

**Each node has a line called Tx (Transmission line) and another one called Rx (Receive Line). The Tx of one node shall be connected to Rx of the other node and vice versa.**

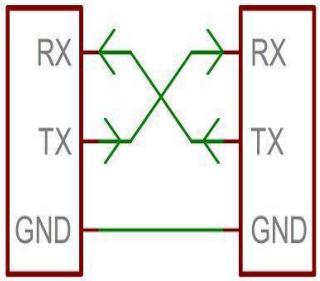



*Figure 46*

## UART frame format

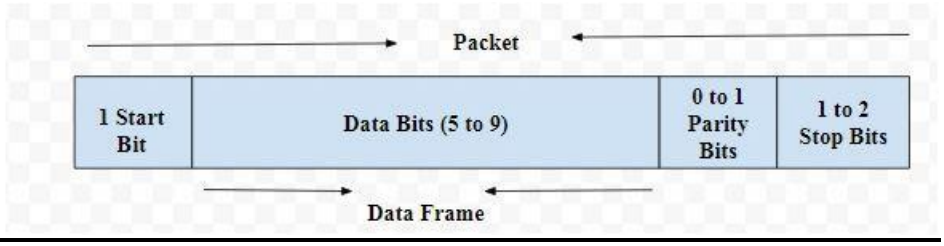



*Figure 47*

•**Start bit**: 1 bit indicates the start of a new frame, always logical low.
• **Data: 5** to **9** bits of sent data.
• **Parity bit:** 1 bit for error checking
•**Even parity:** clear parity bit if number of 1s sent is even.
•**Odd parity:** clear parity bit if number of 1s sent is odd.
•**Stop bit: 1 or 2 bits indicate end of frame, always logic high.**

## UART AVR properties

•Full Duplex Operation (Independent Serial Receive and Transmit Registers).

•Asynchronous or Synchronous Operation.

•Master or Slave Clocked Synchronous Operation.

•High Resolution Baud Rate Generator.

•Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits.

•Odd or Even Parity Generation and Parity Check Supported by Hardware.

•Data Over Run Error Detection.

•Frame and Parity error Detection.

•Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter.

•Three Separate Interrupts on TX Complete, TX Data Register Empty, and RX Complete.

•Double Speed Asynchronous Communication Mode.

## Steps to Set UART peripheral:

1.Set the baud rate in both devices.

2.Set the number of data bits, which needs to be sent/ received, number of stop bits , optional parity bit .

3.Enable the transmitter/receiver according to the desired usage.

4.UART Can be used in Polling or Interrupt. If used in interrupt enable the interrupt bits.

**Advantages**

•Only uses two wires

•No clock signal is necessary

•Has a parity bit to allow for error checking

•The structure of the data packet can be changed as long as both sides are set up for it.

•Well documented and widely used method

**Disadvantages**

•The size of the data frame is limited to a maximum of 9 bits

•Doesn’t support multiple slave or multiple master systems

•The baud rates of each UART must be within 10% of each other

## USART Block Diagram

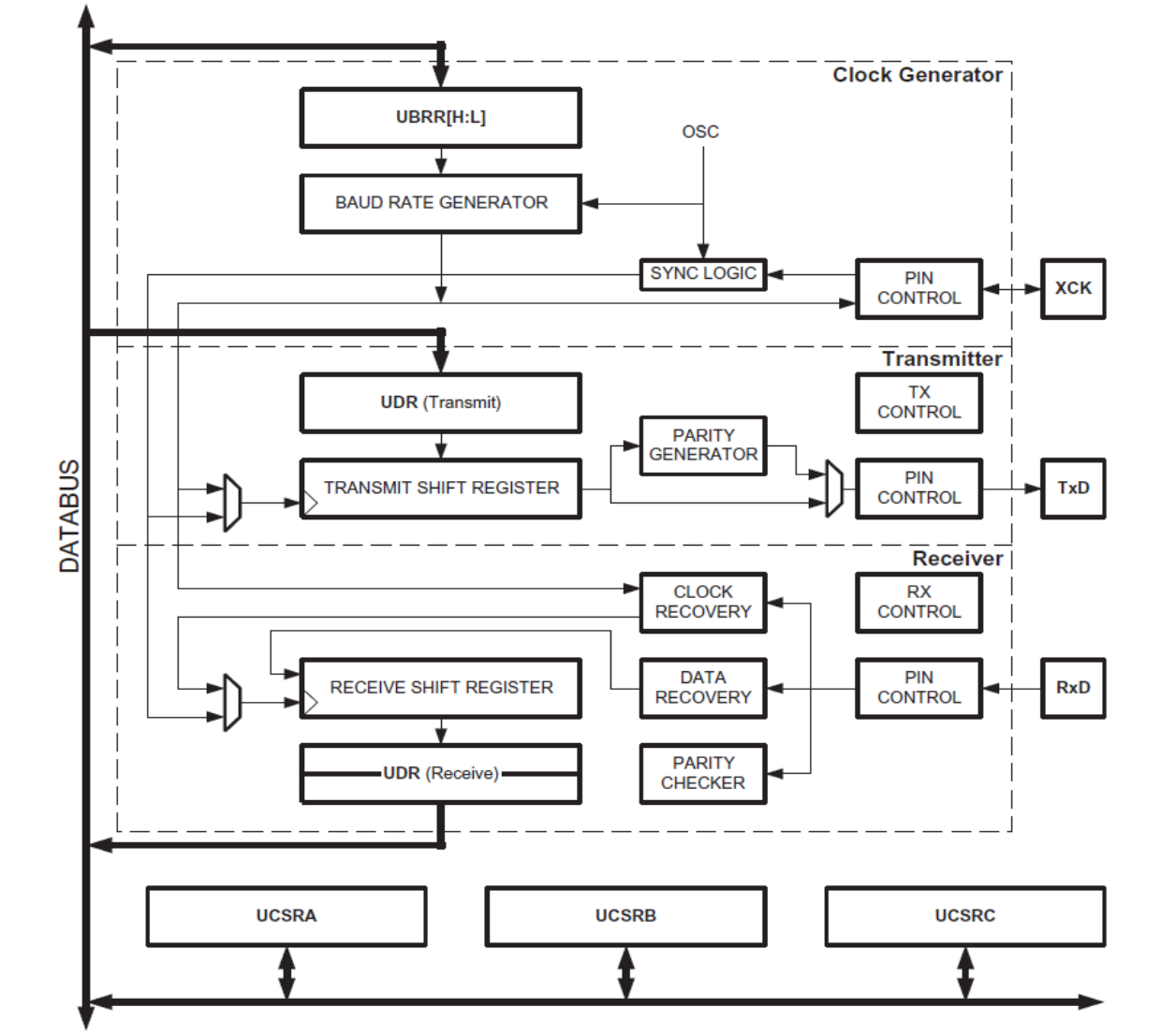



*Figure 48*

The dashed boxes in the block diagram separate the three main parts of the USART (listed from the top): Clock Generator, Transmitter and Receiver. Control Registers are shared by all units. The clock generation logic consists of synchronization logic for external clock input used by synchronous, slave operation, and the baud rate generator. The XCK (Transfer Clock) pin is only used by Synchronous Transfer mode. The Transmitter consists of a single write buffer, a serial Shift Register, parity generator and control logic for handling different serial frame formats. The write buffer allows a continuous transfer of data without any delay between frames. The Receiver is the most complex part of the USART module due to its clock and data recovery units. The recovery units are used for asynchronous data reception. In addition to the recovery units, the receiver includes a parity checker, control logic, a Shift Register and a two level receive buffer (UDR). The receiver supports the same frame formats as the transmitter, and can detect frame error, data overrun and parity errors.

## Clock Generation

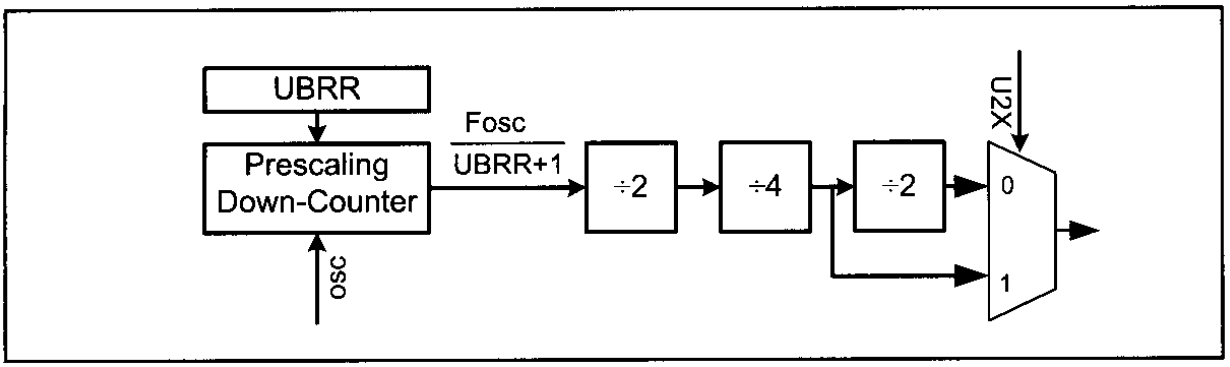



*Figure 49*

In calculating the baud rate we have used the integer number for the UBRR register values because AVR microcontrollers can only use integer values. By dropping the decimal portion of the calculated values, we run the risk of introducing error into the baud rate. There are several ways to calculate this error. One way would be to use the following formula.

Error = (Calculated value for the UBRR - Integer part) / Integer part for example, with XTAL = 8 MHz and U2X = 0 we have the following for

the 9600 baud rate:

UBRR value = (500,000/9600)-1-52.08-151.0851

Error (51.08-51)/51 = 0.16%

Another way to calculate the error rate is as follows:

Error = (Calculated baud rate - desired baud rate) / desired baud rate

Where the desired baud rate is calculated using X = (Fosc / (16(Desired

Baud rate))) - 1, and then the integer X (value loaded into UBRR reg) is used for

the calculated baud rate as follows:

Calculated baud rate = FOSC /(16(X+1)) (for U2X = 0)

For XTAL = 8 MHz and 9600 baud rates, we got X = 51. Therefore, we get the calculated baud rate of 8 MHz/(16(51+1)) = 9765. Now the error is calculated as follows:

Error = (9615-9600)/9600 = 0.16%

## Calculate UBRR for 38400 baud rates

Substitute XTAL with 16 MHz and baud rate with 38400 into the formula:

$UBRR$= ((16×10000000)/ (16×38400)) −1

= (10000000/38400) −1

$UBRR$=26.04−1

$UBRR$=25

# AVR Baud Rate Tables

| 12 Mhz | | |
|---|---|---|
| Baud | UBRR | % of error |
| 300 | 2499 | 0.0 |
| 600 | 1249 | 0.0 |
| 1200 | 624 | 0.0 |
| 2400 | 312 | 0.2 |
| 4800 | 155 | 0.2 |
| 9600 | 77 | 0.2 |
| 14400 | 51 | 0.2 |
| 19200 | 38 | 0.2 |
| 28800 | 25 | 0.2 |
| 38400 | 19 | 2.4 |
| 57600 | 12 | 0.2 |
| 76800 | 9 | 2.4 |
| 115200 | 6 | 7.5 |

| 12.288 Mhz | | |
|---|---|---|
| Baud | UBRR | % of error |
| 300 | 2559 | 0.0 |
| 600 | 1279 | 0.0 |
| 1200 | 639 | 0.0 |
| 2400 | 319 | 0.0 |
| 4800 | 159 | 0.0 |
| 9600 | 79 | 0.0 |
| 14400 | 52 | 0.6 |
| 19200 | 39 | 0.0 |
| 28800 | 26 | 1.2 |
| 38400 | 19 | 0.0 |
| 57600 | 12 | 2.5 |
| 76800 | 9 | 0.0 |
| 115200 | 6 | 5.0 |

| 14.318181 Mhz | | |
|---|---|---|
| Baud | UBRR | % of error |
| 300 | 2982 | 0.0 |
| 600 | 1490 | 0.0 |
| 1200 | 745 | 0.0 |
| 2400 | 372 | 0.0 |
| 4800 | 185 | 0.2 |
| 9600 | 92 | 0.2 |
| 14400 | 61 | 0.2 |
| 19200 | 46 | 0.8 |
| 28800 | 30 | 0.2 |
| 38400 | 22 | 1.3 |
| 57600 | 15 | 3.0 |
| 76800 | 11 | 3.0 |
| 115200 | 7 | 3.0 |

| 14.746 Mhz | | |
|---|---|---|
| Baud | UBRR | % of error |
| 300 | 3071 | 0.0 |
| 600 | 1535 | 0.0 |
| 1200 | 767 | 0.0 |
| 2400 | 383 | 0.0 |
| 4800 | 191 | 0.0 |
| 9600 | 95 | 0.0 |
| 14400 | 63 | 0.0 |
| 19200 | 47 | 0.0 |
| 28800 | 31 | 0.0 |
| 38400 | 23 | 0.0 |
| 57600 | 15 | 0.0 |
| 76800 | 11 | 0.0 |
| 115200 | 7 | 0.0 |

| 16 Mhz | | |
|---|---|---|
| Baud | UBRR | % of error |
| 300 | 3332 | 0.0 |
| 600 | 1666 | 0.0 |
| 1200 | 832 | 0.0 |
| 2400 | 416 | 0.1 |
| 4800 | 207 | 0.2 |
| 9600 | 103 | 0.2 |
| 14400 | 68 | 0.6 |
| 19200 | 51 | 0.2 |
| 28800 | 34 | 0.8 |
| 38400 | 25 | 0.2 |
| 57600 | 16 | 2.1 |
| 76800 | 12 | 0.2 |
| 115200 | 8 | 3.7 |

| 18 Mhz | | |
|---|---|---|
| Baud | UBRR | % of error |
| 300 | 3749 | 0.0 |
| 600 | 1874 | 0.0 |
| 1200 | 937 | 0.1 |
| 2400 | 468 | 0.1 |
| 4800 | 233 | 0.2 |
| 9600 | 116 | 0.2 |
| 14400 | 77 | 0.2 |
| 19200 | 58 | 0.7 |
| 28800 | 38 | 0.2 |
| 38400 | 28 | 1.0 |
| 57600 | 19 | 2.4 |
| 76800 | 14 | 2.4 |
| 115200 | 9 | 2.4 |

| 18.432 Mhz | | |
|---|---|---|
| Baud | UBRR | % of error |
| 300 | 3839 | 0.0 |
| 600 | 1919 | 0.0 |
| 1200 | 959 | 0.0 |
| 2400 | 479 | 0.0 |
| 4800 | 239 | 0.0 |
| 9600 | 119 | 0.0 |
| 14400 | 79 | 0.0 |
| 19200 | 59 | 0.0 |
| 28800 | 39 | 0.0 |
| 38400 | 29 | 0.0 |
| 57600 | 19 | 0.0 |
| 76800 | 14 | 0.0 |
| 115200 | 9 | 0.0 |

| 20 Mhz | | |
|---|---|---|
| Baud | UBRR | % of error |
| 300 | 4166 | 0.0 |
| 600 | 2082 | 0.0 |
| 1200 | 1041 | 0.0 |
| 2400 | 520 | 0.0 |
| 4800 | 259 | 0.2 |
| 9600 | 129 | 0.2 |
| 14400 | 86 | 0.2 |
| 19200 | 64 | 0.2 |
| 28800 | 42 | 0.9 |
| 38400 | 32 | 1.4 |
| 57600 | 21 | 1.4 |
| 76800 | 15 | 1.7 |
| 115200 | 10 | 1.4 |

*Figure 50*

## Frame Error (FE), Parity Error (PE).

When the AVR USART receives a byte, we can check the parity bit and stop bit. If the parity bit is not correct, the AVR will set PE to one, indicating that an parity error has occurred. We can also check the stop bit. As we mentioned before, the stop bit must be one, otherwise the AVR would generate a stop bit error and set the FE flag bit to one, indicating that a stop bit error has occurred. We can check these flags to see if the received data is valid and correct. Notice that FE and PE are valid until the receive buffer (UDR) is read. So we have to read FE and PE bits before reading UDR. You can explore this on your own.

## USART Initialization

The USART has to be initialized before any communication can take place. The initialization process
normally consists of setting the baud rate, setting frame format and enabling the Transmitter or the Receiver depending on the usage. For interrupt driven USART operation, the Global Interrupt Flag should be cleared (and interrupts globally disabled) when doing the initialization.
Before doing a re-initialization with changed baud rate or frame format, be sure that there are no ongoing transmissions during the period the registers are changed. The TXC Flag can be used to check that the Transmitter has completed all transfers, and the RXC Flag can be used to check that there are no unread data in the receive buffer. Note that the TXC Flag must be cleared before each transmission (before UDR is written) if it is used for this purpose.
The following simple USART initialization code examples show one assembly and one C function
that are equal in functionality. The examples assume asynchronous operation using polling (no interrupts enabled) and a fixed frame format. The baud rate is given as a function parameter. For the assembly code, the baud rate parameter is assumed to be stored in the r17:r16 registers. When the function writes to the UCSRC Register, the URSEL bit (MSB) must be set due tothe sharing of I/O location by UBRRH and UCSRC

```
void UART_Init(void)
{
    // Configure RX pin (PORTD_PIN0) as input and TX pin (PORTD_PIN1) as output
    DIO_INIT(PORT_D, PORTD_PIN0, IN);
    DIO_INIT(PORT_D, PORTD_PIN1, OUT);

    // Enable RX complete interrupt and receiver/transmitter
    UCSRB |= 0b10011000;

    // Set frame format: 8 data bits, 1 stop bit
    UCSRC |= 0b10000110;

    // Set baud rate to 38400
    UBRRL = 25;
}
```

*Figure 51*

 **Pin Configuration**: DIO_INIT (PORT_D, PORTD_PIN0, IN); sets the RX pin (PORTD_PIN0) as input. DIO_INIT (PORT_D, PORTD_PIN1, OUT); sets the TX pin (PORTD_PIN1) as output.
 **Enable UART and Interrupt**: UCSRB |= 0b10011000; enables the receiver, transmitter, and RX complete interrupt.
 **Frame Format**: UCSRC |= 0b10000110; sets the frame format to 8 data bits and 1 stop bit.
 **Baud Rate**: UBRRL = 25; sets the baud rate to 38400. This value is pre-calculated based on the system clock and desired baud rate. encounters the null character '\0', sending each byte in turn.

**Receiving a Byte**

```
uint8 UART_ReceiveByte(void)
{
    // Wait for data to be received
    while(GET_BIT(UCSRA, 7) == 0);

    return UDR;
}
```

*Figure 52*

 **Wait for Data**: while (GET_BIT (UCSRA, 7) == 0); waits until data is received (RXC flag in UCSRA is set).
 **Return Data**: The function then returns the received byte from the UDR.

# Bluetooth:

## Background

Introduction: It's a technology used for short range and wireless communication approximately 10 meters there is a newer version of it called BLE (Bluetooth low energy) it uses lower energy and BTW it offers lower speed, this technology uses 2.4GHZ similar to Wi-Fi, both also use Radio waves for high-speed data transfer over short range but of course the key difference between them is the range they cover and this will appear in the applications they are used in.

## Advantages of Bluetooth

- Its wireless technology so this is one of its advantages this makes connections between devices are fast and easy.
- Signals sent can go through walls, so the material of the wall plays a significant role in how well a Bluetooth signal strength.
- Its dominant and accepted technology used in cars, phones, pcs and small projects like ours.
- They avoid connection interference between other devices.
- They can be commonly used in PAN (Personal Area Network).

## Disadvantages of Bluetooth

- Slower compared to other wireless technology like WI-FI.
- security issues and this is the key disadvantage.
- It can lose connection in certain conditions.
- short range communication between devices.

## Architecture of Bluetooth

The architecture of Bluetooth defines two types of networks:

- Piconet is a type of Bluetooth network that contains one primary node called the master node and seven active secondary nodes called slave nodes. Thus, we can say that there is a total of eight active nodes which are present at 10 meters. The communication between the primary and secondary nodes can be one-to-one or one-to-many. Possible communication is only between the master and slave; Slave-slave communication is not possible. It also has 255 parked nodes; these are secondary nodes and cannot take participation in communication unless it gets converted to the active state.
- Scatternet

  It is formed by using various piconets. A slave that is present in one piconet can act as master or we can say primary in another piconet. This kind of node can receive a message from a master in one piconet and deliver the message to its slave in the other piconet where it is acting as a master. This type of node is referred to as a bridge node. A station cannot be mastered in two piconets.

## Bluetooth Module

- It is a basic circuit set of chips which integrated Bluetooth functions there are 2 types of modules HC-05 and HC-06.
- Also, the number associated with the name of the module indicates that the second is better but in fact HC-05 supports both sending and receiving data while HC-06 supports only receiving.
- The most common communication protocol used between microcontroller and Bluetooth module is (UART) there are some other modules that uses I2C for example like RN4020, but we are using HC-05 any way.

## Bluetooth module Pin Description

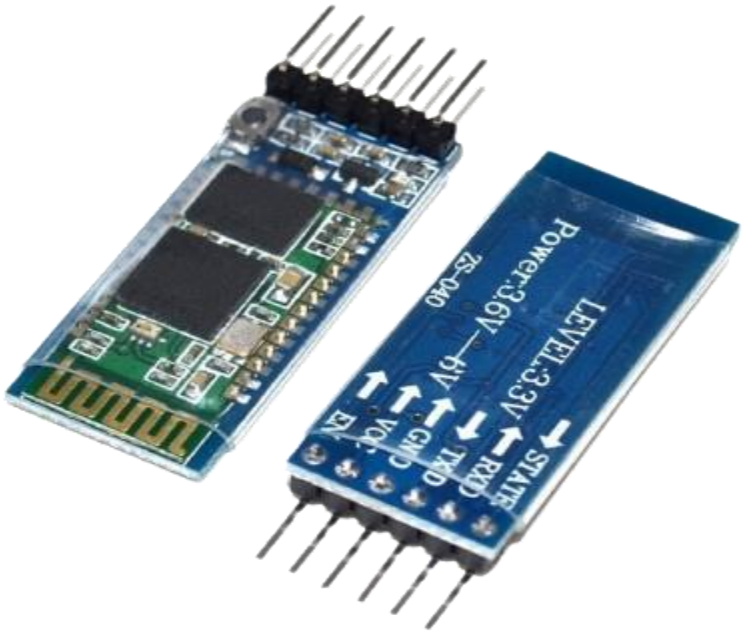



*Figure 53*

- EN: Used for toggling between AT mode (the mode used for Bluetooth module configuration) and this can be done by setting this Pin to high voltage, and Data mode.
- VCC: Used as an input for voltage suppliers usually 5v to 3.3.
- GND: Used as ground pin.
- TXD: The data that been received via the Bluetooth module will be transmitted via serial communication via this bin to the microcontroller for example.
- RXD: The data that the Bluetooth module required to transmit to the outer space will be received via this pin.
- STATE: The state pin is connected to on board LED, it can be used as feedback to check if Bluetooth is working properly.

## Bluetooth module configuration (AT Commands):

- Material needed for configuration:
  - 2 HC-05 one for slave and the other as master
  - Five cables male to female
  - Arduino
- Here are the steps for connection:
  - Connect EN to 3.3V.
  - VCC to 5v.
  - GND to ground.
  - TX to TX (PIN 1 UNO).
  - RX to RX (PIN 0 UNO).
  - STATE can be free.

We can now see that the HC-05 onboard LED blinks slowly after bug-in the USB cable with Arduino indicates that we are in the AT configuration mode, there are two Roles for Bluetooth module:

Slave and Master, by default all modules are slaves and our goal is to set one of them as master.

- Slave: all what we need to do is to know the address of the slave and ensure that this module is a slave and connected correctly.

  AT: By sending this command via serial monitor in Arduino IDE the response shall be OK indicates everything is correct.

  AT+ROLE? the response shall be 0 if the module is set as slave or 1 if set as master in our case its 0.

  AT+ADDR? Module will return its address we can use later for connection.

- Master: all what we need to do is to change the rule of the module to master and assign an address to it.

  AT+ROLE=1: By that we change the role from slave to master.

  AT+CMODE? If the return of this command was 1 that indicates that this module can connect to any near module, so we need to change that to make it connect to desired one.

  AT+CMODE=0: Now it can connect to the desired one we have its address previously.

  AT+BIND=[address]: we can now assign the address for example 98D3,32,208B64.

After this step we are free to transfer data via Bluetooth, but we should first disable AT mode by connecting EN PIN to ground and connecting TX to Rx pin and RX to Tx pin.

- A simple project to Turn ON/OFF VIA Bluetooth:

  This was done by using virtual Serial port Driver to connect pairs between com1 and com2

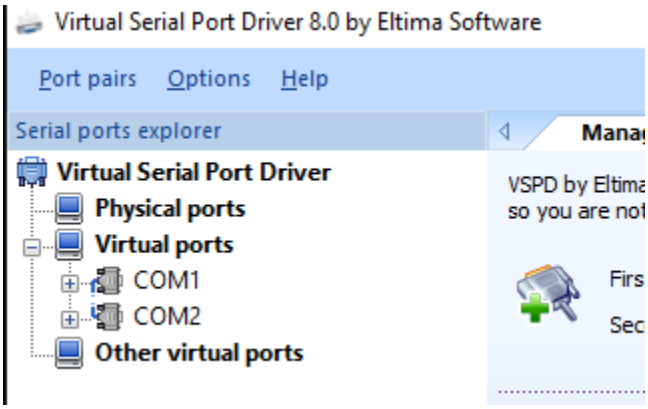



*Figure 54*

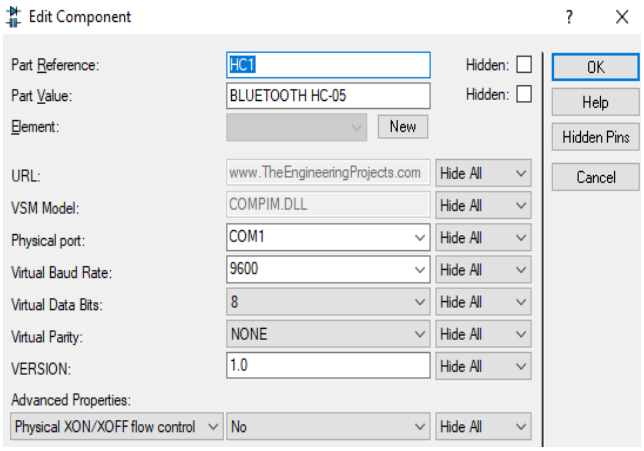



*Figure 55*

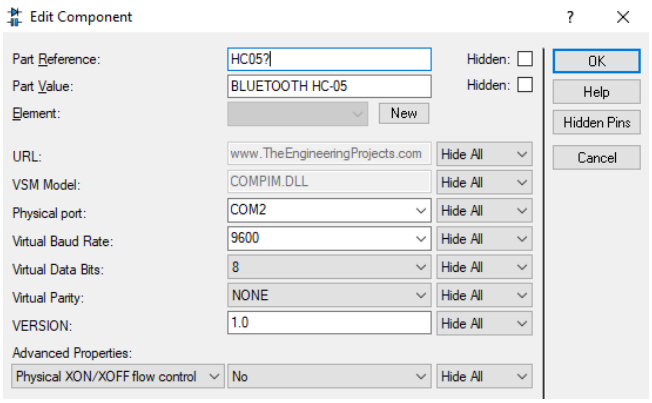



*Figure 56*

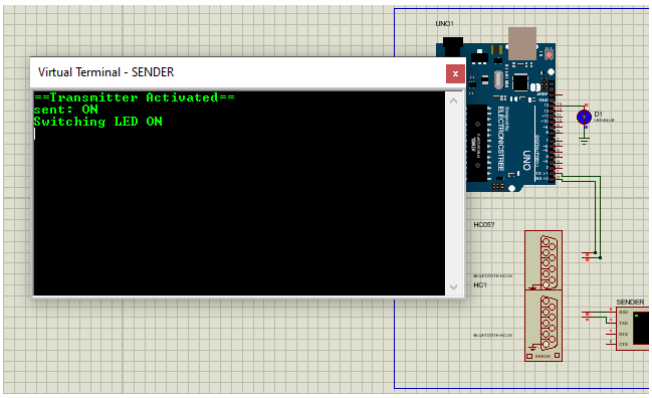



*Figure 57*

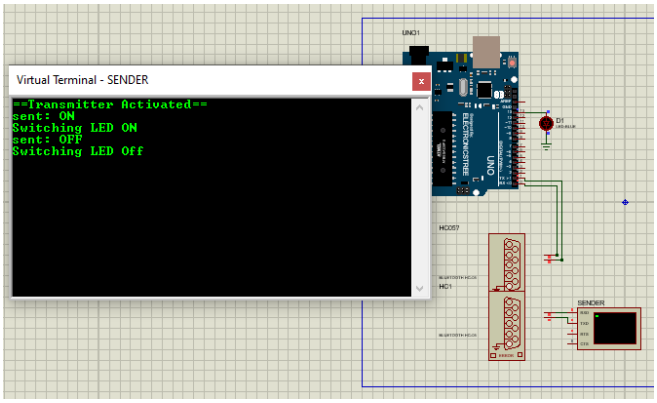



*Figure 58*

This project is simple, but it required some search that I think a good step to be mentioned.

- Code:

```
#define LED 13
void setup() {
  pinMode(LED,OUTPUT);
  Serial.begin(9600);
  Serial.println("==Transmitter Activated==");
}

void loop() {
  if(Serial.available()>0){
    String x =Serial.readString();
    Serial.println("sent: "+ x);
    if(x.equalsIgnoreCase("OFF")){
      Serial.println("Switching LED Off");
      digitalWrite(LED,LOW);
    }else if(x.equalsIgnoreCase("ON")){
      Serial.println("Switching LED ON");
      digitalWrite(LED,HIGH);
    }else{
      Serial.println("ERROR");
```

# Chapter5: Glove Implementation

## Flex Sensor:

### Introduction:

A flex sensor is a device used to measure the amount of bending over it. This can be used for various applications that track physical activities, for example pending hand to let a certain event occur.

### Pin Description:

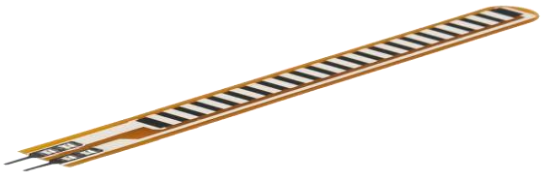


*Figure 59*

As we can see this sensor has 2 pins like all resistance we know, one can be connected to voltage supplier, and one connected to regular resistance then connected it to the microcontroller.

### Working Principle:

Flex sensor is based on the change in resistance. When the sensor is straight, it has a baseline resistance. When it is bent, the conductive material within the sensor stretches, increasing its resistance. This change in resistance can be measured by connecting voltage over the voltage pin so it simply follows voltage divider rule.

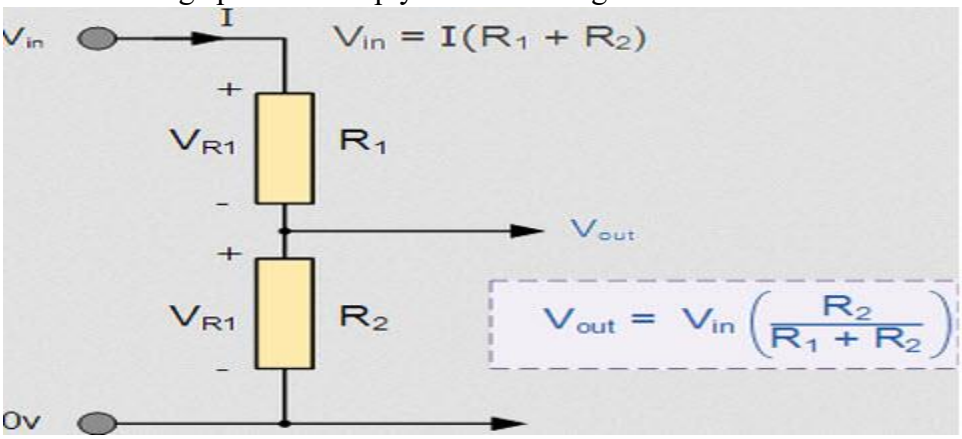



*Figure 61*

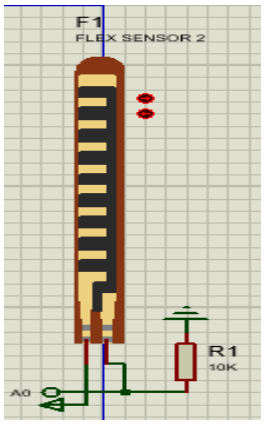



*Figure 60*

Assuming that R1 is the Flex Sensor to Fit the draw on the following schematic

## Pairing the HC-05

Anyone tinkering with the HC-05 Bluetooth module will eventually have to use what are called “AT commands”. AT commands are simple text-based commands that follow the “AT” prefix (“AT” meaning ‘attention’). These commands are similar to the ones used with old modem devices. By sending specific AT commands to the HC-05 module, you can configure it or retrieve basic information about it, such as its name, baud rate, PIN code, role, and so on.

### Configuring the HC-05 Module

To configure the HC-05 module, we put it into command mode and send AT commands over the UART port. While in command mode, any ASCII bytes we send are interpreted as commands. After making changes, we restart the module to apply the settings. When we change the configuration parameters, they are saved until we change them again or perform a factory reset.

### Connecting HC-05 Module to a PC

When it comes to connecting the HC-05 module to a PC, we have two options.

-We can use a USB to TTL converter, directly establishing a connection between the HC-05 module and your PC. This converter serves as a bridge between the USB port on your PC and the TTL-level serial signals that the HC-05 module understands.

-Alternatively, you can use the Arduino as a medium between the PC and the HC-05 module. In this configuration, the Arduino serves as a translator, relaying commands and data between the PC and the HC-05 module.

### With USB to TTL Converter

Let’s connect HC-05 module to PC using a USB to TTL converter.

Connecting the HC-05 Module to a USB to TTL converter is as easy as applying power and wiring up the serial Rx and Tx pins. The Tx of the HC-05 module is connected to the converter’s RXD, the Rx to TXD, GND to GND, and VCC to 5V.

Note that the Rx pin on the HC-05 module is not 5V tolerant. So, if you are using a USB-to-TTL converter that is operating at 5V I/O levels, you need to step down the Tx signal from the converter to 3.3V.


*Figure 62*

### With Arduino to TTL Converter

Similarly, we can connect the HC-05 Module to an Arduino. The TXD of the HC-05 module is connected to the Arduino's D1, the RXD to D0, GND to GND, and VCC to 5V.

Again! The Rx pin on the HC-05 module steps down the Tx signal from the Arduino to 3.3V using a resistor divider. A 1K resistor between HC-05's Rx and Arduino's D0, and a 2K resistor between HC-05's Rx and GND.


*Figure 63*

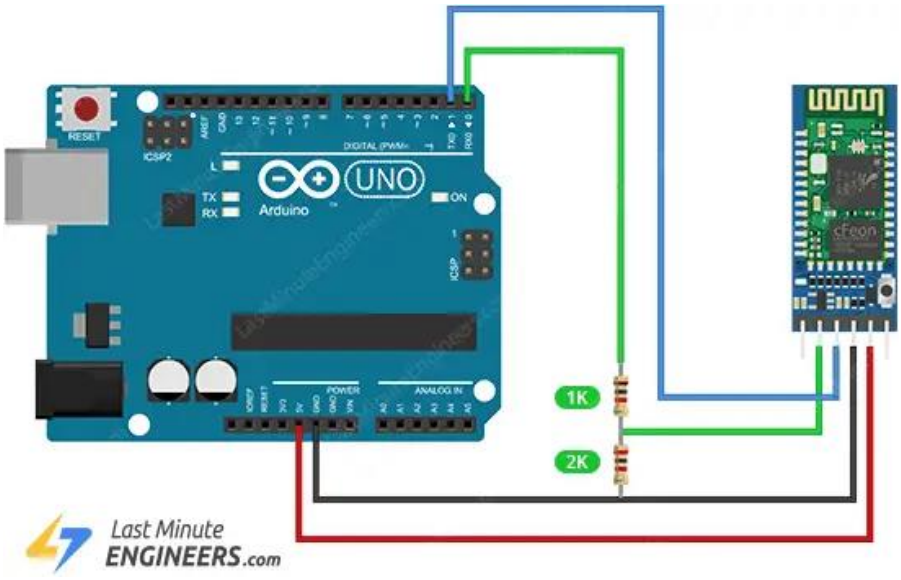



*Figure 64*

## Sending AT Commands

To send AT commands, we will require a Windows terminal program installed on your PC that can communicate with the HC-05 module over UART. For this purpose, we recommend using the “Arduino Serial Monitor”.

-The slave HC-05 module role has to be set to 0 and the address is required in order to bind the master to the slave.

The commands used:

AT +ROLE = 0

AT +ADDR?

AT+UART =38400

-The master HC-05 module role has to be set to 1 at first, then we start to search for the available Bluetooth modules in order to see that if the slave HC-05 is seen by the master or not, if seen, we take the slave address and start to establish the pairing between the two HC05.

The commands used:

AT +ROLE = 1

AT+UART =38400

AT +INQ?

AT +BIND = “Slave Address”

AT +LINK =” Slave Address”

## Flex sensors values mapping

At first, we read the flex sensors values using Arduino's ADC, using analog pins and furthermore analogread () function in Arduino IDE, we used two 37k resistors, as when a flex sensor bends, its resistance changes. By connecting a fixed resistor in series with the flex sensor and supplying a voltage across this combination, we create a voltage divider. The output voltage from the voltage divider is then read by an analog-to-digital converter (ADC) on a microcontroller.

This setup allows the microcontroller to interpret the resistance change as a corresponding change in voltage.

The issue we met is that the flex sensor values can range over 255, and the AVR Microcontroller registers are just 1-byte registers, so we had to do some mapping into the flex values, so we started to test each flex sensor and starting to extrapolate the values range for each flex sensor, and according to that, we built our mapping technique.

```
 /************ Flex 1 "44"*****************************/
if(filter1_avg >100 &&filter1_avg < 110){
 data[0] ='1';


}else if((filter1_avg >=110 &&filter1_avg < 115)){
 data[0] ='2';
}else if((filter1_avg >=115 &&filter1_avg < 120)){
 data[0] ='3';
}
else if((filter1_avg >= 120 &&filter1_avg < 135)){
 data[0] ='4';
}
else if(filter1_avg >=135 ){
 data[0] ='5';
}else {
 data[0] ='0';
}
```

*Figure 65*

# Moving Average Filter Implementation

## Overview

In the glove control system of the robot, flex sensors are used to detect finger movements, which are then translated into control commands for the motors. However, raw sensor readings can be noisy, leading to inaccurate control signals. To mitigate this, a filtering mechanism is implemented to smooth out the data.

Flex sensors are resistive sensors that change resistance based on the amount of bend. When used in the glove control system, these sensors provide variable voltage readings corresponding to finger positions. However, sensor readings can be affected by various types of noise, such as electromagnetic interference, which can lead to fluctuations and inaccuracies.

Digital filtering is essential in signal processing to remove unwanted noise and improve the accuracy of sensor data. Among various filtering techniques, the moving average filter is widely used due to its simplicity and effectiveness in smoothing out short-term fluctuations. In the glove control system, a moving average filter is implemented to stabilize the flex sensor readings.

The moving average filter is implemented by maintaining an array of recent sensor readings and calculating the average of these values. In the code, two arrays flex1_filter and flex2_filter are used to store the readings from two flex sensors. The filter arrays are initialized to zero to avoid garbage data. During each iteration, the current sensor reading is added to the filter array, and the total sum is updated. Once the array is full, the average is calculated, and the index is reset.

## Why using a filter?

In the motor speed control application for the glove interface, it was observed that the raw data from the flex sensors exhibited noticeable fluctuations and distortions. For instance, a sequence of sensor readings such as 100, 98, 99, 75, and 95 could result in erratic motor speeds where the value 75 introduces a significant deviation from the intended speed. To address this issue, a moving average filter was implemented to stabilize the motor speed command and enhance the system’s performance.

The moving average filter operates by computing the average of a set number of recent sensor readings to smooth out fluctuations and provide a more stable control signal. In this case, the filter takes the average of five consecutive flex sensor readings to mitigate the impact of anomalous values and deliver a command that better reflects the user’s intended input.

After implementing the filter, from the previous values, the motor speed would get 94 value which is closer and more noiseless than 75.

```
/*************Filter1*****************************/
flex1_filter[filter1_index] =Flex1_Val; // assign FLEX1 value to the filter array
total_1+=flex1_filter[filter1_index];   // Adding FLEX1 values
filter1_index++;                        //Increment filter 1 index

if(filter1_index >= num_readings ){
  filter1_avg =total_1 /num_readings;   //Calcualte the average of 10 flex values
  filter1_index =0;                     // if index is bigger than the filter size, assign it to zero
  total_1 =0;
}
```

*Figure 66*

By employing the moving average filter, we achieve several key benefits:

**Noise Mitigation:** The filter averages multiple readings to reduce the effect of random fluctuations or transient noise, providing a more consistent signal.

**Data Stabilization:** By averaging the values, the filter minimizes the impact of erratic data points, resulting in smoother and more reliable control commands for the motor.

**Enhanced Accuracy:** The filter ensures that the motor speed command reflects a more accurate representation of the user's intended input by reducing the influence of isolated anomalous readings.

## UART to Bluetooth

Now, after the values has been filtered and mapped, it is time to send them to the Bluetooth module to start transmission, the only way to send data to Bluetooth from Arduino is UART, and on the other hand the slave HC05 send data to the microcontroller through UART, so the UART baud rate have been set to 38400 in both Arduino and MC.

## Car Movement via Glove control

Suppose the data has been sent properly to the Microcontroller, what next?

If 1st Flex sensor → '2' or '3' or '4' && 2nd Flex sensor →'0' (Forward Case)

If 1st Flex sensor → '0' && 2nd Flex sensor →'2' or '3' or '4' (Backward Case)

If 1st Flex sensor && 2nd Flex sensor! = '0' or '1' (Rotation Case)

In detailed, if the first flex sensor was bended, the robot will move forward, if the second flex sensor was bended, the robot will move backward, if both of them were bended the robot will rotate.

# Chapter 6: Obstacle Avoidance

## Background

Timer 1 in the ATmega32 microcontroller is a versatile 16-bit timer/counter. It can be used for a variety of timing and counting tasks, such as generating precise time delays, creating pulse-width modulation (PWM) signals, and measuring time intervals. The 16-bit resolution allows for longer timing periods and finer control compared to 8-bit timers.

### The main features

- **16-bit Resolution:** Timer 1 is 16 bits wide, allowing it to count from 0 to 65535 (2^16 - 1).
- **Multiple Modes of Operation:** It supports various modes such as Normal, CTC (Clear Timer on Compare Match), Fast PWM, and Phase Correct PWM.
- **Two Independent Output Compare Units:** allow to generate a different signal with different frequency and duty cycle.
- **Input Capture:** This feature allows precise timing of external events.
- **Prescaler:** It includes a prescaler that can divide the input clock by several factors to adjust the timing resolution and duration.
- **Interrupts:** Timer 1 can generate interrupts on compare matches, overflows, and input capture events, allowing for precise event handling.

### Operation Modes

1. **Normal Mode**: The timer counts from 0 to 65535 and then overflows, restarting from 0. Useful for simple timing operations.
2. **CTC Mode**: The timer counts from 0 to a specified value (in OCR1A or OCR1B), then resets to 0. Ideal for generating precise time intervals.
3. **Fast PWM Mode**: Generates PWM signals with a high frequency, where the duty cycle is defined by the value in OCR1A or OCR1B.
4. **Phase Correct PWM Mode**: Generates PWM signals where the timer counts up and then down, providing a symmetrical PWM waveform.

## Configuring Timer 1 Driver for AVR:

Creating a driver for Timer 1 in MCAL layer for ATmega32 involves setting up the timer for specific tasks like generating delays, PWM signals, or handling interrupts. A well-structured driver makes it easier to integrate Timer 1 into various applications.

## Driver Initialization:

Initialize the mode for timer 1 from the two registers (TCCR1A, TCCR1B) from data sheet of atmega32 we drive a function to help in use the timer mode and the scaler you want.

**Table 47.** Waveform Generation Mode Bit Description(1)

| Mode | WGM13 | WGM12 (CTC1) | WGM11 (PWM11) | WGM10 (PWM10) | Timer/Counter Mode of Operation | TOP | Update of OCR1x | TOV1 Flag Set on |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | Normal | 0xFFFF | Immediate | MAX |
| 1 | 0 | 0 | 0 | 1 | PWM, Phase Correct, 8-bit | 0x00FF | TOP | BOTTOM |
| 2 | 0 | 0 | 1 | 0 | PWM, Phase Correct, 9-bit | 0x01FF | TOP | BOTTOM |
| 3 | 0 | 0 | 1 | 1 | PWM, Phase Correct, 10-bit | 0x03FF | TOP | BOTTOM |
| 4 | 0 | 1 | 0 | 0 | CTC | OCR1A | Immediate | MAX |
| 5 | 0 | 1 | 0 | 1 | Fast PWM, 8-bit | 0x00FF | TOP | TOP |
| 6 | 0 | 1 | 1 | 0 | Fast PWM, 9-bit | 0x01FF | TOP | TOP |
| 7 | 0 | 1 | 1 | 1 | Fast PWM, 10-bit | 0x03FF | TOP | TOP |
| 8 | 1 | 0 | 0 | 0 | PWM, Phase and Frequency Correct | ICR1 | BOTTOM | BOTTOM |
| 9 | 1 | 0 | 0 | 1 | PWM, Phase and Frequency Correct | OCR1A | BOTTOM | BOTTOM |
| 10 | 1 | 0 | 1 | 0 | PWM, Phase Correct | ICR1 | TOP | BOTTOM |
| 11 | 1 | 0 | 1 | 1 | PWM, Phase Correct | OCR1A | TOP | BOTTOM |
| 12 | 1 | 1 | 0 | 0 | CTC | ICR1 | Immediate | MAX |
| 13 | 1 | 1 | 0 | 1 | Reserved | – | – | – |
| 14 | 1 | 1 | 1 | 0 | Fast PWM | ICR1 | TOP | TOP |
| 15 | 1 | 1 | 1 | 1 | Fast PWM | OCR1A | TOP | TOP |

*Figure 67*

Initialize the scaler for the timer 1 depend on the frequency of the atmega32 used in our project we used 16 MHZ crystal so we used scaler 8 for our design and we can configure the scaler from the register (TCCR1B).

**Table 48.** Clock Select Bit Description

| CS12 | CS11 | CS10 | Description |
|---|---|---|---|
| 0 | 0 | 0 | No clock source (Timer/Counter stopped). |
| 0 | 0 | 1 | $clk_{I/O}$/1 (No prescaling) |
| 0 | 1 | 0 | $clk_{I/O}$/8 (From prescaler) |
| 0 | 1 | 1 | $clk_{I/O}$/64 (From prescaler) |
| 1 | 0 | 0 | $clk_{I/O}$/256 (From prescaler) |

*Figure 68*

Initialize the output compare units mode we have two units that their mode depend on the mode of the timer that used we driver a function for every output compare unit to choose the Mode you want according to data sheet there is a three timer modes (Non-PWM-Mode , Fast-PWM-Mode , Phase- Correct-Mode ) that the output compare units modes different for them.

**Table 45.** Compare Output Mode, Fast PWM(1)

| COM1A1/COM1B1 | COM1A0/COM1B0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC1A/OC1B disconnected. |
| 0 | 1 | WGM13:0 = 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM13:0 settings, normal port operation, OC1A/OC1B disconnected. |
| 1 | 0 | Clear OC1A/OC1B on compare match, set OC1A/OC1B at TOP |
| 1 | 1 | Set OC1A/OC1B on compare match, clear OC1A/OC1B at TOP |

*Figure 70*

ase Correct and Phase and Frequency Correct

| 30 | Description |
|---|---|
| | Normal port operation, OC1A/OC1B disconnected. |
| | WGM13:0 = 9 or 14: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM13:0 settings, normal port operation, OC1A/OC1B disconnected. |
| | Clear OC1A/OC1B on compare match when up-counting. Set OC1A/OC1B on compare match when downcounting. |
| | Set OC1A/OC1B on compare match when up-counting. Clear OC1A/OC1B on compare match when downcounting. |

*Figure 71*

| cription |
|---|
| mal port operation, OC1A/OC1B connected. |
| gle OC1A/OC1B on compare match |
| ar OC1A/OC1B on compare match (Set ut to low level) |
| OC1A/OC1B on compare match (Set ut to high level) |

*Figure 69*

Handling interrupts in timer 1 we have four interrupts in timer one so to handle every one we want we made a function for everyone to enable and disable it by the (TIMSK) register and we made a set call back function to pass for here the function need to implement when interrupt happen
Our four interrupts:

**Input Capture Interrupt:** can be with rising edge and falling.
**Output Compare A Match Interrupt.**
**Output Compare B Match Interrupt.**
**Overflow Interrupt:** happen when timer count ends the count and back to zero

## Timer 1 implementation:

Timer1 is used to generate two PWM signals with different duty cycle and same frequency from the two output compare units to control the speed for the DC Motors and the rotation method for the robot and used for measurement the distance for our ultrasonic at the same time to avoid optical using input capture unit and interrupts to handle this situation in timer one.

We used the Fast-PWM-10-bit Mode to generate PWM.
Used the scaler of 8 for our atmega32 16MHZ.
Used the non-inverting mode for two outputs compare unit.

```
void Timer1_ICU_InterruptEnable(void)
{
    SET_BIT(TIMSK,TICIE1);
}
void Timer1_ICU_InterruptDisable(void)
{
    CLR_BIT(TIMSK,TICIE1);
}
```

*Figure 72*

```
void PWM_Init(void)
{
    Timer1_Init(TIMER1_FASTPWM_10BIT_MODE,TIMER1_SCALER_8);
    Timer1_OCRB1Mode(OCRB_NON_INVERTING);
    Timer1_OCRA1Mode(OCRA_NON_INVERTING);
}
```

*Figure 73*

## Ultrasonic sensors

on this part we are talking about how to protect the robot from crashing obstacles when it is moving using ultrasonic sensor that is a common technique in autonomous vehicles, robotics, and various smart systems.

An ultrasonic sensor is a device used to measure distance by using sound waves. It operates on the principle of echolocation, similar to how bats navigate and detect objects in their environment.

**Operation:**

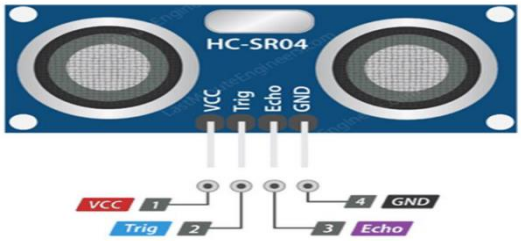



*Figure 74*

1. The sensor emits a burst of ultrasonic sound waves, typically at a frequency of around 40 kHz, which is above the range of human hearing.
2. The sound waves travel through the air until they encounter an obstacle and reflect back towards the sensor.
3. The sensor has a receiver that detects the reflected sound waves (the echo).
4. The sensor measures the time it takes for the sound waves to travel to the obstacle and back.
5. Using the speed of sound (approximately 343 meters per second in air at room temperature), the sensor calculates the distance to the obstacle using the formula:

$$\text{Distance} = \frac{Time \times Speed\ of\ sound}{2}$$

### Ultrasonic module pins

1. VCC: 5V DC.
2. Trigger: Pulse Input.
3. Echo: Pulse Output.
4. GND: 0V.

### Ultrasonic implementation:

- In our project we need to use two ultrasonic for the front direction and the back direction to avoid the robot from crashing into any obstacles.
- To measure distance, we started by choosing two GPIO Pins, one for sending trigger and the other for receiving echo.
- Both pins are either high or low, so it was better to be used in digital mode.
- To start ultrasonic, we need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8-cycle burst of ultrasound at 40 kHz and raise its echo.
- The Echo is a distance object that is pulse width and the range in proportion.
- You can calculate the range through the time interval between sending trigger signal and receiving echo signal.

### Problems observed:

- To drive the two ultrasonic or more to atmega32 the problem that we have just a one input capture unit that need to share two ultrasonic.
- Collision can happen between the two ultrasonic signals when we use one input capture unit.
- How to handle the distance calculation while the timer1 is used for generating the PWM.

### Optimization:

- First, we use a diode for every echo pin for two ultrasonic to prevent the collision of the two ultrasonic signals.
- Second, we cannot start the two ultrasonics at the same time so we can make them work in series like one of them start measurement and wait until it finishes and the next one start and so on.
-

### Ultrasonic timing diagram

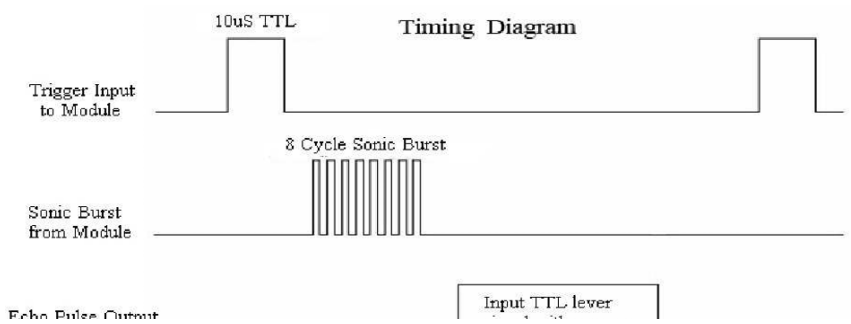



*Figure 75*

## Connection to the atmega32:

Connect the trigger pins for the two ultrasonic to: PINC0, PINC1.
Connect the echo pins for the two ultrasonic with diodes and to: PIND6 (ICU).
Connect the VCC and GND.

**Drive ultrasonic code:**

Ultrasonic drivers are part of HAL layer architecture.
Start with a function initialization to the ultrasonic by enable the input capture unit interrupt and the function that executed when ICU interrupt happen and timer1 overflow happen

```
void Ultrasonic_Init(void)
{
    Timer1_ICU_SetCallBack(f1);
    Timer1_ICU_InterruptEnable();
    Timer1_OVF_SetCallBack(f2);
}
```

*Figure 76*

Bulit a function to measure distance that needs just the number of the ultrasonic that you want to get the distance from by putting high on the trigger pin for 10us and put low.
So to detect the change of the echo pin we use the ICU interrupt at the first time with the rising edge to detect the signal when it stat from the ultrasonic and then we change the detection with the falling edge to detect when the signal is back and we using the busy wait method to wait until the signal is back but to make optimize if the signal is take so long as we know the ultrasonic can measure just to 400cm so we make counter to count in the busy wait if this counter end he break the loop and return the last distance as the distance he measure is out of range and to protect the project from stuck in this loop if the ultrasonic burned.
At the point when echo starts to release a signal at this time we get the first time for the pulse and wait until the pulse to back the ultrasonic to determine the time the pulse take but to get

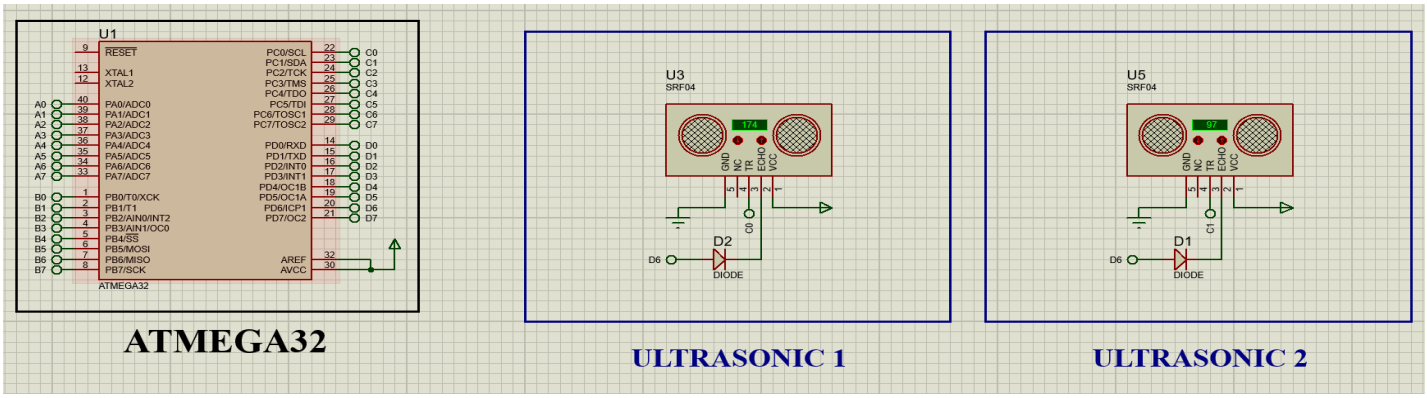



*Figure 77*

the right time as we use timer1 PWM 10-bit mode the timer is count to 1023 and then reset to 0 so what if the pulse take more than 1023 that can make a problem in calculation so we can handle this by using the timer1 overflow interrupt we can make a global counter and make the interrupt to pulse it every time overflow is happen so we can calculate the time in this time without error.

```
time = (t2-t1) + ((u32)1024*count);
d = time/116;
```

*Figure 78*

To calculate the distance, we need to divide the reading from the timer by 2 as we use 16MHZ crystal and now we can calculate the distance in CM using the equation:

## Distance equation in CM

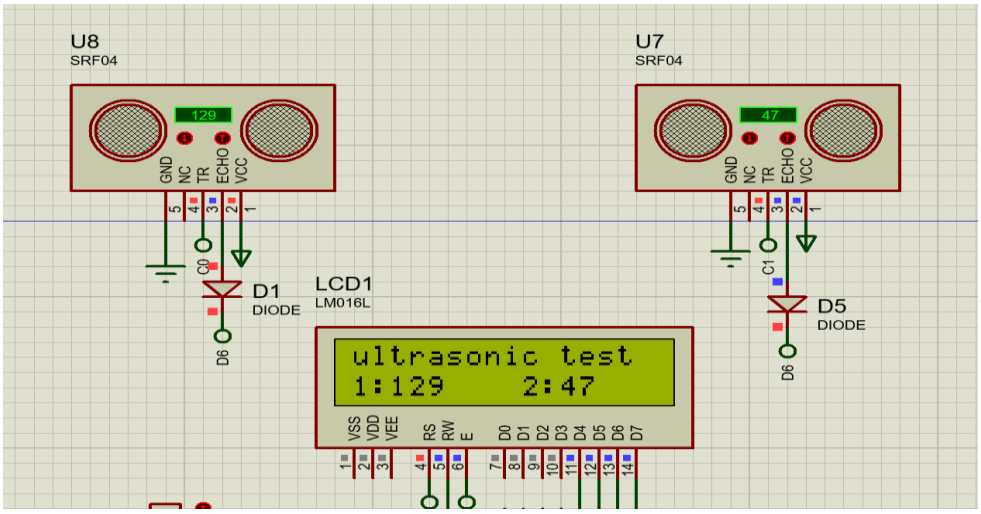



*Figure 79*

# Chapter7: DC Motors Control

## INTRODUCTION TO DC MOTOR

### What is DC Motor?

A DC motor is an electrical device that converts direct current electrical energy into mechanical energy. It operates based on the principle of electromagnetic induction, where a current-carrying conductor (usually a coil of wire) placed in a magnetic field experiences a force, causing it to rotate. This rotation is harnessed to perform mechanical work.

DC motors have a wide range of applications, including use in robotics, electric vehicles, various types of industrial machinery, and household devices. They are particularly useful in situations where speed control is essential. Consequently, DC motors are frequently employed in trolleys, electric trains, production systems, and elevators.

### DC Motor Definition:

DC motor, also known as a direct current motor, is an electric motor that converts mechanical energy from the electrical energy of direct current.

### DC Motor Diagram:

Diagram of Direct Current Motor is shown below:

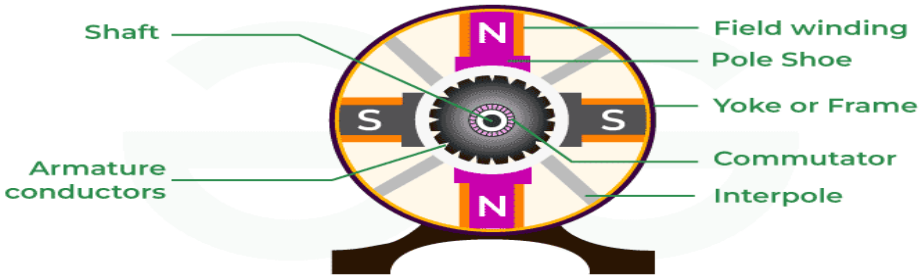



*Figure 80*

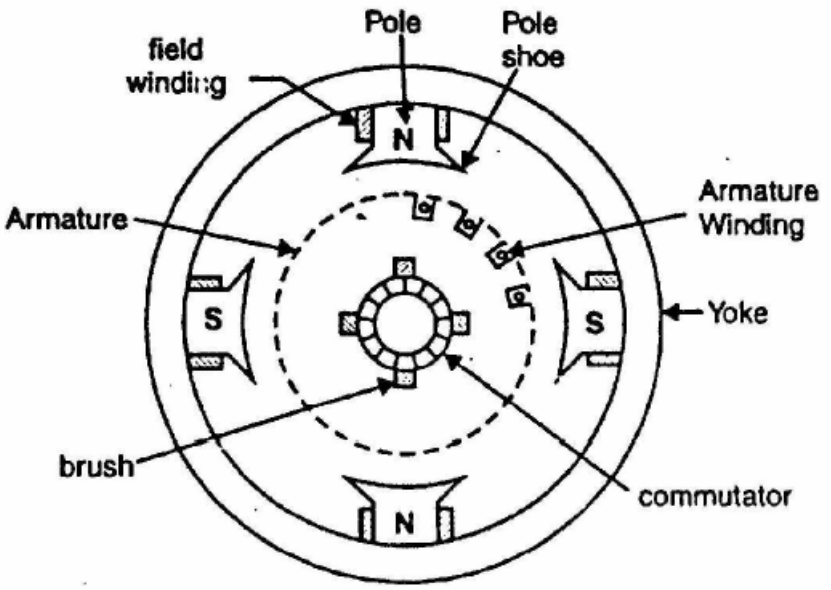



*Figure 81*

## Construction of DC Motor.

A DC motor consists of fundamental components, including a stator, which is the stationary part responsible for producing the magnetic field, and a rotor, which rotates and carries the winding or coil. When DC voltage is applied to the coil, current flows through it, generating an electromagnetic field. The interaction between the magnetic field of the rotor and that of the stator induces a torque, causing the rotor to begin spinning.

## DC Motor Parts:

- Field System or Stator
- Armature
- Commutator
- Brushes

**Field Coil or Stator:**

As the name implies, the field coil or stator is the stationary part of the DC motor around which the coil is wound to produce a magnetic field.

**The stator comprises various components:**

- Yoke
- Pole Core
- Pole Body
- Pole Shoe
- Field Winding
- End Plates

**Yoke**: The yoke of a DC machine functions to create the magnetic circuit between the poles.

**Pole Core**: The pole core is typically made of laminated iron or another magnetic material. Its purpose is to provide a path for the magnetic flux generated by the field winding.

**Pole Body**: The pole body works in conjunction with the pole core. When electric current passes through the field winding, a magnetic flux is established not only in the pole core but also around it. The poles and their supports are referred to as the pole body.

**Pole Shoe**: The pole shoe is analogous to one of the brushes inside an electric motor. DC motors use brushes to maintain contact with the rotating armature, and these are typically made of carbon.

**Field Winding**: The field winding is located on the pole core adjacent to the stator. It uses insulated copper wire wound around the pole core. When this coil on the pole core is energized with direct current, it produces a magnetic flux.

**End Plates**: End plates encase the entire motor, providing a housing for all internal parts, including the armature, commutator, and brushes, and sometimes also the field windings.

**Armature:**

The armature constitutes the rotating component of the motor responsible for generating mechanical energy. The armature core contains windings and is constructed from high magnetic

strength silicon steel laminations, typically ranging from 0.3 to 0.5 mm in thickness. Each sheet is coated with a thin layer of varnish.

**Commutator:**

Commutators are used in DC appliances such as DC Motors and DC Generators. It periodically reverses the current between the armature and the circuit and produces steady torque.

**Brushes:**

Brushes, commonly referred to as carbon brushes, are composed of graphite. In DC motors, these brushes are responsible for supplying current to the armature windings.

## DC Motor Working Principle

When a current carrying conductor is placed in a magnetic field, a mechanical force acts on it, which can be determined by Fleming's left-hand rule. Due to this force the conductor becomes mobile in the direction of the force.

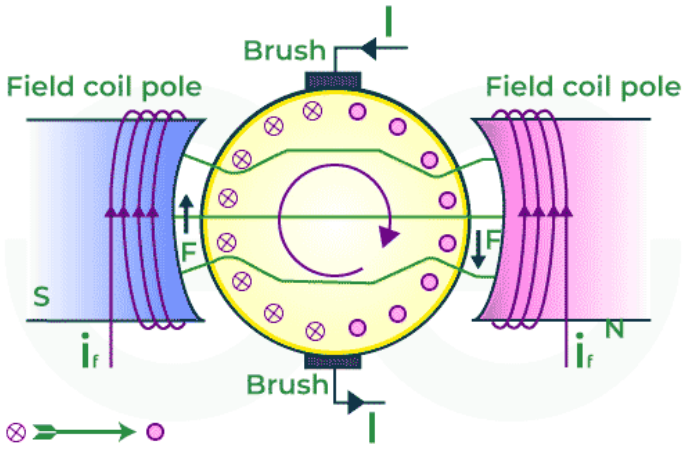



*Figure 82*

**DC Motor Operation:**

Imagine that a current-free conductor (which is not connected to the supply) is placed in the main magnetic field the and without the magnetic field flowing through the conductor. Assume, there is an air gap from N pole to S pole.

Current is flowing in the conductor but the magnetic effect of N pole and S pole has been removed. In this situation the conductor will maintain its own magnetic field. The magnetic field lines of force of the conductor will be clockwise according to the cork screw rule.

Current is flowing in the conductor and main magnetic field is also present. The magnetic field produced due to the current in the conductor acts along with the main field above the conductor but opposes the main field below the conductor. The result is that flux accumulates in the region above the conductor and flux density reduces in the region below.

From this it is clear that when the force is acting on the conductor, it works to push the conductor downwards. If the direction of current in the conductor is changed, the flux will accumulate downwards and will try to move the conductor upwards.

## DC Motor Characteristics:

If length of conductor = $\boldsymbol{L}$ meter, field intensity = $\boldsymbol{B}$ **weber per square meter** $\boldsymbol{Wb/m^2}$ and current flowing in the conductor = $\boldsymbol{i}$ ampere, then the force experienced by the conductor will be $\boldsymbol{F = iBL}$ Newton.

$$\boldsymbol{F = B.L.I.Sin\theta}$$

**where,**

- $\boldsymbol{B}$ = flux density (in Tesla).
- $\boldsymbol{L}$ = length of conductor (in meters).
- $\boldsymbol{I}$ = current flowing in the conductor.
- sin $\boldsymbol{\theta}$ = angle between the conductor and magnetic lines of force.

## Types of DC Motors

DC Motors can be classified into various categories based on the application and winding connections. Based on the winding of armature DC Motor is of two types

- Self-Excited DC Motors
- Separately Excited DC Motors

**Self-Excited DC Motor**

DC motors that excite themselves have a part and coil of field connected in series or partly so, same for parallel connection. They can also have combination of series and Parallel connections. They also get power from only one place. Three types of self-excited DC motors exist:

- Series DC motor
- Shunt DC motor
- Compound DC Motor

**Series DC Motor**

It is a motor in which the field is in series with the armature and its starting torque is very high. This means that the same flow of current goes through both the coil and armature. Series motors always go in one direction and their speed is affected by the physical load.

**Shunt DC Motor**

Shunt motor is a DC motor in which the field is jointed in parallel to the armature and its starting torque is less than that of series motor. Inside the shunt motor the field winding is connected parallel to the armature winding. The field winding is made of more turns of thin wire. Shunt motors are used in applications where continuous speed is required.

**Compound DC Motor**

Compound Motor is a Motor in which both series and parallel fields are added. Compound DC motors use both parallel and connected field windings. In the armature winding, everything is in series. However, field coils can be shunt or series types.

**Separately Excited DC Motor**

In DC motors with a separate excite, field coil to make permanent magnets. But, the armature and field coils are not connected electrically to each other. They work separately and do not bother the other. But, the result of the engine is added up with both.

**More Types of DC Motor**

Some more types of DC Motor on the various parameters are discussed below:

**1. Based on Commutation:**

- **Brushed Motors:** Use brushes and a commutator for current reversal.

- **Brushless Motors:** Switch currents in the windings by electronic commutation.

**2. Based on Application:**

- **DC Servo Motors:** Exact control of position and speed. Applied in robotics, CNC machine tools and automation.
- **Stepper Motors**: Take small steps, accurate position control. Widely employed in printers and CNC machines.
- **Hysteresis Motors:** The materials 'motion makes use of their magnetic hysteresis. Used in devices such as record players, rather simple.
- **Brushed DC Gear Motors:** Have a gearbox for high torque at low speeds. Applied in robotics, electric vehicles and automation.
- **DC Traction Motors:** Intended for use in traction devices, including electric vehicles and trains.

**3. Based on Speed-Torque Characteristics:**

- **High-Speed DC Motors:** Designed for uses involving high rotational speeds. Examples include tools and appliances.
- **Low-Speed DC Motors:** Offer high torque at low speeds. Commonly used in industrial machinery.

**4. Based on Size and Shape:**

- **Micro DC Motors:** Applications such as consumer electronics call for very small motors.
- **Large DC Motors:** As used in industrial purposes such as steel mills, paper industries and ship propulsion.

**5. Based on Control System:**

- **Closed-Loop DC Motors**: Feedback mechanisms can be employed for more precise control. Applications where precision control is important.
- **Open-Loop DC Motors**: Control without feedback, simpler design. In applications where control is not so critical.

## Applications of DC Motors

**D**irect current (DC) motor is a rotary electrical machine that converts direct current electrical energy into mechanical energy. DC motors exhibit a wide range of sizes and power capacities, from small motors used in toys and household appliances to large motors employed in powering vehicles.

**Applications of DC motors include:**

- Traction work
- Machine tools
- Industrial machinery
- Elevators

**Key characteristics and uses of specific types of DC motors:**

- **DC motors** offer the widest variation in speed compared to other motor types.
- **DC series motors** possess exceptionally high starting torque, making them ideal for initiating high inertia loads such as trains, elevators, and hoists.
- **Shunt DC motors** are commonly utilized in drilling machines and centrifugal machines.

## DC MOTOR SPEED CONROL

How to get a DC motor to rotate at a required speed?

The torque-speed characteristic curve of a DC motor shifts with variations in the drive voltage. This implies that the desired operational speed can be attained by merely adjusting the drive voltage.

Referring to the graph below, if the objective is to achieve a rotational speed of $\omega_1$ when the load torque is $T_0$ then:

- A drive voltage of $V_4$ is insufficient, resulting in a speed of $\omega_2$.
- A drive voltage of $V_0$ is excessive, leading to a speed of $\omega_0$
- Applying an intermediate drive voltage of $V_3$ is optimal, enabling the motor to reach the desired speed of $\omega_3$.

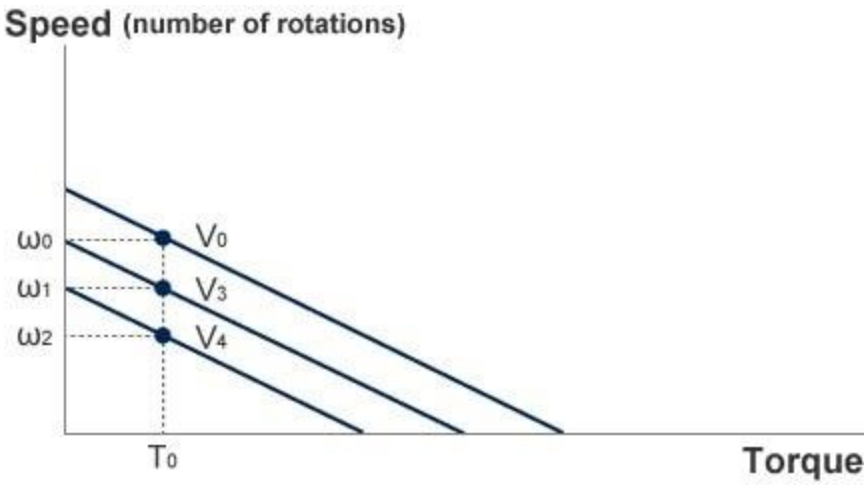



*Figure 83*

By adjusting the drive voltage in this way, a DC motor can be made to rotate at a desired speed regardless of load torque.

## Drive voltage control techniques

There are two primary methods for adjusting the drive voltage of a DC

motor: **linear control** and **pulse-width modulation (PWM) control.**

**Linear Control:**

Linear control involves placing a variable resistor in series with the motor and adjusting the resistance to vary the voltage across the motor. Although a transistor or other semiconductor device can serve as the variable resistor, this method is inefficient due to significant heat generation by the resistor (or semiconductor). Consequently, linear control is rarely utilized in contemporary applications.

### PWM Control:

In contrast, PWM control modulates the voltage applied to the motor by rapidly switching a semiconductor device (such as a transistor or an FET) on and off. The effective voltage is determined by the relative durations of the on and off pulses (duty cycle). This technique boasts high efficiency and is, therefore, the most commonly used method in modern motor control systems.

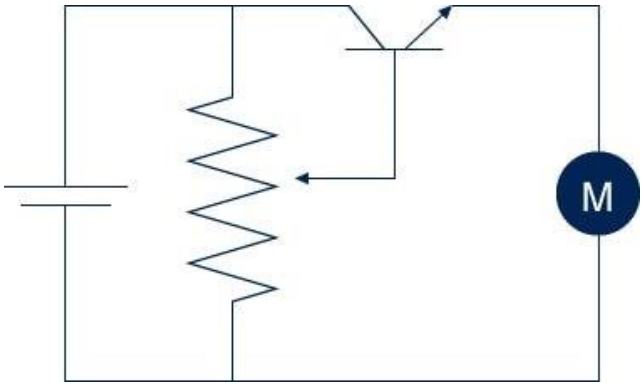



*Figure 85*

Linear Control

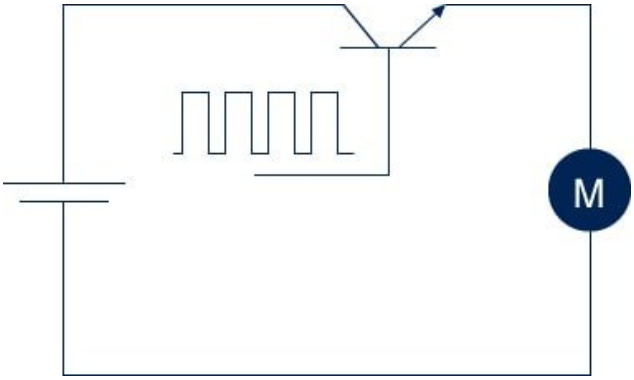



*Figure 84*

PWM Control

## Motor speed control

Use of these techniques allows the flexible adjustment of DC motor speed. However, additional control is required to keep a motor running at a constant speed. This is because the motor's load torque varies due to the load itself, as well as other factors such as temperature, humidity, and changes over time. Simply driving the motor with a constant voltage will result in its speed fluctuating with changes in load.

Maintaining a constant speed despite a variable load requires that the drive voltage be continually adjusted in response to these changes in load. The graph below shows an example where the load torque for a motor operating at a speed of $\omega_0$reduces from $T_1$ to $T_0$ , in which case reducing the drive voltage to $V_0$keeps the motor speed at $\omega_0$. If the torque instead increases to $T_2$, maintaining a constant motor speed of $\omega_0$requires that the drive voltage increase to $V_2$.

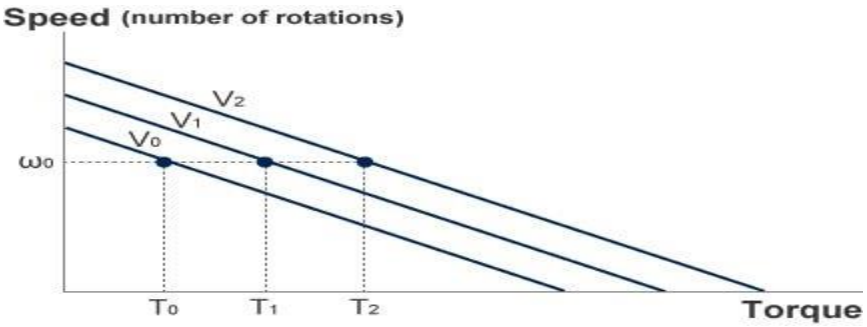



*Figure 86*

The speed is measured by a sensor attached to the motor. The difference between the measured and desired motor speed (speed error) is calculated, and the drive voltage is controlled in such a way that it is increased if the speed is too slow and reduced if the speed is too high. Doing this maintains a constant motor speed. While past practice was to use op amps or other analog circuits to control the drive voltage, the use of microcomputers has become the norm in recent years.

## Microcontroller Based DC Motor

### Motor Drivers.

Motor drivers, specifically H-Bridge drivers, are essential components used to control the direction and speed of DC motors.

**Motor Drivers (H-Bridges)**

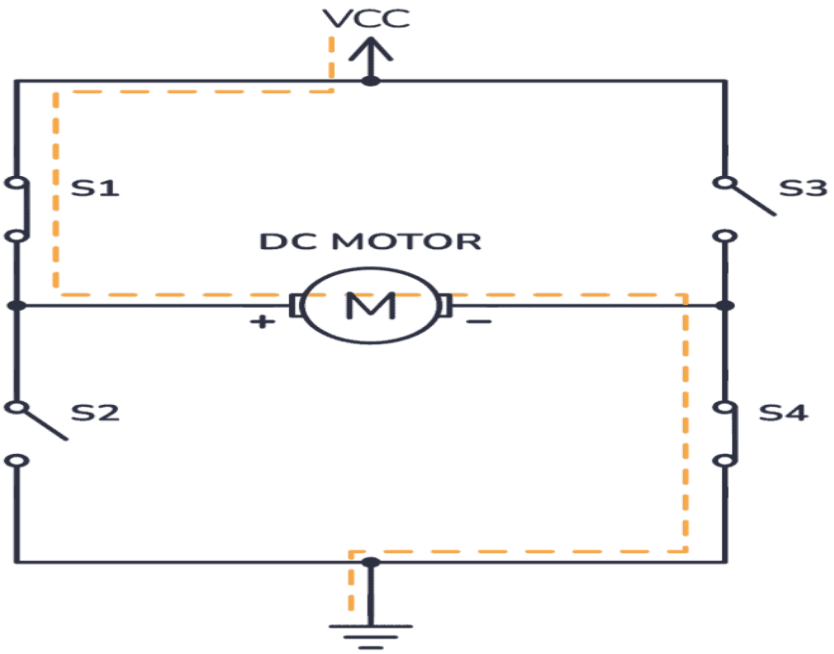



*Figure 87*

Definition**:** Motor drivers are electronic circuits that enable a microcontroller or other control devices to manage the power delivered to a motor. H-Bridge motor drivers are a specific type that allows for bidirectional control of the motor.

### H-Bridge Configuration:

- Named for its resemblance to the letter "H" in circuit diagrams.
- Consists of four switches (transistors or MOSFETs) arranged in a configuration that can direct current through the motor in either direction.

### Key Functions:

- **Direction Control**: By switching the appropriate transistors, the current through the motor can be reversed, changing the motor's direction.
- **Speed Control**: By applying Pulse Width Modulation (PWM) signals, the average voltage (and thus speed) of the motor can be controlled.

### Working Principle:

- **Forward Direction:** Switches S1 and S4 are closed, while S2 and S3 are open, allowing current to flow in one direction through the motor.
- **Reverse Direction:** Switches S2 and S3 are closed, while S1 and S4 are open, reversing the current flow through the motor.
- **Braking and Stopping:** By shorting both ends of the motor or by applying no voltage, the motor can be stopped.
-

## Integration of DC Motor, ATMEGA32 and the Motor Driver:

### Components Needed:

1. **DC Motor:** Determine the specifications (voltage, current) of the DC motor you intend to control.
2. **Motor Driver:** Select a suitable motor driver IC based on your motor's specifications. Common choices include L293D, L298N, or more advanced drivers like DRV8833.
3. **ATmega32 Microcontroller:** Prepare your ATmega32 microcontroller board along with necessary tools for programming.

## Integration Steps:

1. **Power Supply Connections**:

- **Motor Driver Power**: Connect the motor driver's power supply (VM and VCC pins) to an appropriate DC power source. Ensure the voltage matches the motor and driver requirements.
- **ATmega32 Power**: Power the ATmega32 microcontroller according to its specifications (typically 5V or 3.3V).

2. **Motor Connections**:

- Connect the DC motor to the motor driver's output terminals (often labeled as OUT1 and OUT2 for bidirectional control).
- Ensure proper polarity and match the motor driver's current handling capabilities with the motor's requirements.

3. **Control Signals**:

- Configure GPIO pins on the ATmega32 to connect to the motor driver's control inputs (e.g., IN1, IN2 for L293D).
- Designate one GPIO pin for enabling the motor driver (e.g., EN pin on L293D) if applicable.

4. **Programming the ATmega32**:

- Write firmware code in C or assembly language to control the motor driver.
- Use GPIO outputs to send control signals (HIGH/LOW or PWM signals) to the motor driver:
    - Toggle control signals to control motor direction (forward/reverse).
    - Implement PWM (Pulse Width Modulation) signals for speed control if supported by the motor driver.

5. **Testing and Debugging**:

- Test basic motor functionality (e.g., verify motor spins in both directions).
- Debug any issues related to wiring, power supply, or code logic.
- Monitor signals using LEDs or a multimeter to ensure correct operation.

6. **Safety and Considerations**:

- Implement safety measures such as current limiting and proper heat sinking for the motor driver IC.
- Protect the microcontroller from voltage spikes using appropriate diodes or capacitors.

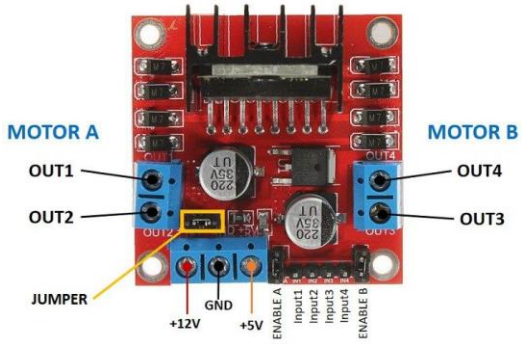



*Figure 88*

**L298 Dual H-Bridge Motor Driver**

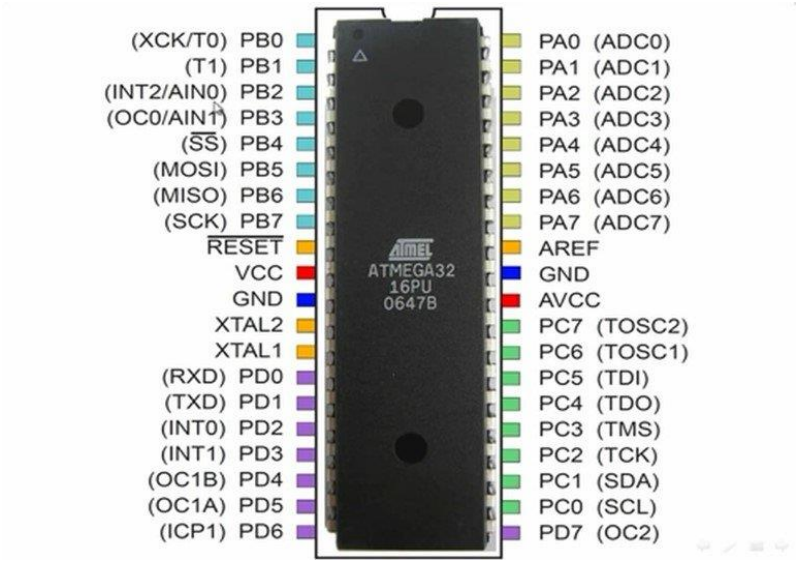



*Figure 89*

**ATmega32 MC**

*Figure 90*

## 12 Volt DC Motor

## Proteus Schematic Diagram:

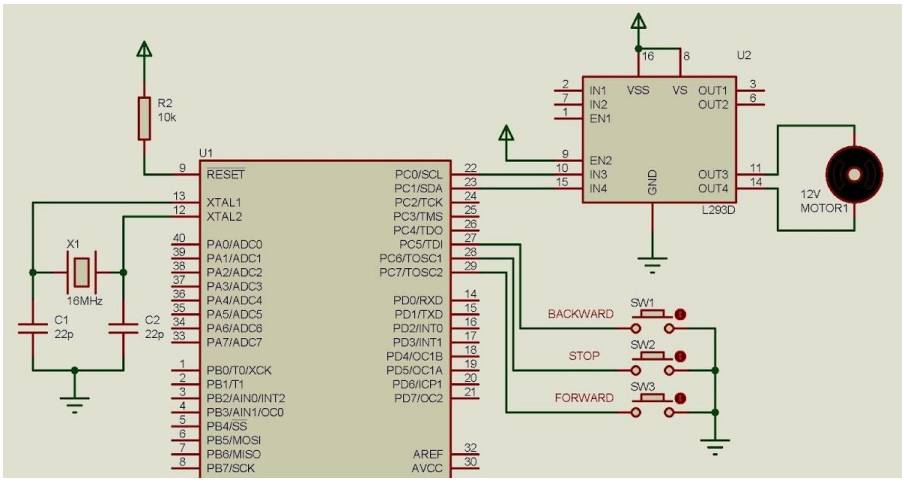



*Figure 91*

## AVR Microcontroller (Pseudo Code):

```
// Define motor control pins
#define MOTOR_A_FORWARD_PIN 1
#define MOTOR_A_BACKWARD_PIN 2
#define MOTOR_B_FORWARD_PIN 3
#define MOTOR_B_BACKWARD_PIN 4

// Function to move forward
void moveForward() {
   digitalWrite(MOTOR_A_FORWARD_PIN, HIGH);
   digitalWrite(MOTOR_A_BACKWARD_PIN, LOW);
   digitalWrite(MOTOR_B_FORWARD_PIN, HIGH);
   digitalWrite(MOTOR_B_BACKWARD_PIN, LOW);
}
```

```
// Function to move backward
void moveBackward() {
  digitalWrite(MOTOR_A_FORWARD_PIN, LOW);
  digitalWrite(MOTOR_A_BACKWARD_PIN, HIGH);
  digitalWrite(MOTOR_B_FORWARD_PIN, LOW);
  digitalWrite(MOTOR_B_BACKWARD_PIN, HIGH);
}

// Function to turn right
void turnRight() {
  digitalWrite(MOTOR_A_FORWARD_PIN, HIGH);
  digitalWrite(MOTOR_A_BACKWARD_PIN, LOW);
  digitalWrite(MOTOR_B_FORWARD_PIN, LOW);
  digitalWrite(MOTOR_B_BACKWARD_PIN, HIGH);
}

// Function to turn left
void turnLeft() {
  digitalWrite(MOTOR_A_FORWARD_PIN, LOW);
  digitalWrite(MOTOR_A_BACKWARD_PIN, HIGH);
  digitalWrite(MOTOR_B_FORWARD_PIN, HIGH);
  digitalWrite(MOTOR_B_BACKWARD_PIN, LOW);
}

void setup() {
  // Set motor control pins as outputs
  pinMode(MOTOR_A_FORWARD_PIN, OUTPUT);
  pinMode(MOTOR_A_BACKWARD_PIN, OUTPUT);
  pinMode(MOTOR_B_FORWARD_PIN, OUTPUT);
  pinMode(MOTOR_B_BACKWARD_PIN, OUTPUT);
}

void loop() {
  // Example movement sequence
  moveForward();
  delay(2000); // Move forward for 2 seconds
  turnRight();
  delay(1000); // Turn right for 1 second
  moveForward();
  delay(2000); // Move forward for 2 seconds
  turnLeft();
  delay(1000); // Turn left for 1 second
  moveBackward();
  delay(2000); // Move backward for 2 seconds
}
```

# Timer 1

In the Atmega32 microcontroller, timers are essential hardware modules that count clock cycles and generate timing events.

**Timer1** specifically is a 16-bit timer with various modes of operation, used for generating precise time delays, generating PWM signals, and measuring elapsed time. It's configurable through control registers to set its mode, pre-scaler (which divides the clock frequency), and output behavior.

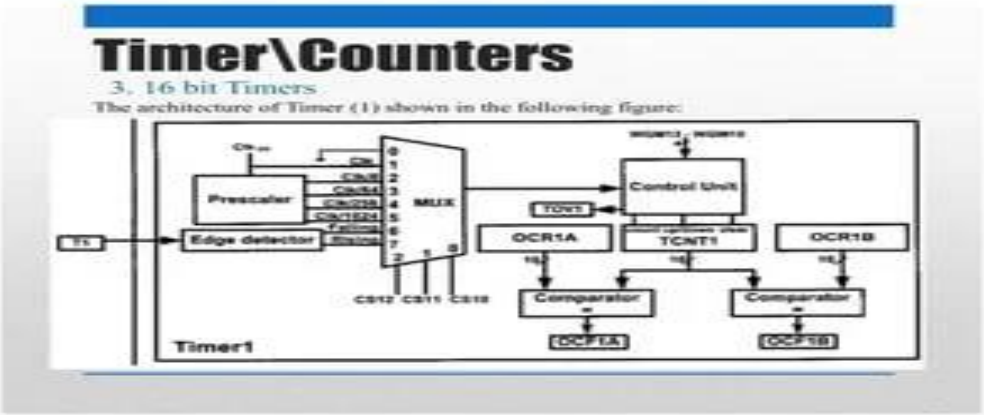



*Figure 92*

## OCR1A and OCR1B (Output Compare Registers 1A and 1B)

The Output Compare Registers OCR1A and OCR1B are associated with Timer/Counter 1 (Timer1) in Atmega microcontrollers. They play crucial roles in generating PWM signals and defining output compare values for Timer1.

OCR1A and OCR1B Functions:

1. **PWM Generation:**
    - In PWM modes (such as Fast PWM or Phase Correct PWM), OCR1A and OCR1B determine the duty cycles of the PWM signals.
    - The values stored in OCR1A and OCR1B dictate when the output pins associated with Timer1 (typically OC1A and OC1B) toggle their states based on the counter value of Timer1.
    - This allows for the generation of precise PWM signals for applications such as motor control, LED dimming, and other tasks requiring variable pulse width.

2. **Output Compare Function:**
    - In non-PWM modes, OCR1A and OCR1B can be used to set specific values against which the Timer1 counter is compared.
    - When the Timer1 counter matches the value stored in OCR1A or OCR1B, it can trigger actions such as generating interrupts or toggling the state of the output pins (depending on the configuration of the COM bits in the TCCR1A and TCCR1B registers).
    - This functionality is useful for creating precise timing events and generating output signals with specific timing requirements.

3. **Register Size:**
    - OCR1A and OCR1B are 16-bit registers since Timer1 is a 16-bit timer. This means they can store values from 0 to 65535.
    - The 16-bit resolution provides finer granularity for timing and PWM applications, allowing for more precise control compared to 8-bit timers.

## PWM Wave Generation

PWM (Pulse Width Modulation) in the Atmega32 microcontroller refers to a technique where the width of a digital pulse is varied while keeping the frequency constant. This is achieved by controlling the duty cycle of the pulse, which is the percentage of time the signal is high (ON) compared to the total period of the signal.

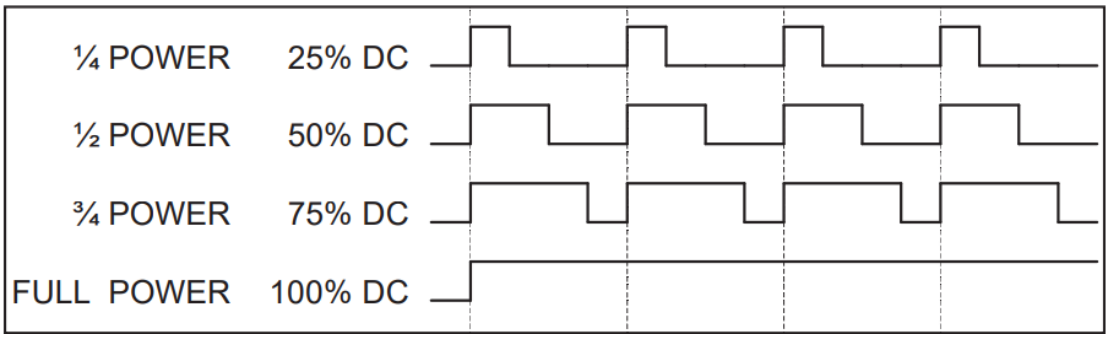



*Figure 93*

Figure. Pulse Width Modulation Comparisons.

### Fast PWM Mode for Timer1 in AVR Microcontrollers

Fast PWM (Fast Pulse Width Modulation) mode in Timer1 of the AVR microcontroller operates similarly to Fast PWM in Timer0 but with a higher resolution due to Timer1 being a 16-bit timer. Here are the key characteristics and behavior of Fast PWM mode for Timer1:

## Key Characteristics of Fast PWM Mode in Timer1:

1. **Counting Sequence:**
    - In Fast PWM mode, Timer1 counts from 0 (BOTTOM) to a specified TOP value and then resets to 0. The TOP value can be set using the ICR1 or OCR1A register, allowing for flexible PWM frequency control.
    - The default TOP value is 0xFFFF, providing a full 16-bit range, but it can be adjusted for different PWM frequencies.

2. **Waveform Generation:**
    - The PWM waveform is generated on the output pins (typically OC1A and OC1B for Timer1) based on compare matches between the Timer1 value (TCNT1) and the values stored in the OCR1A and OCR1B registers.
    - The waveform generated is fast and symmetric, which is ideal for applications requiring high-speed and precise PWM signals.
3. **Output Behavior:**
    - The behavior of the output pins (OC1A and OC1B) is controlled by the Compare Output Mode (COM) bits in the TCCR1A register.
    - The COM1A and COM1B bits determine how the output pins react on a compare match:
        - **Clear on Compare Match (non-inverting mode):** The output pin is cleared (set to low) when the Timer1 counter matches the OCR1A or OCR1B value and set (high) when the counter reaches BOTTOM.
        - **Set on Compare Match (inverting mode):** The output pin is set (high) when the Timer1 counter matches the OCR1A or OCR1B value and cleared (low) when the counter reaches BOTTOM.
        - **Toggle on Compare Match:** The output pin toggles its state on a compare match (useful for generating square waves).

## Reaction of the Waveform Generator in Fast PWM Mode:

- **Non-Inverting Mode (Clear on Compare Match):**
    - When the counter (TCNT1) matches the value in OCR1A (or OCR1B), the output pin OC1A (or OC1B) is cleared (set to low).
    - When the counter resets to 0 (BOTTOM), the output pin is set (high).
- **Inverting Mode (Set on Compare Match):**
    - When the counter matches the value in OCR1A (or OCR1B), the output pin OC1A (or OC1B) is set (high).
    - When the counter resets to 0 (BOTTOM), the output pin is cleared (low).
- **Toggle Mode:**
    - The output pin toggles its state on each compare match, creating a square wave with a frequency determined by the value in the OCR1A (or OCR1B) register.

**Frequency of the generated wave in Fast PWM mode:**

Fast PWM mode for Timer1, the timer counts from 0 to a specified TOP value and then rolls over. Timer1 is a 16-bit timer, which means it can count up to 0xFFFF (65535). However, the TOP value can be adjusted using the ICR1 or OCR1A register, allowing for different PWM frequencies.

The frequency of the generated wave is determined by the timer clock frequency divided by the number of timer's counts per period. The timer clock frequency is derived from the main system clock frequency divided by the pre-scaler value (N).

$$F_{PWM} = \frac{F_{timer\ clock}}{N(TOP + 1)}$$

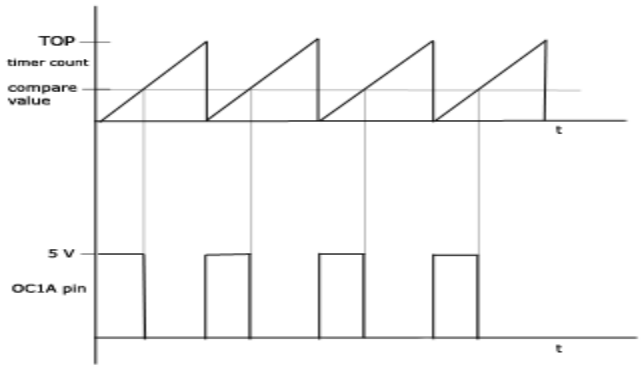



*Figure 94*

**Duty cycle of the generated wave in Fast PWM mode:**

For Timer1 in Fast PWM mode, the duty cycle of the generated PWM signal is determined using the OCR1A and OCR1B registers. The duty cycle can be calculated based on the value in these registers relative to the TOP value. Below are the formulas for both non-inverted and inverted modes:

$$Duty\ Cycle = \frac{OCR1n+1}{TOP+1} \times 100$$

Similarly, the duty cycle formula for inverted mode is as follows:

$$Duty\ Cycle = \frac{TOP-OCR1n}{TOP+1} \times 100$$

**PWM Programming in C Language:**

Header File:

```
#ifndef PWM_CORE_H_
#define PWM_CORE_H_

#include "../../LIB/7"
#include "../../LIB/MCU.h"

#define PWM_ENABLE    1U
#define PWM_DISABLE   0U

#define PWM_FAST           0U
#define PWM_PHASE_CORRECT   1U

#define PWM_INVERTED      0U
#define PWM_NONINVERTED   1U

#define PWM_PRESC8     8U
#define PWM_PRES32     32U
#define PWM_PRESC64    64U
#define PWM_PRESC128   128U
#define PWM_PRESC256   256U
#define PWM_PRESC1024  1024U

#define  PWM0_STATE  PWM_ENABLE
#define  PWM1_STATE  PWM_DISABLE
#define  PWM2_STATE  PWM_DISABLE

#define  PWM0_TYPE  PWM_FAST
#define  PWM1_TYPE  PWM_FAST
#define  PWM2_TYPE  PWM_FAST

#define  PWM0_MODE  PWM_NONINVERTED
#define  PWM1_MODE  PWM_NONINVERTED
#define  PWM2_MODE  PWM_INVERTED

#define  PWM0_PRESC  PWM_PRESC1024
#define  PWM1_PRESC  PWM_PRESC256
#define  PWM2_PRESC  PWM_PRESC8

void PWM_Init(void);
```

```
void PWM0_Generate(uint8 Duty_Cycle);

void PWM1_Generate(uint8 Frewquancy , uint8 Duty_Cycle);

#endif /* PWM_CORE_H_ */
```

PWM Implementation

```
#include "PWM_Core.h"

void PWM_Init(void)
{

	#if (PWM1_STATE == PWM_ENABLE)
	TCCR1A |= 0x02;
	TCCR1B |= 0x18;
	TCCR1A |= 0x80;*/
	#endif if (PWM1_STATE == PWM_ENABLE)
}

void PWM1_Generate(uint8 Frequancy , uint8 Duty_Cycle)
{
	ICR1_16BIT_ACCESS = (16000000U / (Frequancy * PWM1_PRESC));
	OCR1A_16BIT_ACCESS = ((Duty_Cycle * (ICR1_16BIT_ACCESS)) /100);
	TCCR1B |= 0x05;
}
```

**DC Motor Implementation**

**Header File**

```
#ifndef HAL_DC_MOTOR_DC_MOTOR_H_
#define HAL_DC_MOTOR_DC_MOTOR_H_
//**************************INCLUDES*************************//
#include "../../LIB/MCU.h"
#include "../../LIB/BIT_MATH.h"
#include "../../MCAL/DIO/DIO.h"
#include "../../MCAL/TIMER1/TIMER1.h"
//**************************MACROS*************************//
#define CW 1
```

```
#define CCW 0
//*************************FUNCTIONS**********************//
void DC_Motor_INIT(void);
void DC_Motor1_SetSpeed(uint8 Speed);
void DC_Motor2_SetSpeed(uint8 Speed);
void Both_motors(uint8 Speed);
void DC_Motor_SetDir(uint8 Dir);
void DC_Motor2_SetDir(uint8 Dir);
void DC_Motor_Emergency_Stop(void);
void DC_Motor2_Emergency_Stop(void);
void DC_Motor_Stop(void);
void DC_Motor2_Stop(void);
#endif /* HAL_DC_MOTOR_DC_MOTOR_H_ */
```

C File

```
#include "Dc_Motor.h"

/****************************** Functions *********************/
void DC_Motor_INIT(void){
	//PWM
	DIO_INIT(PORT_D,PORTD_PIN4,OUT);
	DIO_INIT(PORT_D,PORTD_PIN5,OUT);
	//Motor 1
	DIO_INIT(PORT_C ,PORTC_PIN4,OUT);
	DIO_INIT(PORT_C ,PORTC_PIN5,OUT);
	//Motor2
	DIO_INIT(PORT_C ,PORTC_PIN6,OUT);
	DIO_INIT(PORT_C ,PORTC_PIN7,OUT);

}
void DC_Motor1_SetSpeed(uint8 Speed){
	PWM_Duty_A(Speed);
}
void DC_Motor2_SetSpeed(uint8 Speed){
	PWM_Duty_B(Speed);
}
void Both_motors(uint8 Speed){
	PWM_Duty(Speed);
}
void DC_Motor_SetDir(uint8 Dir){
	switch(Dir){
	case CCW :
		DIO_Write(PORT_C,PORTC_PIN4 ,HIGH);
		DIO_Write(PORT_C,PORTC_PIN5 ,LOW);
		break;
	case CW :
```

```
                DIO_Write(PORT_C,PORTC_PIN4 ,LOW);
                DIO_Write(PORT_C,PORTC_PIN5 ,HIGH);
                break;
        default:
                break;
        }
}

void DC_Motor2_SetDir(uint8 Dir){
        switch(Dir){
        case CCW :
                DIO_Write(PORT_C,PORTC_PIN6 ,HIGH);
                DIO_Write(PORT_C,PORTC_PIN7 ,LOW);
                break;
        case CW :
                DIO_Write(PORT_C,PORTC_PIN6 ,LOW);
                DIO_Write(PORT_C,PORTC_PIN7 ,HIGH);
                break;
        default:
                break;
        }
}

void DC_Motor_Emergency_Stop(void){
        DIO_Write(PORT_C ,PORTC_PIN4 ,HIGH);
        DIO_Write(PORT_C ,PORTC_PIN5 ,HIGH);
}

void DC_Motor2_Emergency_Stop(void){
        DIO_Write(PORT_C ,PORTC_PIN6 ,HIGH);
        DIO_Write(PORT_C ,PORTC_PIN7 ,HIGH);
}

void DC_Motor_Stop(void){
        DIO_Write(PORT_C ,PORTC_PIN4 ,LOW);
        DIO_Write(PORT_C ,PORTC_PIN5 ,LOW);
}

void DC_Motor2_Stop(void){
        DIO_Write(PORT_C ,PORTC_PIN6 ,LOW);
        DIO_Write(PORT_C ,PORTC_PIN7 ,LOW);
                                        }
```

# Chapter 8: ARTIFICIAL INTELLIGENCE in Surveillance

## Flask: Integrating Ai with Mobile and Embedded:

To address the overlapping connections within our project, we have devised a solution called STATION. This serves as an organizing component, tackling several challenges that arise when directly connecting multiple parts.

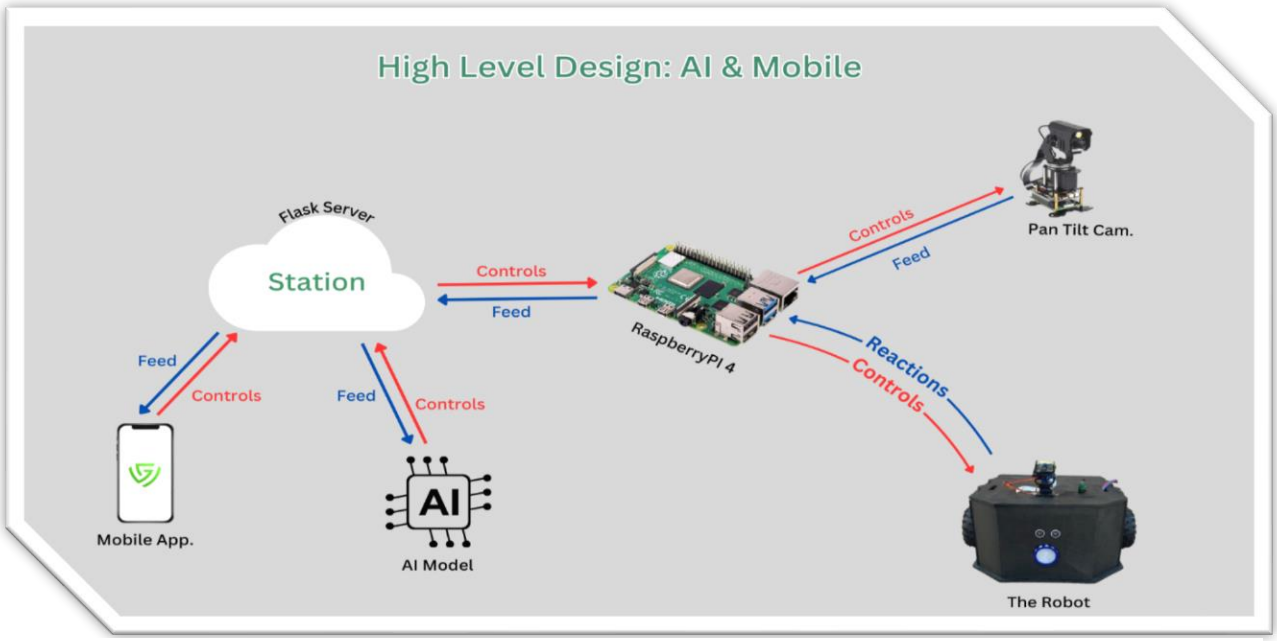



*Figure 95*

### Challenges

Mobile Issues:

- The video feed from the Raspberry Pi, provided by the Flask code, can only be accessed by one client at a time. This limitation prevents simultaneous reading by both the mobile device and the AI model.
- Switching between modes needs to be managed in a preliminary stage, enabling the user to control both the AI and the Raspberry Pi concurrently.

AI Issues:

- Similar to the video feed problem, the AI must be able to read frames and then post its processed output to the same server. This will allow the mobile user to access the live processed data.
- There must be an exchange of data and arguments between the mobile device, the AI, and the Raspberry Pi, as well as with the robot.

## Solutions

- Video Feed:
    - We established the (/video_feed) endpoint as a dedicated reference point that can always be accessed by the mobile device to watch the live feed of the project. By implementing robust code, we developed an algorithm to manage various user requests and modes effectively.
    - In the AI mode, the AI model accesses the Raspberry Pi endpoint and then posts its processed feed to the same video feed endpoint.
    - In mobile mode, the mobile device directly accesses the video feed connected to the Raspberry Pi camera feed.
    - For the glove mode, there are no changes, and raw feed is used.

## Controller

To execute the AI model code based on user requests, we handle two main cases:

1. AI Main Model: The recognize.py script.

2. Adding a New User to the Dataset: The add_person.py script.

This process involves first releasing the video feed endpoint from the mobile device. The mobile device then posts its selfie camera feed to the same server and video feed endpoint, allowing the AI model to read the feed and train on the user's face.

Other Functions:

- **(/set_opt):** receive the mode's option from the user and control the required action by the station.
- **(/pan_tilt):** control the pan tilt camera's angles.
- **(/car_mv):** control the speed of the robot's wheels.
- **(/frame_ids):** where the model posts the real time faces' ids.
- **(/chosen_id):** where the user posts its chosen id out of the frame ids, so the robot takes the required action to track the target.
- **(/chosen_name):** where the user posts the chosen name so the robot can search for and track.
- **(/notify):** where the user gets the notification to be updated of how the model is working during the real time tracking period.
- **(/new_person):** where the user sends the new person's data so the model can train on.

# Ai Model

## Introduction

In the initial stage of our project, the AI's primary role is to detect, track, and recognize people's faces. Based on this recognition, certain actions are taken by the robot in response to AI and user requests. Human supervision plays a significant role in overseeing the AI due to the ambiguity of the environment and to ensure a secure and enhanced user experience.

The general role of Ai in this project is helping the user track specific target (Human being) depending on the ability of recognition and with distance estimation and robot tracking function, an easy tracking should be given, rather than manual controlling.

Supervised control, in general, is very crucial in some situations and environments such like earthquakes, floods and any other nature disaster. And with our implementation in this UGV, a new uses is ahead in the field of surveillance, monitoring and other military purposes.

### AI Role Recap

The AI workflow is as follows:

1. **Raspberry Pi Server:** The Raspberry Pi opens its Flask server and streams its live feed from the attached camera module.
2. **Station Server Connection:** The station server connects to the Raspberry Pi server and sets up the necessary endpoints.
3. **AI Model Execution:** Upon request, the AI model accesses the live feed from the station.
4. **Processing and Output:** The model generates a processed feed and outputs data. Some of this data is posted for the robot to act on, while other data is exchanged with mobile devices and the Raspberry Pi.
5. **Robot Actions:**
    - **Face Tracking:** Using a pan-tilt camera, the robot can track faces.
    - **Target Tracking:** The robot can directly follow a specific target by calculating the target's position and distance.
    - **Executing Orders:** The robot performs specific tasks as ordered, based on the recognized and tracked target.

## AI Model Breakdown:

Our system comprises three interacting models ensuring stable experiment and enhanced surveillance tool:

### 1- Face Detection: SCRFD

Stands for (Single-Shot Scale-Aware Face Detector), is designed for real-time face detection across various scales. It is particularly effective in detecting faces at different resolutions within the same image.

- **Why is it the best solution in our implementation?**
  The model's accuracy and efficiency (due to its single-shot nature) make it suitable for real-time applications like video surveillance, where detecting faces accurately and swiftly is essential.
- **Implementation:**

1- Initializing:

a. Downloading the model's weights

```
# Face detector
detector = SCRFD(model_file="face_detection/scrfd/weights/scrfd_2.5g_bnkps.onnx")
```

*Figure 96*

b. Locating detection class.

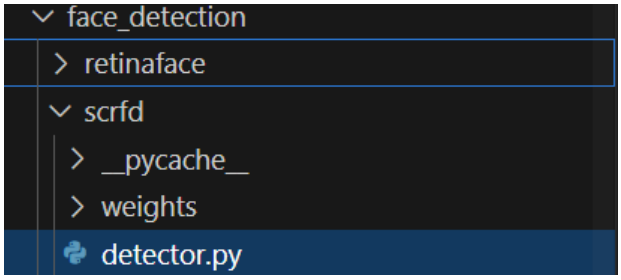



*Figure 97*

c. Importing:

```
from face_detection.scrfd.detector import SCRFD
```

*Figure 98*

2- Detect tracking:
    a. **Frame Pass:** We pass the raw frame to the model with initial Size.
    b. **Resizing:** The input frame is first resized to match the initial size required by the model.
    c. **Prepare the Input Blob:** The resized image is then prepared as an input blob. This involves normalizing the image and converting it into a format suitable for the neural network (e.g., converting to a blob with specific mean and scale values).
    d. **Run Forward Pass:** The input blob is fed into the model to perform a forward pass, which generates the raw output predictions including scores, bounding box predictions, and optionally keypoint predictions.
    e. **Process Model Outputs:** The raw outputs are processed to decode the predictions. The bounding box coordinates and scores are extracted and scaled appropriately.
    f. **Non-Maximum Suppression (NMS):** NMS is applied to the detected bounding boxes to filter out overlapping and redundant boxes, keeping only the most relevant detections.
    g. **Final Scaling:** results are scaled back to the original image size.
    h. **Returning:** final detections, width and height, the bounding boxes and landmarks.

And depending on these outputs, we need to use Tracker model to keep tracking, so we move to the second step, tracking.

## Face Tracking: BYTE Track

Byte Track is a cutting-edge multi-object tracking algorithm designed for exceptional performance. Unlike traditional methods that discard detections with low confidence scores, Byte Track considers all detections. This allows it to effectively track objects even when they're partially hidden or in challenging lighting conditions. The model's strength lies in its simple yet powerful association method, enabling it to achieve state-of-the-art results while maintaining real-time processing speeds.

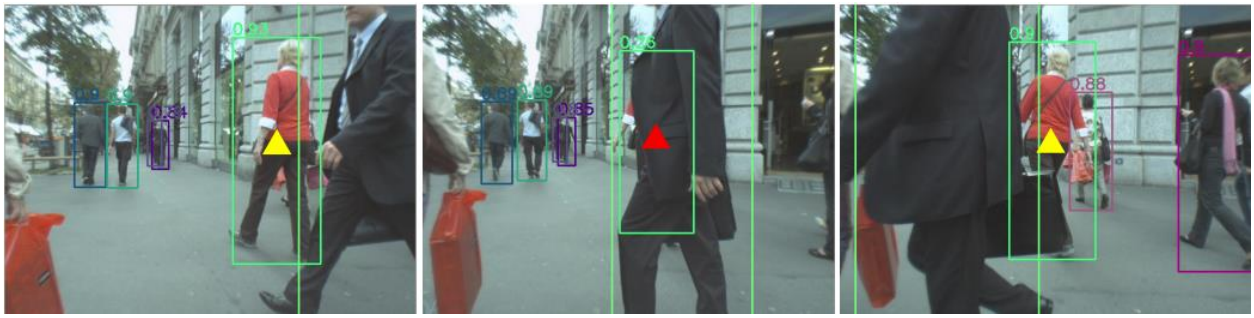


*Figure 99*

## Why is it the best solution in our implementation?

So we would be able to keep tracking the faces detected without having to process every single frame. and this helps decrease the latency and predict as fast as possible where should the moving face in this implementation be very quick.

## Implementation:

1- Initialization:

```
tracker = BYTETracker(args=args, frame_rate=30)
```

*Figure 100*

2- Update:

a. Passing inputs: detected boxes, width and height of frame and Minimum size threshold for objects to be tracked.

b. Iterates over the faces to retrieve:

i. Tlwh: bounding box coordinates.

ii. Tid: Tracked id.

iii. Checks if the tracked object's aspect ratio is above a specified threshold and if Minimum area threshold for tracked objects is considered.

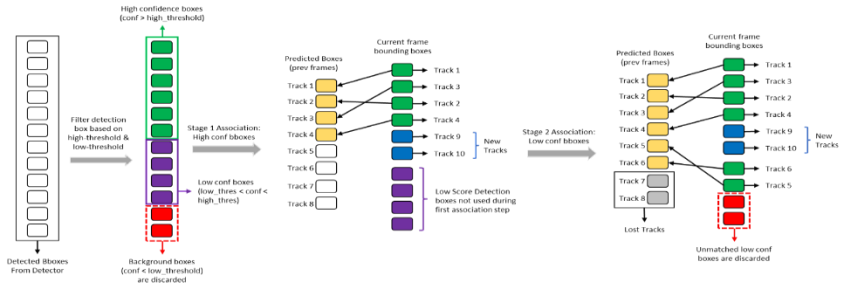



*Figure 101*

c. Stores:
    i. tracking_tlwhs: the tlwh of tracked objects.
    ii. tracking_scores: the confidence score of tracked objects.
    iii. tracking_ids: the unique ID (track_id) of tracked objects.

### Merging Detection and Tracking:

Overall, we integrate face detection (SCRFD), object tracking (BYTETracker), and prepares data (tracking_bboxes, tracking_tlwhs, tracking_scores, tracking_ids) for further processing: visualization, face recognition (ArcFace) and Robot actions.

But now the challenge is that we now have two different boxes for the same face recognized, so we had to find a way to make them one. And here we go with the concept of Intersection over Union (IOU).

### Intersection over Union (IOU):

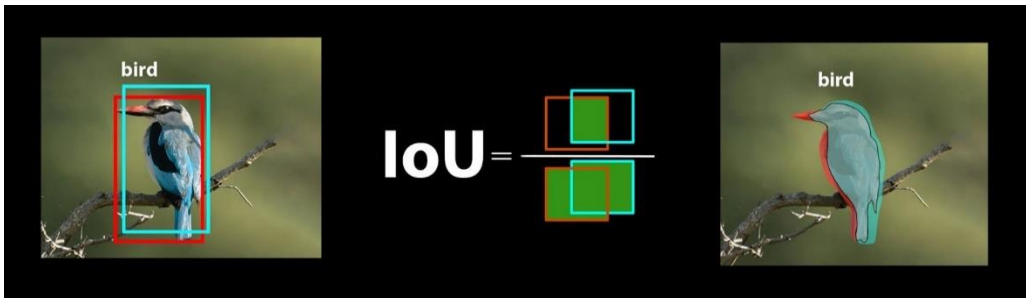



*Figure 102*

The IOU metric is used to evaluate the accuracy of object detection models during training and testing. Typically, a threshold value is set for IOU, and any predicted bounding box with an IOU greater than the threshold is considered a true positive detection.

## Pseudocode:

**Algorithm 1:** IoU computation

**Data:** $\text{box}_1 = [L_1, T_1, R_1, B_1]$, $\text{box}_2 = [L_2, T_2, R_2, B_2]$ - two bounding boxes in terms of left, top, right and bottom coordinates.

**Result:** IoU - the Intersection of Union score for $\text{box}_1$ and $\text{box}_2$.

$L_{inter} \leftarrow \max(L_1, L_2)$
$T_{inter} \leftarrow \max(T_1, T_2)$
$R_{inter} \leftarrow \min(R_1, R_2)$
$B_{inter} \leftarrow \min(B_1, B_2)$
**if** $(R_{inter} < L_{inter})$ *or* $(B_{inter} < T_{inter})$ **then**
  **return** 0;
**end**
$A_{inter} \leftarrow (R_{inter} - L_{inter}) \times (B_{inter} - T_{inter})$
$A_1 \leftarrow (R_1 - L_1) \times (B_1 - T_1)$
$A_2 \leftarrow (R_2 - L_2) \times (B_2 - T_2)$
$A_{union} \leftarrow A_1 + A_2 - A_{inter}$
$\text{IoU} \leftarrow A_{inter}/A_{union}$
**return** IoU;

*Figure 103*

## Implementation:

```python
def mapping_bbox(box1, box2):
    """
    Calculate the Intersection over Union (IoU) between two bounding boxes.

    Args:
        box1 (tuple): The first bounding box (x_min, y_min, x_max, y_max).
        box2 (tuple): The second bounding box (x_min, y_min, x_max, y_max).

    Returns:
        float: The IoU score.
    """
    # Calculate the intersection area
    x_min_inter = max(box1[0], box2[0])
    y_min_inter = max(box1[1], box2[1])
    x_max_inter = min(box1[2], box2[2])
    y_max_inter = min(box1[3], box2[3])

    intersection_area = max(0, x_max_inter - x_min_inter + 1) * max(
        0, y_max_inter - y_min_inter + 1
    )

    # Calculate the area of each bounding box
    area_box1 = (box1[2] - box1[0] + 1) * (box1[3] - box1[1] + 1)
    area_box2 = (box2[2] - box2[0] + 1) * (box2[3] - box2[1] + 1)

    # Calculate the union area
    union_area = area_box1 + area_box2 - intersection_area

    # Calculate IoU
    iou = intersection_area / union_area

    return iou
```

*Figure 104*

## Threshold condition:

We assigned a 0.9 value thresholding the value of IOU and in this case only, the next step of recognition starts.

```python
for i in range(len(tracking_bboxes)):
    for j in range(len(detection_bboxes)):
        mapping_score = mapping_bbox(box1=tracking_bboxes[i], box2=detection_bboxes[j])
        if mapping_score > 0.9:
            face_alignment = norm_crop(img=raw_image, landmark=detection_landmarks[j])
```

*Figure 105*

## Face Recognition: Arc-Face

ArcFace is a state-of-the-art face recognition algorithm that focuses on learning highly discriminative features for face verification and identification. It is known for its robustness to variations in lighting, pose, and facial expressions.

The mathematical formulas of the softmax and an Additive Angular Margin loss (AAM for abbreviation) are as follows:

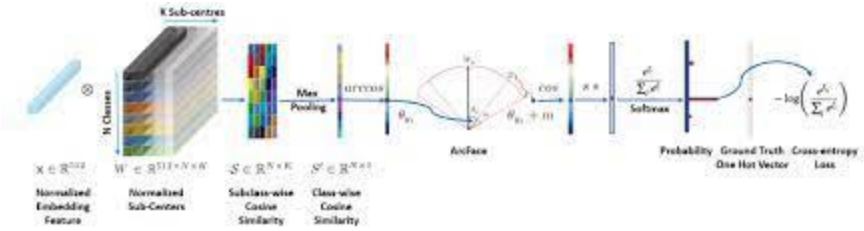



*Figure 106*

$$L_{\text{softmax}} = -\log \frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{N} e^{W_j^T x_i + b_j}} \tag{1}$$

$$L_{\text{AAM}} = -\log \frac{e^{s\cos(\theta_{y_i}+m)}}{e^{s\cos(\theta_{y_i}+m)} + \sum_{j=1, j\neq y_i}^{N} e^{s\cos(\theta_j)}} \tag{2}$$

*Figure 107*

As a premise, they want to make intra-class points closer but keep inter-class points away from each other. Then, they focus on the cosine similarity between data points.

$W^t$ x can be described as the dot product of W and x. So, we can transform it to the formula below using formulas

$$W_j^T x_i = \|W_j\| \|x_i\| \cos\theta_j \tag{5}$$

$$W_j^T x_i = \cos\theta_j \quad \text{(if W and x are unit vectors)} \tag{6}$$

*Figure 108*

$\theta_j$is the angle between $W_j$ and $x_i$. When we fix the bias term b = 0 for simplicity, the formula (1) will be as follows:

$$L_{\text{softmax}} = -\log \frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{N} e^{W_j^T x_i + b_j}} \tag{7}$$

$$= -\log \frac{e^{\cos \theta_{y_i}}}{\sum_{j=1}^{N} e^{\cos \theta_j}} \tag{8}$$

$$= -\log \frac{e^{\cos \theta_{y_i}}}{e^{\cos \theta_{y_i}} + \sum_{j=1, j \neq y_i}^{N} e^{\cos \theta_j}} \tag{9}$$

*Figure 109*

The formula (9) looks closer to the formula (2)(I rewrite the summation in the denominator for the following process). Next, if we can make intra-class samples' angles smaller but inter-class samples' angles larger, that sounds good. To achieve it simultaneously, the authors introduced margin to the angle.

$$-\log \frac{e^{\cos(\theta_{y_i} + m)}}{e^{\cos(\theta_{y_i} + m)} + \sum_{j=1, j \neq y_i}^{N} e^{\cos \theta_j}} \tag{10}$$

*Figure 110*

Compared to the formula (9), the model needs to learn the intra-class angle smaller because of the margin in the formula (10) case. If not, the model cannot classify samples correctly because each class area will be mixed.

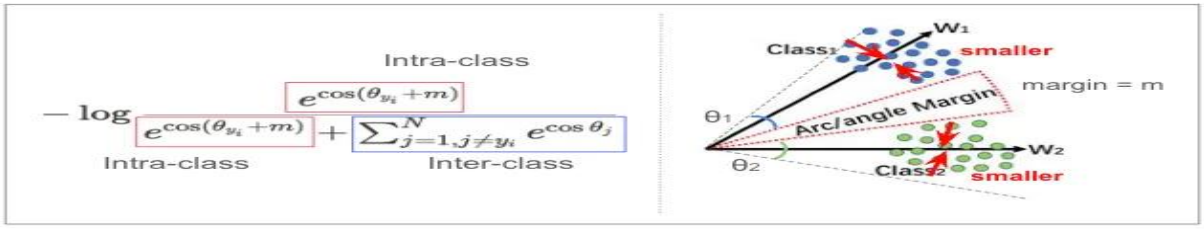



*Figure 111*

Finally, we re-scale the cosine parameters to mitigate the effect that the correct label logits tend to have smaller values. You can imagine that the numerator value is smaller if we have many classes(typical face recognition problem).

$$L_{\text{AAM}} = -\log \frac{e^{s \cos(\theta_{y_i} + m)}}{e^{s \cos(\theta_{y_i} + m)} + \sum_{j=1, j \neq y_i}^{N} e^{s \cos(\theta_j)}}$$

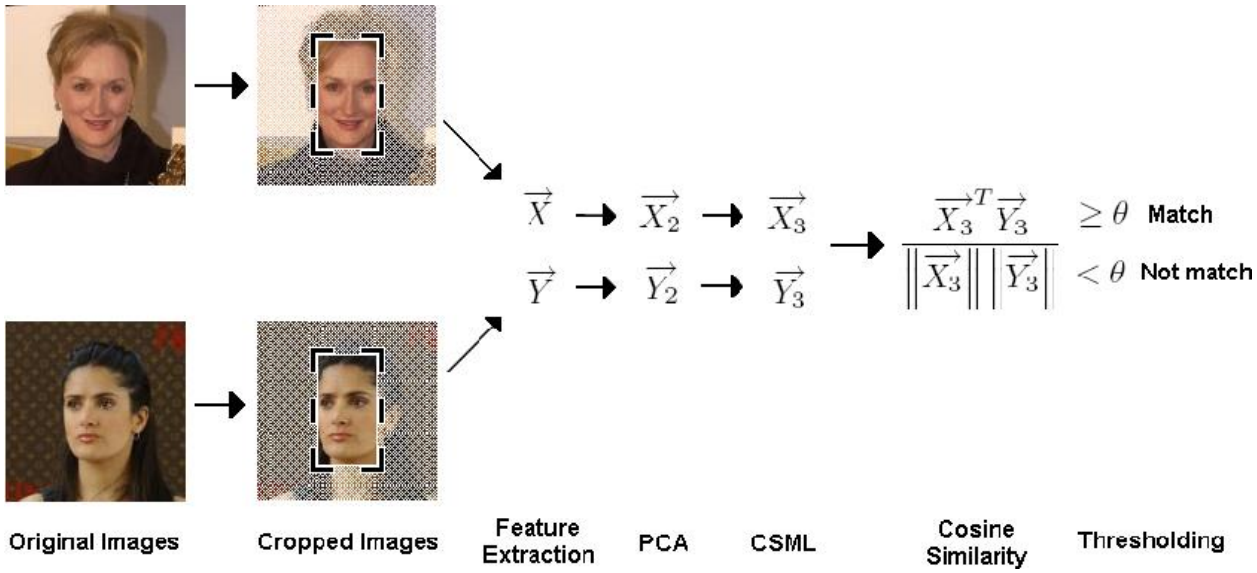
## Implementation

a. Initialization:

```
recognizer = iresnet_inference(
    model_name="r100", path="face_recognition/arcface/weights/arcface_r100.pth", device=device
)
```

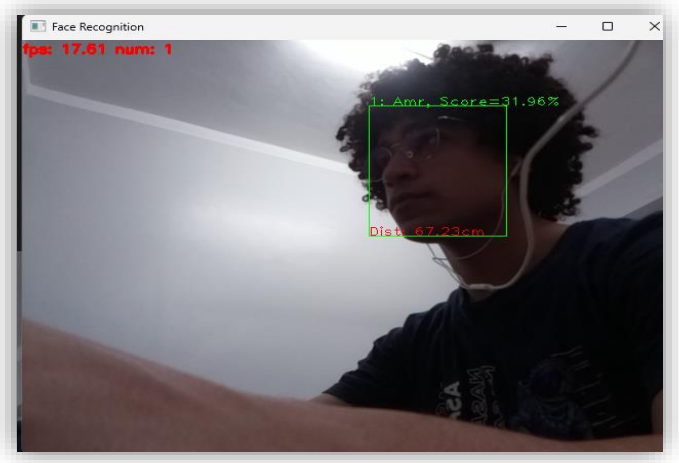
b. Cosine Similarity:

```
def compare_encodings(encoding, encodings):
    sims = np.dot(encodings, encoding.T)
    pare_index = np.argmax(sims)
    score = sims[pare_index]
    return score, pare_index
```



## Visualization

For visualizing the processed feed to the user, we assigned some features eases the user experience and makes a reasonable output.




- Frame rate and Face counter.
- Name
- ID
- Score
- Estimated Distance.
- Colorful boxes turn green in cond. of known faces.

## Unique ID

And to fulfill the project requirements, we had to assign unique od for each person instead of a new id assigned every loop.

So instead of the dictionary that saves all the faces detected since the executing of the code, we had to make a new dictionary expresses only the real-time-faces, so the user, after all, can interact with now-shown-faces instead of no longer shown ones.

- Implementation and sending the ids to the user:

```
id_face_mapping[tracking_ids[i]]=[caption[0], caption[1], face_id, round(Distance,2), deltaX,deltaY,caption[2]]
# print(id_face_mapping)

# Priority choosing for known faces only
min_id=float('inf')
for trackid in tracking_ids:
    real_time_ids[trackid] = id_face_mapping.get(trackid, ['Unknown', 'Unknown', float('inf'), 0, 0, 0, 'Unknown'])
    if  real_time_ids[trackid][6]!='UN_KNOWN' and real_time_ids[trackid][2]<min_id :
        min_id=real_time_ids[trackid][2]
        general_id=trackid
        is_known_face=True

print(real_time_ids)

current_ids = [data[2] for data in real_time_ids.values()]
print("Current IDs in frame: ", current_ids)

# Post current_ids to Flask Station
asyncio.run(frameIDs(current_ids))
```

# Add New Person

## Introduction

Add new person is a critical part of our face recognition system. It facilitates the addition of new persons facial data into the recognition database. This process involves capturing images via a webcam, detecting faces within these images, encoding the facial features, and storing these encodings for future recognition tasks. Below is a detailed explanation of the code, its functionality, and the benefits it provides to the overall project.

## Overview

- Used libraries
- Capturing photos using a webcam.
- Detecting and saving faces from the captured images.
- Encoding the facial features of the saved images.
- Saving the encoded data for later use in face recognition
- Priority Choosing for Known Faces
- Real-Time Processing

## Used libraries:

```
import argparse
import os
import shutil, requests

import cv2 ,time
import numpy as np
import torch
from torchvision import transforms
import argparse
import os
import shutil, requests
```

```
import cv2 ,time
import numpy as np
import torch
from torchvision import transforms


from face_detection.scrfd.detector import SCRFD
from face_recognition.arcface.model import iresnet_inference
from face_recognition.arcface.utils import read_features
from Flask.endpoints import url_station_video_feed,url_get_person
from Flask.notification import SendNotifications
```

Capturing photos using from mobile video:

```
cam = cv2.VideoCapture(url_station_video_feed)
cam.set(3, 640) # set video width
cam.set(4, 480) # set video height
```

The first step in the add_persons involve capturing photos using a webcam. This process is essential to gather the raw data needed for face detection and recognition This is a crucial step as it provides the necessary data for further processing.

This contains more steps:

1. **Initialization:** The webcam of mobile is initialized to capture video frames. The resolution is set to ensure the captured images are clear and suitable for face detection.
    - **Webcam Initialization:** We use the **cv2.VideoCapture** function to start the webcam feed and set its resolution to 640x480 pixels. This ensures a good balance between image quality and processing speed.

2. **Face Detection:** A pre-trained face detection model is used to identify faces in the video frames. This is achieved using the **Haar Cascade** classifier, which is well-suited for detecting frontal faces.

```
face_detector = cv2.CascadeClassifier('haarcascade_frontalface_alt.xml')
```

   - **Haar Cascade Classifier:** The **cv2.CascadeClassifier** is used to load the Haar Cascade model, which detects faces in the grayscale images. This model is highly efficient for real-time face detection.

3. **Capturing and Saving Images:** Once a face is detected, the system captures multiple images of the person. These images are saved in a designated directory for further processing.
   - **Image Capture and Storage:** The system captures 50 images of the person, saving each detected face as a grayscale image. This ensures a comprehensive dataset for each individual, improving the accuracy of the face recognition system.

```
while(True): # while opt=n


    time.sleep(5) # wait till camera of mobile is streaming to vid_back to be shown at video_feed
~ sec


    #reads frames from video_feed
    ret, img = cam.read()
    img = cv2.flip(img, 1) # flip video image vertically
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # to reduce preprocessing for frame as
rgb need three chanels
    faces = face_detector.detectMultiScale(gray, 1.3, 5)
    for (x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
        count += 1
        # Save the captured image into the datasets folder
```

```
    cv2.imwrite(f"datasets/new_persons/{face_name}/User." + str(face_id) + '.' +
                str(count) + ".jpg", gray[y:y+h,x:x+w])
    cv2.imshow('image', img)
k = cv2.waitKey(100) & 0xff # Press 'ESC' for exiting video
if k == 27:
    break
elif count >= 50: # Take 50 face sample and stop video
     #send ch to notify mob to close camera()
     SendNotifications('c')
     break
```

### Detecting and Saving Faces from the Captured Images:

After capturing the images, the next step involves detecting faces within these images and saving them for encoding.

1. **Face Detection:** Advanced face detection models, such as SCRFD, are used to accurately detect faces in the captured images.

```
detector = SCRFD(model_file="face_detection/scrfd/weights/scrfd_2.5g_bnkps.onnx")
```

   - **SCRFD Model:** The SCRFD model is a state-of-the-art face detector that offers high accuracy and speed. It detects faces and key landmarks, which are essential for precise face recognition.
2. **Saving Detected Faces:** The detected faces are cropped from the images and saved in a separate directory. This step ensures that only the relevant part of the image (the face) is used for encoding.
   - **Face Cropping and Storage:** The system crops the detected faces from the images and saves them in a dedicated folder. This organized storage facilitates easy access and further processing of the face images.

## Encoding the Facial Features of the Saved Images

```
recognizer = iresnet_inference(
    model_name="r100", path="face_recognition/arcface/weights/arcface_r100.pth", device=device
)
```

1. **Face Recognition Model:** A pre-trained face recognition model, such as **iResNet**, is used to extract unique features from each face. These features are then used for recognition.
    - **iResNet Model:** The iResNet model is a powerful deep learning model trained specifically for face recognition. It converts face images into a high-dimensional feature vector that uniquely represents each face.
2. **Feature Extraction:**

    The face images are pre-processed and passed through the recognition model to extract these feature vectors These vectors are normalized to ensure consistency.

```
def get_feature(face_image):
    """
    Extract facial features from an image using the face recognition model.

    Args:
        face_image (numpy.ndarray): Input facial image.

    Returns:
        numpy.ndarray: Extracted facial features.
    """
    # Define a series of image preprocessing steps
    face_preprocess = transforms.Compose(
        [
            transforms.ToTensor(),
```

```
        transforms.Resize((112, 112)),
        transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
    ]
)

# Convert the image to RGB format
face_image = cv2.cvtColor(face_image, cv2.COLOR_BGR2RGB)

# Apply the defined preprocessing to the image
face_image = face_preprocess(face_image).unsqueeze(0).to(device)

# Use the model to obtain facial features
emb_img_face = recognizer(face_image)[0].cpu().numpy()

# Normalize the features
images_emb = emb_img_face / np.linalg.norm(emb_img_face)
return images_emb
```

- **Image Pre-processing:** The images are resized, normalized, and converted to a suitable format before being fed into the recognition model. This pre-processing step is crucial for achieving accurate and reliable feature extraction.
- **Feature Vector:** The output of the recognition model is a feature vector that represents the unique characteristics of each face. These vectors are essential for comparing and recognizing faces.

### Saving the Encoded Data for Later Use in Face Recognition

The final step involves saving the encoded facial features along with the associated names. This data is stored in a file for future use in recognizing individuals.

1. **Combining and Saving Features:** The extracted features from the new faces are combined with existing features in the database. This ensures that the system can recognize both new and previously known individuals.
    - **Feature Storage:** The feature vectors and corresponding names are saved in a compressed file format. This makes it easy to load and use the data for future recognition tasks.
2. **Organized Backup:** The images and data of the new persons are moved to a backup directory to keep the working directory clean and organized.
    - **Data Management:** By moving the data to a backup directory, the system maintains an organized structure, making it easier to manage and update the face recognition database.

Usage of add new person

```
def add_persons(backup_dir, add_persons_dir, faces_save_dir, features_path):
    """
    Add a new person to the face recognition database.

    Args:
        backup_dir (str): Directory to save backup data.
        add_persons_dir (str): Directory containing images of the new person.
        faces_save_dir (str): Directory to save the extracted faces.
        features_path (str): Path to save face features.
    """
```

```
# Initialize lists to store names and features of added images
images_name = []
images_emb = []

# Read the folder with images of the new person, extract faces, and save them
for name_person in os.listdir(add_persons_dir):
    person_image_path = os.path.join(add_persons_dir, name_person)

    # Create a directory to save the faces of the person
    person_face_path = os.path.join(faces_save_dir, name_person)
    os.makedirs(person_face_path, exist_ok=True)

    for image_name in os.listdir(person_image_path):
        if image_name.endswith(("png", "jpg", "jpeg")):
            input_image = cv2.imread(os.path.join(person_image_path, image_name))

            # Detect faces and landmarks using the face detector
            bboxes, landmarks = detector.detect(image=input_image)

            # Extract faces
            for i in range(len(bboxes)):
                # Get the number of files in the person's path
                number_files = len(os.listdir(person_face_path))

                # Get the location of the face
                x1, y1, x2, y2, score = bboxes[i]

                # Extract the face from the image
                face_image = input_image[y1:y2, x1:x2]
```

```
            # Path to save the face
            path_save_face = os.path.join(person_face_path, f"{number_files}.jpg")

            # Save the face to the database
            cv2.imwrite(path_save_face, face_image)

            # Extract features from the face
            images_emb.append(get_feature(face_image=face_image))
            images_name.append(name_person)

# Check if no new person is found
if images_emb == [] and images_name == []:
    print("No new person found!")
    SendNotifications('n')
    return None

# Convert lists to arrays
images_emb = np.array(images_emb)
images_name = np.array(images_name)

# Read existing features if available
features = read_features(features_path)

if features is not None:
    # Unpack existing features
    old_images_name, old_images_emb = features

    # Combine new features with existing features
```

```
    images_name = np.hstack((old_images_name, images_name))
    images_emb = np.vstack((old_images_emb, images_emb))

    print("Update features!")

# Save the combined features
np.savez_compressed(features_path, images_name=images_name, images_emb=images_emb)

# Move the data of the new person to the backup data directory
for sub_dir in os.listdir(add_persons_dir):
    dir_to_move = os.path.join(add_persons_dir, sub_dir)
    shutil.move(dir_to_move, backup_dir, copy_function=shutil.copytree)

########## notify mob that the process succeeded()!
SendNotifications('s')
print("Successfully added new person!")
```

1. **Efficient Data Collection:** The automated process of capturing multiple images ensures a diverse and comprehensive dataset, enhancing the accuracy of face recognition.
2. **Accurate Face Detection and Encoding:** Utilizing advanced models for face detection and recognition ensures high accuracy and reliability in identifying faces.
3. **Reusable Encodings:** The encoded facial features are stored in a reusable format, allowing for quick and efficient recognition without needing to re-process the images.
4. **Scalability:** The system is designed to easily add new individuals, making it scalable and capable of handling a large number of faces.
5. **Practical Implementation:** The code provides a practical solution for real-world face recognition applications, covering all essential steps from data collection to recognition.

## Real-Time Processing

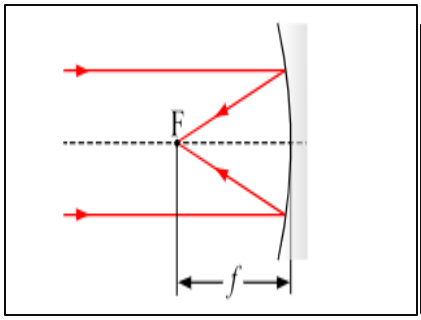
1. **Continuous Video Capture:**
    - **Objective:** Capture live video frames from a webcam continuously.
    - **Implementation:** Use OpenCV (cv2) to initialize and stream frames from the webcam. Ensure the video feed is stable and handles different lighting conditions.
2. **Face Detection and Recognition:**
    - **Challenge:** Perform efficient face detection and recognition in real-time.
    - **Implementation:** Use optimized face detection models like Haar Cascade, Dlib, or the more advanced SCRFD for faster detection. Integrate a pre-trained face recognition model (e.g., iResNet) to extract and compare facial features.
3. **Feature Extraction and Encoding:**
    - **Objective:** Extract unique features from detected faces for identification.
    - **Implementation:** Pre-process images by resizing, normalizing, and converting them to the required format before feeding them into the recognition model. Extract feature vectors that represent facial characteristics robustly across different conditions.

4. **Database Integration:**
    - **Challenge:** Integrate real-time updates with a central database of known faces.
    - **Implementation:** Store feature vectors and associated metadata (names, IDs) in a database system (e.g., SQLite, MySQL) or a structured file format. Implement mechanisms for adding new faces to the database and updating existing entries efficiently.

## Focal Length Calculation:

### What is focal length:

**Focal Length (f):** The distance from the optical center of a lens to the focal point, where parallel rays of light converge after passing through the lens. A positive Focal length indicates that a system converges light, while a negative focal length indicates that the system diverges light. Converges lenses convex lenses, which bulge outward, converge light rays to a focal point. The focal length is positive for these lenses. Diverging Lenses (Concave Lenses) Concave lenses, which curve inward, diverge light rays. The focal length is negative for these lenses. The focal length is measured to the point from which the diverged light rays appear to originate



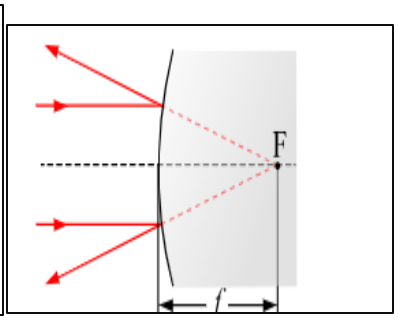



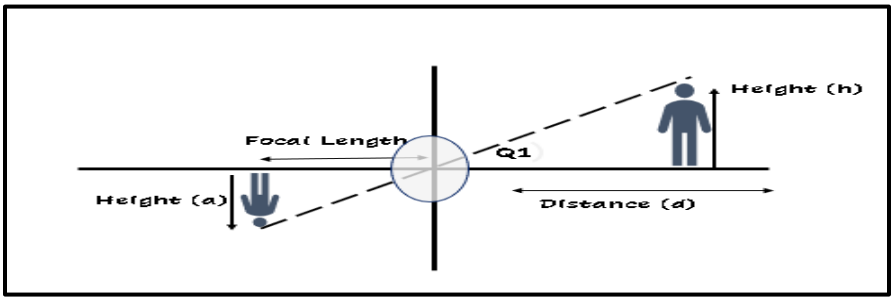

*Figure 112*

*Figure 113*

### Formula and Calculation:

**Focal Length**: In camera optics, the focal length is the distance between the camera lens and the image sensor when the subject is in focus. It determines how magnified the subject appears.
**Image Sensor**: The sensor captures the image formed by the lens. The dimensions of the image sensor and the focal length affect the field of view and the apparent size of objects in the image. In this project, the focal length is essential for distance estimation. Given the size of an object in the real world and its size in the captured image, the focal length allows us to compute the distance from the camera to the object.

The focal length (f) can be calculated using similar triangles formed by the object and its image on the sensor plane.



Using similar triangles, it found that to find the focal length using the following relationship:

$$Tan\theta = \frac{a}{F} = \frac{h}{d} \qquad (1)$$

So after using the above equation the focal length will be

$$F = \frac{d \times \mathrm{a}}{h} \qquad (2)$$

To apply this equation from physics in terms of code implementation the relationship will change a little bit to be:

$$F = \frac{P \times \mathrm{D}}{W} \qquad (3)$$

Where:

P: Width of the object in reference image (in pixels)

D: Measured distance from the camera to the object (in cm)

W: Real width of the object in real life (in cm)

### Focal length Implementation:

To implement focal length function which need to pass some important parameters such as real face width which measured in cm and measured distance from camera to object in cm too and finally width in reference image. To find this width by using Haar cascade model which can detect face in frames and it return list contains detected faces which the list consist of tuples of (x,y,h,w) X and Y represent the coordinates of top left corner and H represent the height of detected face and W represent the width of the detected face in pixels .Now we can implement the focal length using the above equation

 focal length (): It is a function take measured distance and real face width and width in reference image (frame) and apply focal equation and return focal length

## Distance Estimation:

### Explanation of Distance Estimation

Distance estimation in computer vision involves calculating the distance from a camera to an object based on the object's size in an image or video frame. This process is crucial for various applications such as robotics, augmented reality, autonomous vehicles, and surveillance systems. Here's a detailed discussion on distance estimation. After we found focal length from known measured distance now we need to estimate distance of object from a camera if the object moved towards or backwards from the camera. Using focal length and known object size in real world then apply mathematical relationship

Known object is called as reference object as its width is known to apply the relationship and to calibrate the system

## Mathematical Formula:

Once the focal length is known from previous part and equation  then the distance object from camera can be estimated as if the object change its distance from camera towards or backwards the focal length remain constant and it doesn't affect by this moving but what is change is width of image or frame in pixels at the focus of the lens of the camera .It can be calculated using similar triangles formed by the object and its image on the sensor plane.

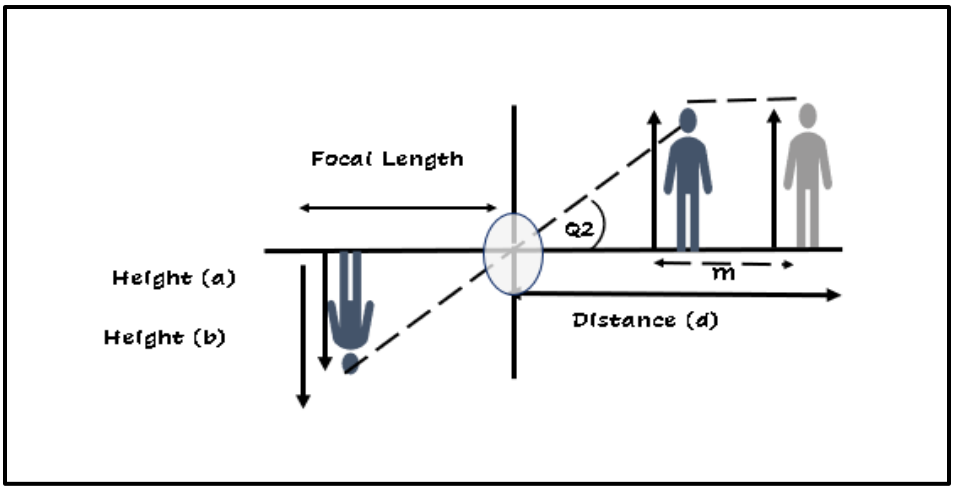



*Figure 114*

Using similar triangles, it found that to find the focal length using the following relationship:

$$\frac{b}{f} = tan\,\theta_2 = \frac{h}{d-m} \qquad (4)$$

Divide equation (1) with equation (2):

$$\frac{a}{b} = \frac{h}{d} \times \frac{d-m}{h}$$

$$\frac{a}{b} = \frac{d-m}{d} = 1 - \frac{m}{d}$$

$$\frac{m}{d} = 1 - \frac{a}{b}$$

Then the estimation distance will be $\boldsymbol{d = \frac{m}{1-\frac{a}{b}}}$ (5)

To apply this equation from physics in terms of code implementation the relationship will change a little bit to be from equation number (3)

$$D = \frac{W \times \mathrm{F}}{P} \qquad (6)$$

By this equation the distance can be estimated whether the object moving towards or backwards from the camera

### Distance Estimation Implementation:

To implement distance estimation function which need to pass some important parameters such as real face width which measured in cm and focal length that had found from focal length function before and finally width in reference image in pixels. To find this width by using Haar cascade model which can detect face in frames and it return list contains detected faces which the list consist of tuples of (x,y,h,w) X and Y represent the coordinates of top left corner and H represent the height of detected face and W represent the width of the detected face in pixels .Now we can implement the focal length using the above equation

 distance finder (): It is a function take focal length and real face width and width in reference image (frame) and apply distance equation and return distance object from the camera.

## Pan-Tilt functionality in Face tracking system:

### Explanation of Pan-Tilt System:

A pan-tilt system is a mechanical setup that allows a camera or other sensor to be moved in two dimensions: horizontally (pan) and vertically (tilt). This enables the camera to have a wide range of motion, which is essential for tracking and recognizing faces in dynamic environments. The pan-tilt system operates by receiving commands from a controlling software, which specifies the desired angles for pan and tilt. These angles are calculated based on the position of the object to be tracked within the camera's field of view. In the given code, the pan-tilt system is used to dynamically adjust the camera's orientation to keep the tracked face in the center of the frame.

### Mathematical equations for Angle Calculation

To Calculate the angles for the pan and tilt movements (Angle X and AngleY) based on the position of the detected face within the camera frame there are some critical parameters. First parameter is to know the width and height of the frame which is the frame dimension in pixels. Then to find the detected face center after using Haar cascades model to detected the faces in the frame and then plot the box around it. The top left corner of the box its coordinates is known. Now it is easy to find the detected face center coordinates x and y by add half of box width to x and half of box height to y.

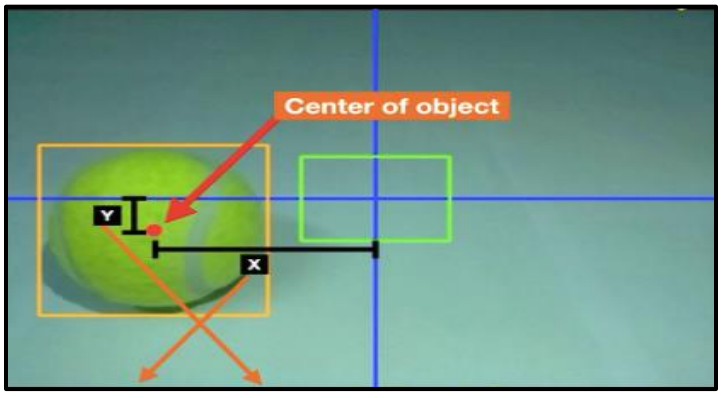



*Figure 115*

Then calculating the error which is the difference between x and y of face or object center and the center of the frame.

Using some basic mathematics to find error in both x coordinate and y coordinate

$$\Delta X = face\_CenterX - (Frame\_Width/2) \quad (7)$$

$$\Delta Y = face\_CenterY - (Frame\_Height/2) \quad (8)$$

After find delta of x and y which also called errors represent the difference in distance between the center of detected face in frame and the center of frame in pixels.

### Adjust Angels Based on Deviation:

The adjustment of angles is based on how much the face center deviates from the frame center. The larger the deviation the larger is the angle and the smaller the deviation the smaller is the angle. which is mean that angles adjust proportional to the deviation the face from the center of the frame.

$$\text{Angle X}=\frac{\Delta X}{N} \qquad \text{Angle y}=\frac{\Delta Y}{N} \quad (9)$$

N represent to some fraction from 0 to 100 to convert pixels into degrees so pan move whether right or left. By some try and error the optimal N for better movement for pan and tilt system is 45 .

For better movement and avoid excessive movement by applying constraints to angles. For Angle x stay within -90 to 90 range .For Angle y stay within 5 to 90 range.

### The Pan-Tilt mechanism:

First by find the errors which called deltas in both y and x coordinates. By subtract face width in frame from the center of the frame in both y and x coordinates.

```
face_centerX= tracking_bboxes[i][0] + (face_width_in_frame/2)
face_centerY = tracking_bboxes[i][1] +(face_height_in_frame/2)

deltaX=face_centerX-(640//2)
deltaY=face_centerY-(480//2)
```

*Figure 116*

Initializes the pan angle attribute to 0 degrees. This attribute represents the horizontal angle. Initializes the tilt angle attribute to 45 degrees. This attribute represents the vertical angle. And then an asynchronous method named move that takes two parameters: XDelta d YDelta. The make the Pan angel proportional to Xdelta or errors which represent how far an object from center of frame. Then ensure that the pan angle in range of 90 to -90 which it can scan 180 degrees in right and left sides and ensure also it won't exceed -90 degrees to keep it in a correct position. Make tilt angle in constant range from 5 to 90degrees.

```python
class PanTiltController:
    def __init__(self):
        self.pan_angle = 0
        self.tilt_angle = 45

    async def move(self, xDelta, yDelta):
        self.pan_angle += xDelta // 45
        self.tilt_angle += yDelta // 45

        self.pan_angle = max(-90, min(90, self.pan_angle))
        self.tilt_angle = max(5, min(90, self.tilt_angle))

        print(self.pan_angle,self.tilt_angle)

        # data = {"pan": self.pan_angle, "tilt": self.tilt_angle}
        # async with aiohttp.ClientSession() as session:
        #     async with session.post(url_R_pt, json=data) as response:
        #         if response.status == 200:
        #             pass
        #         else:
        #             print("Failed to send Pan-Tilt command")
        # return self.pan_angle

    async def order(self, pan, tilt):

        self.pan_angle = max(-90, min(90, pan))
        self.tilt_angle = max(5, min(90, tilt))

        print(self.pan_angle,self.tilt_angle)
```

*Figure 117*

The Pan Tilt Controller class is designed to manage the pan (horizontal) and tilt (vertical) angles of a device, such as a camera. The class initializes with a pan angle of 0 degrees and a tilt angle of 45 degrees. The `move` method is an asynchronous function that takes two parameters: `xDelta` and `yDelta`, representing changes in the pan and tilt angles, respectively. The method adjusts the pan and tilt angles by dividing `xDelta` and `yDelta` by 45 and adding the results to the current angles. To ensure the angles remain within a specified range, the pan angle is constrained between -90 and 90 degrees, while the tilt angle is constrained between 5 and 90 degrees. These constraints are applied using the `max` and `min` functions. Finally, the adjusted pan and tilt angles are printed to the console, providing a way to monitor the changes. The `Pan Tilt Controller` class, with its `move` method, offers a straightforward approach to controlling and adjusting the orientation of a device within defined limits .The pan-tilt system, combined with intelligent face tracking and recognition algorithms, allows the camera to dynamically adjust its orientation to keep the target face in view. This system is crucial for applications where the camera needs to monitor and interact with moving subjects, ensuring that the face is always centered and within the field of view. The searching algorithm further enhances this capability by systematically scanning the environment when the target face is not immediately visible.

## Searching Algorithm Overview

The searching algorithm is designed to locate a specific person based on their face. The algorithm operates by receiving input from the mobile application, detecting faces in real-time, recognizing the faces, and then adjusting the camera to keep the recognized face in the center of the frame.

### Key Steps

1. **Activation**: The search condition is set to true upon receiving the chosen name from the mobile application.
2. **Face Detection**: The system continuously captures frames and detects faces within them.
3. **Face Recognition**: Recognized faces are compared against the chosen name.
4. **Tracking and Notification**: If a match is found, the camera is centered on the person, and a notification is sent to the user.

The search condition is activated based on input from the mobile application. The user selects a name then a communication process via a flask occurs ordering the autonomous car system to start search process

For better understanding how the whole thing work Let's talk about our data structure first then we will continue.

real_time_ids is a dictionary  updated using tracking_ids to maintain real-time information about each tracked face,  id_face_mapping dictionary maps tracking IDs to a list of attributes associated with each face, including the name, id_face_mapping[tracking_ids[i]] = [name, round(score * 100, 2), face_id, round(Distance, 2), deltaX, deltaY, 'KNOWN'] , The search looping began  after the conditions are met and the movement of pan tilt is started by moving 180 degrees from left to right in forward direction this is done while incrementing the movement 45 degrees and after each increment the processes of detection and recognition are working through every detected face in front of the camera and the comparison process of every recognized name and the one that was chosen previously by the user

```
elif chosenName_bymobile:  # Remember to reset the chosen name after finishing the search/attack (make it none at
    print("searching for the name in data")
    chosenName_general_id = None
    for track_id, real_time_data in real_time_ids.items():
        print("looping")
        if chosenName_bymobile == real_time_data[0]:
            chosenName_general_id = track_id
            is_found=True
            break
```

*Figure 118*

While looping, if the comparison came to true result the Boolean variable is_found is set to True which will break the second condition of the searching process .

```
if SearchingCond and not is_found:
    asyncio.run(fetch_chosen_name())
    asyncio.run(SearchingAlgorithm())
```

*Figure 119*

How ever if it is not the search loop continues and the movement of pan tilt as well, after moving 180 degrees from left to right and still we don't find the chosen person the movement began from right to left 180 degrees but in the backward direction which will satisfy a total of 360 degrees if the person is found during the whole process it returns a notification 'f' to the mobile and then the camera is centered in the direction of the specified person.

If the person is not found after checking all angles a notification 'x' is sent to the mobile application then the reset name function is called

```
elif not SearchingCond:
    print("process is killed")
    await ResetName()
    return

# await Rotate(180)

await asyncio.sleep(2)
print("Searching Process finished without finding any faces")
await PanTiltMoving.order(0,45)
await SendNotifications('x') # send notification
await ResetID()
await ResetName()
# SearchingCond=False
```

*Figure 120*

## Implementation

**1-Initialize Camera Angles**:

```
pan=-90
tilt=45
print('pan= -90')
await PanTiltMoving.order(pan,tilt)
await asyncio.sleep(1)
```

*Figure 121*

**2-Left-to-Right Scan:**

```
while pan<90 and not is_found and SearchingCond:
    pan=pan+45
    print('pan++ =',pan)
    await PanTiltMoving.order(pan,tilt)
    await asyncio.sleep(2)
```

*Figure 122*

**3-Check if Target is Found:**

```
if is_found:
    print("is found")
    await SendNotifications('f') # send notification
    await ResetID() # to stop fetching id / executing search algo.
    return
```

*Figure 123*

**4-Handle Interrupted Search:**

```
elif not SearchingCond:
    print("process is killed")
    await ResetName()
    return
```

*Figure 124*

**5-Right-to-Left Scan and End Search Without Finding Target:**

```
    while pan>-90 and not is_found and SearchingCond:
        pan=pan-45
        print('pan-- =',pan)
        await PanTiltMoving.order(pan,tilt)
        await asyncio.sleep(2)


    await asyncio.sleep(2)
    print("Searching Process finished without finding any faces")
    await PanTiltMoving.order(0,45)
    await SendNotifications('x') # send notification
    await ResetID()
    await ResetName()
    # SearchingCond=False
    return
```

*Figure 125*

## Conclusion

The searching algorithm is a critical component of the autonomous car face recognition system. It enables the system to dynamically locate and track a specific person based on their facial features. The integration of this algorithm with real-time face detection, recognition, and camera control ensures efficient and accurate identification in various environments.

## The Car attack

plays a crucial role in navigating and interacting with targets detected by the system. Here's a detailed breakdown of its importance and functionality:

### Importance

**Target Alignment and Tracking**:

The function ensures that the car is properly aligned with a target. It continuously checks and adjusts the car's orientation until it faces the target within an acceptable tolerance level.

**Distance Calculation**:

It computes the real distance to the target using the provided x, y, and z coordinates. This is critical for determining how far the car needs to move forward or when to stop.

**Dynamic Movement**:

Based on the calculated distance, the car adjusts its movement. If the target is far away, it moves forward; if it's close enough, it stops. This dynamic behavior is essential for precise navigation and interaction with the environment.

**Safety and Accuracy**:

By incorporating ultrasonic sensor checks (UltraSonic function), it ensures that the car avoids obstacles and maintains a safe distance, enhancing the overall reliability and safety of the system.

when there is a targeted person where distance between the center of the camera and the center of that person is known but that person is standing right the car with positive delta so we want to steer the car right to make the person faces the camera not sided face to it to have a clear vision, the car has two wheels if we want to steer right for example the right wheel should have a speed lower than the lift wheel but not zero speed until the distance between us and the target is 95 cm or some appropriate value, we need to adjust the speed of wheels and do some calculations to make it done.

## Calculating Desired Steering Angle:

The desired angle is calculated using trigonometry to get the arctangent of the y offset over x offset.

This gives us the ideal angle required to align the target face with the center of the camera.

**Getting Current Steering Angle:**

We need to know the current orientation of the car to determine how much it needs to steer.

**Calculating Steering Delta**

- Steering delta is the difference between desired and current angle:
    - steer_delta = desired_angle - current_angle
- This tells us the amount of adjustment needed in the steering.

**Mapping Delta to Wheel Speeds**

- To steer, we adjust the speeds of left and right wheels differently
- A positive delta means we need to steer right
- So for a right turn:
    - Reduce right wheel speed
    - Keep left wheel at max speed
- The amount reduced is proportional to the steer_delta
- e.g. right_speed = MAX_SPEED - steer_delta

**Gradual Acceleration/Deceleration**

- As distance reduces, slowly increase speeds from MIN to MAX
- This provides a smoother brake rather than abrupt stop
- We divide the current distance by max distance and multiply to speed

## Priority Choosing for Known Faces

### Introduction

The priority choosing mechanism is a critical part of your face recognition system, designed to optimize the identification and tracking process. This section focuses on ensuring that the system accurately identifies and prioritizes known faces, enhancing the overall efficiency and reliability of the recognition process.

The main idea behind priority choosing is to identify and prioritize the most relevant individuals in the camera's field of view. This is achieved by assigning priority to known faces, which are individuals whose facial data has already been stored and recognized by the system. The process involves comparing the tracking IDs of individuals detected in real-time with a database of known faces and selecting the one with the highest priority (usually the one with the lowest ID value).

real_time_ids → it is dictionary that save real faces in video with these facial coordinates

```
# Priority choosing for known faces only
min_id=float('inf')
for trackid in tracking_ids:
    real_time_ids[trackid] = id_face_mapping.get(trackid, ['Unknown', 'Unknown', float('inf'), 0, 0, 0, 'Unknown'])
    if  real_time_ids[trackid][6]!='UN_KNOWN' and real_time_ids[trackid][2]<min_id :
        min_id=real_time_ids[trackid][2]
        general_id=trackid
        is_known_face=True

current_ids = [data[2] for data in real_time_ids.values()]
print("Current IDs in frame: ", current_ids)
```

## Usage and functionality

1. **Initialization:**
    - The system starts by initializing a variable to hold the minimum ID value (min_id) and setting it to infinity. This ensures that any valid ID will be less than the initial value.
2. **Iterating Through Tracking IDs:**
    - The system iterates through the list of tracking_ids obtained from the face detection and tracking module. For each trackid, it retrieves the corresponding data from a dictionary (id_face_mapping) that maps IDs to facial data.
3. **Prioritizing Known Faces:**
    - During the iteration, the system checks if the face associated with the current trackid is a known face (i.e., not labeled as 'UN_KNOWN').
    - If the face is known and its ID is less than the current min_id, the system updates min_id and sets this trackid as the general_id. This general_id represents the face with the highest priority among those currently being tracked.
4. **Fetching and Handling IDs:**
    - Once the iteration is complete, the system collects the IDs of all individuals currently in the frame and prints them for logging purposes. These IDs are then posted to a Flask endpoint for further processing and logging.
5. **Action Based on Priority:**
    - If a known face with the highest priority is identified (is_known_face is True) and no specific ID has been chosen from a mobile interface (chosen_id_bymobile is None), the system updates the tracking coordinates for the identified face.
    - It then triggers actions such as moving the camera to keep the face centered (PanTiltMoving) and possibly initiating an attack sequence (Attack) based on the face's coordinates.
6. **Handling Chosen ID from Mobile:**
    - If a specific ID is chosen via a mobile interface, the system searches for this ID in the real-time tracking data and updates the tracking coordinates accordingly. This allows users to manually prioritize a specific individual.

7. **Handling Chosen Name from Mobile:**
    - ❖ Similarly, if a specific name is chosen from the mobile interface, the system searches for this name in the real-time tracking data. If found, it updates the tracking coordinates and initiates relevant actions.

## Continuously fetching IDs

### Introduction

Fetching IDs is an integral part of your face recognition system, enabling the dynamic identification and prioritization of individuals in real-time. This process ensures that the system can adapt to user inputs and maintain accurate tracking of faces, whether the input is received from a mobile interface or another external source.

The main idea behind fetching IDs is to continuously monitor and retrieve specific identifiers (IDs or names) from an external source, such as a mobile interface, and use this information to update the tracking priorities and behaviors of the system. This process involves asynchronous network requests to fetch the latest chosen IDs or names and update the system's state accordingly.

### Three cases for Fetching IDs

**Using Default Priority ID:**

```
if is_known_face and chosen_id_bymobile is None:
xDelta=real_time_ids[general_id][4]
yDelta=real_time_ids[general_id][5]
zTrack=real_time_ids[general_id][3]
# asyncio.run(Attack(xdelt,ydelt,zTrack))


if prev_xDelta is None or prev_yDelta is None or abs(xDelta - prev_xDelta) > threshold or abs(yDelta - prev_yDelta) > threshold:
    asyncio.run(PanTiltMoving.move(xDelta,yDelta))
```

```
    prev_xDelta, prev_yDelta = xDelta, yDelta
print('min_id: ',min_id,', general_id: ',general_id)
```

- Provide a fallback ID when the mobile device does not specify any ID.
- **Implementation:**
  - **Default ID Selection:** Choose a minimum or priority ID (e.g., smallest available ID in the system).
  - **Usage:** If the mobile does not send an ID, recognize.py defaults to using this ID for recognition purposes.

### Sending Real-Time IDs from Video Feed:

```
elif isinstance(chosen_id_bymobile, int): #as chosen id could be not none but 's'
    chosen_general_id = None
    for track_id, real_time_data in real_time_ids.items():
        if chosen_id_bymobile == real_time_data[2]:
            chosen_general_id = track_id
            break


    if chosen_general_id is not None:
        xDelta = real_time_ids[chosen_general_id][4]
        yDelta = real_time_ids[chosen_general_id][5]
        zTrack = real_time_ids[chosen_general_id][3]


        if prev_xDelta is None or prev_yDelta is None or abs(xDelta - prev_xDelta) > threshold or
abs(yDelta - prev_yDelta) > threshold:
            asyncio.run(PanTiltMoving.move(xDelta,yDelta))
            prev_xDelta, prev_yDelta = xDelta, yDelta


        # asyncio.run(Attack(xdelt,ydelt,zTrack))
        asyncio.run(Attack(xDelta, yDelta, zTrack))
        print('chosen_id_bymobile: ',chosen_id_bymobile,', chosen_general_id: ',
chosen_general_id)
```

```
    else:
        print(f"Chosen ID {chosen_id_bymobile} not found in real_time_ids")
```

- Allow the mobile device to select an ID in real-time from the live video feed.
- **Implementation:**
  - **Real-Time Video Feed:** Continuously stream video frames to recognize.py.
  - **ID Extraction:** Detect faces in real-time and assign IDs to these faces.
  - **Mobile Interaction:** Send these real-time IDs to the mobile device for selection.
  - **Selection Process:** Mobile users choose an ID from the options provided by recognize.py for person recognition.

Name-Based Recognition and Search:

```
elif chosenName_bymobile:
    print("searching for the name in data")
    chosenName_general_id = None
    for track_id, real_time_data in real_time_ids.items():
        print("looping")
        if chosenName_bymobile == real_time_data[0]:
            chosenName_general_id = track_id
            is_found=True
            break


    if chosenName_general_id is not None:
        xDelta = real_time_ids[chosenName_general_id][4]
        yDelta = real_time_ids[chosenName_general_id][5]
        zTrack = real_time_ids[chosenName_general_id][3]
        # asyncio.run(Attack(xdelt,pan,zTrack))
        asyncio.run(Attack(xDelta, yDelta, zTrack))
        if prev_xDelta is None or prev_yDelta is None or abs(xDelta - prev_xDelta) > threshold or
abs(yDelta - prev_yDelta) > threshold:
```

```
            asyncio.run(PanTiltMoving.move(xDelta,yDelta))
            prev_xDelta, prev_yDelta = xDelta, yDelta
        print('chosenName_bymobile: ',chosenName_bymobile,', chosenName_general_id: ',
chosenName_general_id)
    else:
        print(f"Chosen Name {chosenName_bymobile} not found in real_time_ids yet")
```

- Enable recognition based on a person's name sent from the mobile.

## Implementation

- **Mobile Input:** Receive a person's name from the mobile device.
- **Database Search:** Query the face recognition system to find faces associated with the provided name.
- **Recognition:** Perform recognition based on the identified faces linked to the provided name.
- **Feedback:** Provide recognition results to the mobile device for confirmation or further action.

## Usage and functionality:

**Continuous Fetching**: The system continuously fetches the ID at regular intervals. This ensures that the system always has the latest ID to track or recognize. The interval can be adjusted based on the application's requirements. A shorter interval means the system gets updated IDs more frequently, which is useful in dynamic environments where the target ID might change quickly.

```
async def fetch_chosen_id():
    global chosen_id_bymobile,SearchingCond,is_found
    try:
```

```
        async with aiohttp.ClientSession() as session:
            async with session.get(url_chs_id) as response:
                if response.status == 200:
                    data = await response.json()
                    if data['id']=='s':
                        SearchingCond=True
                        # print("Searching Cond.")
                    else:
                        SearchingCond=False
                        is_found=False
                        chosen_id_bymobile = int(data['id'])
                        if chosen_id_bymobile==0:
                            chosen_id_bymobile=None #at the end of attack, ai would post (0) to
/chosen_id
                        else:
                            print("Fetched chosen ID:", chosen_id_bymobile)
                else:
                    # print(f"Failed to fetch chosen ID: {response.status}")
                    pass
    except Exception as e:
        print(f"Error fetching chosen ID: {e}")


async def continuously_fetch_chosen_id(interval=2):
    while True:
        await fetch_chosen_id()
        await asyncio.sleep(interval)
```

**Error Handling**: Robust error handling is in place to ensure the system can handle scenarios where the server might not respond, or the response might not be as expected. This is crucial for maintaining the stability and reliability of the system.

**ID Update Logic**: When a new ID is fetched, the system updates its internal state. If the fetched ID is a specific signal (like 's'), it sets a searching condition. If the ID is 0, it resets the chosen ID, indicating the end of an operation or a reset state. For other IDs, it updates the chosen ID, which the system will then use for further operations.

### Fetching Names

Similarly, the system can fetch names, which might be necessary for applications where the identification of individuals by name is required.

```
async def fetch_chosen_name():
    global chosenName_bymobile,is_found
    try:
        async with aiohttp.ClientSession() as session:
            async with session.get(url_chs_nm) as response:
                if response.status == 200:
                    data = await response.json()
                    chosenName_bymobile = data['name']
                    is_found=False
                    if chosenName_bymobile==0: chosenName_bymobile=None #at the end of attack, ai would post (0) to /chosen_name
                    print("Fetched chosen Name:", chosenName_bymobile)
                    return
                else:
                    # print(f"Failed to fetch chosen Name: {response.status}")
                    pass
    except Exception as e:
        print(f"Error fetching chosen Name: {e}")
```

- **Continuous Fetching**: Just like IDs, names are fetched continuously at regular intervals. This ensures that the system can respond to updates or changes in the list of names it needs to recognize or track.
- **Name Update Logic**: When a name is fetched, the system updates its internal state. If the fetched name is 0, it resets the chosen name. For other names, it updates the chosen name, which the system will then use for further operations.

### Usage Scenarios

- **Security Systems**: In security systems, fetching IDs and names allows the system to dynamically update the list of persons it needs to recognize. This can be useful in scenarios like access control, where the list of authorized persons might change frequently.
- **Event Management**: In events, fetching IDs and names allows the system to dynamically update the list of attendees. This can be useful for managing access to different areas within an event.
- **Retail and Customer Service**: In retail, fetching IDs and names can help the system recognize VIP customers and provide personalized service.

### Priority Choosing for Known Faces

#### Concept and Importance

In face recognition systems, priority choosing is a method used to ensure that the most relevant or important faces are given precedence during recognition and tracking. This is particularly important in scenarios where multiple faces are detected, and the system needs to focus on the most significant ones.

#### Known Faces Priority

The system prioritizes known faces based on certain criteria, ensuring that individuals already recognized by the system are tracked and identified more accurately. This involves:

- **Minimum ID Check**: The system maintains a mapping of track IDs to face data, including a unique identifier. By comparing these IDs, the system can identify the face with the smallest ID value, which is often used as a proxy for priority or importance. This face is then given priority in tracking and recognition tasks.
- **Updating Real-time IDs**: The system continuously updates a list of real-time IDs and their associated data. This allows it to keep track of all detected faces and their current status (known or unknown).

## Usage Scenarios

- **Security and Surveillance**: In security applications, prioritizing known faces ensures that high-risk individuals or persons of interest are tracked accurately, enabling quick responses to potential threats.
- **Customer Service**: In customer service, prioritizing known faces allows the system to recognize returning customers, facilitating personalized interactions and enhancing customer experience.
- **Access Control**: In access control systems, prioritizing known faces ensures that authorized individuals are granted access promptly, while unknown or unauthorized individuals are flagged for further verification.

## Operational Workflow

1. **Detection and Tracking**: The system detects faces and assigns a unique track ID to each detected face.
2. **Real-time ID Mapping**: For each track ID, the system updates a real-time mapping with face data, including whether the face is known or unknown.
3. **Priority Check**: The system checks the IDs to find the face with the highest priority (smallest ID value).
4. **Action Based on Priority**: Depending on the priority face, the system can perform various actions, such as alerting security personnel, updating customer records, or controlling access gates

# Chapter 9 Mobile Application Interface and Control

## Mobile Application

### Introduction

Smartphones have become an essential part of our lives in recent years, with everyone carrying at least one smartphone 90% of the time. These devices come equipped with numerous features such as wireless communication and impressive processing power. Considering this, smartphones can be utilized for a wide range of applications, from security to controlling other smart devices. In this project, we leveraged the capabilities of Bluetooth Low Energy (BLE), Wi-Fi, and smartphone processing power to introduce features like car authentication and car controlling, searching and video streaming.

## Flutter

Flutter is a framework created by Google, it allows building high-performance, cross-platform applications for android, iOS, web, and desktop from a single codebase. Flutter was first released in 2017 and has gained significant popularity among developers due to its fast development cycles, expressive and flexible UI, native like performance.

### Key Features of Flutter

**Single Codebase**: With Flutter, developers can write code once and deploy it on multiple platforms such as iOS, Android, Web, Windows and Linux. This "write once, run anywhere" approach helps save time and effort.

Dart Programming Language: Flutter uses the Dart programming language, which was also created by Google. Dart is known for its simplicity, readability, and performance. It has a modern syntax, supports both object-oriented and functional programming paradigms, and includes features like hot reload for rapid development.

Widgets: Flutter utilizes a reactive and component-based architecture. The entire UI is composed of widgets, which are reusable and customizable building blocks. Flutter provides an extensive set of pre-designed widgets for constructing user interfaces, as well as the ability to create custom widgets.

Hot Reload: One of Flutter's standout features is its hot reload capability. Developers can make changes to the code and instantly see the results in the app without restarting or losing the app state. This significantly speeds up the development process and helps in experimenting and iterating on designs.

Native-Like Performance: Flutter applications are compiled to native code, allowing them to achieve high performance and near-native speed. Flutter uses its own rendering engine, called Skia, to draw graphics and UI elements. It also provides access to platform-specific APIs and services, giving developers full control and access to native features.

Great Community: Flutter has a vibrant and growing ecosystem with a wide range of community created packages and libraries available through the Flutter package manager, called Pub. These packages cover various functionalities such as networking, state management, databases, and more, allowing developers to leverage existing solutions and accelerate development.

Material Design and Cupertino Widgets: Flutter offers ready-to-use widgets that follow the Material Design guidelines for Android apps and Cupertino design for iOS apps. This helps in creating apps that look and feel native to each platform, providing a consistent user experience.

State Management: flutter has Options for managing state using Stateful Widgets, Inherited Widgets, and more, Support for popular state management packages like Provider, Riverpod, Bloc, and more.

Animation and Motion**:** flutter has Rich Animation Libraries for creating complex animations and motion effects.

Testing Support: Comprehensive support for unit testing, widget testing, and integration testing and has Tools for writing automated tests to ensure code reliability.

### Why Flutter?

Google tries to make the cross-platform development production-ready using the Dart programming language and the Flutter UI framework. Flutter provides a powerful framework for building beautiful, fast, and cross-platform application with a single codebase, along with dart's simple syntax, and the many ready to use packages and libraries available on pub.dev the Flutter package manager it makes one of the best choices for a project like this one. Flutter renders everything by itself in a very good way and it does not use any intermediate bridge to communicate with the OS. It compiles directly to ARM, for mobile, or optimized JavaScript, for the web. The app you create in Flutter will look the same on any platform. That is because of the Skia graphics engine. Moreover, it is easy to both create and quickly update the UI, which with other tools is usually more cumbersome. Another thing to notice is plugins and platform channels that help utilize OS-level features without extra hassle at the top of their potential.

### API with Flutter

Flutter provides http package to use http resources. The http package uses await and async features and provides many high-level methods such as read, get, post, put, head, and delete methods for sending and receiving data from remote locations. These methods simplify the development of REST-based mobile applications. the core methods of the http package are as follows:

- Read: This method is used to read or retrieve the representation of resources. It requests the specified URL by using the get method and returns the response as Future<String>

- Get: This method requests the specified URL from the get method and returns a response as Future<response>. Here, the response is a class, which holds the response information.
- Post: This method is used to submit the data to the specified resources. It requests the specified URL by posting the given data and return a response as Future<response>
- Put: This method is utilized for update capabilities. It updates all the current representation of the target resource with the request payloads. This method requests the specified URL and returns a response as Future<response>
- Head: It is like the Get method, but without the response body.
- Delete: This method is used to remove all the specified resources.

The http package also provides a standard http client class that supports the persistent connection. This class is useful when a lot of requests to be made on a particular server. It should be closed properly using the close () method. Otherwise, it works as a http class.

## Vision Guard Application

In this project, the car is controlled using three options: mobile, AI, and glove, and it is possible to choose between them easily. We use the application for video streaming and control the car, and it is also used to search for the user, as the camera is used to share a live on the application, and I can control the car by the controller, and I can also search for a user inside the database, so the car searches for him until Track it, and finally we can add a new user by entering the name and ID, and then the mobile camera opens to receive the user, and then the new user becomes available user. The application also contains a login system so that only users can access the application, It contains a monthly subscription system in which payment is made through Stripe, and all user data including name, ID, and photos, are placed in the user profile. Finally, notifications were created in the application in order to facilitate the user and make the process of using the application highly efficient and distinct.

### Application Objectives

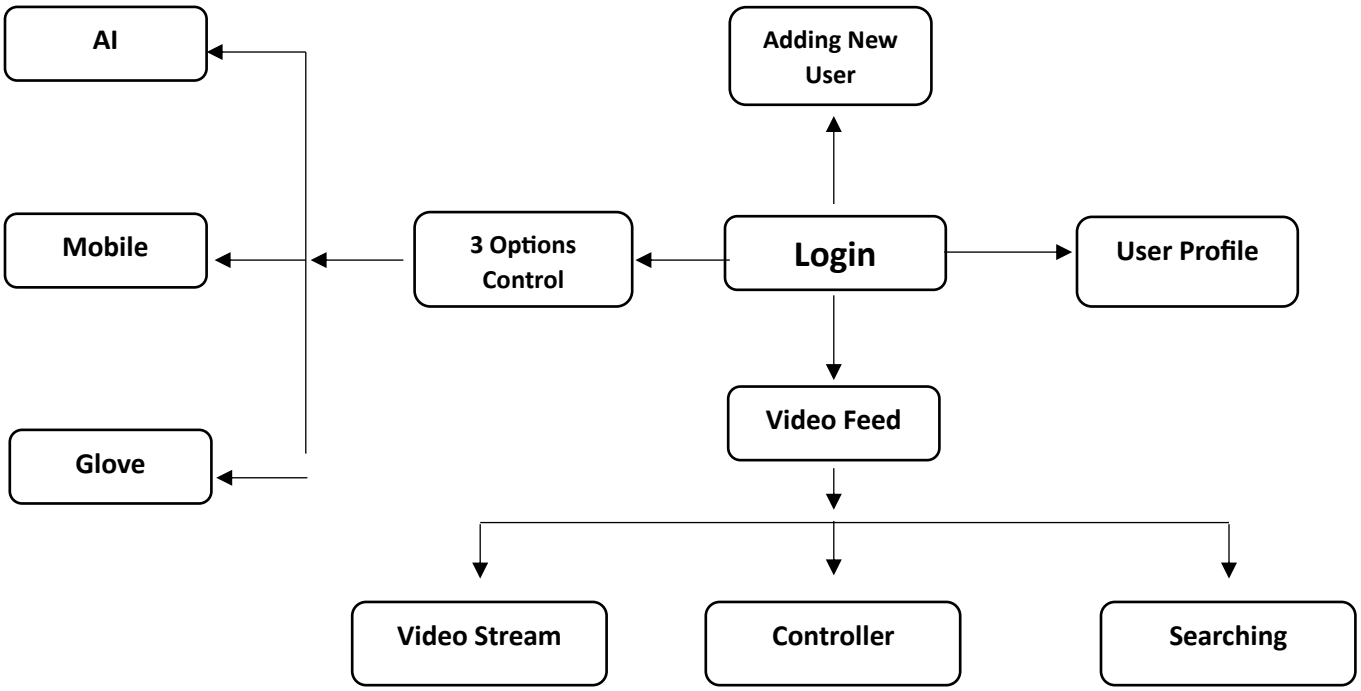
- Mange car leader options
- Car Control using mobile phone.
- Video Streaming
- Searching for users
- Adding new users
- Interactive User Interface with a Positive User Experience

### Programs and Packages Used

- **Visual studio and Android Studio with Flutter Framework**: The IDE and framework recommended by google to develop a cross-platform mobile flutter application.

- **Bloc**: State management package, allows control of the apps states, mainly to limit the number of UI updates that's not needed, which leads to better performance while providing easier access to variables and functions.
- **go router** :is a package for Flutter that simplifies navigation and deep linking in Flutter applications. It provides an easy and declarative way to define routes and handle navigation, making it a great choice for complex navigation scenarios.
- **Shared preferences**: allow you to persist simple data types, such as String, int, double, bool, and List<String>, across app launches. It's a key-value storage mechanism that's perfect for storing user settings, simple data, or app state.
- **Dio**: is a powerful HTTP client for Dart, which supports making API requests, handling responses, and more. It's commonly used in Flutter applications for network operations.
- **Equitable** : is a Dart package that helps to implement equality comparisons for objects in Dart, particularly useful for comparing states in Bloc or other state management solutions.
- **Flutter_svg** : This package allows to render SVG files and strings as Flutter widgets.
- **Flutter_screenutil**: is a great tool It helps in create a responsive design that adapts to different screen sizes and densities.
- **Image-picker**: provides access to the system's UI for selecting images and videos from the phone's library or taking a photo with the camera
- APIs that Flutter application can use to communicate with three very powerful native libraries (uCrop, TOCropViewController and Cropper. js) to crop and rotate images.
- **Flutter VLC Player**: VLC-powered alternative to Flutter's video player that supports iOS and Android.
- **Flutter Joystick**: is a virtual joystick package for Flutter applications that provides interactive joystick components for user interface design.
- **camera**: provides tools to get a list of the available cameras, display a preview coming from a specific camera, and take photos or videos.
- **Path provider**: aids in finding commonly used locations on the host platform's file systems.
- **Cached network image:** A flutter library to show images from the internet and keep them in the cache directory.
- **Shimmer:** A package provides an easy way to add shimmer effect in Flutter project
- **Flutter secure storge:** Flutter Secure Storage provides API to store data in secure storage. Keychain is used in iOS, KeyStore based solution is used in Android**,** we used it to store our token

-Application Flow Diagram



- Application architecture

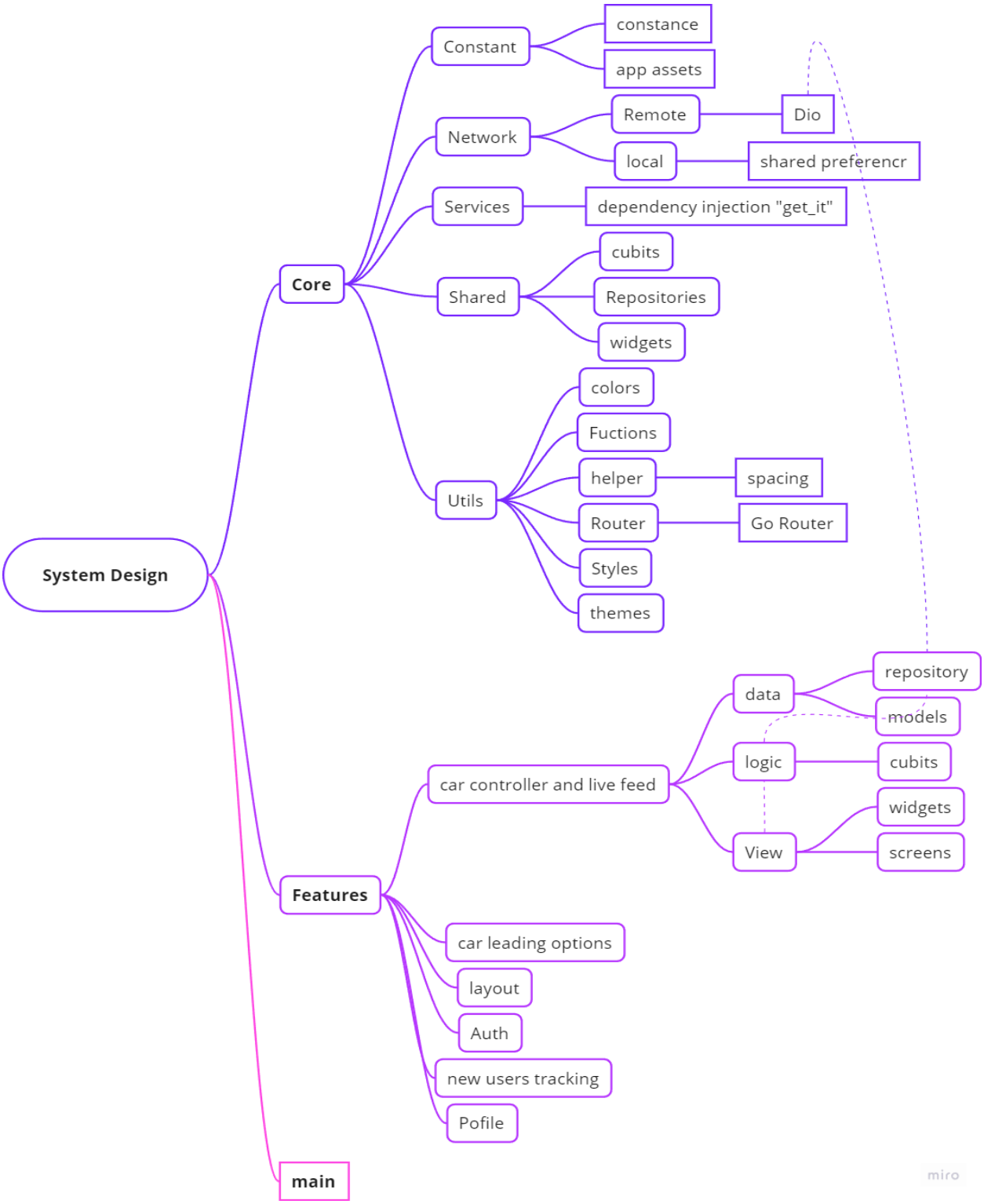



*Figure 126*

As show above we use Dio package to handle https requests so we create this file to easily call Dio methods in our repository files :

```
class DioHelper {
  static late Dio _dio;
  static init() {
    _dio = Dio(
      BaseOptions(
        baseUrl: AppConstance.apisBaseURL,
        receiveDataWhenStatusError: true,
      ),
    );
  }

  static Future<Response> getdata({
    required String endPoint,
    Map<String, dynamic>? query,
    String? bearer,
  }) async {
    if (bearer == null) {
      _dio.options.headers = {
        'Content-Type': 'application/x-www-form-urlencoded',
        'Accept-Language':'en-Us',
        "Accept": "application/json"
      };
    } else {
      _dio.options.headers = {
        'Content-Type': 'application/x-www-form-urlencoded',
```

```
      'Accept-Language': 'en-Us',
      'Authorization': 'Bearer $bearer',
      "Accept": "application/json"
    };
  }
  return await _dio.get(endPoint, queryParameters: query);
}

static Future<Response> postdata({
  required String endPoint,
  data,
  bool isFormData = false,
  query,
  String? bearer,
}) async {
  if (bearer == null) {
    _dio.options.headers = {
      'Content-Type':
          !isFormData ? 'application/json' : 'multipart/form-data',
      'lang': 'en-Us',
      "Accept-Language": "application/json"
    };
  } else {
    _dio.options.headers = {
      'Content-Type':
          !isFormData ? 'application/json' : 'multipart/form-data',
      'Accept-Language': 'en-Us',
      'Authorization': 'Bearer $bearer',
      "Accept": "application/json"
```

```
      };
    }
    return await _dio.post(
      endPoint,
      queryParameters: query,
      data: data,
      options: Options(
          followRedirects: false,
          validateStatus: (status) {
            return status! < 500;
          }),
    );
  }
static Future<Response> postVideoFrame({
  required String endPoint,
  required dynamic data,
  bool isFormData = false,
  Map<String, dynamic>? query,
  String? bearer,
}) async {
  try {
    _dio.options.headers = {
      'Content-Type': 'multipart/form-data',
      if (bearer != null) 'Authorization': 'Bearer $bearer',
    };

    return await _dio.post(
      endPoint,
      queryParameters: query,
```

```
      data:  data,
      options: Options(
        followRedirects: false,
        validateStatus: (status) {
          return status != null && status < 500;
        },
      ),
    );
  } catch (e) {
    print('Error posting video frame: $e');
    rethrow;
  }
}

  static Future<Response> putdata({
    required String endPoint,
    required data,
    query,
  }) async {
    _dio.options.headers = {
      'Content-Type': 'application/json',
      'Accept-Language': 'en-Us',
    };
    return await _dio.put(endPoint, queryParameters: query, data: data);
  }

  static Future<Response> deleteData({
    required String endPoint,
    data,
```

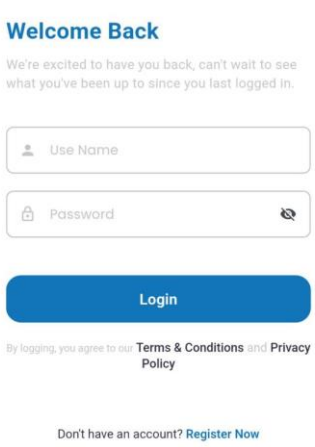
```
    query,
    String? bearer,
  }) async {
    if (bearer == null) {
      _dio.options.headers = {
        'Content-Type': 'application/x-www-form-urlencoded',
        'lang': 'en-Us',
        "Accept-Language": "application/json"
      };
    } else {
      _dio.options.headers = {
        'Content-Type': 'application/x-www-form-urlencoded',
        'Accept-Language': 'en-Us',
        'Authorization': 'Bearer $bearer',
        "Accept": "application/json"
      };
    }
    return await _dio.delete(
      endPoint,
      queryParameters: query,
      data: data,
      options: Options(
          followRedirects: false,
          validateStatus: (status) {
            return status! < 500;
          }),
    );
  }
}
```

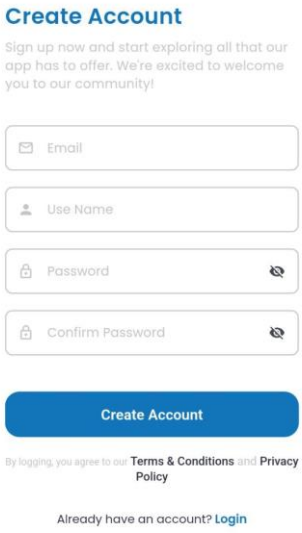
## - Screens and Functionality:

Auth Screens:

Login features two input fields: one for the username and one for the password, which includes a visibility toggle. Users can press the "Login" button to authenticate their credentials. Successful logins result in the authentication token being securely stored using flutter secure storage

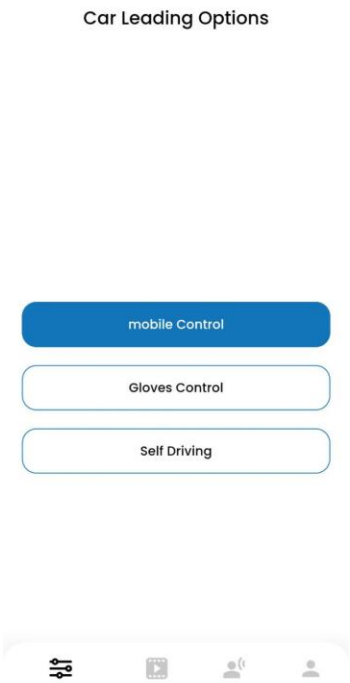
Register features four input fields: email, username, password, and confirm password, each with a placeholder text. After filling out the form, users can create their account by pressing the "Create Account" button. Upon successful registration, the authentication token is securely stored also  using flutter secure storage.



Car Leading options screen:

Here, users can easily change the car leader option among "mobile", "gloves", and "AI" by sending a character to our Flask server via an API.




## Car Controller Option Cubit Function:

```
CarOptionsCubit(this.carOptionsRepo) : super(CarOptionInitial());


  Future<void> changeCarControllerOption(String  option) async {
    AppConstance. carControllerOption = option ;
   emit(ChangeCarOptionLoadingState());
   var response =
       await carOptionsRepo.changeCarControllerOptions(option: option);
   response.fold((failure) {
     emit(ChangeCarOptionErorrState());
   }, (loginModel) {
```

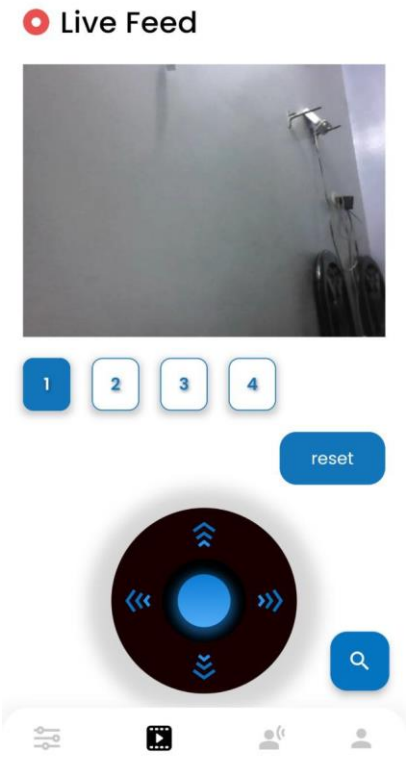
```
      emit(ChangeCarOptionSuccessState());
    });
  }
```

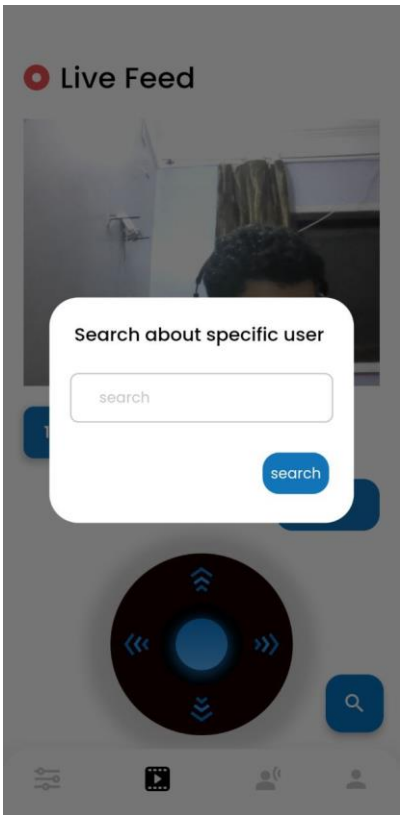
Repository Function:

```
 Future<Either<String, dynamic>> changeCarControllerOptions({required
String  option}) async {
    try {
      var response = await DioHelper.postdata(
          endPoint: EndPoints.setCarOption,
          data: {
            'opt':option
          }
         );
       print(response.data);
        return Right(response.data);

    } catch (error) {
      print(error);
      return Left('Error');
    }
  }
```

## Car Controller and Live feed Screen:

1- In this screen we have a live video feed from the car's movable camera, displayed using VLC player. Users can control the camera direction by swiping on the video. These swipe gestures are mapped to appropriate pan-tilt values for the camera.
2- We display real-time available IDs for tracking by our car, allowing users to easily set a target or reset the tracking.
3- Users can also search for a specific person by name, prompting the car to move in different directions to locate them.
4- Lastly, our joystick controller enables the car to move in various directions by mapping joystick inputs to appropriate values for the car's left and right drivers.



Camera Directions mapping values:

```
onPanUpdate: (details) {
                  if (details.delta.dx > 0 && details.delta.dy > 0) {
                    if (AppConstance.pan <= 88) {
                      AppConstance.pan += 2;
                    }
                    if (AppConstance.tilt >= 7) {
                      AppConstance.tilt -= 2;
                    }

                    CarControllerCubit.get(context)
                        .changePanTiltMovementDirections(
                            pan: AppConstance.pan, tilt: AppConstance.tilt);
                  } else if (details.delta.dx < 0 && details.delta.dy > 0) {

                    if (AppConstance.pan >= -88) {
                      AppConstance.pan -= 2;
                    }

                    if (AppConstance.tilt >= 7) {
                      AppConstance.tilt -= 2;
                    }
                    CarControllerCubit.get(context)
                        .changePanTiltMovementDirections(
                            pan: AppConstance.pan, tilt: AppConstance.tilt);
                  } else if (details.delta.dx > 0 && details.delta.dy < 0) {
                    if (AppConstance.pan <= 88) {
                      AppConstance.pan += 2;
                    }
```

```
                    if (AppConstance.tilt <= 88) {
                      AppConstance.tilt += 2;
                    }
                    CarControllerCubit.get(context)
                        .changePanTiltMovementDirections(
                            pan: AppConstance.pan, tilt: AppConstance.tilt);
                  } else if (details.delta.dx < 0 && details.delta.dy < 0) {
                    if (AppConstance.pan >= -88) {
                      AppConstance.pan -= 2;
                    }
                    if (AppConstance.tilt <= 88) {
                      AppConstance.tilt += 2;
                    }
                    CarControllerCubit.get(context)
                        .changePanTiltMovementDirections(
                            pan: AppConstance.pan, tilt: AppConstance.tilt);
                  }
                },
```

joystick controller Mapping Function:

```
(details) {
                  double l = 0, r = 0;
                  if (details.x < 0 && details.y <= 0) {

                    l = 199;
                    r = (-1 * 0.24) * ((details.x * 100).round() * (-1.0)) + 99;
                  } else if (details.x < 0 && details.y > 0) {
```

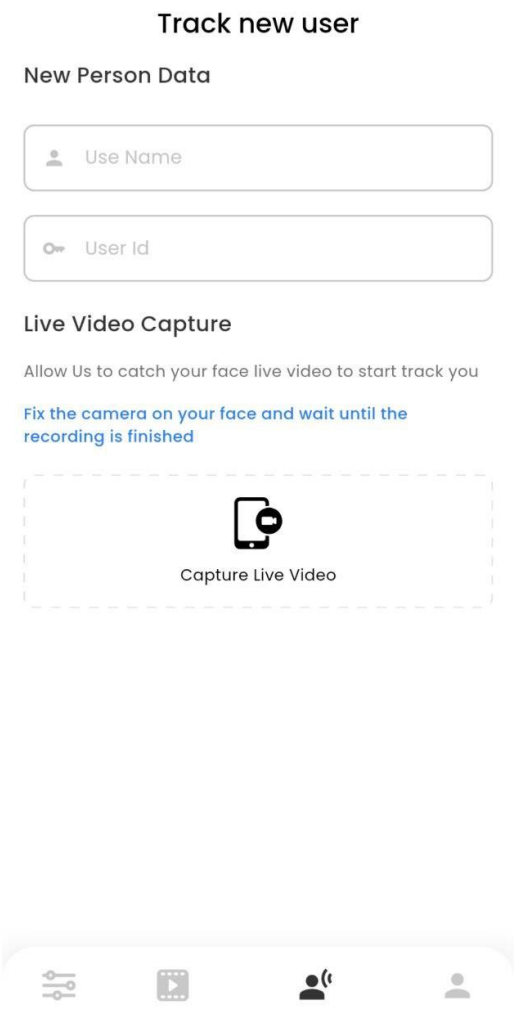
```
          l = 0.24 * (details.y * 100).round() + 125;
          r = 49;
        } else if (details.x > 0 && details.y <= 0) {

          l = ((details.y * 100).round() * (-1.0));
          l = 0.24 * l + 175;
          r = 99;
        } else if (details.x > 0 && details.y > 0) {

          l = 149;
          r = -0.24 * (details.x * 100).round() + 49;
        } else if (details.x == 0 && details.y == 0) {
          l = 101;
          r = 1;
          CarControllerCubit.get(context)
              .carMovementDirection(r: r.round(), l: l.round());
          CarControllerCubit.get(context)
              .carMovementDirection(r: r.round(), l: l.round());
          CarControllerCubit.get(context)
              .carMovementDirection(r: r.round(), l: l.round());
          CarControllerCubit.get(context)
              .carMovementDirection(r: r.round(), l: l.round());
          CarControllerCubit.get(context)
              .carMovementDirection(r: r.round(), l: l.round());
        }

        CarControllerCubit.get(context)
            .carMovementDirection(r: r.round(), l: l.round());
      },
```

## Track New User Screen:

On this screen, we ask the user to enter a new username and ID, then press a button to open the live video stream. This allows our AI model to capture the new user's face and add them to our tracking list.

The camera streams video when the user starts it, and we close the stream after receiving a notification indicating that the AI model has finished its work.

*Figure 127*



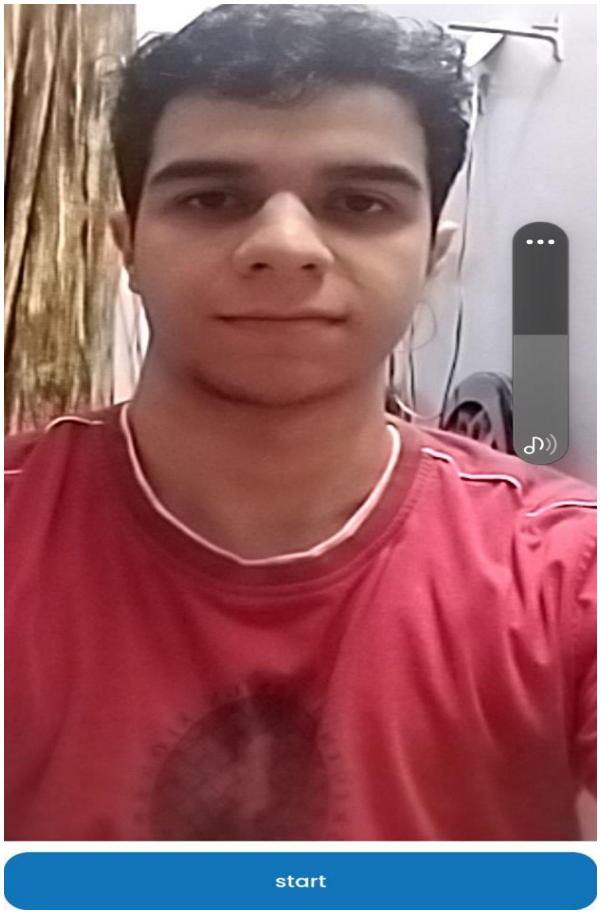




*Figure 128*

Camera Live Stream code:

```
Future<void> startStreaming() async {
  VideoStreamingCubit.get(context).addnewUserToTrack();
  if (controller != null && controller!.value.isInitialized) {

      var tempFile = controller?.takePicture();
      if (tempFile != null) {
        var file = await tempFile;
        VideoStreamingCubit.get(context)
            .sendVideoStreamFrame(encodedFrame: File(file.path));

      }
  }
}
```

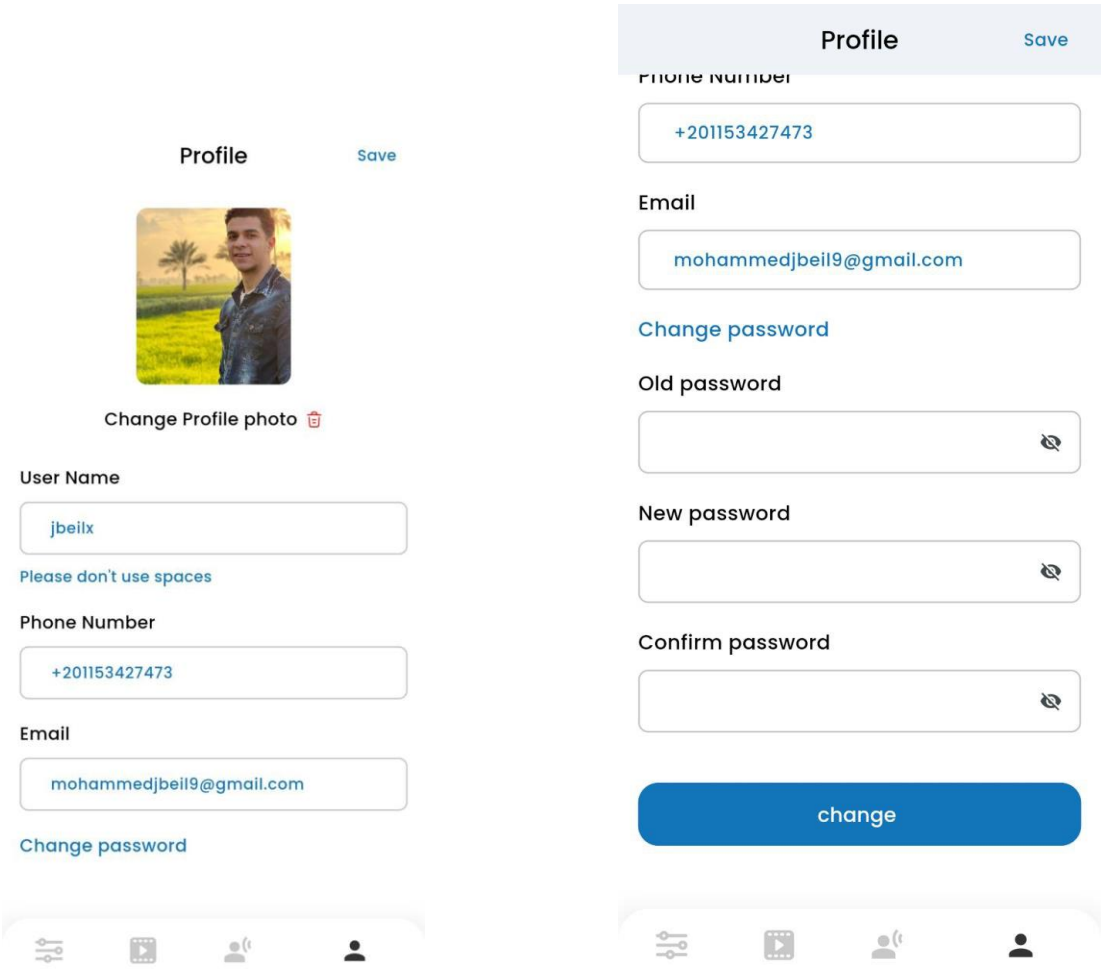
Listen to Notifications Status to Close Camera Function:

```
BlocListener<NotificationsCubit, NotificationsStates>(
     listener: (context, state) {
       if (state is GetNotificationsSuccessState) {
         if (state.message == 'n') {
           RepeatedFunctions.showSnackBar(context,
               message: 'No New Person Found!');
         }
```

```
        if (state.message == 's') {
          RepeatedFunctions.showSnackBar(context,
              message: 'Successfully add new person');
          GoRouter.of(context).pop() ;
          GoRouter.of(context).pop() ;
        }
      }
```

## Profile Screen:

On this screen, we allow the users to manage all their app data, including their avatar, email, phone number, and password.



# Chapter 10: Authentication and Subscription Management

**1. User Input:**

- **Explanation of User Input**: Users provide their credentials, typically a username/email and a password, to log into the system.
- **Key Considerations**:
  - Ensuring fields are not empty.
  - Validating email format.
  - Ensuring password meets security requirements.

**2. API Endpoint:**

- **Explanation of API Endpoint**: An endpoint that receives the login request from the client and processes it.
- **Key Considerations**:
  - Accepting POST requests with user credentials.
  - Routing the request to appropriate backend services.

**3. Backend Validation:**

- **Explanation of Backend Validation**: Validates the format and presence of necessary fields to ensure data integrity.
- **Key Considerations**:
  - Ensuring email is properly formatted.
  - Ensuring password meets minimum length requirements.

**4. Authentication:**

- **Explanation of Authentication**: The process of verifying user credentials against stored data.
- **Key Considerations**:
  - Retrieving user data from the database.
  - Comparing provided password with the stored hashed password.

**5. Database Query:**

- **Explanation of Database Query**: Query the database to find user records based on provided credentials.
- **Key Considerations**:
  - Efficiently querying the database for user data.
  - Handling cases where the user does not exist.

**6. Password Comparison:**

- **Explanation of Password Comparison**: Compare the provided password with the hashed password stored in the database.
- **Key Considerations**:
    - Using secure hashing algorithms ( bcrypt).
    - Ensuring timing-safe comparisons to prevent attacks.

**7. Token Generation:**

- **Explanation of Token Generation**: Generate JWT tokens for session management.
- **Key Considerations**:
    - Creating access and refresh tokens.
    - Setting appropriate expiration times for tokens.

**8. Error Handling:**

- **Explanation of Error Handling**: Handle errors and provide appropriate responses (invalid credentials, server errors).
- **Key Considerations**:
    - Returning meaningful error messages.
    - Logging errors for troubleshooting.

**9. Email/SMS Notification:**

- **Explanation of Email/SMS Notification**: Send notifications for successful logins, failed attempts, or account-related alerts.
- **Key Considerations**:
    - Configuring email and SMS services.
    - Ensuring notifications are sent in a timely manner.

**10. Response to Client:**

- **Explanation of Response to Client**: Send the appropriate response back to the client (success or error message, tokens).
- **Key Considerations**:
    - Structuring the response in a consistent format.
    - Including necessary data such as tokens and user information.

**11. Subscription Handling:**

- **Explanation of Subscription Handling**: Manage subscription status and updates (e.g., checking, creating, updating subscriptions).
- **Key Considerations**:
    - Handling subscription lifecycle events.
    - Integrating with payment gateways if applicable.

# User input and API Endpoint:

## 1-First add new register:

We will take about signup function that handles user registration. When a user submits their email, password, and username, the function first checks if the email already exists in the database. If the email is found, it returns a 400 status with a message indicating the email already exists.

If the email does not exist, the function proceeds to hash the password using bcrypt for security. Next, it creates a customer record in Stripe using the provided email, which is essential for handling payments and subscriptions.

The function then creates a new user record in the database with the hashed password, username, and Stripe customer ID. Upon successfully saving the user, it generates a JWT token for email confirmation and a refresh token.

An email containing the confirmation link and a request for a new link is sent to the user. The confirmation link includes the JWT token, which the user must click to verify their email address.

Finally, the function sends a success response with a 201 status and includes the email message and token. If saving the user fails, a 400 status with an error message is returned. Any errors encountered during the process are caught, logged, and a 500 status is sent back to the client, ensuring robust error handling.

## 2-now to login we should first confirm email:

The confirm Email function handles email confirmation for user registration. When a user clicks the confirmation link, the function extracts the token from the URL parameters
after clicking the link:
t updates the user's record in the database, setting confirm Email to true for the user with the corresponding ID who has not yet confirmed their email. If the update is successful, the function responds with a 201 status and a message indicating the email is confirmed and the user can log in.

**3- sign In function:**

The sign In function handles user login by verifying email and password credentials. Upon receiving the login request, it searches for a user with the provided email. If the user is not found, it give me invalid account.

If the user exists but has not confirmed their email, it prompts the user to confirm their email. If the email is confirmed, the function compares the provided password with the stored hashed password using bcrypt. If the passwords do not match, it responds with an "Invalid password" message.

When the password matches, a JWT token is generated, signed with the user's ID and a logged-in status, and set to expire in 30 hours. The user’s online status is updated in the database, and a successful login response is sent with the token, user role, username, and status code.

3-Backend Validation:

```
const dataMethods = ['body', 'params', 'query', 'headers']
export const validation = (schema) => {
    return (req, res, next) => {
        const validationErr = []
        dataMethods.forEach(key => {
            if (schema[key]) {
                console.log(key);
                const validationResult =
schema[key].validate(req[key], { abortEarly: false })
                if (validationResult?.error?.details) {
                    validationErr.push(validationResult.error.details)
                }
            }
        })
        if (validationErr.length) {
            res.json({ message: "validation error", err: validationErr
})
        } else {
            next()
        }
    }
}
```

We designed it to validate incoming HTTP request data against a specified schema. The dataMethods array contains the keys 'body', 'params', 'query', and 'headers', representing different parts of the request that can be validated.

The validation function takes a schema object as an argument, which outlines the validation rules for each part of the request. Inside the function, an empty array validationErr is initialized to collect validation errors. The function then iterates over dataMethods and checks if the schema has validation rules for each method. If rules exist, it validates the corresponding request data using the schema's validate method. If validation errors are found, they are added to validationErr.

Finally, if validationErr contains any errors, the function responds with a JSON object detailing the validation errors. If no errors are found, it calls next() to proceed to the next middleware.

### Register validation:

We used Joi library to ensure that the data entering your application adheres to the defined rules, thus preventing invalid or malicious data from causing issues. By validating input at the boundaries of your application, you can catch errors early and provide clear feedback to users.

```
import joi from "joi";

export const signUp = {
  body: joi
    .object()
    .required()
    .keys({
      userName: joi.string().required(),
      email: joi.string().email().required(),
      role: joi.string(),
      password: joi
        .string()
        .pattern(
          new RegExp(/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[a-zA-
Z]).{8,}$/)
        )
        .required(),
      cPassword: joi.string().valid(joi.ref("password")).required(),
    }),
  params: joi.object().required().keys({
  }),
};

export const signIn = {
  body: joi
    .object()
    .required()
    .keys({
      email: joi
        .string()
        .email({ minDomainSegments: 1, tlds: { allow: ["com", "net",
"edu"] } })
        .required()
        .messages({
          "any.required": "Email is Required",
```

```
          "string.empty": "not allowed to be empty",
          "string.base": "only string is allowed",
        }),
      password: joi
        .string()
        .pattern(
          new RegExp(/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[a-zA-
Z]).{8,}$/)
        )
        .required(),
    }),
};

export const logOut = {
  headers: joi
    .object()
    .required()
    .keys({
      authorization: joi.string().required(),
    })
    .options({ allowUnknown: true }),
};
export const forgetPassword ={
  body: joi
    .object()
    .required()
    .keys({
      email: joi
        .string()
        .email({ minDomainSegments: 1, tlds: { allow: ["com", "net",
"edu"] } })
        .required()
        .messages({
          "any.required": "Email is Required",
          "string.empty": "not allowed to be empty",
          "string.base": "only string is allowed",
        }),
        password: joi
```

```
.string()
        .pattern(
          new RegExp(/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[a-zA-
Z]).{8,}$/)
        )
        .required(),
      cPassword: joi.string().valid(joi.ref("password")).required(),
      code: joi.number().required()
    }),

}
```

In this code we use a complex regular expression to ensure it includes:

-At least one digit ((?=.*\d))

-At least one lowercase letter ((?=.*[a-z]))

-At least one uppercase letter ((?=.*[A-Z]))

-At least one character that is either a letter or digit ((?=.*[a-zA-Z]))

-A minimum length of 8 characters (.{8,})

## 4-Authentication:

Authentication is the process of verifying the identity of a user or system. It's a critical aspect of security in computer systems and networks, ensuring that only authorized individuals or entities can access resources, data, or services and there is many types of authentication but we used Token-Based Authentication:

Involves issuing a token (a small piece of data) to users upon successful login. The token is then used to access resources without repeatedly entering credentials. Examples include JWT (JSON Web Tokens) and OAuth tokens.

**Code :**

```
import jwt from "jsonwebtoken";

import { userModel } from "../DB/models/user.js";
export const auth = () => {
  return async (req, res, next) => {
    try {
      const { authorization } = req.headers;
      if (!authorization?.startsWith(process.env.BearerKey)) {
        res.json({ message: "In-valid token or In-valid Bearer Key"
});
      } else {
        console.log({ authorization });
        const token = authorization.split(process.env.BearerKey)[1];
        console.log({ token });
        const decoded = jwt.verify(token, process.env.TOKENSIGNATURE);
        console.log(decoded);
        if (!decoded?.id || !decoded.isLoggedIn) {
          res.json({ message: "In-valid payload" });
        } else {
          const user = await userModel
            .findById(decoded.id)
            .select("userName email"); //{} null
          if (!user) {
            res.status(400).json({ message: "In-valid Token user" });
          } else {
            req.user = user;
            next();
          }
        }
```

```
      }
    } catch (error) {
      res.status(500).json({ message: "Catch error", error });
    }
  };
};
```

**Breakdown of the Code**

**1-Token Extraction:**

The authorization header from the incoming request is checked. If it does not start with the expected Bearer key (defined in **process.env.BearerKey**), an error message is returned.

**2-Token Validation:**

The token is extracted from the authorization header by splitting the string with the Bearer key.

The extracted token is then verified using the jwt.verify method, which checks the token against the secret signature (defined in **process.env.TOKENSIGNATURE**).

**3-Payload Check:**

The decoded token payload is checked to ensure it contains both an id and isLoggedIn property. If either is missing, an error message is returned.

**4-User Verification:**

If the payload is valid, the code attempts to find the user in the database using the userModel.findById method.

If the user exists, their information (username and email) is attached to the req object, allowing subsequent middleware or route handlers to access it.

If the user does not exist, an error message is returned.

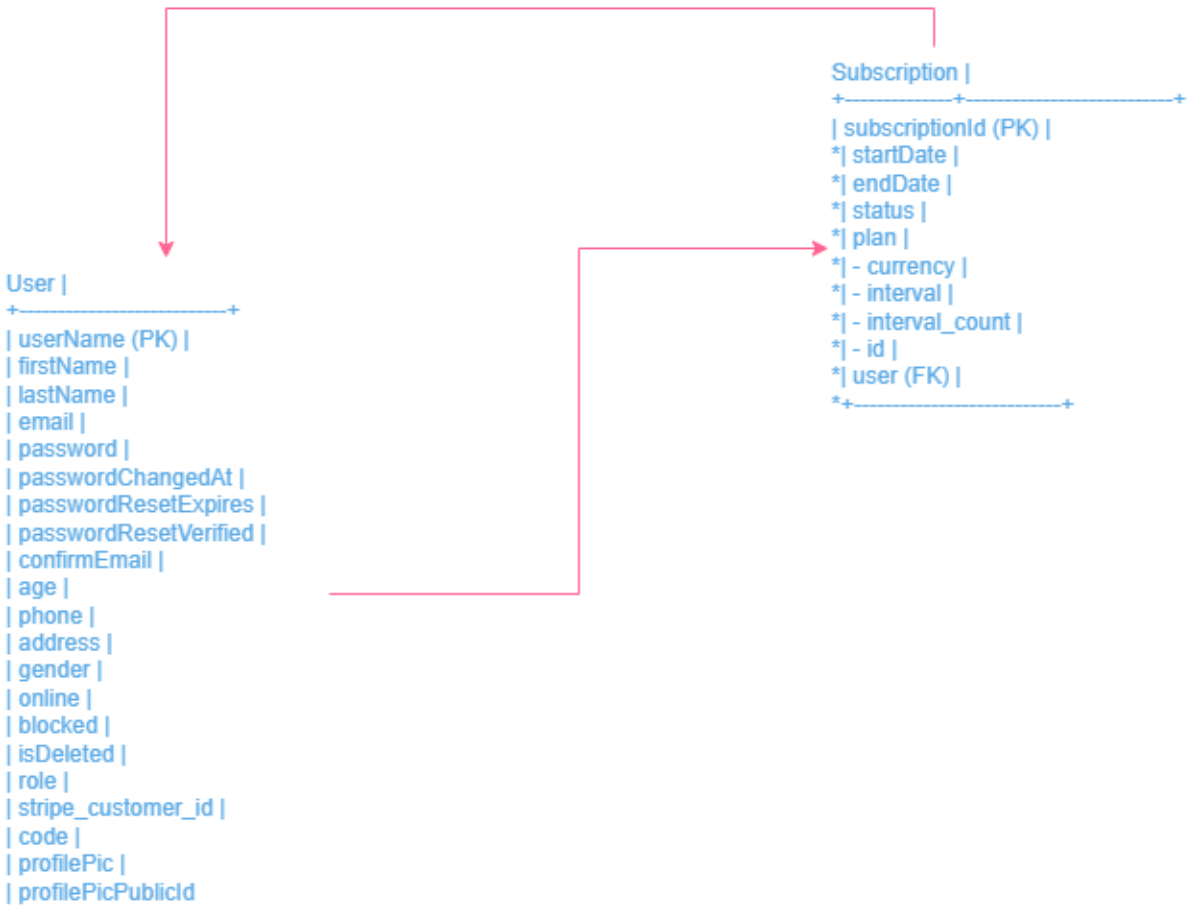
**5-Error Handling:**

Any errors during the process are caught and a generic error message is returned with a 500 status code.

## 5-Database Query:

We use in our project Mongo database and MongoDB is a NoSQL database known for its flexibility, scalability, and ease of use. It stores data in JSON-like documents, which allows for dynamic schema design. This document-oriented approach supports embedded data models and rich queries, making MongoDB well-suited for applications requiring rapid development and iteration. MongoDB is horizontally scalable through sharding, distributing data across multiple servers for high availability and performance. It's widely used in various industries for its ability to handle large volumes of data and support for complex operations, making it a popular choice for modern web, mobile, and big data applications.

**ER diagram:**



**Models code:**
**1-user model:**

```
import { Schema, model } from "mongoose";


const userSchema = new Schema(
  {
    firstName: String,
    lastName: String,
    userName: {
      type: String,
      required: [true, 'userName is required'],
    },
    email: { type: String,
      uniqe: [true, 'email must be uniqe'],
      required: [true, 'email is required'],
    },
    password: {
      type: String,
      required: [true, 'password is required'],
    },
    passwordChangedAt: Date,
    passwordResetExpires: Date,
    passwordResetVerified: Boolean,
    confirmEmail: {
      type: Boolean,
      default: false
    },
    age: Number,
    phone: String,
    address: String,
    gender: {
      type: String,
      default: "Male",
      enum: ["Male", "Female"]
    },
    online: {
      type: Boolean,
      default: false
```

```
    },
    blocked: {
      type: Boolean,
      default: false
    },
    isDeleted: {
      type: Boolean,
      default: false
    },
    role: {
      type: String,
      default: "free",
      enum: ["free", "vip"]
    },
    stripe_customer_id: String,
    subscriptions: [{
      type: Schema.Types.ObjectId,
      ref: 'Subscription'
    }],
    code: String,
    profilePic: {
      type: String
    },
    profilePicPublicId: {
      type: String
    }
  },
  {
    timestamps: true,
  }
);

export const userModel = model("User", userSchema);
```

**2-subscription model:**

```
import { Schema, model } from "mongoose";

const subscriptionSchema = new Schema(
  {
    subscriptionId: { type: String, required: true },
    startDate: { type: Date, required: true },
    endDate: { type: Date },
    status: { type: String, required: true },
    plan: {
      currency: { type: String, required: true },
      interval: { type: String, required: true },
      interval_count: { type: Number, required: true },
      id: { type: String, required: true }
    },
    user: { type: Schema.Types.ObjectId, ref: 'User', required: true
},
  },
  {
    timestamps: true,
  }
);

export const subscriptionModel = model("Subscription",
subscriptionSchema);
```

## 6- Password Comparison:

Password comparison is a critical aspect of authentication systems, ensuring that users provide the correct password when attempting to access their accounts. Here's a detailed breakdown of how password comparison works and best practices for implementing it securely.

**1. Hashing Passwords**

Before comparing passwords, it's essential to understand that storing plain text passwords is highly insecure. Instead, passwords should be hashed using a cryptographic hash function.

- **Hash Functions**: Convert a password into a fixed-size string of characters, which is typically a hash.
- **Salt**: A random value added to the password before hashing to ensure that even identical passwords produce different hashes.

**2. Storing Passwords**

When a user creates an account or changes their password, the application should:

1. Generate a random salt.
2. Hash the password along with the salt.
3. Store the salt and the hashed password in the database.

**3. Comparing Passwords**

1. **Retrieve Salt and Hashed Password**: Fetch the stored salt and hashed password for the given username/email from the database.
2. **Hash Input Password**: Hash the input password using the same hashing algorithm and the retrieved salt.
3. **Compare Hashes**: Compare the newly generated hash with the stored hash. If they match, the password is correct.

```
if (!user.confirmEmail) {
       res.json({ message: "Please confirm your email first" });
     } else {
       const match = await bcrypt.compare(password, user.password);
       if (!match) {
         res.json({ message: "Invalid password" });
       } else {
         const token = jwt.sign(
           { id: user._id, isLoggedIn: true },
           process.env.TOKENSIGNATURE,
           { expiresIn: '30h' }
```

## 7- Token Generation:

**1. What is a Token?**

A token is a string of data that represents a user's identity and permissions. It is typically used in place of traditional session IDs to maintain a user session in stateless protocols like HTTP. Tokens are commonly used in APIs to authenticate and authorize users.

**2. Types of Tokens**

- **Access Tokens**: Short-lived tokens used to access protected resources. They are typically included in the Authorization header of HTTP requests.
- **Refresh Tokens**: Long-lived tokens used to obtain new access tokens once the current access token expires. Refresh tokens are usually stored securely and used less frequently.
- **ID Tokens**: Contain user information and are primarily used in OpenID Connect to authenticate users.

```
const saveduser = await newUser.save();
      const token = jwt.sign(
        { id: saveduser._id },
        process.env.confirmEmailToken,
        // { expiresIn: year }
      );
      console.log(token);
      const rfToken = jwt.sign(
        { id: saveduser._id },
        process.env.confirmEmailToken
      );
```

**Refresh Token:**

```
export const requestRefToken = asyncHandler( async (req, res) => {
  try {
    const { token } = req.params;
    const decoded = jwt.verify(token, process.env.confirmEmailToken);
    if (!decoded?.id) {
      res.status(400).json({ message: "In-valid token payload" });
    } else {
      const user = await userModel.findById(decoded.id);
      if (user?.confirmEmail) {
        res.json({ message: "Already confirmed" });
      } else {
        const token = jwt.sign(
          { id: user._id },
          process.env.confirmEmailToken,
          {
            expiresIn: 60 *60*60*60* 2,
          }
        );
        const emailMessage = `<a
href='${req.protocol}://${req.headers.host}${process.env.baseURL}/auth
/confirmEmail/${token}'>Follow me to confrim Your account</a>`;
        sendEmail(user.email, "Confirm-Email", emailMessage);
        res.status(201).json({ message: "Done" });
      }
    }
  } catch (error) {
    res.status(500).json({ message: "Catch error", error });
  }
});
```

## 8-Error Handling:

Error handling is a crucial aspect of software development that ensures applications can gracefully manage and recover from unexpected situations. Proper error handling improves the reliability, usability, and maintainability of software. Here's a comprehensive guide on error handling, specifically focusing on Node.js applications.

**1. Types of Errors**

**Syntactic Errors**: Mistakes in the code that prevent it from being parsed, such as missing semicolons or incorrect use of language syntax.

**Runtime Errors**: Occur during the execution of the program, such as trying to read an undefined variable or a failed network request.

**Logical Errors**: Errors in the logic that lead to incorrect behavior, such as off-by-one errors or incorrect algorithm implementations**.**

**Code :**

```
export const asyncHandler = (fn) => {
    return (req, res, next) => {
        fn(req, res, next).catch(err => {
            return next(new Error(err))
        })
    }
}
export const globalErrorHandling = (err, req, res, next) => {
    if (err) {
        if (process.env.MOOD == "DEV") {
            return res.status(err.cause || 500).json({ message:
err.message, err, stack: err.stack })
        }
        return res.status(err.cause || 500).json({ message:
err.message })
    }
}
```

## 9-Email/SMS Notification:

We used in our project to send:

**email**: we used nodeoutlook and it nodeoutlook is a Node.js module that simplifies sending emails using Microsoft Outlook, Office 365, or Exchange servers. It's especially useful for Node.js applications that

need to send emails via Outlook's SMTP service. Here’s an overview of how to use nodeoutlook for sending emails.

**Code:**

```
import nodeoutlook from "nodejs-nodemailer-outlook";

export function sendEmail(dest, subject, message) {
  nodeoutlook.sendEmail({
    auth: {
      user: process.env.SenderEmail,
      pass: process.env.SenderPassword,
    },
    from: process.env.SenderEmail,
    to: dest,
    subject,
    html: message,

    onError: (e) => console.log(e),
    onSuccess: (i) => console.log(i),
  });
}
```

**SMS:**

We used Infobip and it is a global communications platform that provides a variety of services for businesses to connect with their customers via SMS, email, voice, and chat applications. Infobip's platform is designed to help businesses automate and manage their communication channels efficiently

## 10-Response to Client:

refers to how a server or application sends feedback or information back to a client after processing a request. This is a fundamental aspect of client-server communication, especially in web development, APIs, and any networked application. Here's a detailed overview of handling responses to clients, focusing on HTTP responses in the context of Node.js applications:

**1. Understanding HTTP Responses**

An HTTP response typically consists of three main parts:

Status Line: Contains the HTTP version, status code, and status message.

Headers: Provide metadata about the response, such as content type, content length, server information, etc.

Body: The actual content or data being sent back to the client (e.g., HTML, JSON, XML).

**2. Status Codes**

HTTP status codes indicate the result of the client's request:

1xx (Informational): Request received, continuing process.

2xx (Successful): The request was successfully received, understood, and accepted.

200 OK: The request was successful.

201 Created: The request was successful and a resource was created.

204 No Content: The request was successful but there's no content to send in the response.

3xx (Redirection): Further action needs to be taken by the user agent to fulfill the request.

301 Moved Permanently: The resource has been moved to a new URL.

302 Found: The resource is temporarily available at a different URL.

4xx (Client Error): The request contains bad syntax or cannot be fulfilled.

400 Bad Request: The server could not understand the request due to invalid syntax.

401 Unauthorized: The client must authenticate itself to get the requested response.

403 Forbidden: The client does not have access rights to the content.

404 Not Found: The server cannot find the requested resource.

5xx (Server Error): The server failed to fulfill a valid request.

500 Internal Server Error: The server encountered an unexpected condition.

**11-Subscription Handling Overview**

1. **Subscription Status**: Check the status of a user's subscription and update user roles.
2. **Check and Notify Expired Subscriptions**: Daily job to check for expired subscriptions and notify users.
3. **Create Subscription**: Create a new subscription for a user.
4. **Update Subscription**: Update an existing subscription to a new plan or price.

**Create Subscription code :**

```
export const createSubscription = asyncHandler(async (req, res) => {
    try {
        const user = await userModel.findById(req.user.id);

        if (!user) {
            return res.status(404).json({ message: "User not found"
});
        }

        const session = await stripe.checkout.sessions.create({
            mode: "subscription",
            payment_method_types: ["card"],
            line_items: [
                {
                    price: req.body.priceId,
                    quantity: 1,
                },
            ],
            customer: user.stripe_customer_id,
            success_url: process.env.STRIPE_SUCCESS_URL,
            cancel_url: process.env.STRIPE_CANCEL_URL,
        });

        if (!session) {
            return res.status(400).json({ message: "Unable to create
subscription session" });
        }

        const paymentIntent = await stripe.paymentIntents.create({
            amount: session.amount_total,
            currency: session.currency,
            customer: user.stripe_customer_id,
```

```
            setup_future_usage: 'off_session', // Helps in future off-
session payments
            metadata: { integration_check: 'subscription_payment' }
        });

        res.json({ clientSecret: paymentIntent.client_secret });
    } catch (err) {
        console.error(err);
        res.status(500).json({ message: "Internal Server Error" });
    }
});
```

**2- Subscription Status code** :

```
export const subscriptionStatus = asyncHandler(async (req, res) => {
    try {
        const user = await userModel.findById(req.user.id);
```

```
        console.log(req.user.id);
        if (!user) {
            return res.status(404).json({ message: "User not found"
});
        }
        const subscriptions = await stripe.subscriptions.list({
        customer: user.stripe_customer_id,
        status: "all",
        expand: ["data.default_payment_method"],
    });
    if (!subscriptions.data.length) {
        return res.status(404).json({ message: "No subscriptions
found" });
    }
      // Assuming there's only one subscription
    const subscription = subscriptions.data[0];
    console.log(subscription);
    if (subscription && subscription.status === "active") {
        // Update user role to "VIP"
        user.role = "vip";
    }
      // Save subscriptions data to the Subscription model
    const savedSubscriptions = [];
    for (const sub of subscriptions.data) {
    const existingSub = await subscriptionModel.findOne({
subscriptionId: sub.id });
            const subscriptionData = {
            subscriptionId: sub.id,
            startDate: sub.start_date ? new Date(sub.start_date *
1000) : null,
            endDate: sub.current_period_end ? new
Date(sub.current_period_end * 1000) : null,
            status: sub.status,
            plan: {
                currency: sub.plan.currency,
                interval: sub.plan.interval,
                interval_count: sub.plan.interval_count,
                id: sub.plan.id,
            },
```

```
            user: user._id,
            };
        if (existingSub) {
          // Update existing subscription
            Object.assign(existingSub, subscriptionData);
            await existingSub.save();
            savedSubscriptions.push(existingSub._id);
            } else {
            // Create new subscription
            const newSubscription = new
subscriptionModel(subscriptionData);
            await newSubscription.save();
            savedSubscriptions.push(newSubscription._id);
            }
        }
        // Update user document with subscription references
        user.subscriptions = savedSubscriptions;
        await user.save();
        res.status(200).json({ message: 'User is VIP', user });
    } catch (err) {
    console.error(err);
    res.status(500).json({ message: "Internal server error" });
        }
    });
```

**3- Update Subscription code** :

```
export const updateSubscription = asyncHandler(async (req, res) => {
    const { newPriceId } = req.body;
    const user = await userModel.findById(req.user._id);
```

```
    if (!user) {
    return res.status(400).json({ message: "User not found" });
    }

    const subscriptions = await stripe.subscriptions.list({
    customer: user.stripe_customer_id,
    status: "all",
    expand: ["data.default_payment_method"],
    });

    if (!subscriptions.data.length) {
    return res.status(404).json({ message: "No subscriptions found"
});
    }

    const subscription = subscriptions.data[0];

    // Update the subscription plan with proration
    const updatedSubscription = await
stripe.subscriptions.update(subscription.id, {
    cancel_at_period_end: false,
    proration_behavior: 'create_prorations',
    items: [{
        id: subscription.items.data[0].id,
        price: newPriceId,
    }]
    });

    res.status(200).json({ message: "Subscription updated
successfully", subscription: updatedSubscription });
}
)
```

**4- Check and Notify Expired Subscriptions**:

```
const notifyUserByEmail = async (user, subscription) => {
        const message = `Dear ${user.userName}, your subscription with
ID ${subscription.subscriptionId} has expired.`;
```

```
        try {
        sendEmail(user.email, 'Subscription Expired', message);
        console.log(`Email sent to ${user.email}`);
        } catch (error) {
        console.error(`Failed to send email to ${user.email}:
${error.message}`);
        }
    };

    const checkAndNotifyExpiredSubscriptions = async () => {
        try {
        const now = new Date();
        const expiredSubscriptions = await subscriptionModel.find({
endDate: { $lte: now }, status: 'active' }).populate('user');

        for (const subscription of expiredSubscriptions) {
            const user = subscription.user;

            // Send notification to user
            await notifyUserByEmail(user, subscription);
            // Update subscription status to expired
            subscription.status = 'expired';
            await subscription.save();
        }
        } catch (error) {
        console.error(`Failed to check and notify expired
subscriptions: ${error.message}`);
        }
    };
    cron.schedule('0 0 * * *', () => {
        console.log('Running daily subscription expiry check...');
        checkAndNotifyExpiredSubscriptions();
    });
```

# Chapter 10: Designing Website for displaying the project:

## -Introduction:

As part of our graduation project, we designed and developed the website to showcase our project. The primary goal was to create an engaging, informative, and user-friendly platform to present our work effectively. This section details the design process, the technologies used, and the features implemented on the website.

### -Technologies Used to build website:

to create a robust and dynamic website, I utilized a combination of several web technologies:

1. **HTML**: The backbone of our website, HTML was used to structure the content, ensuring a clear and organized layout.
2. **CSS**: For styling the website, CSS was employed to enhance the visual appeal and ensure a responsive design across different devices.
3. **JavaScript**: JavaScript was used to add interactivity and dynamic behavior to the website, making it more engaging for users.
4. **Bootstrap**: To streamline the design process and ensure a consistent look and feel, Bootstrap was used. This framework facilitated the development of a responsive and mobile-first website.
5. **Laravel**: Laravel, a PHP framework, was used for routing and to handle various backend functionalities. It helped in creating a seamless navigation experience and managing the content effectively.

## - Website Structure and Features:

The website consists of several key sections, each designed to provide specific information about our project. Below is an overview of the main components and their functionalities:

### -Home Page:

The home page serves as the entry point to our website and includes the following elements:

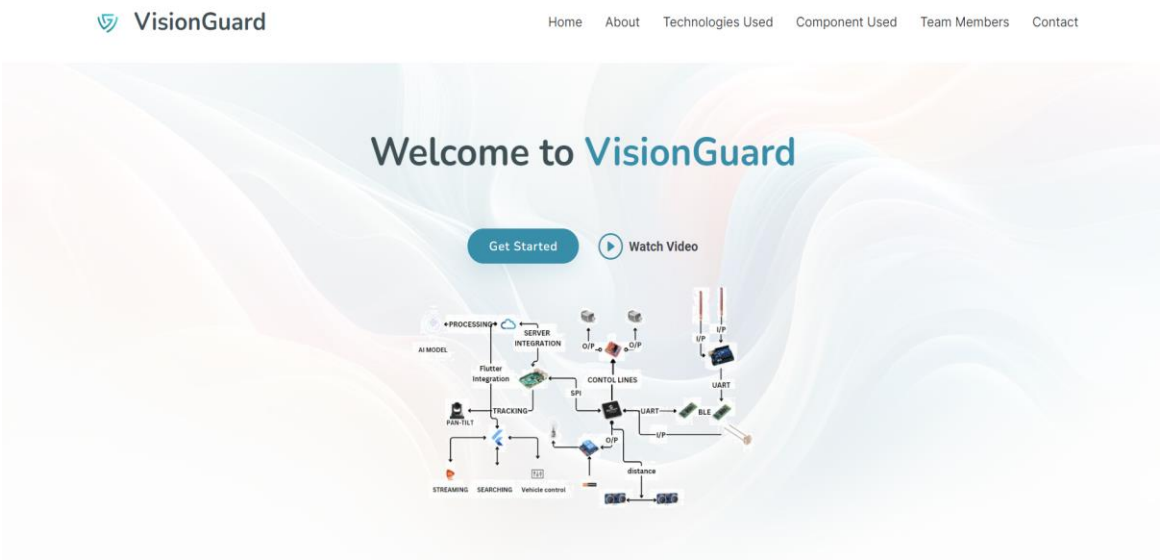
- **Navbar**: The navigation bar at the top of the page includes links to different sections: Home, About, Technologies Used, Components Used, Team Members, and Contact. Each link directs the user to the respective section within the homepage. The Navbar is designed using Bootstrap classes, ensuring it is fully responsive and collapses into a hamburger menu on smaller screens.
- **Welcome Message**: A welcoming message (h2) is centered on the page to greet visitors. This message sets the tone for the website and provides a brief introduction to what the visitors can expect. The welcome message is styled with CSS to ensure it stands out and grabs the visitor's attention immediately.

- **Buttons**: Below the welcome message are two buttons:

1- **Watch Video**: This button plays a YouTube video about our project directly on the same page. Integrating the video using an embedded iframe tag ensures that visitors do not have to leave the website to watch the video.
2- **Get Started**: This button takes the user to the About section. This button scrolls the page smoothly to the desired section, enhancing the user experience.

3- Following these buttons, a schematic diagram related to the project is displayed, providing a visual representation of the project's components and their interconnections. This diagram is designed to give visitors a quick overview of the technical aspects of the project.

*Figure 129*




Features Section: Below the buttons, three key features of our project are highlighted to give visitors a quick overview. These features are displayed using Bootstrap cards, which provide a clean and organized layout.

About Section:

The About section offers a concise overview of the project, including its objectives and significance and photos to the project. This section is crafted to give visitors a quick yet comprehensive understanding of what our project is about and why it matters. The section concludes with a "Read More" link that redirects to a detailed page (using Laravel routing), where visitors can find an in-depth discussion of the project. This page, located at the /about route, dives deeper into the project's background, goals, development process, and outcomes. By offering both a brief and a detailed view, we cater to different types of visitors—those looking for a quick summary and those seeking comprehensive details.

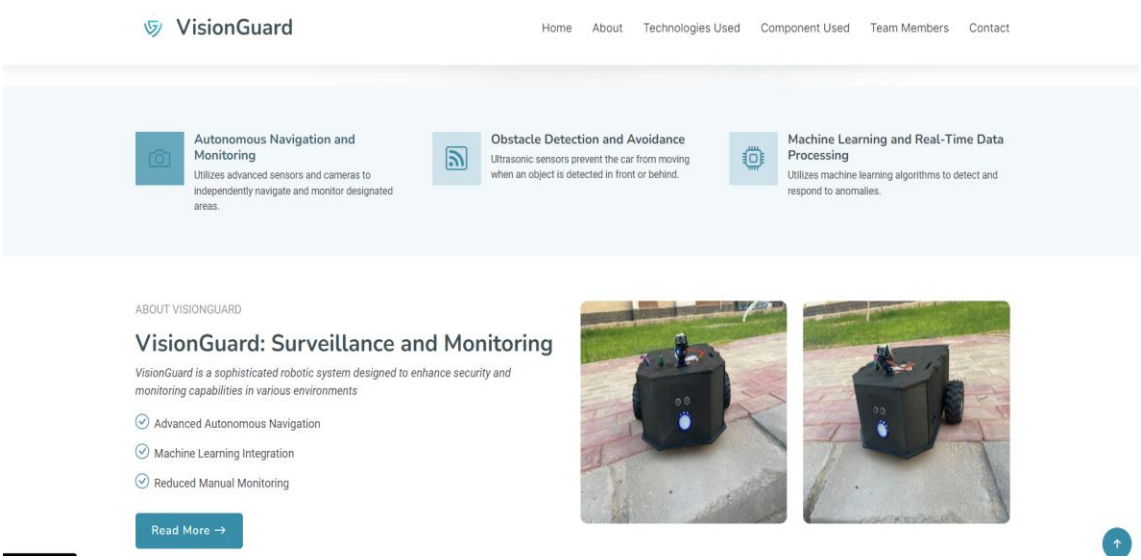



*Figure 130*

## Technologies Used:

This section lists and describes the technologies used in the development of our project. Each technology is briefly explained, highlighting its role and importance.

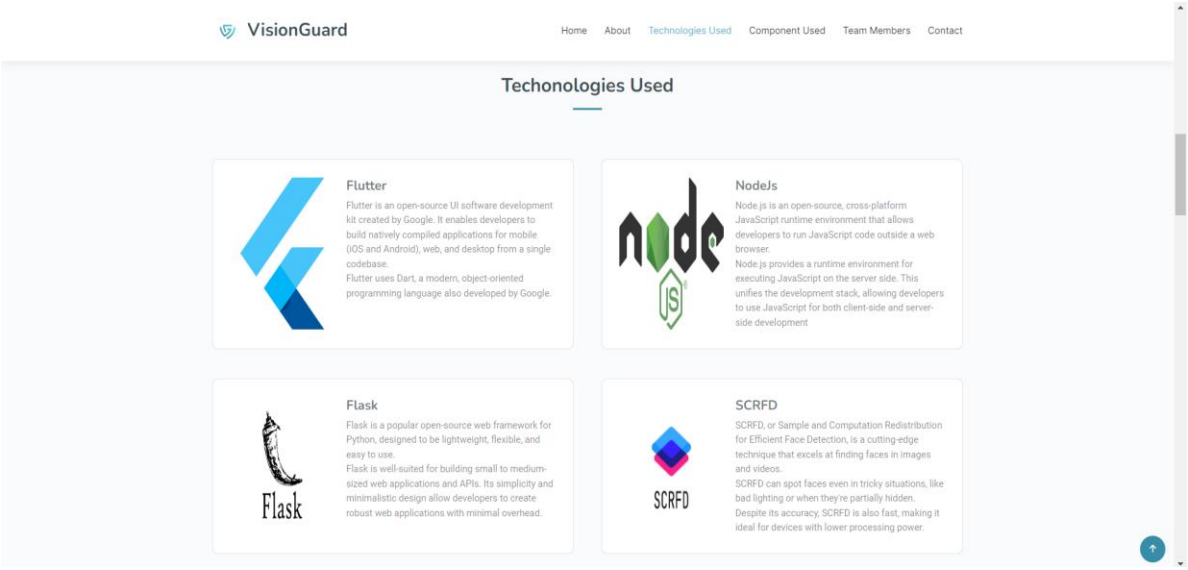



*Figure 131*

### Component Used Section:

Here, the various components used in the project are displayed, Here, the various components used in the project are displayed. Each component is described in detail. This section helps visitors appreciate the complexity and technical depth of the project.

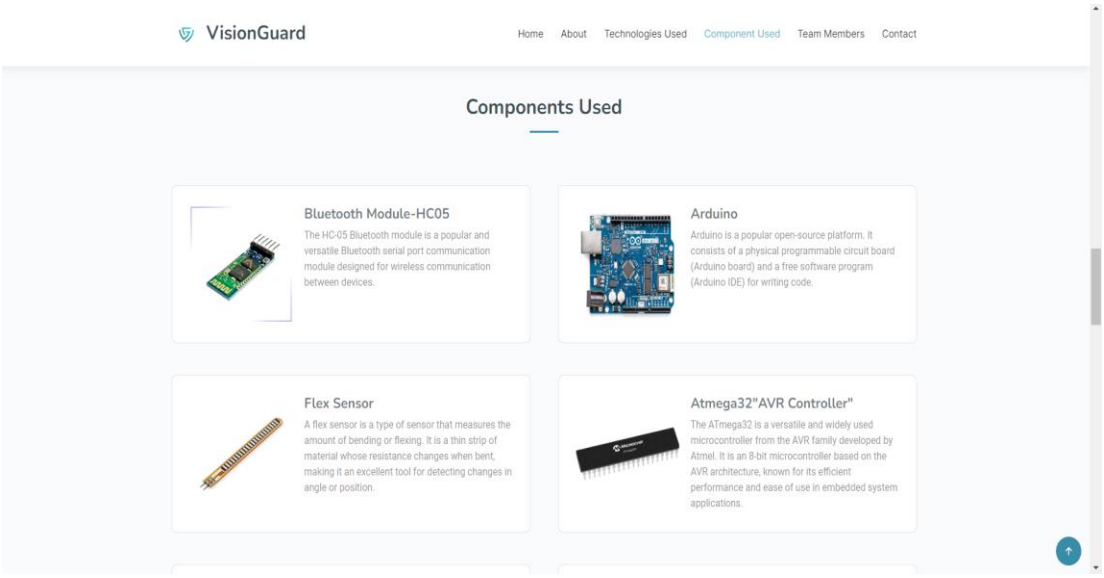



*Figure 132*

## Team Members Section:

This section introduces the team members involved in the project. Each member's role and contributions are highlighted, providing a personal touch to the project presentation.

By showcasing the people behind the project, we aim to add a human element and give credit to the hard work and dedication of each team member. The section includes:

1- **Profiles:** Brief profiles of each team member, including their role in the project and their specific contributions.

2- **CV:** External link to the resume of each team member

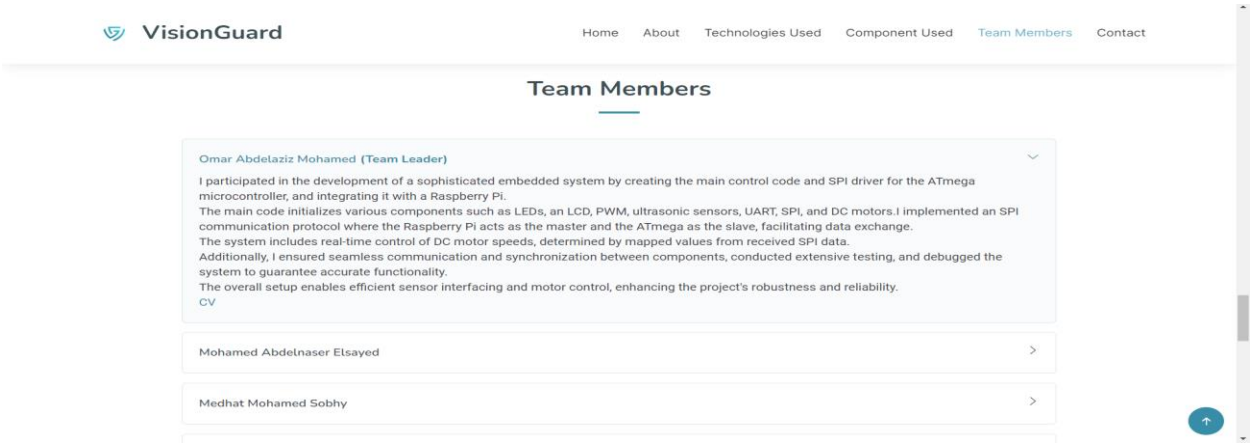



*Figure 133*

## Contact Section:

The contact section provides information on how to reach the team. This includes email address and phone number. By providing multiple ways to get in touch, we aim to make it easy for visitors to connect with us. The section is designed to be straightforward and accessible, ensuring that visitors can find the information they need without any hassle.

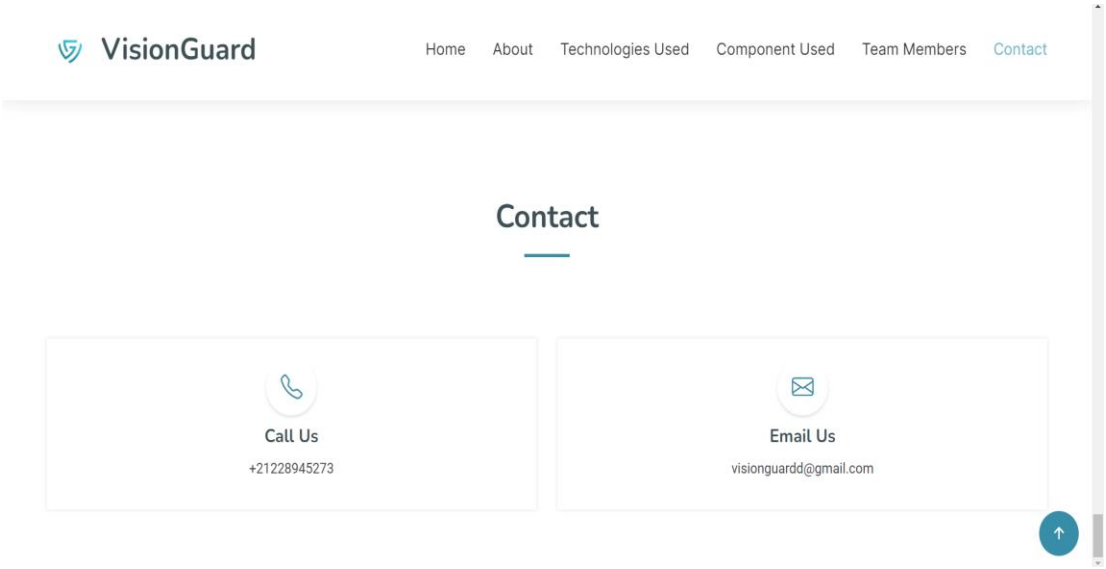



*Figure 134*

**Conclusion**:

The website serves as a comprehensive showcase of our graduation project. By utilizing modern web development technologies and frameworks, we created an informative and visually appealing platform. This project not only demonstrates our technical skills but also our ability to collaborate and present our work effectively. The process of designing and developing the website was a valuable learning experience, allowing us to apply our knowledge in a practical setting and produce a professional-quality site that effectively communicates our project to the world.

# Conclusion

In conclusion, the "Vision Guard: Monitoring and Surveillance" project demonstrates the potential for innovation in the field of security and surveillance. By integrating multiple control methods and incorporating artificial intelligence, this project offers a versatile and efficient solution for monitoring and tracking activities in various environments.

The robot's ability to be controlled via a glove, a mobile app, and its autonomous functionality showcases the flexibility and adaptability of the system. The real-time video streaming and user recognition features significantly enhance the robot's effectiveness, making it a valuable tool for ensuring safety and security.

This project has successfully addressed several challenges, providing a reliable and advanced surveillance solution. The practical applications of this technology are vast, including home security, industrial monitoring, and public safety, all contributing to safer environments.

Overall, the "Vision Guard: Monitoring and Surveillance " project not only achieves its intended goals but also opens up new possibilities for future developments in automated surveillance systems. It highlights the importance of combining innovative ideas with practical solutions to create impactful advancements in technology. This project serves as a testament to the potential of engineering to solve real-world problems and improve our daily lives.