كلية الهندسة بحلوان
قسم هندسة الحاسبات والنظم

جامعة حلوان
HELWAN UNIVERSITY

Computer and Systems Engineering Department
Faculty Of Engineering
Helwan University

Smart Drowning Detection System

SWIDRO

BY

Taqi Eldeen Moanes Mohamed
Ahmed Ayman Samir
Ahmed Othman Ali
Michael Samy Henein

Supervised by: Dr. Rasha F. Aly

Contents

List of figures

Acknowledgement

All praises to Allah and His blessing for the completion of this project. We thank God for all the opportunities, trials and strength that have been showered on us to finish this project.

First and foremost, we would like to sincerely thank our supervisor **Dr. Rasha F. Aly** for her guidance, understanding, patience and most importantly, she has provided positive encouragement and a warm spirit to finish this project. It has been a great pleasure and honor to have her as our supervisor.

We are also grateful to all the Teaching staff in the Computer Engineering Department, for their consistent support and assistance. It was great sharing premises with all of you during the last five years.

Abstract

Drowning is considered a leading cause for death in the world, specially in summer where people love to swim in beaches and pools. Therefore, there is a need for a system that will help reduce the risk of drowning and getting injured by long term damage.

SWIDRO is a smart drowning detection system that focuses on early detection of drowning and alerting the lifeguards to take immediate actions in saving the victim early. The system will provide an insight into the swimmer’s status using an AI approach which will make use of heart rate, SPO2, and movement pattern in determining the status of the swimmer. The status is displayed in a user-friendly web application that gives insight on the pools and current lifeguards assigned to the pools and the status of the swimmers in the pools.

The system consists of a wrist wearable band that will collect the data needed by the AI and send it to the gateway that will send the data to our server using MQTT protocol and finally the status of the swimmer will be displayed using a user-friendly web application that can display different pools and lifeguards assigned to their pools and alerting the lifeguards.

1 Introduction

1.1 Problem statement:

Drowning is considered amongst the top 10 causes of unintentional death, according to the World Health Organization (WHO). Drowning tends to be silent, and victims rarely move in a convulsive manner. Instead, they expend significant energy trying to keep their heads above water and may be unable to call or signal for help.

Lifeguards need to be focused on emergency situations, also need to be able to make quick and decisive decisions and must be able to stay alert and attentive for long periods of time. So, by human nature we tend to make mistakes and distracted, by using an additional system that will increase the performance and efficiency of lifeguards we will reduce the risk of drowning incidents.

By using a drowning detection system, we will make pools and beaches a safer place to swim in. Increasing the lifeguards' performance by using this system will also have a great impact on the safety of hotels and beach resorts.

The proposed system comprises three main components, embedded device (wearable device), web application, artificial intelligence. The embedded device will be used to collect data from different sensors (Oximeter, heart rate, movement measurement). The web application will be used to monitor the swimmers and alert if there is a health issue or pre-drowning detection. Machine learning will be used to predict the users' health to get the related action needed for the situation.

1.2 Objectives

Our idea came from the fact that the victims cannot call for help most of the time. So, we came up with a smart drowning detection wearable device that will help reduce the risk of drowning by informing the lifeguards about a drowning situation.

The project objective is to early detection of drowning situations for normal swimmers in swimming pools or beach resorts to reduce the risk of drowning among people by informing the lifeguards early through a GUI application by firing an alarm indicating someone is in a drowning situation.

The system should be able to accurately detect the drowning situation by using different sensors that are related to his health situation. By using the heart rate, we can detect how normal his heart beats, by using Spo2 measurements we can detect how well his respiration is, by using the accelerometer we can detect his movements.

2 Literature review

We found some good examples on our idea that we can build on it and take it as a reference for our project including:

2.1 Wearable Pulse Oximeter for Swimming Pool Safety [1]

Research stated that they came up with an algorithm for wearable devices to detect and prevent people from drowning by detection of pre-drowning symptoms and alerting the lifeguards immediately.

2.2 An Early Drowning Detection System for Internet of Things (IoT) Applications [3]

Research provides a system that can detect early stages of drowning by using a headband and IOT applications including mobile and web application.

2.3 A Smart Multi-Sensor Device to Detect Distress in Swimmers [4]

This research provides a system that has different sensors for detection of distress in swimmers by using different sensors and a threshold value to inform about drowning situations.

2.4 SEAL SwimSafe [5]

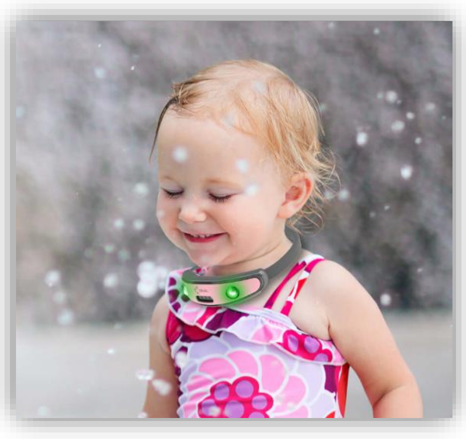


Figure 1 SEAL SwimSafe product

A product provides a neck band for children and a hub that receives the signals to alert parents about drowning of their children.

2.5 <u>MYLO [6]</u>

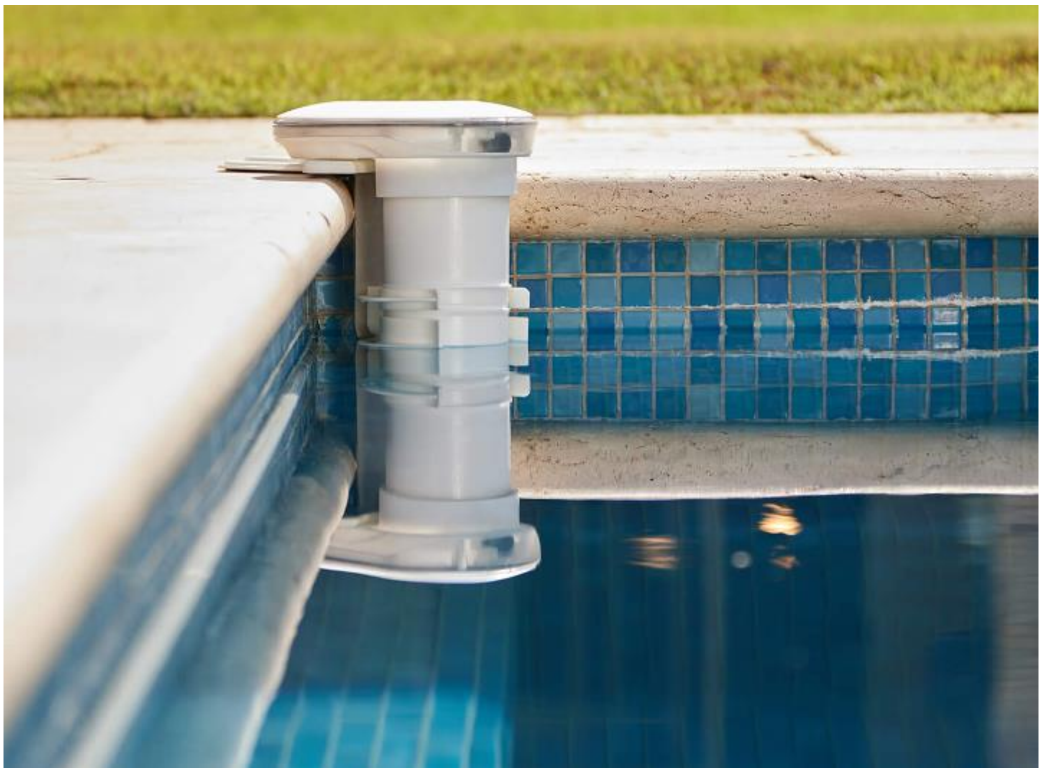


Figure 2 MYLO product in pool

MYLO is the world’s first A.I. Lifeguard System that actively monitors your pool 24/7. Smart technology allows MYLO to detect drowning risks and identify drowning scenarios to send a series of escalating alarms via mobile notifications and home alarm system.
It uses computer vision to detect drowning situations in pools.

2.6 Comparison

	1	2	3	4	5
Used parameters	-Heartbeats -SPO2 -Acceleration	-Heartbeats -Duration of submersion	-Heartbeats -SPO2 -Water depth -Duration of submersion	-Duration of submersion	- Computer vision
Wear location	Wrist band	Head band	wrist band	necklace	Fixed pool location
Cost	undefined	undefined	undefined	449 $	1500 $

3 Analysis and Requirements

3.1 System requirements

a) Enumerated Functional Requirements

REQ-x	Description
REQ-1	System should detect pre-drowning situations and alert the lifeguards
REQ-2	System should add new staff (Lifeguards) information to the DB
REQ-3	Assign a SWIDRO device to a swimmer
REQ-4	System should add swimmer information to the DB
REQ-5	System should monitor users' health status

b) Enumerated Nonfunctional Requirements

REQ-6	SWIDRO device should be able to communicate a long-range using a low RF signal
REQ-7	SWIDRO should have gateway to connect all devices
REQ-8	Device should have a relatively long battery life
REQ-9	Device should connect to the local network using a light weight MQTT protocol
REQ-10	Device should be small and compact

REQ-11	System should have low latency
REQ-12	System will monitor users' health real-time using heart rate, SPO2 and acceleration

3.2 Functional requirements specification

a) Stake holders

- Hotels
- Beach resorts
- Parents
- Public pools managers

b) Actors and goal

- ***Swimmers***

They are direct users with the SWIDRO device.

- ***Staff / Lifeguards***

They are involved in using the system to monitor the swimmers' status.

- ***Admins***

They are responsible for adding new lifeguards and new users.

- ***Parents***

The system could be commercially available.

c) Use cases

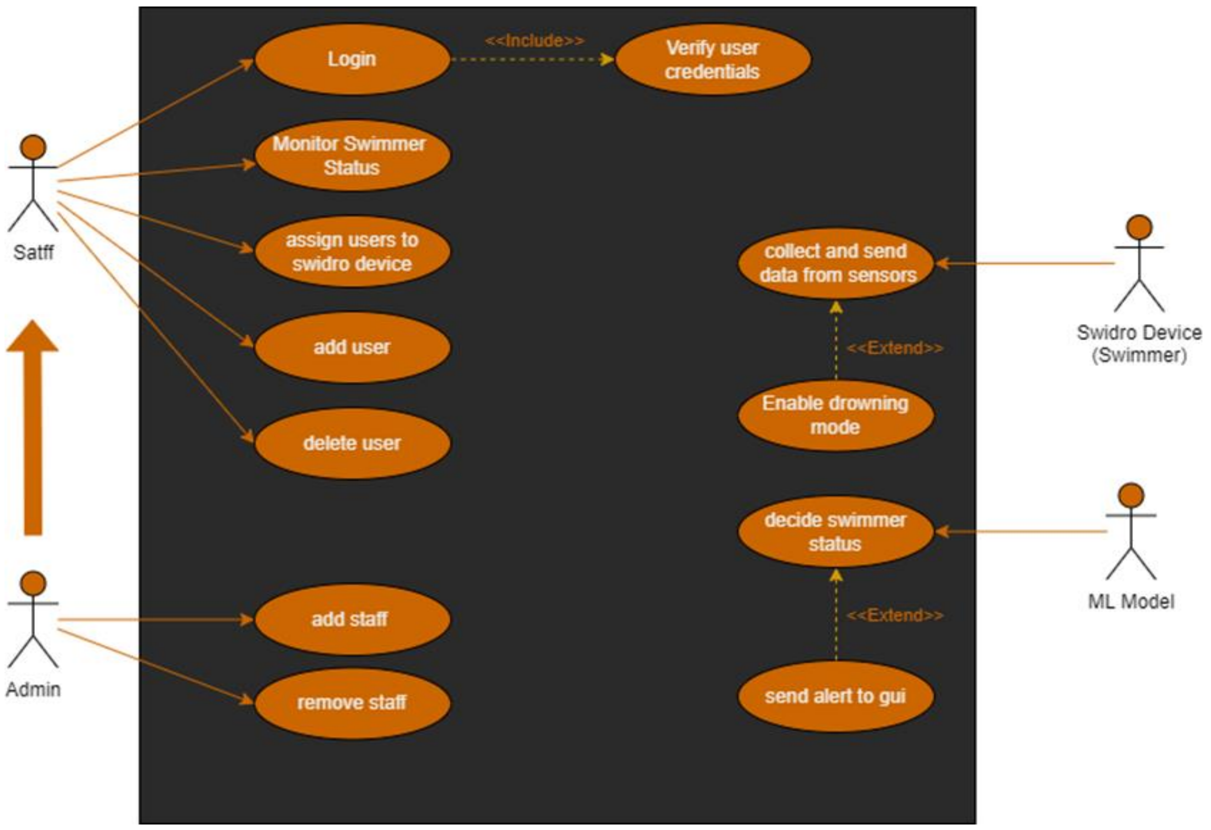


Figure 3 System use case

- **Login**: Staff / Lifeguards and Admins should be able to login to the system using a valid user credentials.
- **Monitor swimmer status**: Staff / Lifeguards and admin should be able to monitor swimmers’ status
- **Add user**: Staff / Lifeguards and admin should be able to add users to the system
- **Delete user**: Staff / Lifeguards and admin should be able to delete users
- **Add staff**: Admin should be able to add lifeguards to the system
- **Remove staff**: Admin should be able to remove lifeguards
- **Collect and send data from sensors**: Device should be able to send and collect data from sensors to the application.
- **Decide swimmer status**: machine learning model should be able to decide the swimmers’ status accurately.

4 Embedded System

4.1 Embedded software architecture

Software Architecture Definition

The software architecture is a structuring way used to define software elements and relationships between them. In Embedded Systems we use a major type of software architecture called ***Layered Architecture***.

In the layered architecture the software is divided into small parts called software components. Software components related to each other are organized in a horizontal layer. Each layer is performing a specific role.

4.1.1 Static architecture

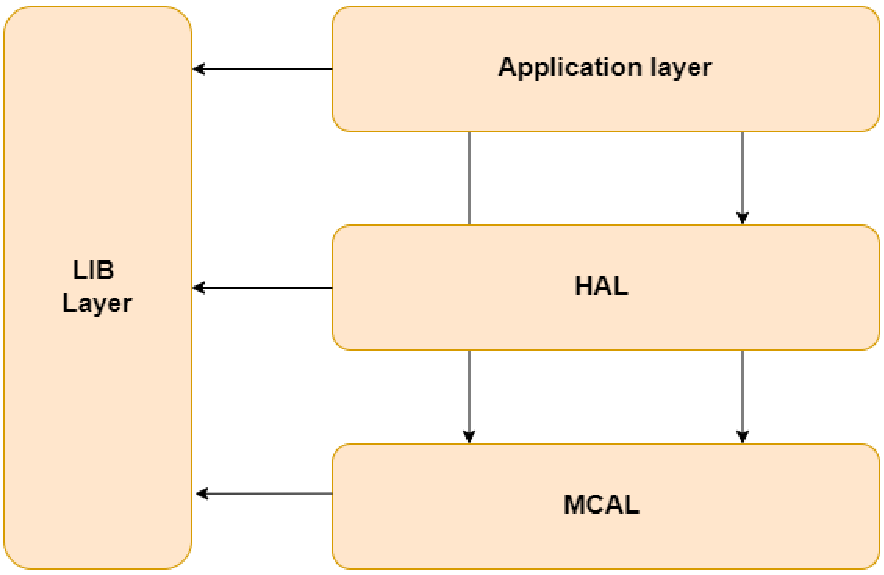


Figure 4 Layered view for embedded systems

As seen in figure 4 we separate our code into layers where each layer will contain its related software modules. For example, in the LIB layer we add the needed libraries for our project and for the MCAL “Microcontroller abstraction layer” we add all modules related to microcontroller peripherals such as DIO, TIMER, etc… as seen in figure 5.

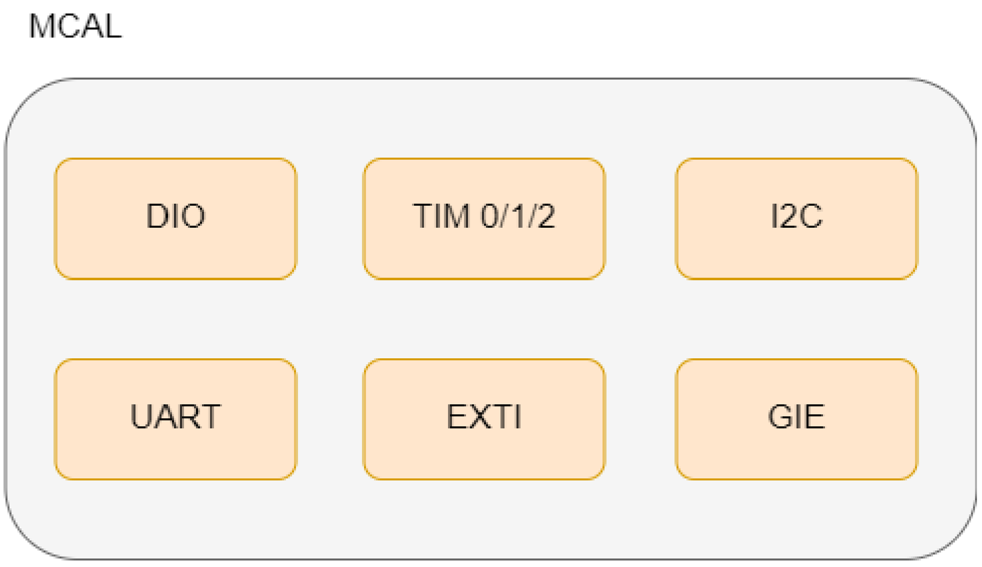


Figure 5 MCAL modules

While in HAL “Hardware abstraction layer” we add all the necessary modules for the sensors used and all HW outside the microcontroller as seen in figure 6.

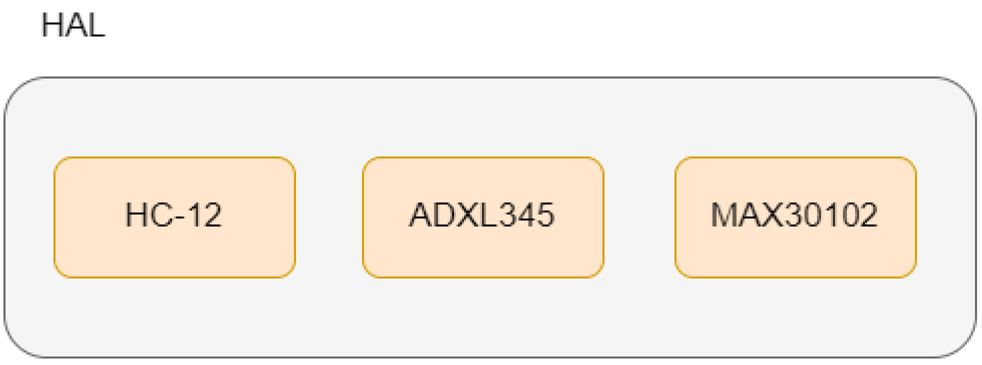


Figure 6 HAL modules

4.1.2 Dynamic architecture

Super loop

In an Embedded C Application, there are set of statements which need to be executed forever. Because there are no operating system to return to or an embedded device is running until the power supply is removed. So, to run set of statements, we need a loop that must not be finished, such kind of loops are known as 'Super Loop' or 'Infinite Loop'.

There is only one difference between 'Super Loop' and 'Infinite Loop': There may only one 'Super Loop' but the 'Infinite Loop' may be infinite (i.e. there is no limit of infinite loops in a program).

Interrupt

Interrupt is an event the disturbs the normal execution sequence of the code and makes the processor to jump to another piece of code to execute called Interrupt Service Routine (ISR). After the processor finish executing of the ISR, it returns back to the interrupted code to continue from the interrupted instruction.

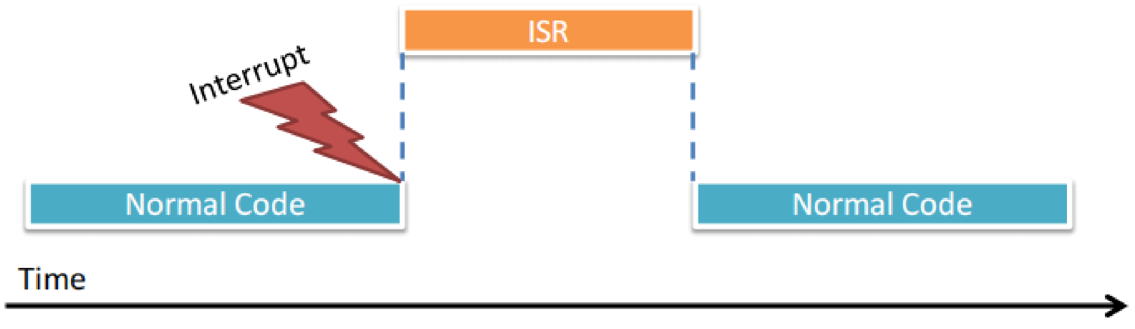


Figure 7 Interrupt

Foreground-Background Architecture

As the name suggests, the architecture consists of two main parts: the endless, While(1){…} background loop inside the main() function and the interrupt service routines (ISRs) comprising the foreground. The ISRs running in the foreground preempt the background loop, but they always return to the point of preemption. The two parts of the system communicate through shared variables To avoid race conditions due to asynchronous ISRs, these shared variables must be protected by briefly disabling interrupts around any access to them from the background.

4.2 Microcontrollers

4.2.1 STM32F103

Introduction

The STM32F103 is a microcontroller from the STM32 family of ARM Cortex-M based microcontrollers, manufactured by STMicroelectronics. It is part of the STM32F1 series, which is based on the ARM Cortex-M3 core. The STM32F103 microcontroller is widely used in various embedded systems and applications due to its features, performance, and cost-effectiveness.

Characteristics of the STM32F103 microcontroller

1. ARM Cortex-M3 Core: The STM32F103 is built around the ARM Cortex-M3 32-bit RISC processor core, which offers a good balance between performance, power efficiency, and code density. The Cortex-M3 core operates at a maximum frequency of up to 72 MHz.
2. Flash Memory: The microcontroller typically comes with various Flash memory options, ranging from 16 KB up to 512 KB. This Flash memory is used for storing program code, firmware, and non-volatile data.
3. RAM: The STM32F103 includes on-chip RAM for storing data and

variables during program execution. RAM sizes vary depending on the specific microcontroller variant, typically ranging from 16 KB to 64 KB.

4. GPIO: The microcontroller provides a set of general-purpose input/output (GPIO) pins that can be configured as digital inputs or outputs. These pins are used for interfacing with external devices, such as sensors, actuators, or other peripherals.
5. Peripherals: The STM32F103 microcontroller includes a range of on-chip peripherals, such as UART (USART), SPI (including a dedicated I2S interface for audio applications), I2C, USB, ADC (analog-to-digital converter), timers, PWM (pulse-width modulation), and more. These peripherals enable communication with other devices, data acquisition, signal generation, and control functionalities.
6. Clock and Reset Systems: The microcontroller provides a flexible clock system, including an internal high-speed oscillator, multiple internal low-power oscillators, and the ability to use an external clock source. It also includes reset and power management circuitry for reliable system startup and operation.
7. Interrupt and DMA Controller: The STM32F103 microcontroller features an interrupt controller that manages and prioritizes interrupts from various sources, allowing for efficient event handling and real-time responsiveness. It also includes a DMA (Direct Memory Access) controller, which offloads data transfer tasks from the CPU, freeing up its processing power.

4.2.2 ATmega32

Introduction

The ATmega32 is a microcontroller from the Atmel AVR family, which is a popular series of 8-bit microcontrollers known for their low power consumption, versatility, and ease of use. The ATmega32 is widely used in various embedded systems and applications due to its features, performance, and affordability.

Characteristics of the ATmega32 microcontroller:

1. 8-bit AVR Core: The ATmega32 is built around an 8-bit AVR RISC processor core. This architecture offers a good balance between performance and power efficiency, making it suitable for a wide range of applications.
2. Flash Memory: The microcontroller typically comes with 32 KB of in-system programmable Flash memory. This memory is used for storing the program code, firmware, and non-volatile data. The Flash memory can be easily reprogrammed even after the microcontroller is soldered onto a PCB.
3. SRAM: The ATmega32 includes 2 KB of on-chip SRAM (Static Random Access Memory). SRAM is used for storing data and variables during program execution, and it provides fast access for efficient data processing.
4. EEPROM: The microcontroller also features 1 KB of on-chip EEPROM (Electrically Erasable Programmable Read-Only Memory). EEPROM is non-volatile memory that can be electrically erased and reprogrammed, allowing for storage of persistent data that needs to be retained even when power is removed.
5. GPIO: The ATmega32 provides a set of general-purpose input/output (GPIO) pins. These pins can be individually configured as digital inputs or outputs, enabling interfacing with various external devices, such as

sensors, switches, LEDs, or other peripherals.

6. Peripherals: The microcontroller includes a range of on-chip peripherals, including UART (USART), SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), timers/counters, PWM (Pulse Width Modulation) outputs, analog-to-digital converters (ADC), and more. These peripherals provide communication capabilities, timing functions, analog signal acquisition, and other essential features for many embedded applications.
7. Interrupt and Timer/Counter System: The ATmega32 features an interrupt controller that manages and prioritizes interrupts from various sources. It also includes multiple 8-bit and 16-bit timer/counter units that can be used for timing, event counting, and waveform generation.
8. Clock and Reset Systems: The microcontroller offers a flexible clock system with internal oscillators, such as an 8 MHz RC oscillator and a 32 kHz crystal oscillator, as well as the option to use an external clock source. It also includes reset and power management circuitry for reliable system startup and operation.

4.2.3 ATmega328p

Introduction

The ATmega328P is a microcontroller from the AVR family, manufactured by Microchip Technology (formerly Atmel Corporation). It is an 8-bit microcontroller that belongs to the ATmega series and is widely used in various embedded systems and Arduino-based projects.

Characteristics of the ATmega328p microcontroller:

1. Architecture: The ATmega328P is based on the AVR 8-bit RISC (Reduced Instruction Set Computing) architecture. It features a Harvard architecture with separate program and data memories, which allows for simultaneous instruction fetching and data access.
2. Flash Memory: The microcontroller includes on-chip Flash memory for storing program code. The ATmega328P typically comes with 32 KB of Flash memory, which is non-volatile and can be electrically erased and reprogrammed.
3. RAM: The microcontroller has on-chip RAM for storing data and variables during program execution. The ATmega328P usually has 2 KB of SRAM (Static Random Access Memory), which is volatile and loses its contents when power is removed.
4. GPIO: The ATmega328P provides a set of general-purpose input/output (GPIO) pins that can be individually configured as digital inputs or outputs. These pins can be used for interfacing with external devices, such as sensors, actuators, or other peripherals.
5. Peripherals: The microcontroller offers various on-chip peripherals, including UART (USART), SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), timers/counters, PWM (Pulse Width Modulation) channels, analog-to-digital converter (ADC), and more. These peripherals enable communication with other devices, precise timing control, analog signal acquisition, and other functionalities.
6. Clock and Reset Systems: The ATmega328P features an on-chip oscillator and can be operated with an external crystal or resonator for accurate timing. It also includes reset and power management circuitry for reliable

system startup and operation.

7. Interrupt Controller: The microcontroller includes an interrupt controller that manages and prioritizes interrupts from various sources, allowing for efficient event handling and real-time responsiveness.

4.3 Communication protocols

Communication in Embedded systems

- Exchanging data between different subsystems within the same system.
- Reduce the complexity of a system by splitting it into different subsystems.
- transfer the data on different distances and on different mediums.

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 two each other without any constraints.

- A protocol is a defined method of communication by defining two main aspects:

- Hardware Interface

This activity defines the hardware connections (wires) between the communicating nodes (ECUs)

- Data Frame Format

This activity defines the data frame transmitted of the wires between the nodes including the number of bits and arrangement.

- In our system we use two communication protocols:

- UART

- I2C

4.4 UART

What is UART?

UART stands for Universal Asynchronous Receiver Transmitter. It is a serial communication protocol that consists of one wire for transmitting data and one wire to receive data. A common parameter is the baud rate known as "bps" which stands for bits per second. If a transmitter is configured with 9600bps, then the receiver must be listening on the other end at the same speed.

Characteristics of the UART protocol:

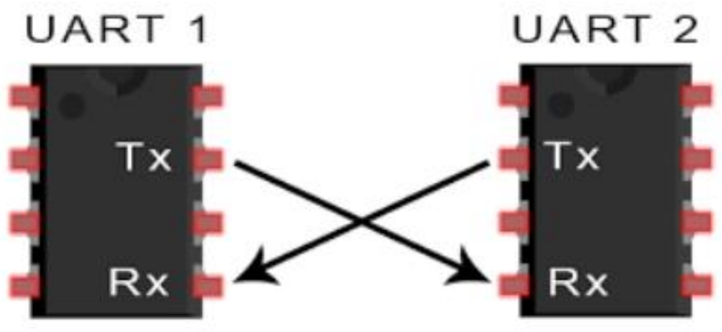
1. ***Asynchronous Communication:*** UART is an asynchronous protocol, which means that data is transmitted without the use of a clock signal. Instead, each transmitted byte is accompanied by start and stop bits, which frame the data and allow the receiving device to synchronize and extract the data.
2. ***Serial Communication****:* UART uses a serial data transmission format, where the data bits are sent one after another over the communication line. The bits are usually transmitted in a least significant bit (LSB) first order, but this can vary depending on the configuration.
3. ***Baud Rate****:* The baud rate determines the speed at which data is transmitted over the UART interface. It represents the number of bits per second that can be transmitted. Common baud rates include 9600, 19200, 115200, and so on. The transmitting and receiving devices must be configured with the same baud rate for successful communication.
4. ***Start and Stop Bits****:* Each byte of data transmitted over UART is preceded by a start bit and followed by one or more stop bits. The start bit is always

low (0) and indicates the beginning of a new byte. The stop bit or bits are always high (1) and signal the end of the byte.

5. ***Data Format:*** UART allows for various data formats, including the number of data bits, parity, and stop bits. The number of data bits typically ranges from 5 to 9 bits, with 8 bits being the most common. Parity can be used for error detection, with options like no parity, even parity, or odd parity. The number of stop bits is usually 1 or 2.
6. ***Half-Duplex Communication***: UART supports half-duplex communication, which means that data can only flow in one direction at a time. In a typical setup, one device acts as a transmitter, while the other device acts as a receiver. However, UART can also be used for full-duplex communication by utilizing separate TX and RX lines for each direction

UART Hardware Interface

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.



UART frame format

Figure 8 UART Hardware Interface

- 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.

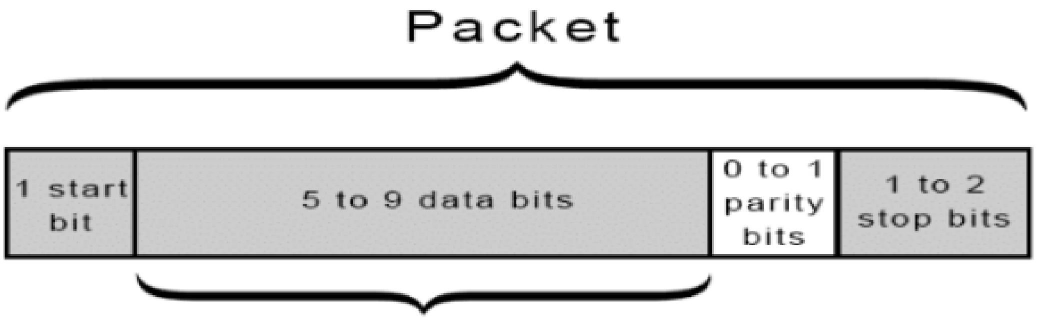


Figure 9 UART Data frame format

4.5 I2C

What is I2C?

I²C stands for Inter-Integrated Circuit (Pronounced I Two C or I Squared C), is a serial communication protocol at which the devices are hooked up to
the I2C bus with just two wires.

- It is sometimes referred to as Two Wire Interface, or the TWI
- Devices could be the CPU, IO Peripherals like ADC, or any other device which supports the I2C protocol.
- All the devices connected to the bus are classified as either being Master or Slave.

Characteristics of the I2C protocol:

1. Master-Slave Architecture: In I2C, communication typically takes place

between a master device (e.g., microcontroller) and one or more slave devices (e.g., sensors, memory chips). The master device initiates and controls the communication, while the slave devices respond to the master's commands.

2. Serial Communication: I2C employs a serial communication method, using only two wires: a data line (SDA) and a clock line (SCL). This makes it a cost-effective choice, as it requires fewer pins and wires compared to parallel communication protocols.
3. Synchronous and Bit-Serial: I2C is a synchronous protocol, meaning that data is transmitted in synchronized cycles based on the clock signal provided by the master device. It uses a bit-serial data transmission format, where data bits are sent one after another over the data line.
4. Multi-Master Capability: I2C supports multi-master communication, allowing multiple master devices to share the same bus. Each master device can take control of the bus and initiate communication with the slave devices as needed. This feature enables more complex systems with distributed control.
5. Addressing: Devices in an I2C network are identified using unique 7-bit or 10-bit addresses. The 7-bit addressing scheme is more commonly used and allows up to 128 different addresses. The 10-bit addressing scheme expands the address range to accommodate a larger number of devices.
6. Start and Stop Conditions: Communication in I2C is initiated with a Start condition, where the master device pulls the data line (SDA) low while the clock line (SCL) remains high. The Start condition signals the beginning of a transmission. Similarly, a Stop condition occurs when the master releases the data line, transitioning from low to high while the clock line is high, indicating the end of a transmission.

7. Acknowledgment: After each byte of data transmission (including the address byte), the receiving device, either the master or the slave, sends an acknowledgment (ACK) or non-acknowledgment (NACK) signal. The ACK or NACK is sent by pulling the data line (SDA) low during a specific clock pulse to indicate whether the data was successfully received.
8. Clock Speed: The clock speed in I2C is typically configurable and determines the rate at which data is transmitted. Standard clock speeds for I2C are 100 kHz (standard mode) and 400 kHz (fast mode). High-speed modes with clock speeds up to several MHz are also supported in some systems.

4.6 **Hardware design**

4.6.1 SWIDRO band

The SWIDRO band is the wrist wearable device that will be used by the swimmers. It contains the following:

- ATmega328p microcontroller
- HC-12 module
- MAX30102 pulse oximeter sensor
- ADXL345 acceleration sensor

The following is the schematic for our SWIDRO band and its PCB design.

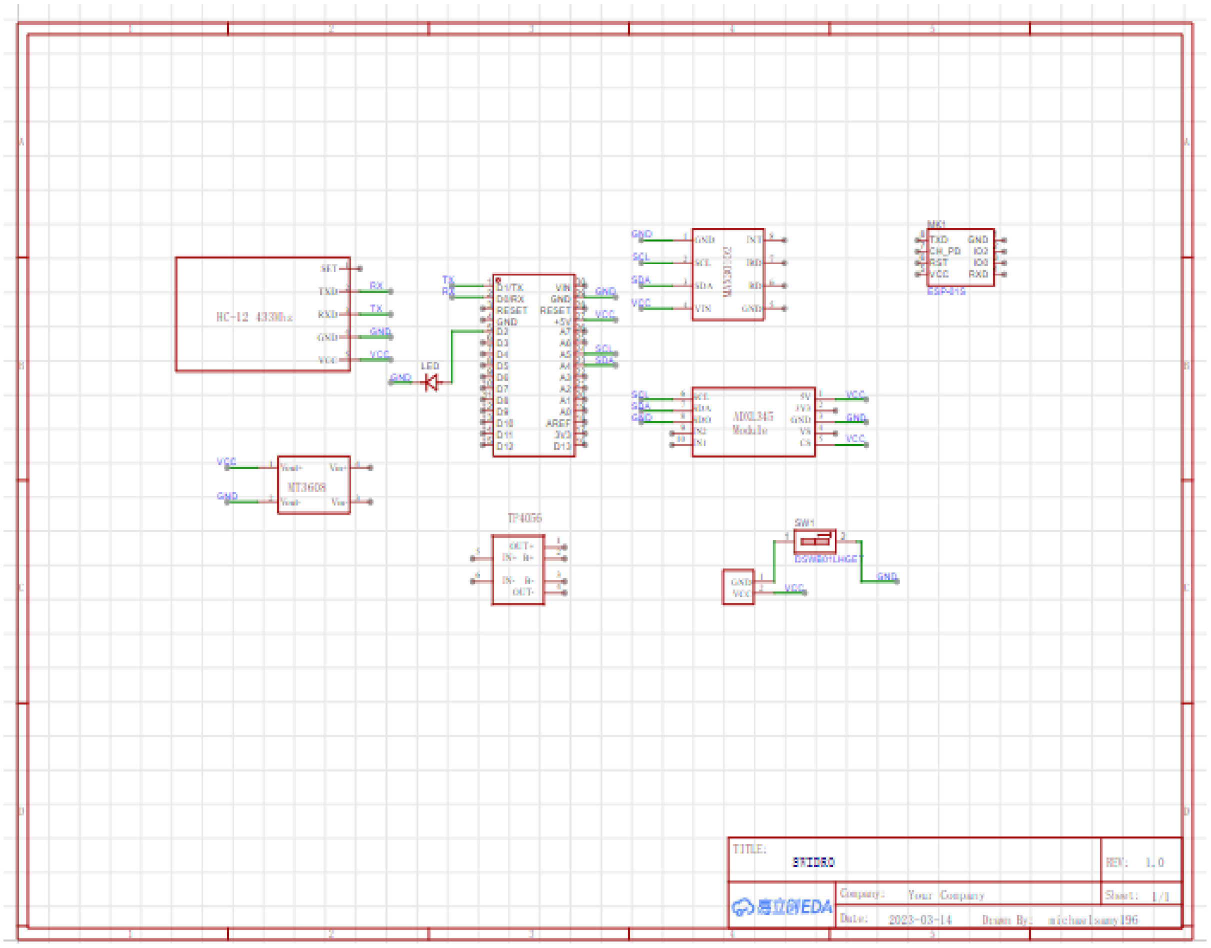


Figure 10 Band Schematic

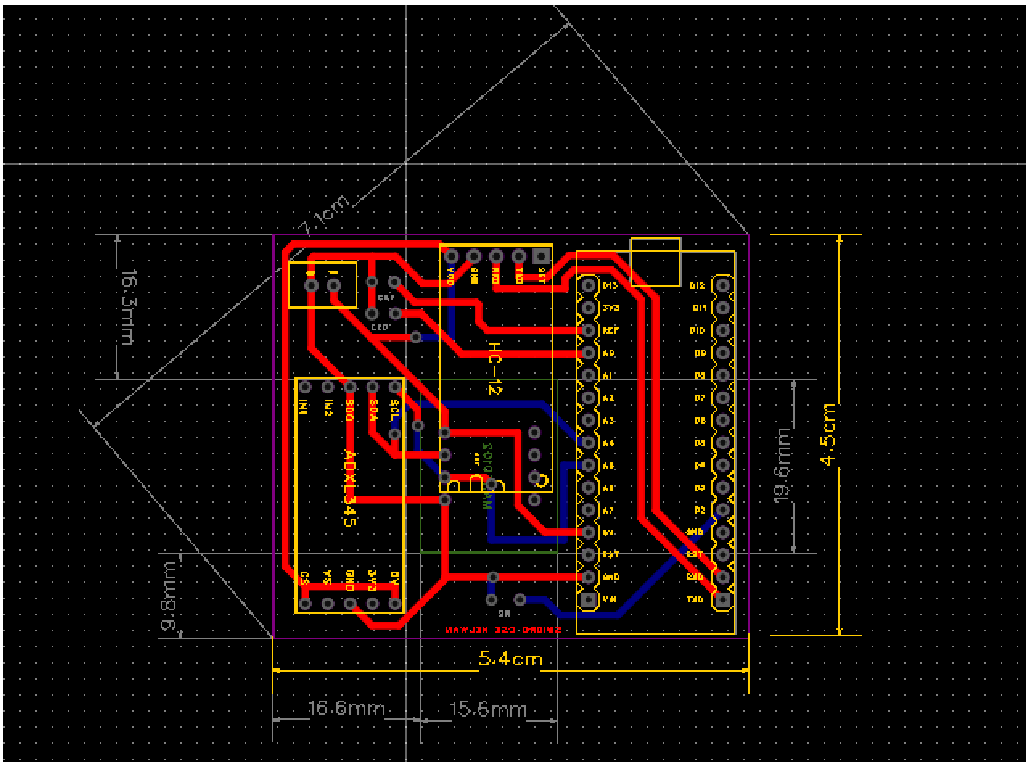


Figure 11 SWIDRO band PCB design

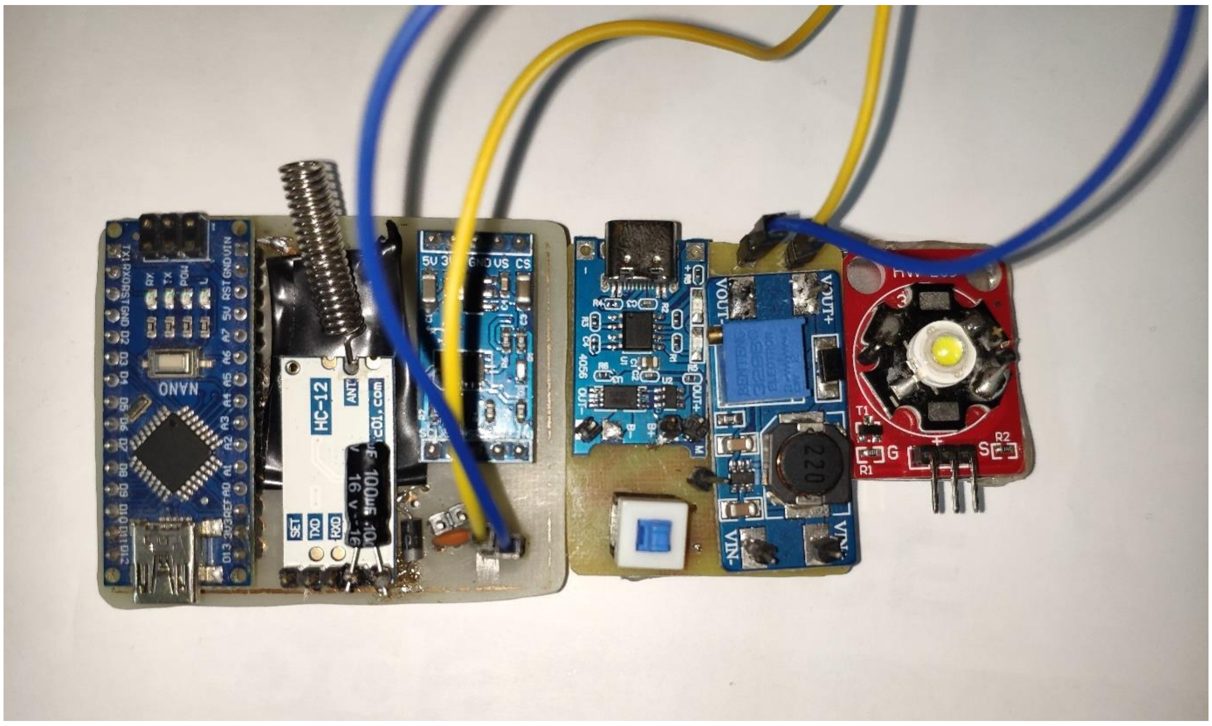


Figure 12 Band implemented.

4.6.2 SWIDRO gateway

The SWIDRO gateway is responsible for collecting the signals from the SWIDRO bands in the pools and forward it through MQTT to our server. It consists of the following:

- STM32F103 microcontroller
- HC-12 module
- ESP-01 WI-FI module

The following is the schematic and PCB design for our SWIDRO gateway.

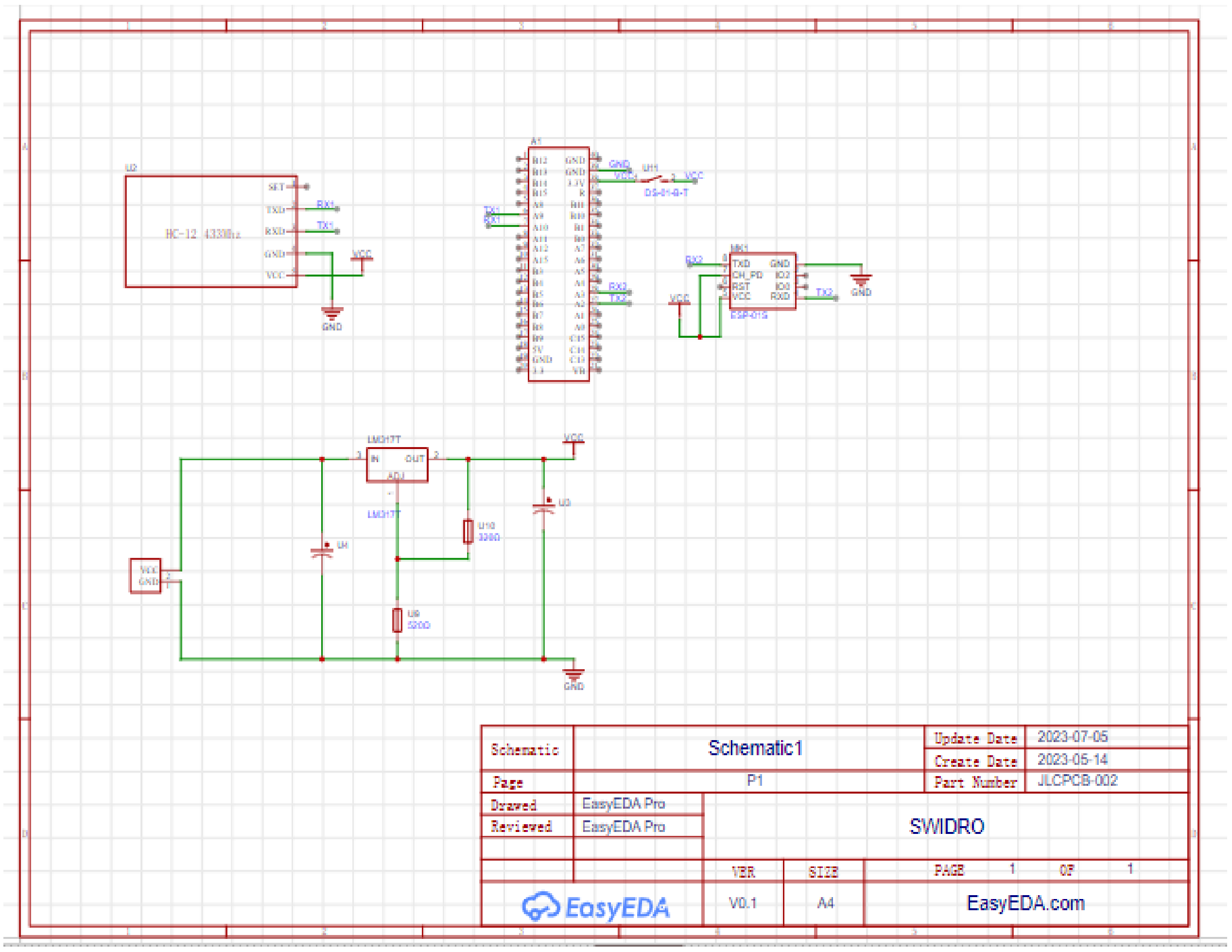


Figure 13 SWIDRO gateway schematic

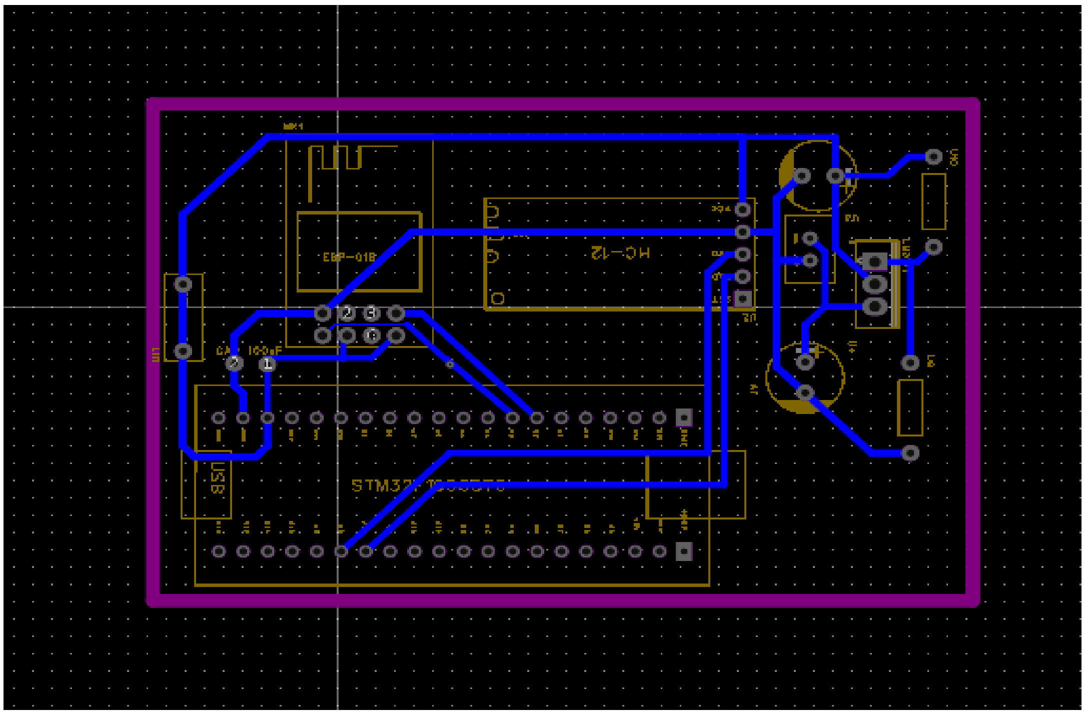


Figure 14 Gateway PCB

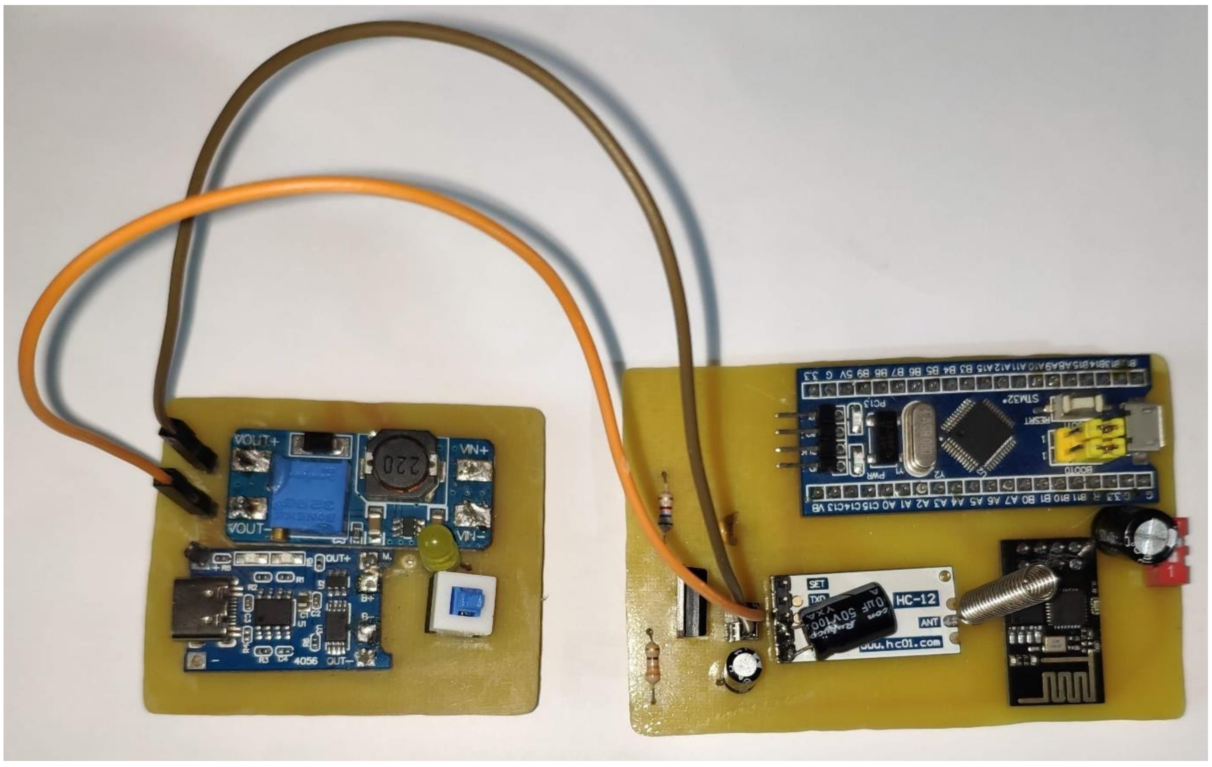


Figure 15 Gateway implemented.

4.6.3 Power circuit

The SWIDRO band and gateway needs to be powered standalone. So we have made a power circuit that uses MT3608 boost converter to increase the voltage level of the 3.7 li-po battery to 5v and use TP4056 charger module to charge the battery.

We made two of this circuit for each band and gateway. The following is the PCB design of the power circuit.

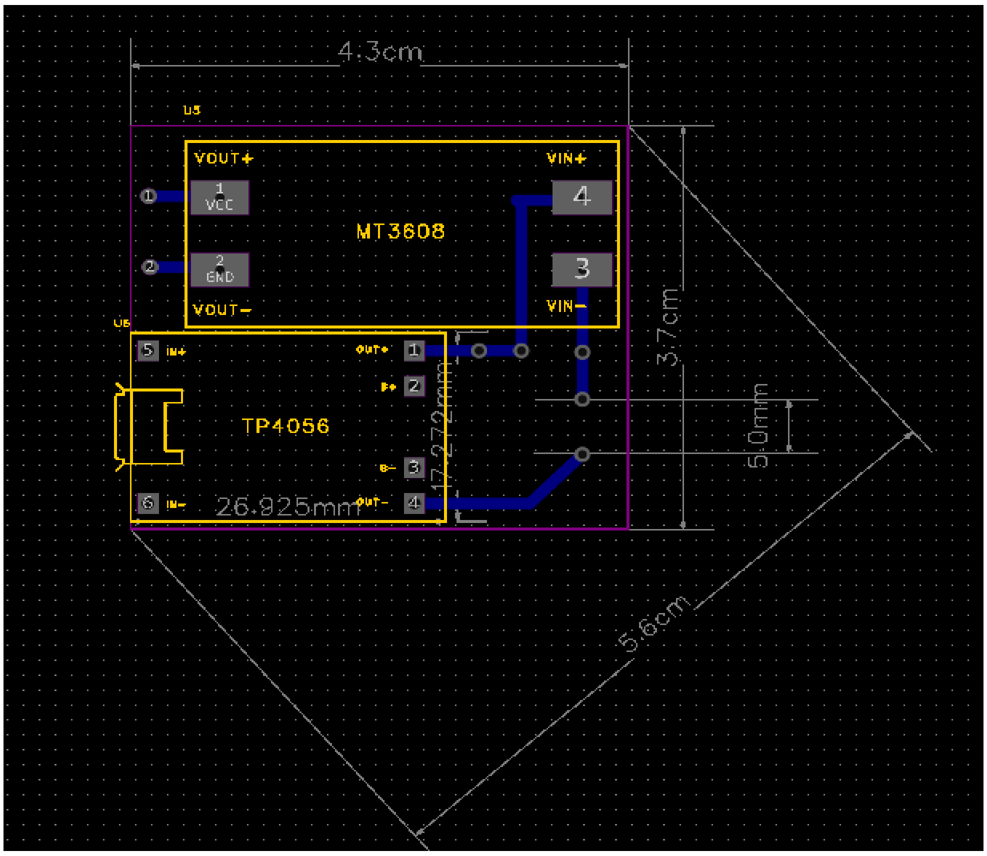


Figure 16 Power supply circuit

4.7 Sensors

Based on the requirements of the system we started to search for sensors to achieve our goal. We started to look for sensors for:
1- Acceleration.
2- Heart rate and SPO2.
3- Communication.

4.7.1 ADXL345 Accelerometer

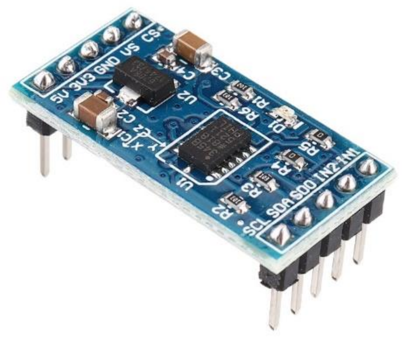


Figure 17 ADXL345 sensor module

Introduction:

The sensor is a small, low power, complete 3-axis accelerometer modules with both I2C and SPI interfaces. The ADXL345 board feature on-board 3.3V voltage regulator and level shifter, which makes it simple to interface with 5V microcontrollers such as the Arduino.

Theory of operation:

It measures both dynamic acceleration resulting from motion or shock and static acceleration, such as gravity, that allows the device to be used as a tilt sensor
The sensor is a polysilicon surface-micromachined structure built on top of a

silicon wafer. Polysilicon springs suspend the structure over the surface of the wafer and provide a resistance against forces due to applied acceleration.

Deflection of the structure is measured using differential capacitors that consist of independent fixed plates and plates attached to the moving mass. Acceleration deflects the proof mass and unbalances the differential capacitor, resulting in a sensor output whose amplitude is proportional to acceleration. Phase-sensitive demodulation is used to determine the magnitude and polarity of the acceleration.

How ADXL345 works?

1. Sensing Principle: The ADXL345 utilizes the principle of capacitance change to measure acceleration. It consists of a small, thin, flexible structure called a microstructure that moves in response to applied acceleration. The microstructure contains tiny capacitors, and the movement of the structure changes the capacitance values.
2. Sensing Axes: The ADXL345 can measure acceleration along three axes: X, Y, and Z. Each axis has a corresponding sensing element, which consists of a movable microstructure and fixed plates. Acceleration along an axis causes the microstructure to deflect, changing the capacitance between the movable and fixed plates.
3. Capacitance Sensing: The capacitance change is detected by an analog front-end circuit within the ADXL345. The circuit converts the capacitance change into electrical signals proportional to the applied acceleration.
4. Analog-to-Digital Conversion: The electrical signals from the analog front-end circuit are converted into digital values using an analog-to-digital converter (ADC). The ADXL345 has an embedded 10-bit ADC that

converts the analog signals into digital representations.

5. Digital Output: The ADXL345 provides digital output in the form of acceleration values for each axis. These values are typically expressed in units of g (gravity), where 1 g is approximately 9.8 m/s^2. The digital output can be accessed by reading the appropriate registers via the communication interface (I2C or SPI).
6. Configuration and Control: The ADXL345 has configuration registers that allow you to customize its behavior. These registers control settings such as data rate, measurement range, power modes, and interrupts. By writing specific values to these registers, you can adjust the operation of the accelerometer to suit your application requirements.
7. Communication Interface: The ADXL345 supports both I2C (Inter-Integrated Circuit) and SPI (Serial Peripheral Interface) communication interfaces. These interfaces allow you to communicate with the accelerometer and read the digital output values.

4.7.2 MAX30102 Pulse oximeter and Heart rate sensor

General Description

The MAX30102 is an integrated pulse oximetry and heart-rate monitor module. It includes internal LEDs, photodetectors, optical elements, and low-noise electronics with ambient light rejection. The MAX30102 provides a complete system solution to ease the design-in process
for mobile and wearable devices.
The MAX30102 operates on a single 1.8V power supply and a separate 3.3V power supply for the internal LEDs. Communication is through a standard I2C-compatible interface. The module can be shut down through software with zero

standby current, allowing the power rails to remain powered at all times.

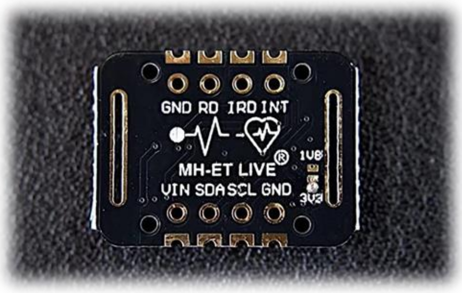


Figure 18 MAX30102 sensor module

How MAX30102 Pulse Oximeter and Heart Rate Sensor Works?

The MAX30102, or any optical pulse oximeter and heart-rate sensor for that matter, consists of a pair of high-intensity LEDs (RED and IR, both of different wavelengths) and a photodetector. The wavelengths of these LEDs are 660nm and 880nm, respectively.

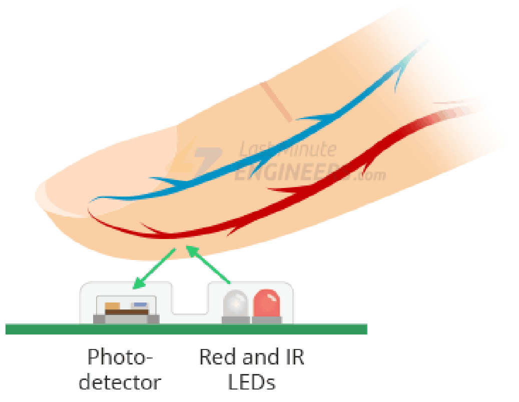


Figure 19 Working of MAX30102

The MAX30102 works by shining both lights onto the finger or earlobe (or essentially anywhere where the skin isn't too thick, so both lights can easily penetrate the tissue) and measuring the amount of reflected light using a photodetector. This method of pulse detection through light is called Photoplethysmogram.

The working of MAX30102 can be divided into two parts: Heart Rate Measurement and Pulse Oximetry (measuring the oxygen level of the blood).

Heart Rate Measurement

The oxygenated hemoglobin (HbO2) in the arterial blood has the characteristic of absorbing IR light. The redder the blood (the higher the hemoglobin), the more IR light is absorbed. As the blood is pumped through the finger with each heartbeat, the amount of reflected light changes, creating a changing waveform at the output of the photodetector. As you continue to shine light and take photodetector readings, you quickly start to get a heart-beat (HR) pulse reading.

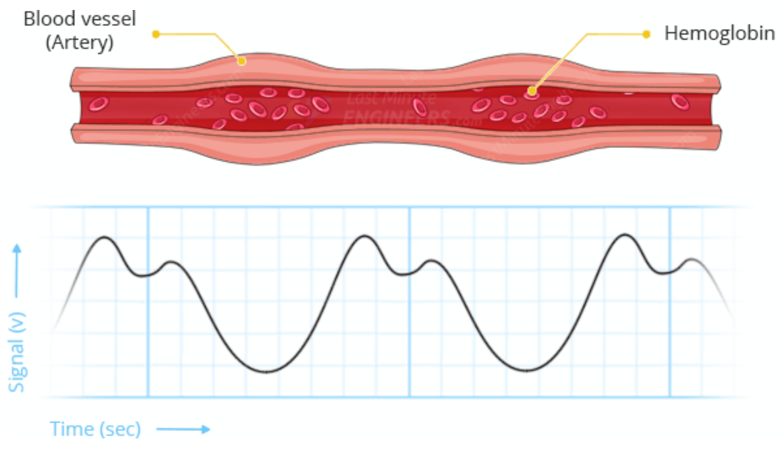


Figure 20 Signal of heartbeats

Pulse Oximetry

Pulse oximetry is based on the principle that the amount of RED and IR light absorbed varies depending on the amount of oxygen in your blood. The following graph is the absorption-spectrum of oxygenated hemoglobin (HbO2) and deoxygenated hemoglobin (Hb).

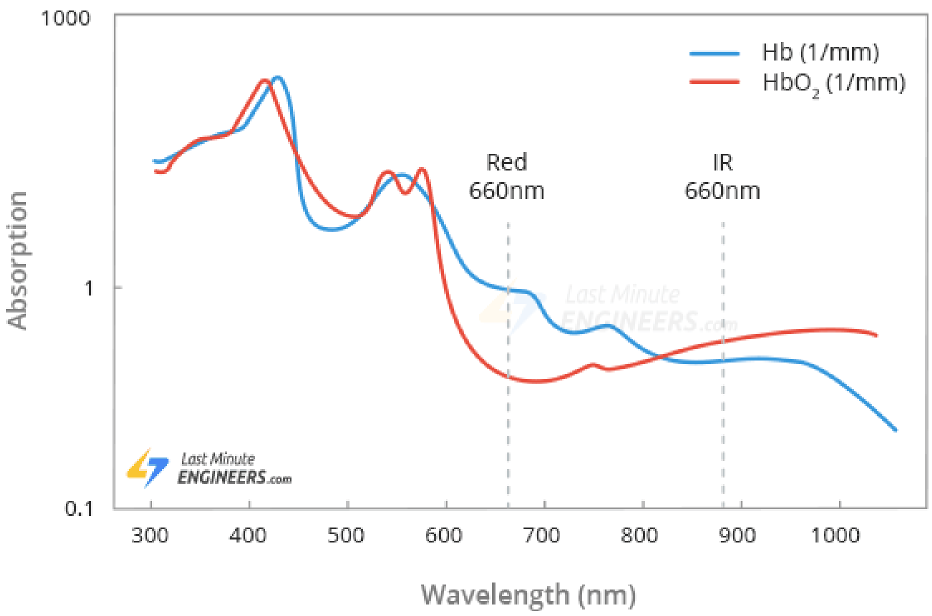


Figure 21 Absorbtion of different wavelengths by HBO2 and HB

4.7.3 MT3608 Boost converter

General Description

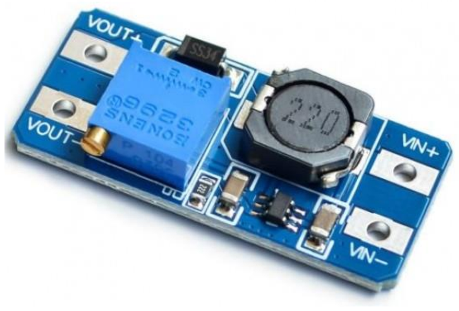


Figure 22 MT3608 boost converter module

- The MT3608 is a constant frequency, 6-pin SOT23 current mode step up converter intended for small, low power applications. The MT3608 switches at 1.2MHz and allow the use of tiny, low cost capacitors and inductors 2mm or less in height. Internal soft-start results in small inrush current and extends battery life
- The MT3608 is a step-up (boost) DC-DC converter module that allows you to

increase a lower input voltage to a higher output voltage. It is a popular and widely used module due to its low cost, small size, and efficiency.
- The MT3608 module can accept input voltages as low as 2V and can boost the output voltage to up to 28V

- It has a maximum current output of 2A, making it suitable for powering small electronic devices such as Arduino and Raspberry Pi boards, sensors, and small motors.
- The MT3608 module is often used in battery-powered applications such as solar-powered systems, portable electronic devices, and small robots, where the input voltage from the battery is too low to power the required components.
- The module can also protect the battery from over-discharge by shutting off when the input voltage drops below a certain threshold.

The operation of the MT3608 module:

1. Connect the input voltage to the VIN pin on the module. The input voltage should be between 2V and 24V.
2. Connect the ground (GND) pin to the ground of your circuit.
3. Connect the load (the device you want to power) to the VOUT pin on the module. The output voltage can be adjusted using a potentiometer on the module.
4. Apply power to the input voltage source.
5. The module will start boosting the input voltage to the desired output voltage. The output voltage can be adjusted by turning the potentiometer clockwise or counterclockwise.
6. The module has over-temperature and over-current protection features to prevent damage to the module or the load. If the temperature of the module

exceeds a certain threshold or the load draws too much current, the module will shut down to protect itself.

- The MT3608 module is designed to be highly efficient, with a typical conversion efficiency of up to 96%. This means that most of the input power is converted into output power, with very little wasted as heat.
- The MT3608 module has a number of built-in protection features, including over-temperature protection, over-current protection, and short-circuit protection. These features help to prevent damage to the module and connected devices.

4.7.4 TP4056 Lithium battery charger

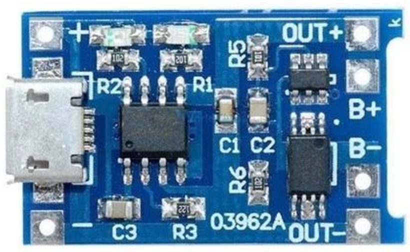


Figure 23 TP4056 charger module

General Description

- TP4056 is a lithium-ion battery charging controller IC (integrated circuit) that is commonly used for charging single-cell lithium-ion or lithium-polymer batteries.
- The TP4056 chip also prevents overcharging of the battery by stopping the charging process when the battery is fully charged and provides various protection features such as overcharge protection, over-discharge protection, short-circuit protection, and over-current protection

- The working principle of the TP4056 module is based on a constant-current/constant-voltage (CC/CV) charging algorithm.

The constant-current/constant-voltage (CC/CV) charging algorithm is a popular charging method used for rechargeable batteries, especially Lithium-ion batteries. It is designed to charge the battery in **two stages:**

1. **Constant Current (CC) Stage:** In this stage, the battery is charged with a constant current until it reaches a specified voltage. During this stage, the charging current remains constant, and the battery voltage increases gradually.
2. **Constant Voltage (CV) Stage:** In this stage, the charging voltage is maintained at a constant level while the charging current decreases gradually. This stage continues until the battery is fully charged, and the charging current reaches a predetermined level.

- When the TP4056 module is first connected to a battery, it begins charging the battery with a constant current. The charging current is determined by the value of the external resistor connected to the module's "PROG" pin. Typically, a resistor value of 1.2 kΩ will result in a charging current of 1A.

- The TP4056 module has an LED indicator that shows the charging status of the battery. The LED is red when the battery is being charged and turns off when the battery is fully charged.

- The TP4056 module can be powered by a USB port or a DC power supply. The input voltage range is 4.5V to 5.5V and the maximum input current is 1A.

- The TP4056 module provides a constant output voltage of 4.2V to charge the battery. The charging current can be set using a resistor, typically between 1A to 2A.

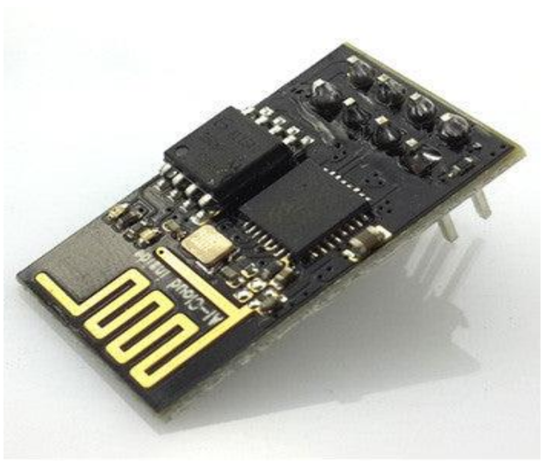


Figure 24 ESP-01 wifi module

4.7.5 ESP-01 WiFi Module

Introduction

The ESP-01 is a low-cost, small-sized WIFI module that is based on the ESP8266 chipset. The ESP8266 is a highly integrated chip that includes a microcontroller, WIFI connectivity, and a networking stack. It was designed to provide wireless connectivity to microcontrollers and other embedded systems.
It has eight pins, but only six of them are usable for communication. It can be powered with a supply voltage of 3.3V and has an 80MHz 32-bit Tensilica processor. The module is designed to work with a variety of development boards, including Arduino and Raspberry Pi.
It can be used in a variety of Internet of Things (IoT) projects, where it can provide wireless connectivity to sensors, actuators, and other devices.

Basic operation

To operate the ESP-01 WiFi module, you will need to follow a few steps:

1. Power the module: The ESP-01 module requires a power supply of 3.3V. You can power it using a voltage regulator or a dedicated power supply module.

2. Connect the module to a microcontroller or development board: The ESP-01 module has eight pins, but only six of them are usable for communication. You will need to connect the RX pin of the ESP-01 module to the TX pin of your microcontroller or development board, and the TX pin of the ESP-01 module to the RX pin of your microcontroller or development board. You will also need to connect the VCC and GND pins of the ESP-01 module to the power supply and ground of your microcontroller or development board.
3. Enable the module: To enable the ESP-01 module, you will need to pull the CH_PD (chip enable) pin high. This can be done using a resistor or a transistor.
4. Configure the module: Once the module is enabled, you can configure it using AT commands. AT commands are a set of commands that allow you to set the WiFi mode, network credentials, and other settings of the module. You can send these commands to the module using a serial communication interface.
5. Use the module: Once the module is configured, you can use it to communicate with other devices over the WiFi network. You can send and receive data using TCP or UDP protocols.

4.7.6 HC-12 module

The HC-12 module is a wireless transceiver module that operates on the 433MHz frequency band. It is commonly used for establishing wireless communication between microcontrollers or other electronic devices over long distances.

Characteristics of the HC-12 module:

1. Frequency: The HC-12 operates in the 433.4 to 473.0 MHz frequency range, and it has 100 communication channels available.
2. Communication Range: The HC-12 module can achieve communication ranges of up to 1 kilometer in open areas with line-of-sight conditions. The actual range may vary depending on environmental factors and obstacles.
3. Serial Communication: It supports serial communication using UART (Universal Asynchronous Receiver-Transmitter) protocol. This allows for simple integration with microcontrollers and other devices using serial communication.
4. Power Supply: The module typically operates at a supply voltage of 3.2 to 5.5 volts DC. It has low power consumption, making it suitable for battery-powered applications.
5. Communication Modes: The HC-12 module supports several modes of operation, including a transparent mode and a fixed-point mode. In transparent mode, it functions as a transparent wireless serial link, transmitting and receiving data packets. In fixed-point mode, it can establish point-to-point or multi-point communication.
6. Adjustable Parameters: The module offers adjustable parameters such as the baud rate, communication channel, transmission power, and air data rate. These parameters can be configured according to the specific requirements of your application.
7. Error Correction: The HC-12 module implements forward error correction (FEC) to enhance the reliability of data transmission over long distances. This helps to mitigate the effects of noise and interference in the wireless channel.
8. Antenna: The module requires an external antenna for optimal

performance. A wire antenna of appropriate length is usually recommended, and the antenna design and placement can have a significant impact on the range and signal quality.

9. Compatibility: The HC-12 module is compatible with a wide range of microcontrollers, such as Arduino boards, STM32, PIC, and others. It can be easily integrated into existing projects and systems.

4.8 SWIDRO system

Now we know the sensors and the software architectures in embedded systems. We can talk in detail about the hardware that achieves the success of this **requirements.**

4.8.1 Band to gateway communication

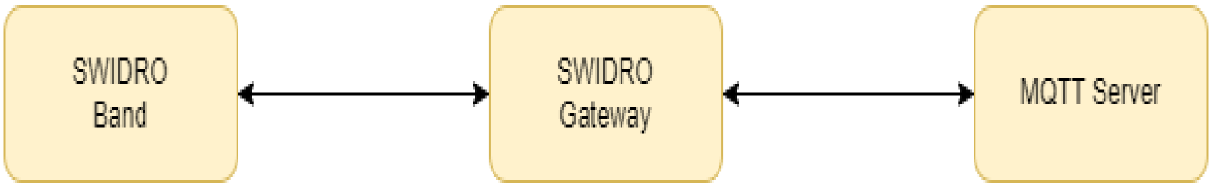


Figure 25 system overview

Since we need to communicate our band to the server, we needed a gateway that will be placed in the pool near all band devices that will take the signals and forward it to the server to make the application update the status.

Band and gateway **are connected using HC-12 module as this module is running in 433MHZ band (ISM band) it will make a lower attenuation in water compared to WI-FI and GSM which is running on a higher frequency band. This allowed us to achieve a better result in water Up to 5-meter radius**

range with 50 cm deep in water.

The band read data from MAX30102 and ADXL345 then calculate the acceleration and heart rate and SPO2 then sends them to the gateway which in turns will forward it to the MQTT server.

4.8.2 Band system

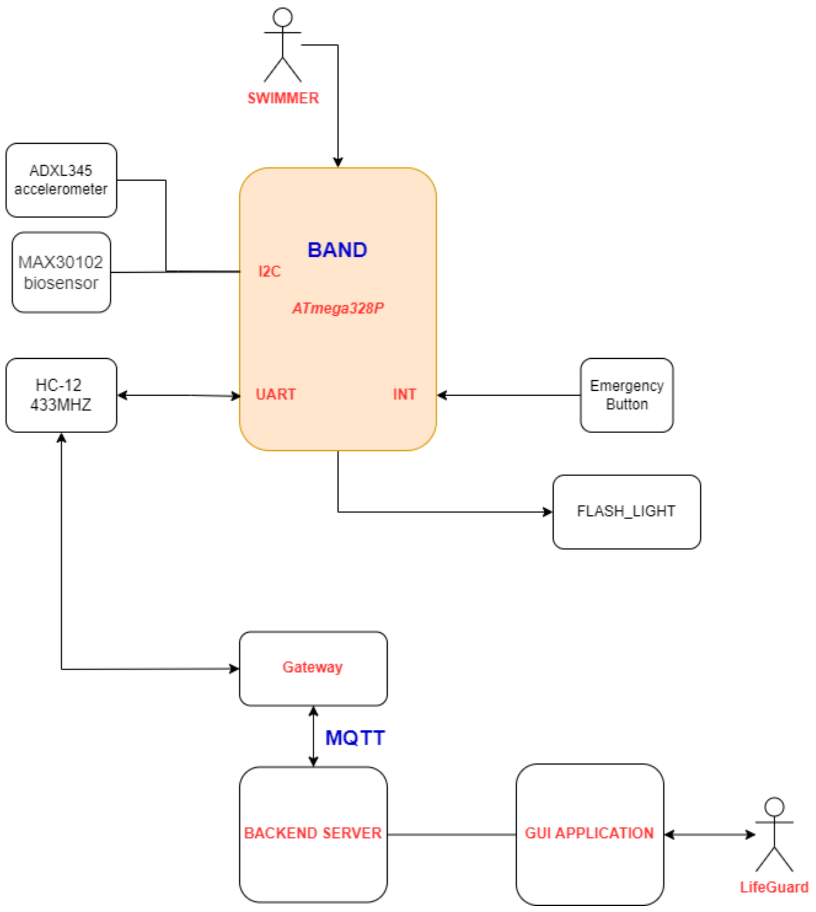


Figure 26 Band detailed diagram

In figure 23 we can see that the band is running on ATmega328p microcontroller and connected to the sensors through I2C communication protocol, the HC-12 is connected to the UART and finally we have an emergency button to send emergency signal when the swimmer feels an emergency to get help from lifeguards. The flashlight is an indication for an emergency which may help lifeguards in determining the location if is dark. The flashlight could be replaced with any kind of life saving technique and can be integrated with any other systems.

4.8.3 Gateway system

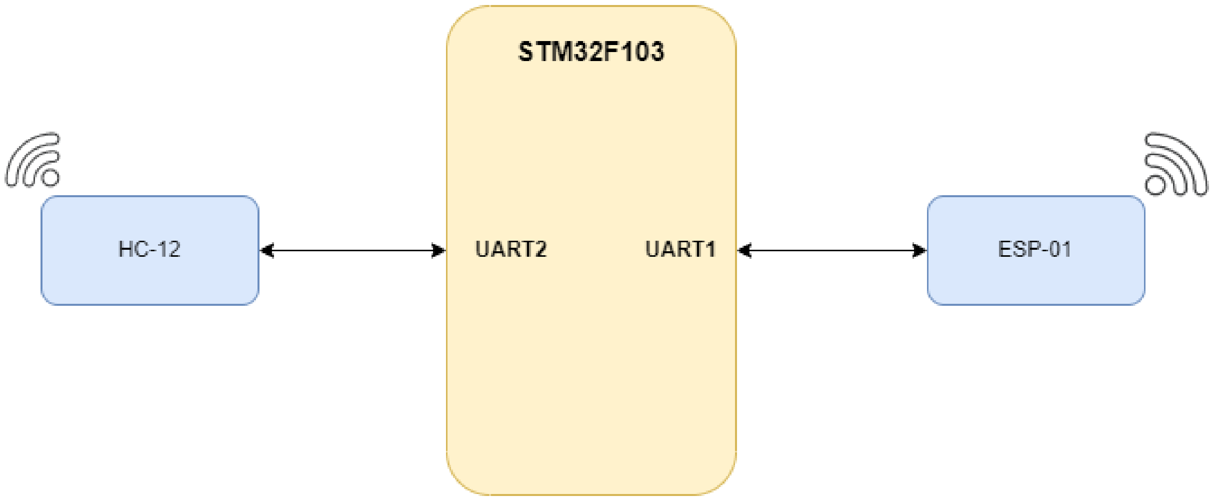


Figure 27 SWIDRO gateway diagram

In the Gateway we chose to use STM32F103 microcontroller as it is a more industrial and reliable than ATmega328p. The gateway is connected to the HC-12 module and ESP-01 module as in figure 20. The HC-12 module is connected to UART2 and is responsible for getting the SWIDRO bands' signals and publish it to the ESP-01. The ESP-01 module is used to publish MQTT messages and subscribes to the ALERT topics if drowning is detected so that it will forward the signal to the band and enable emergency on it.

5 Web platform

5.1 introduction

The purpose of the web platform is to ensure that every lifeguard receives timely notifications about drowning incidents in their designated area so that they can take immediate action to save lives. This functionality is a key aspect of the platform. Lifeguards will be connected to the platform through their phones, enabling them to receive real-time notifications regarding swimmers in distress.

The system is not limited to lifeguards alone; it includes additional pages that enhance the overall convenience of using the system. This dashboard allows for the management of devices, the addition of staff members, and the monitoring of swimmers.

5.2 platform components

5.2.1 front-end

A front-end refers to the user-facing aspect of a **software application or website. It is the visual and interactive layer that users directly interact with. The main purpose of the front-end is to provide an intuitive and seamless experience for users, allowing them to easily navigate and interact with the application.**

The front-end is responsible for presenting information, graphics, and controls in a visually appealing and organized manner**. It incorporates various technologies such as HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), and JavaScript to create a responsive and interactive user interface.**

A well-designed front-end focuses on user experience and usability. It ensu**res that the interface is visually appealing, with a harmonious blend of colors, typography, and graphical elements. The layout is structured and intuitive, guiding users through the application effortlessly.**

Interactivity is a key aspect of the front-end. **It enables users to perform actions, submit forms, and interact with different elements of the application. JavaScript plays a crucial role in adding dynamic functionality, such as form validation, data manipulation, and real-time updates.**

Responsive desi**gn is another essential consideration in front-end**

development. With the prevalence of mobile devices, it is crucial to create interfaces that adapt seamlessly to different screen sizes and orientations. Responsive design ensures that the application is accessible and usable across various devices, enhancing the overall user experience.

Front-end development also involves optimizing the performance of the application. This includes m**inimizing load times, optimizing images and media, and reducing unnecessary requests to enhance speed and efficiency. These optimizations contribute to a smoother user experience and encourage user engagement.**

In summary, a front-end is the user-facing par**t of an application, designed to provide an intuitive and visually appealing experience. It incorporates technologies like HTML, CSS, and JavaScript to create interactive interfaces that adapt to different devices. The front-end focuses on user experience, usability, interactivity, and performance to deliver a seamless and engaging user interface.**

5.2.2 back-end server

The backend of a software application or website is the hidden infrastructure that supports the front-end and handles behind-the-scenes operations. Unlike the front-end, which focuses on the user interface and interactions, the backend is responsible for data storage, processing, and the overall functionality of the application.

At its core, the backend consists of a server, a database, and the logic that connects them. It receives requests from the front-end and processes them, fetching and manipulating data as needed. The backend's primary goal is to provide the necessary information and resources to the front-end, enabling it to deliver a seamless user experience.

Backend development often involves programming languages such as Python, Java, Ruby, or PHP, along with frameworks and libraries specific to each language. These tools allow developers to create the server-side logic and handle data operations efficiently.

One of the essential aspects of the backend is data management. It involves designing and implementing a database structure that can store and retrieve data reliably. The backend handles tasks such as data validation, storage, retrieval, and manipulation, ensuring data integrity and security.

Another critical aspect of the backend is API (Application Programming Interface) development. APIs enable communication between different software systems and allow data exchange. Backend developers create APIs that expose specific functionalities of the application, enabling other systems or external services to interact with it.

Security is a paramount consideration in backend development. The backend must implement robust security measures to protect sensitive data and prevent unauthorized access. This includes measures like encryption, authentication mechanisms, and implementing proper access controls.

Scalability is also a key concern in backend development. As the application grows and the user base expands, the backend should be capable of handling increased traffic and processing demands. This often involves implementing techniques such as load balancing, caching, and distributed systems to ensure optimal performance.

Monitoring and debugging are crucial for maintaining a healthy backend. Developers use various tools and techniques to monitor the performance and identify potential issues. Logging and error handling mechanisms help in detecting and resolving errors or bugs in the backend code.

In summary, the backend of an application or website provides the infrastructure and functionality required to support the front-end. It handles data storage, processing, and communication with other systems. Backend development involves programming languages, databases, APIs, security measures, scalability considerations, and monitoring techniques to ensure efficient and reliable application performance.

5.2.3 server for the ai model

In a backend system that hosts an AI model and handles API requests, the server plays a crucial role in managing the AI functionalities and facilitating the prediction process. Here are more details on how this backend server operates:

Hosting the AI Model: The backend server hosts the AI model, which can be a machine learning model or any other type of AI model trained for a specific task. This involves loading the model into memory and initializing it to make predictions or perform desired operations.

Handling API Requests: The backend server acts as an intermediary between the front-end or external systems and the AI model. It receives API requests containing input data from the clients and forwards them to the AI model for processing. These requests can be made in various formats, such as JSON or multipart/form-data, depending on the requirements of the API.

Preprocessing Input Data: Before passing the data to the AI model, the backend server often preprocesses the input data to ensure it is in the correct format and meets the AI model's input requirements. This may involve tasks such as data normalization, scaling, or feature extraction, depending on the specific needs of the AI model.

Making Predictions: Once the input data is ready, the backend server sends it to the AI model for prediction. The AI model processes the data based on its trained algorithms and produces the desired output, such as a prediction, classification, or recommendation. The server captures the response from the AI model and prepares it for further processing or transmission back to the client.

Post-processing and Response Handling: After receiving the predictions from the AI model, the backend server can perform post-processing tasks on the output, such as formatting the response in a specific structure or applying additional business logic. It prepares the final response and sends it back to the client in a suitable format, typically as an API response, such as JSON or XML.

Performance and Scalability: The backend server must handle multiple API requests simultaneously and efficiently manage the computational resources required by the AI model. This may involve techniques such as load balancing, caching, or parallel processing to ensure optimal performance and responsiveness, especially during periods of high traffic or heavy computational loads.

Error Handling and Logging: The backend server implements error handling mechanisms to capture and manage any errors or exceptions that occur during the API request processing or prediction phase. It logs relevant information about these errors for debugging purposes or to provide valuable insights into the system's behavior.

Security and Access Control: To ensure the security and integrity of the AI model and the data being processed, the backend server incorporates appropriate security measures. This may include implementing authentication and authorization mechanisms, encryption of sensitive data, and ensuring secure communication channels.

In summary, the backend server that hosts an AI model and handles API requests acts as an intermediary between the clients and the AI model. It manages the preprocessing and post-processing of data, makes predictions using the AI model, handles error conditions, ensures performance and scalability, and incorporates security measures to protect the system.
platforms tools

5.2.4 front-end

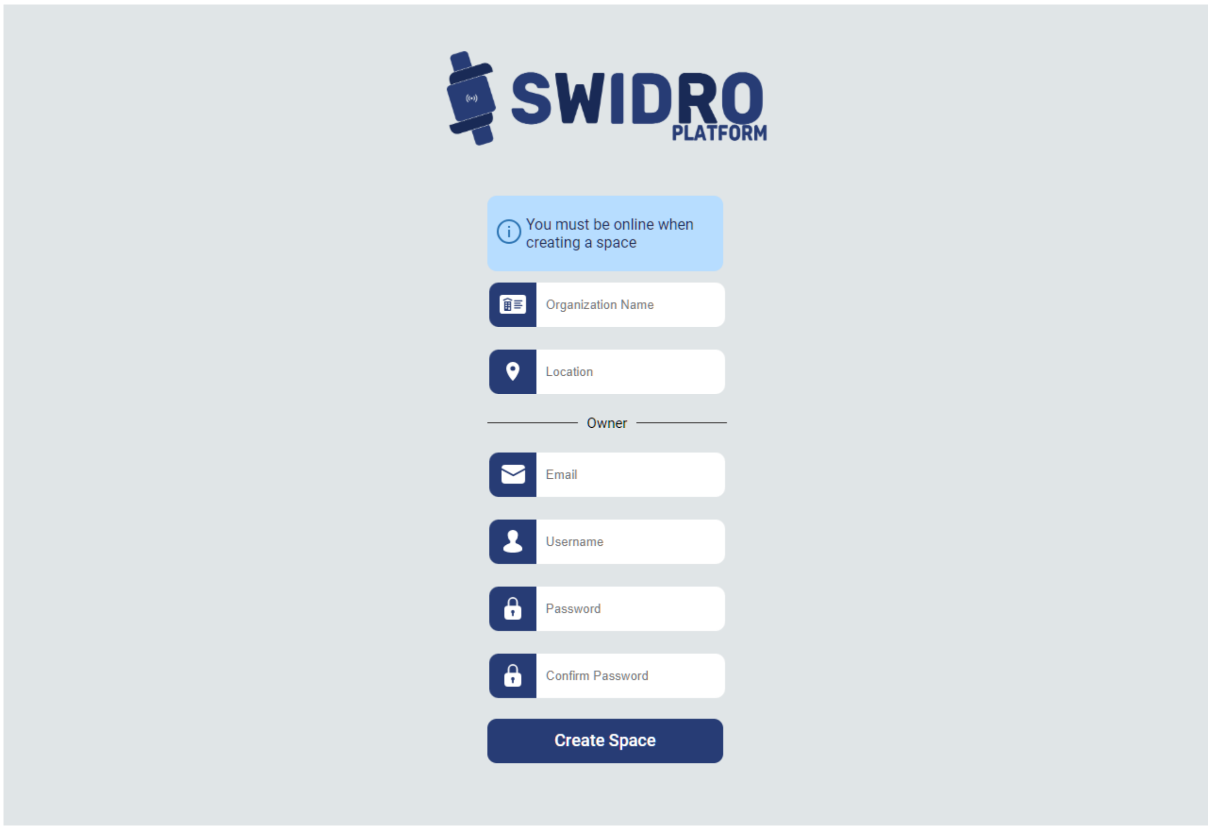
react **framework**

5.2.5 back-end

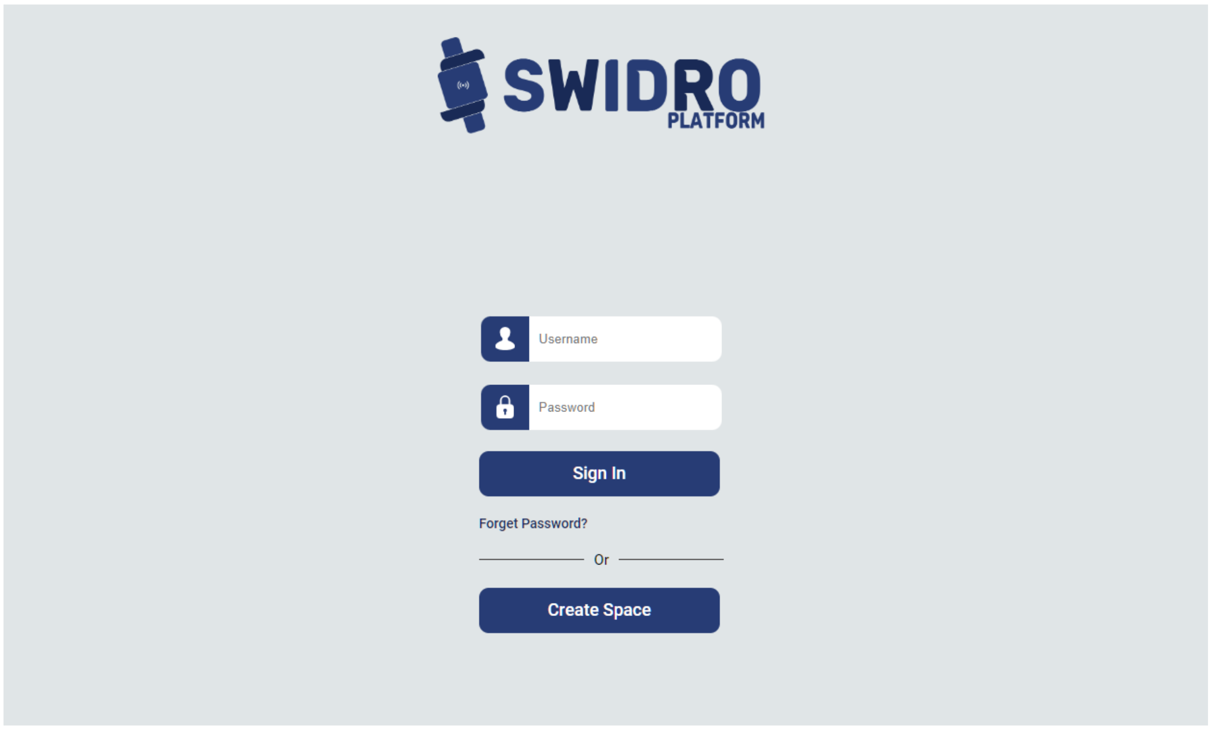
node **JS and express JS and mongo DB**

5.3 platform view

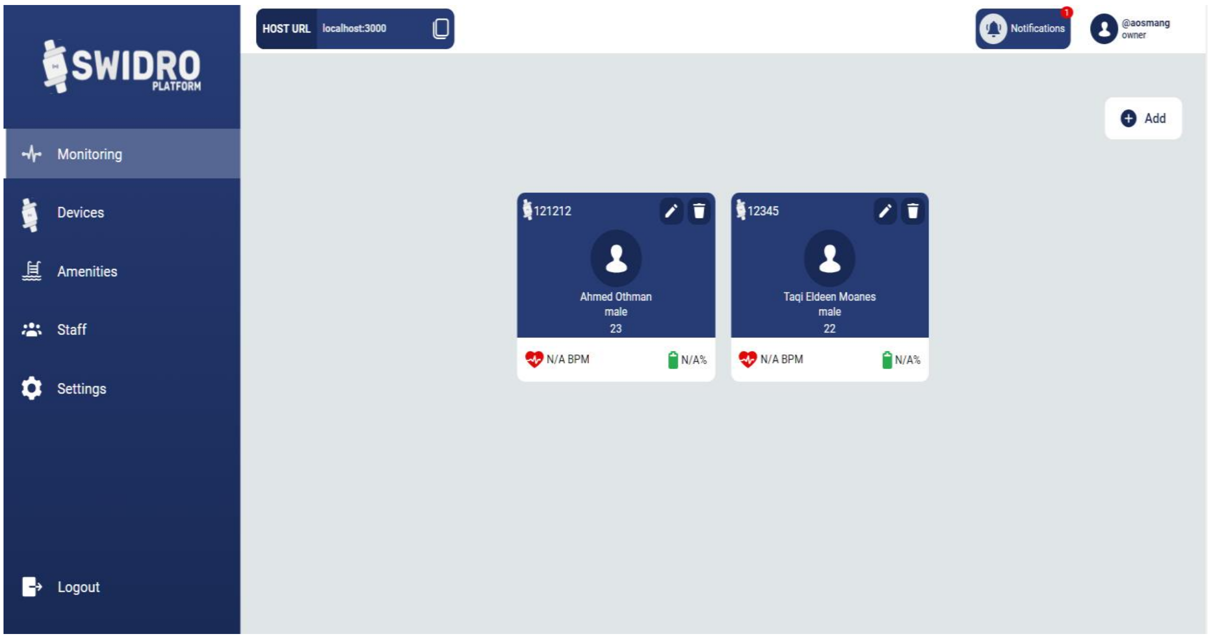
5.3.1 sign up page



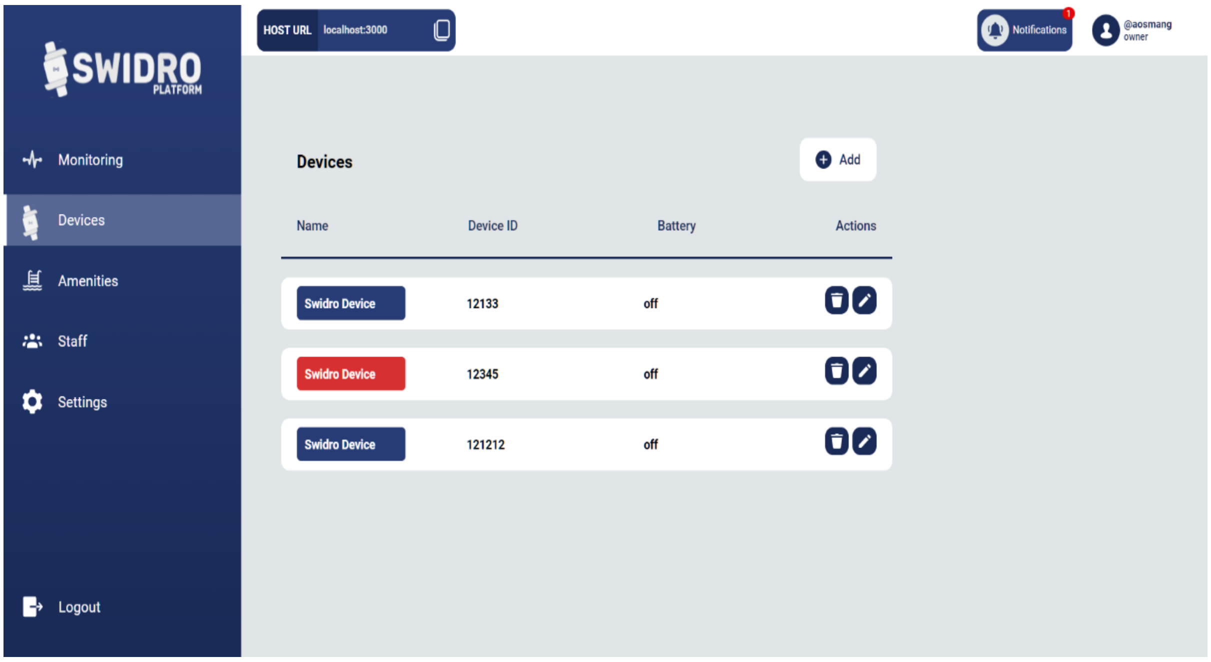
5.3.2 sign in page



5.3.3 monitoring page



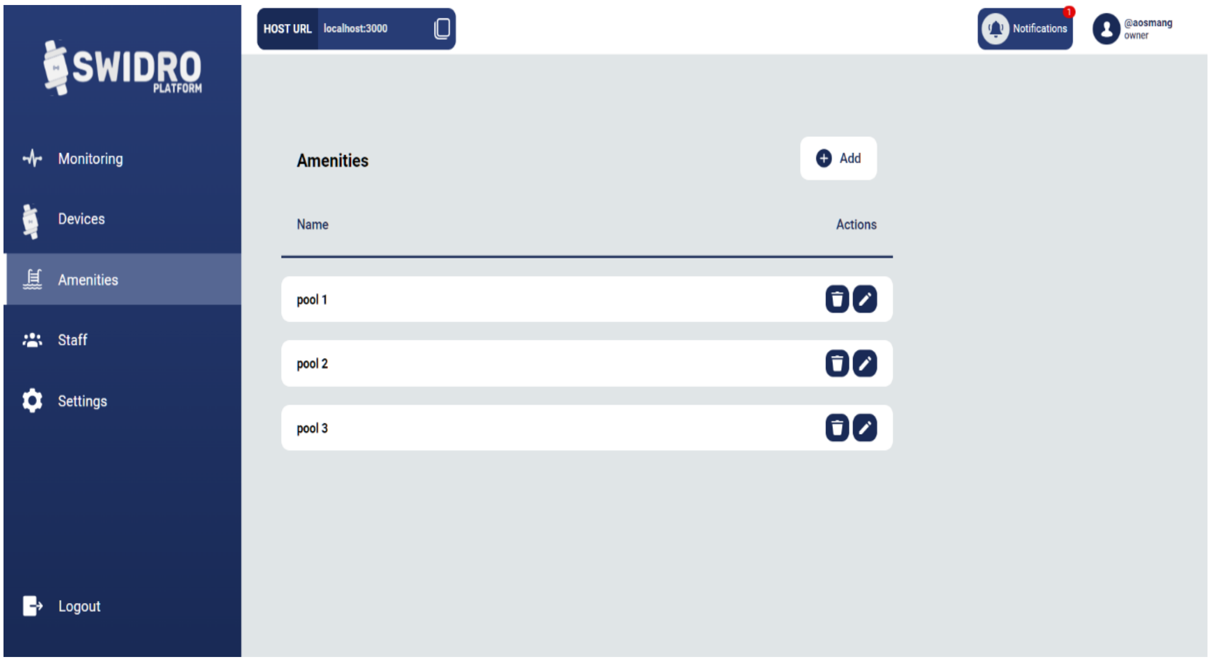
5.3.4 Devices



Devices

Name	Device ID	Battery	Actions
Swidro Device	12133	off	
Swidro Device	12345	off	
Swidro Device	121212	off	

5.3.5 Amenities


HOST URL localhost:3000

Notifications

@aosmang owner

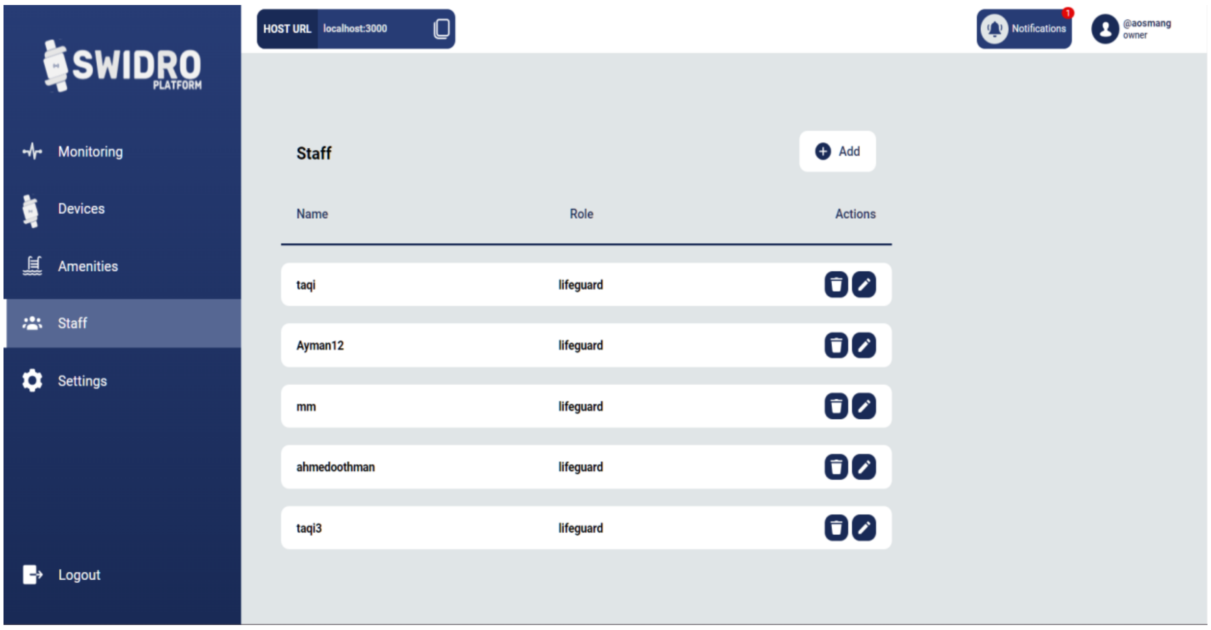
Monitoring
Devices
Amenities
Staff
Settings
Logout

Amenities

Add

Name	Actions
pool 1	
pool 2	
pool 3	

5.3.6 Staff


HOST URL localhost:3000

Notifications

@aosmang owner

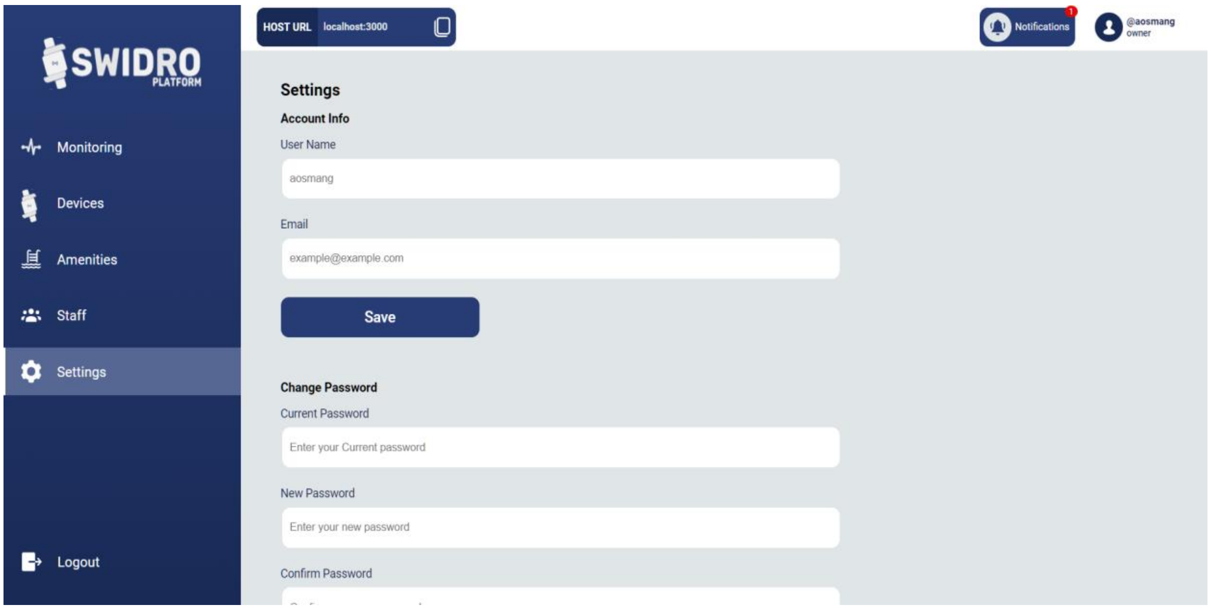
Monitoring
Devices
Amenities
Staff
Settings
Logout

Staff

Add

Name	Role	Actions
taqi	lifeguard	
Ayman12	lifeguard	
mm	lifeguard	
ahmedoothman	lifeguard	
taqi3	lifeguard	

5.3.7 settings



Settings

Account Info

User Name: aosmang

Email: example@example.com

Change Password

Current Password: Enter your Current password

New Password: Enter your new password

Confirm Password

6 Artificial Intelligence

6.1 Intro to AI

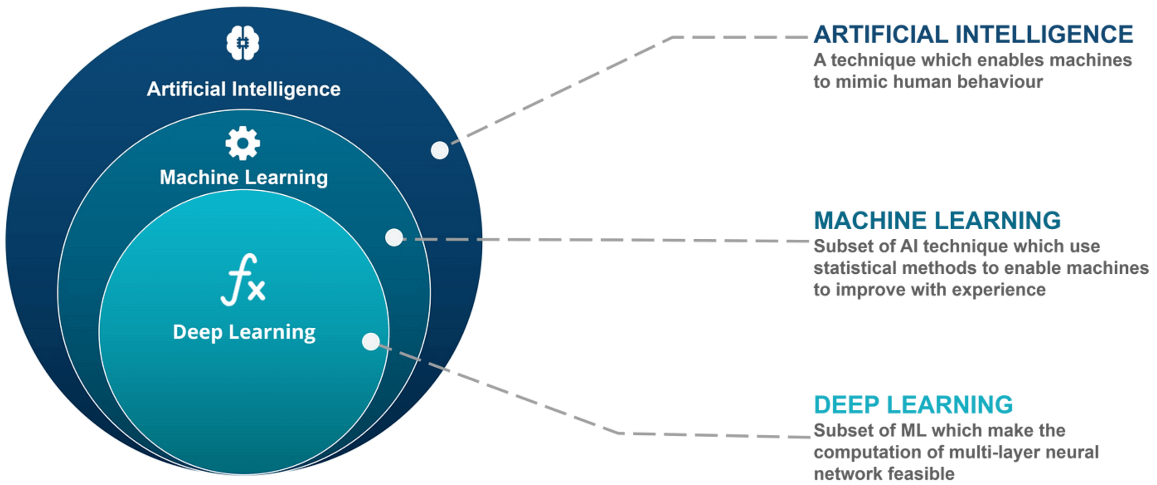

Artificial Intelligence
Machine Learning
Deep Learning
ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour
MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience
DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

Figure 28 AI Branches

Artificial intelligence (AI) is the ability of a **computer, or a robot controlled by a computer to do tasks that usually require human intelligence and discernment. AI research has been highly successful in developing effective techniques for solving a wide range of problems, from game playing to medical diagnosis.**
Artificial Intelligence (AI) is a rapidly evolving field that has gained significant attention and is revolutionizing numerous industries and domains. It refers to the development of intelligent machines capable of performing tasks that typically require human intelligence. AI systems are designed to perceive, reason, learn, and make decisions, often surpassing human capabilities in specific domains.
In recent years, AI has demonstrated its potential to address complex problems and improve various aspects of our lives. From healthcare and finance to transportation and entertainment, AI applications have been widely adopted. One such promising application is in the domain of drowning detection systems.
In the context of drowning detection, AI can play a pivotal role in enhancing the accuracy, efficiency, and reliability of existing methods. Traditional drowning detection systems often rely on human surveillance or basic sensor readings, which can be prone to errors and delays. By leveraging AI algorithms and techniques, we can develop sophisticated systems that analyze data in real-time,

identify potential drowning incidents, and alert the relevant authorities or lifeguards promptly.
The integration of AI in drowning detection systems holds great potential for saving lives and minimizing the risks associa**ted with water-related activities. By combining advanced algorithms with sensor data, we can create intelligent systems that possess the ability to detect subtle patterns, recognize distress signals, and differentiate between normal water activities and emergencies. These systems can significantly reduce response times, enabling timely intervention and potentially preventing tragic outcomes.**

In the following chapters, we will delve into the specifics of how AI is integrated into our drowning detection system. We will explore the data collection process, the AI algorithms and techniques utilized, and the overall system architecture. Additionally, we will discuss the training and evaluation of our AI model, as well as the limitations and future directions of our research.
Through this exploration, we aim to shed light on the immense potential of AI in addressing critical challenges, such as drowning detection, and highlight how advancements in technology can contribute to the safety and well-being of individuals in aquatic environments.

6.2 AI in Drowning Detection Systems

Drowning incidents continue to be a significant concern worldwide, especially in aquatic environments such as pools, beaches, and lakes. Traditional methods of drowning detection heavily rely on human surveillance, which can be limited by factors such as distractions, fatigue, and human error. This is where AI comes into play, offering a promising solution to address the shortcomings of conventional approaches.
By integrating AI into drowning detection systems, we can achieve a paradigm shift in how we monitor and respond to water-related emergencies. Here are some key benefits of using AI in this domain:
Enhanced Accuracy: AI algorithms can analyze data from wearable devices, such

as smartwatches or wearable bands equipped with sensors, to detect potential signs of distress accurately. These devices can measure parameters like heart rate, movement patterns, and body orientation, providing valuable information for the AI system. By leveraging machine learning or deep learning techniques, the AI model can learn patterns associated with distress and drowning incidents, distinguishing them from regular water activities.

Improved Efficiency: Traditional methods of drowning detection often rely on manual monitoring or basic sensor readings, leading to delays in identifying emergencies. AI systems can continuously analyze data in real-time, significantly reducing response times. Wearable devices equipped with AI capabilities can autonomously monitor individuals' vital signs and movement patterns, quickly alerting lifeguards or triggering an emergency response system when necessary. This enhanced efficiency can be crucial in situations where immediate action is required to prevent drowning incidents.

Potential Impact on Saving Lives: The integration of AI in drowning detection systems has the potential to save lives by providing early and accurate alerts. By leveraging AI algorithms, these systems can quickly identify distress signals and provide immediate notifications to lifeguards or emergency services. The combination of wearable devices and AI can create a proactive safety net, helping prevent drowning incidents and enabling timely rescue efforts. This can significantly reduce the loss of life and the long-term impact on affected individuals and their families.

The utilization of wearable devices in AI-powered drowning detection systems offers a unique advantage. These devices can be comfortably worn by individuals engaging in water-related activities, making them unobtrusive and convenient. With the continuous monitoring capabilities provided by wearables, the AI system can provide real-time insights and adapt to changing circumstances, ensuring a high level of safety for swimmers, divers, or anyone near water bodies. By integrating AI into wearable device systems for drowning detection, we can harness the power of technology to create safer aquatic environments. The potential benefits of enhanced accuracy, improved efficiency, and the potential to save lives make AI an indispensable tool in addressing the pressing challenge of drowning incidents.

6.3 <u>Machine Learning</u>

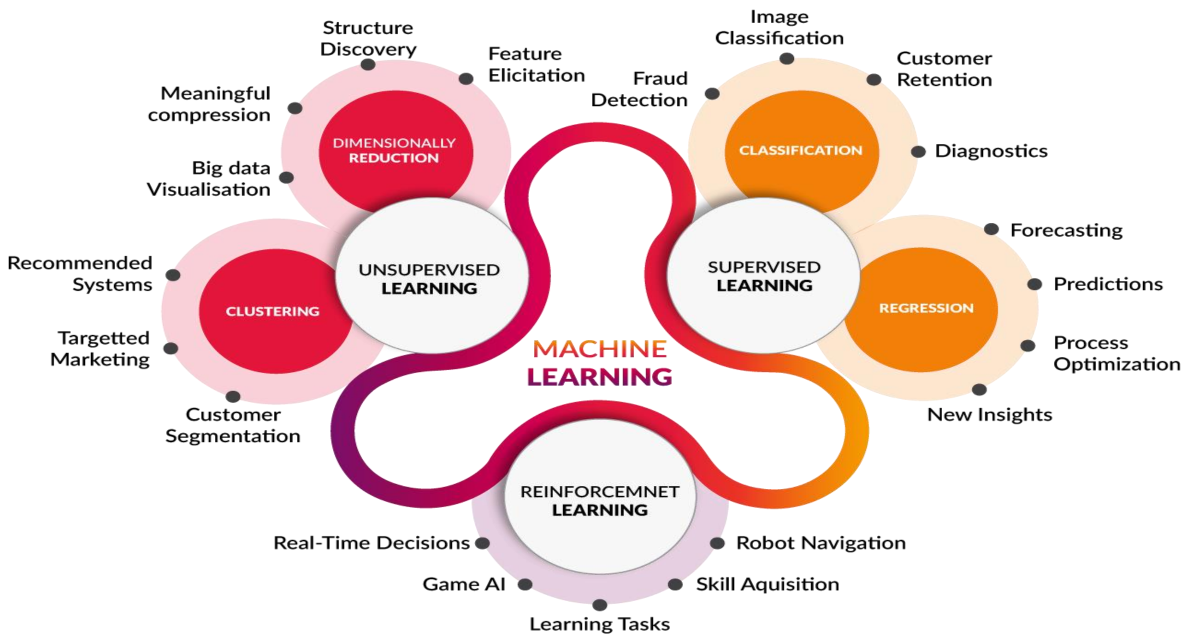


Figure 29 Machine Learning Branches

Machine Learning: Unleashing the Power of Data

6.3.1 <u>Introduction:</u>

Machine learning is an exciting field at the intersection of computer science and statistics that focuses on developing algorithms capable of automatically learning and making predictions or decisions without being explicitly programmed. It has revolutionized numerous industries and domains, enabling advancements in areas such as healthcare, finance, image recognition, natural language processing, and more.

6.3.2 <u>What is Machine Learning?</u>

Machine learning is a subset of artificial intelligence (AI) that empowers computers to learn from data and improve their performance over time.
It revolves around the idea of developing mathematical models or algorithms that automatically learn patterns and relationships in data, enabling the system to make accurate predictions or decisions.

6.3.3 <u>Supervised Learning:</u>

Supervised learning is a popular branch of machine learning where models are trained on labeled data, meaning that the desired output or target value is provided along with the input data.
The models learn from this labeled data to generalize and make predictions or decisions on unseen data.
Common algorithms used in supervised learning include decision trees, support vector machines (SVM), random forests, and gradient boosting.

6.3.4 <u>Unsupervised Learning:</u>

Unsupervised learning involves training models on unlabeled data, where the goal is to discover inherent patterns, structures, or clusters within the data.
This type of learning is useful for tasks such as data exploration, dimensionality reduction, and anomaly detection.
Clustering algorithms, such as k-means clustering and hierarchical clustering, are commonly used in unsupervised learning.
Applications of Machine Learning:

Machine learning has found applications in diverse fields, including healthcare, finance, marketing, autonomous vehicles, recommendation systems, and more.
It enables tasks such as image classification, speech recognition, natural language understanding, sentiment analysis, fraud detection, and personalized

recommendations.

6.3.5 Conclusion:

Machine learning has transformed the way we analyze and utilize data. By harnessing the power of algorithms capable of learning from data, we can uncover valuable insights, make accurate predictions, and automate complex tasks. With ongoing advancements, the potential for machine learning to revolutionize industries and improve various aspects of our lives is truly limitless.

6.3.6 Machine Learning in Drowning Detection Systems: (SVM Model)

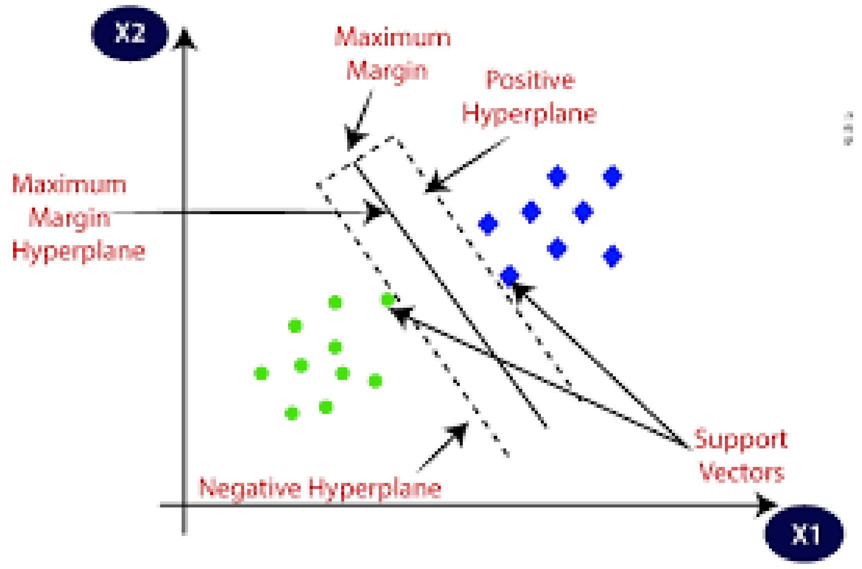


Figure 30 SVM Model

Machine **learning plays a crucial role in the development of AI systems, enabling them to learn from data and make intelligent predictions or decisions. In the context of drowning detection systems, machine learning algorithms can be employed to analyze sensor data and identify patterns associated with potential drowning incidents. One such algorithm that was utilized in our earlier work is the Support Vector Machine (SVM) model.**

The SVM model is a powerful and wid**ely used supervised learning algorithm, particularly effective for classification tasks. It is known for its ability to handle high-dimensional data and capture complex decision boundaries. In the context of drowning detection, the SVM model can be trained to**

distinguish between normal water activities and distress signals by learning from labeled data.

In our project, we employed the SVM model to analyze the sensor data collected from wearable devices. The SVM model was trained using a labeled dataset that consisted of instances representing both regular water activities and simulated distress scenarios. **Each instance in the dataset was associated with specific features extracted from the sensor readings, such as heart rate, movement patterns, and orientation.**

During the training phase, the SVM model learned to create a decision boundary that effectively **separates the different classes of instances. By finding the optimal hyperplane, the SVM model maximizes the margin between the instances of different classes, enabling robust classification. This ability to capture complex decision boundaries allows the SVM model to generalize well to unseen data, enhancing its predictive accuracy.**

After training the SVM model, we evaluated its performance using various metrics, including accuracy, precision, recall, and F1 score. These metrics provided insights into how **well the model could distinguish between normal water activities and potential drowning incidents. By achieving high accuracy and other favorable performance measures, the SVM model demonstrated its effectiveness in detecting distress signals and reducing false positives or false negatives.**

It is important to note that while SVM proved to be effective in our project, there are other machine learning algorithms that can also be explored for drowning detection systems. For example, ensemble methods like Ran**dom Forest or deep learning approaches such as Convolutional Neural Networks (CNNs) could also be considered based on the specific requirements and characteristics of the dataset.**

In conclusion, the use of machine learning, particularly the SVM model, in our earlier work on the drowning detection system allowed us to leverage the power of supervised learning to analyze sensor data and identify distress

signals. By training the SVM model on labeled data and evaluating its performance, we obtained a reliable and accurate classification framework for distinguishing between normal water activities and potential drowning incidents.

6.4 Deep Learning

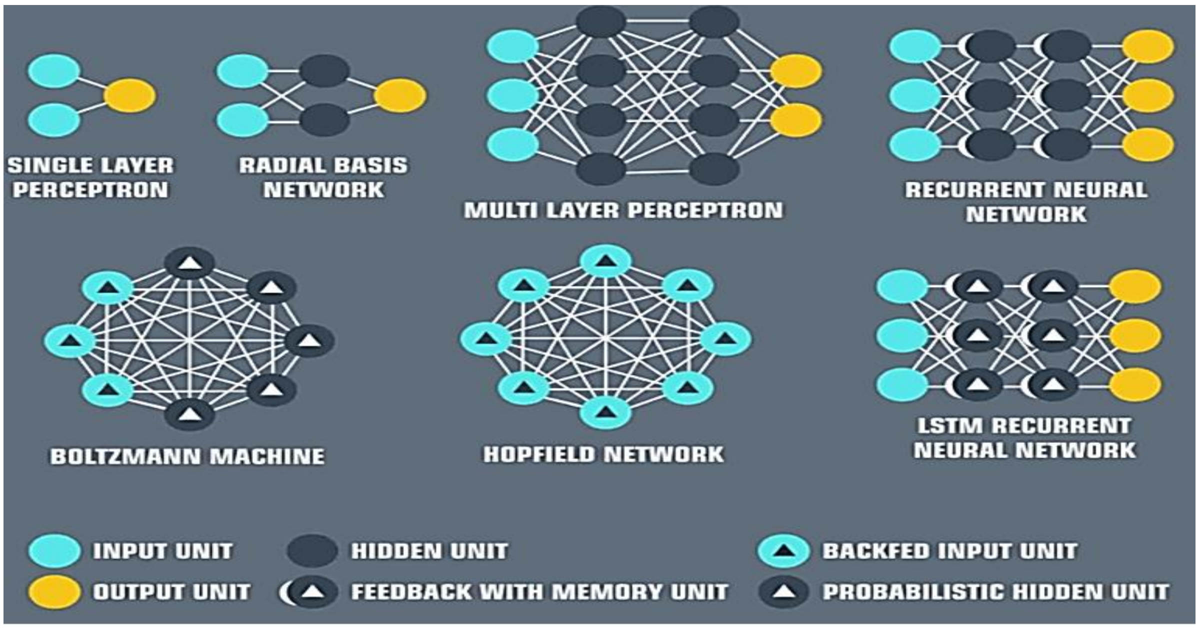


Figure 31 Neural Networks

Deep Learning: Unleashing the Power of Neural Networks

6.4.1 Introduction

Deep learning is a subfield of machine learning that focuses on training deep neural networks capable of learning intricate patterns and representations. It has gained significant attention due to its ability to handle complex tasks, such as image recognition, natural language processing, and speech synthesis.

6.4.2 Neural Networks and Deep Learning

Neural networks are the foundation of deep learning, inspired by the interconnected structure of the human brain.

They consist of an input layer, hidden layers, and an output layer, with each layer containing multiple neurons.

Deep learning models, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have demonstrated remarkable success in various domains.

6.4.3 Deep Learning: Capturing Complex Patterns

Deep learning models excel at capturing complex patterns and dependencies in data, making them suitable for tasks where traditional machine learning approaches may fall short.

The ability of deep neural networks to learn hierarchical representations allows them to automatically extract and understand intricate features from raw data.

Applications of Deep Learning:

Deep learning has found applications in diverse fields, including computer vision, natural language processing, speech recognition, recommendation systems, and more.

It enables tasks such as image classification, object detection, machine translation, sentiment analysis, voice synthesis, and personalized content recommendations.

6.4.4 Advancements in Deep Learning

Ongoing advancements in deep learning, fueled by increased computational power and large-scale datasets, have contributed to its success.
Techniques such as transfer learning, generative adversarial networks (GANs), and reinforcement learning have pushed the boundaries of what deep learning can achieve.

6.4.5 Conclusion

Deep learning has unleashed the power of neural networks, enabling us to tackle complex tasks and gain deeper insights from data. Its ability to capture intricate patterns and representations has revolutionized domains such as computer vision, natural language processing, and speech synthesis. With ongoing advancements and exploration, the potential for deep learning to reshape industries and unlock new possibilities continues to expand.

6.4.6 Deep Learning with LSTM for Drowning Detection in Time Series Sensor Data (LSTM Model)

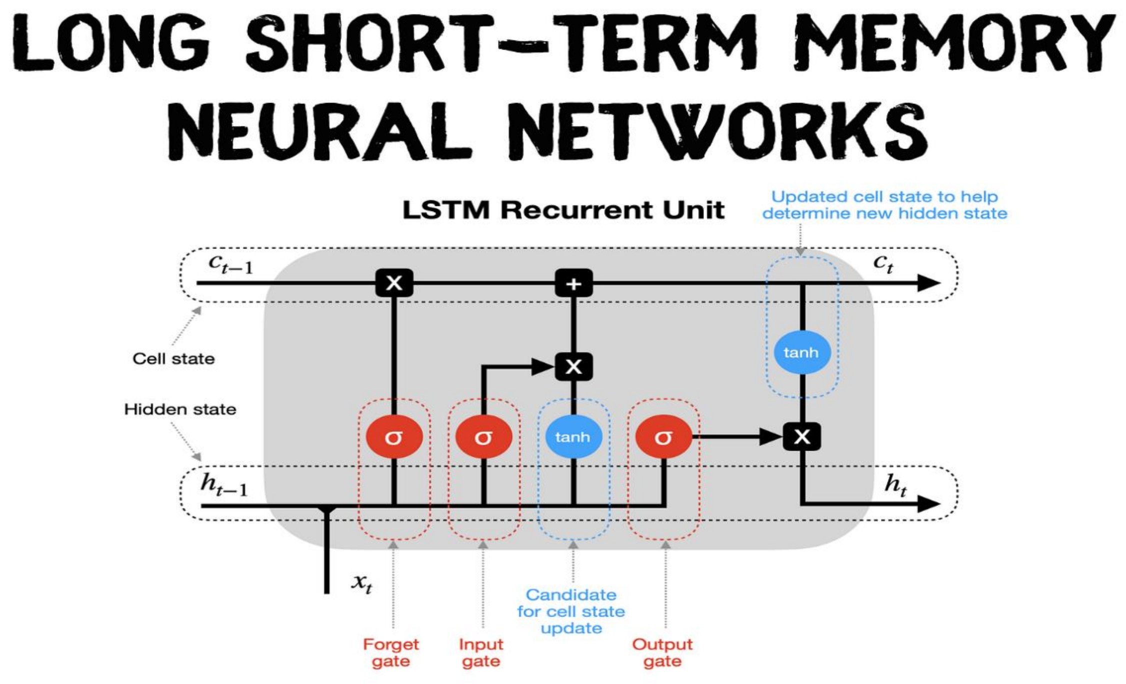

LONG SHORT-TERM MEMORY NEURAL NETWORKS
LSTM Recurrent Unit
Updated cell state to help determine new hidden state
c_{t-1}
c_t
Cell state
Hidden state
tanh
σ
σ
tanh
σ
h_{t-1}
h_t
x_t
Forget gate
Input gate
Candidate for cell state update
Output gate

Figure 32 LSTM Model

Deep learning techniques have gai**ned significant attention in recent years for their ability to effectively handle complex patterns and dependencies in data. In the context of drowning detection systems, deep learning models can be particularly useful for analyzing sensor data collected over time, as they can capture temporal relationships and detect subtle patterns that may indicate distress signals. One popular deep learning model for sequence data analysis is the Long Short-Term Memory (LSTM) model.**

LST**Ms are a type of recurrent neural network (RNN) that are designed to handle and learn from sequential or time series data. Unlike traditional feedforward neural networks, LSTMs can retain information over long sequences, making them suitable for modeling and predicting patterns in**

sensor data collected over time.

In our project, we leveraged the power of LSTM to analyze the time series sensor data obtained from wearable devices. The data c**ollected from various sensors, such as accelerometers or heart rate monitors, were organized as sequential inputs, with each time step corresponding to a specific measurement.**

The LSTM model was trained to learn the temporal dependencies within the sensor **data and identify patterns that indicate potential drowning incidents. By processing the data sequentially, the LSTM model can capture the dynamic relationships between sensor measurements at different time steps, enabling it to make accurate predictions about distress signals.**

During the training phase, the LSTM mode**l learned to understand the underlying patterns in the sensor data and create internal representations that encode relevant features for distinguishing between normal water activities and distress situations. By optimizing the model's parameters through backpropagation and gradient descent, the LSTM model improved its ability to accurately classify instances of drowning incidents.**

To evaluate the performance of the LSTM model, we used appropr**iate metrics such as accuracy, precision, recall, and F1 score. The LSTM model demonstrated superior performance compared to the machine learning models we had previously explored, such as SVM and Random Forest. Its ability to capture the temporal dependencies in the sensor data allowed for more accurate and reliable detection of distress signals, reducing false positives and false negatives.**

It is worth noting that the success of the LSTM model heavily relies on the availability of a sufficient amount of **labeled training data. The more diverse and representative the training data is, the better the LSTM model can generalize and accurately detect drowning incidents.**

In conclusion, the use of deep learning techniques, particularly the LSTM model, in analyzi**ng time series sensor data for drowning detection resulted in improved accuracy compared to traditional machine learning models. The LSTM model's ability to capture temporal dependencies and recognize complex patterns makes it a powerful tool for identifying distress signals in the context of drowning incidents.**

6.5 Detection System Architecture

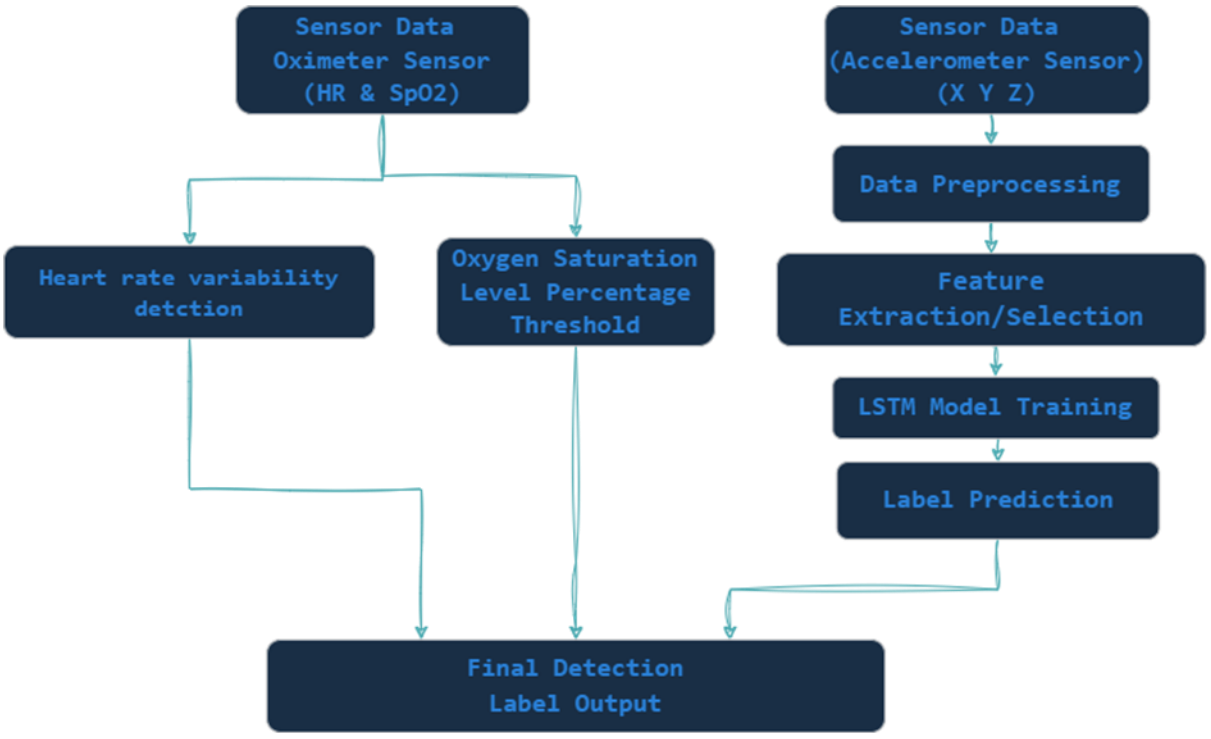


Figure 33 System Architecture Diagram

Explanation of the flowchart:

- Sensor Data: The system collects data from various sensors, including accelerometer, heart rate, and SpO2 (oxygen saturation level) sensors.

- Data Preprocessing: The collected sensor data undergoes preprocessing, which may include steps such as data cleaning, normalization, and handling missing values.

- Feature Extraction/Selection: Relevant features are extracted or selected from the preprocessed data. This step helps in representing the data effectively for the subsequent LSTM model.

- LSTM Model Training: The preprocessed and selected features are used to train the LSTM model. The LSTM model learns the patterns and dependencies in the data to make accurate predictions.

- Label Prediction: The trained LSTM model takes the processed sensor data as input and predicts the label corresponding to the activity state, such as Safe, Active Drowning, or Passive Drowning.

- Drowning Label Output: The system outputs the predicted label based on the LSTM model's prediction. This information can be used to trigger appropriate actions, such as sending alerts to lifeguards or initiating emergency response protocols.

The flowchart illustrates the overall system architecture for the detection system using AI and the LSTM model. It showcases the flow of data from the sensors through preprocessing, feature extraction/selection, model training, and ultimately predicting the drowning label based on the collected sensor data.

6.6 <u>Implementation</u>

6.6.1 <u>Concept</u>

The primary objective of the detection system in this project is to accurately identify the activity state of a person in water, distinguishing between Safe, Active Drowning, and Passive Drowning scenarios. The system utilizes an LSTM model, a type of deep learning model specifically designed for analyzing sequential or time series data.

i. LSTM Model for Activity Detection:
   The core component of the detection system is the LSTM model. It is trained on a labeled dataset consisting of sensor data collected from the accelerometer sensor (measuring movement along x, y, and z axes) and the combined heart rate and oxygen sensor (oximeter sensor). The LSTM model learns patterns and dependencies in the sequential sensor data to predict the activity state accurately.

ii. Detection of Activity Labels:
   The LSTM model takes the sequential sensor data as input and outputs a predicted label for the person's activity state. The labels include Safe, Active Drowning, and Passive Drowning. By learning from the patterns in the data, the LSTM model can differentiate between different activities and identify potential drowning incidents.

iii. Heart Rate Variability (HRV) Function for Safe Label:
   To ensure the Safe label, the system incorporates a function to detect heart rate variability. The HRV function analyzes the heart rate data obtained from the combined heart rate and oxygen sensor. By examining the variability in the intervals between consecutive heartbeats, the system can assess the person's cardiac health and determine if their heart rate falls within a safe range. If the heart rate variability is within the acceptable range, the system assigns the Safe label to the activity.
   {normal_heart_rate_range = (60, 100) - normal_hrv_range = (0.1, 0.3)}

iv. Oxygen Saturation Level (SpO2) Threshold:

The combined heart rate and oxygen sensor (oximeter sensor) also provide data on the person's oxygen saturation level (SpO2). To identify potentially hazardous scenarios, the system sets a threshold for the SpO2 level. If the SpO2 falls below the defined threshold (< 90 %), it indicates a potential oxygen deficiency, which may be a sign of drowning or distress. In such cases, the system predicts either Active Drowning or Passive Drowning as the activity label.

By combining the predictions from the LSTM model, heart rate variability analysis, and oxygen saturation level threshold, the detection system can accurately classify the activity state of a person in water. The system can provide real-time monitoring and generate alerts or trigger appropriate actions when potentially dangerous situations are detected, helping to prevent drowning incidents and ensure swift response and assistance.

It's important to note that the system's accuracy and effectiveness depend on the quality and reliability of the sensor data, the training of the LSTM model, and the calibration of the heart rate variability function and oxygen saturation level threshold. Continuous validation, refinement, and optimization are essential to improve the system's performance and enhance its ability to detect and respond to drowning incidents accurately.

6.6.2 <u>Coding:</u>

Accelerometer Sensor:

Dataset Description[8]:

Graphs identifying data:

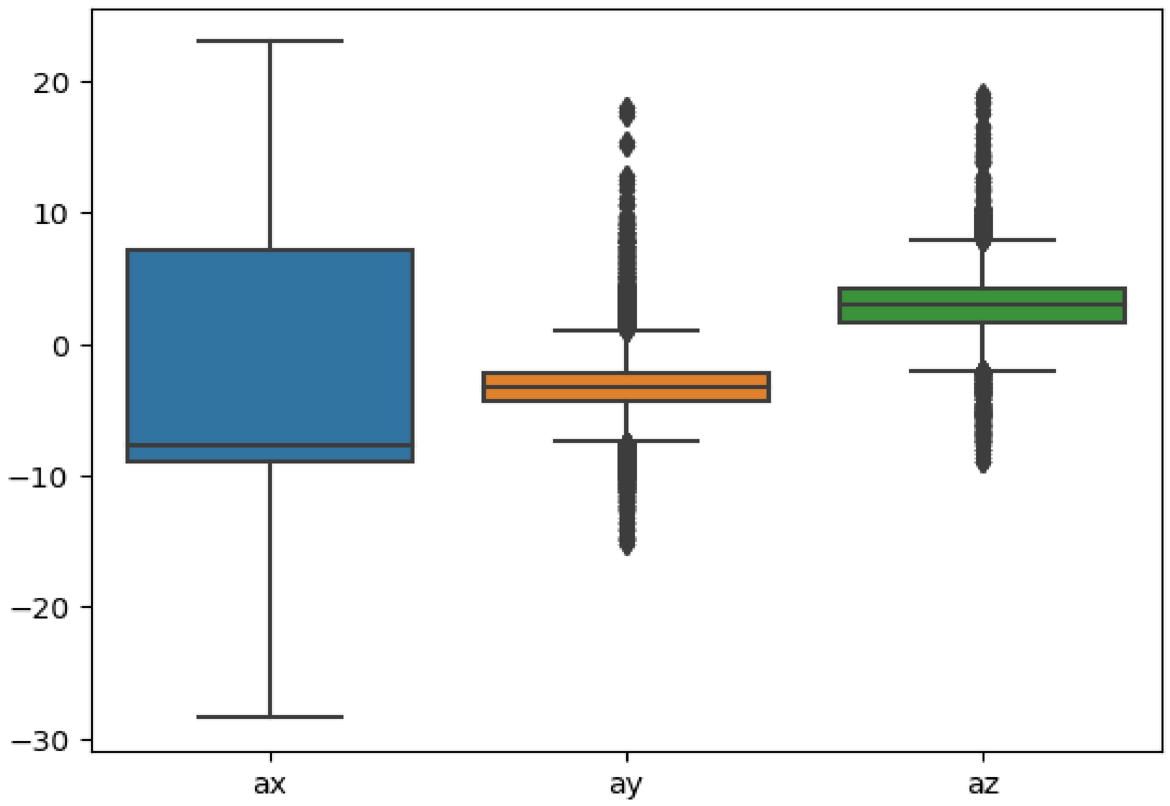


Figure 34 Safe Swimming Data

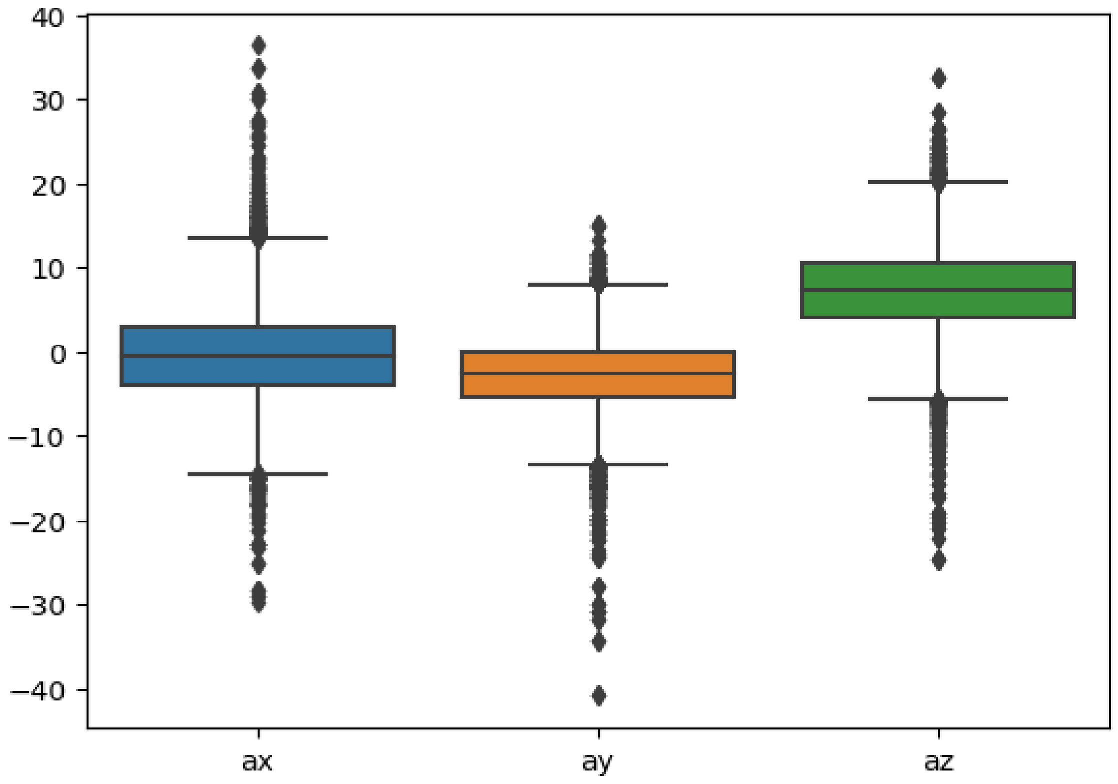


Figure 35 Active Drowning Data

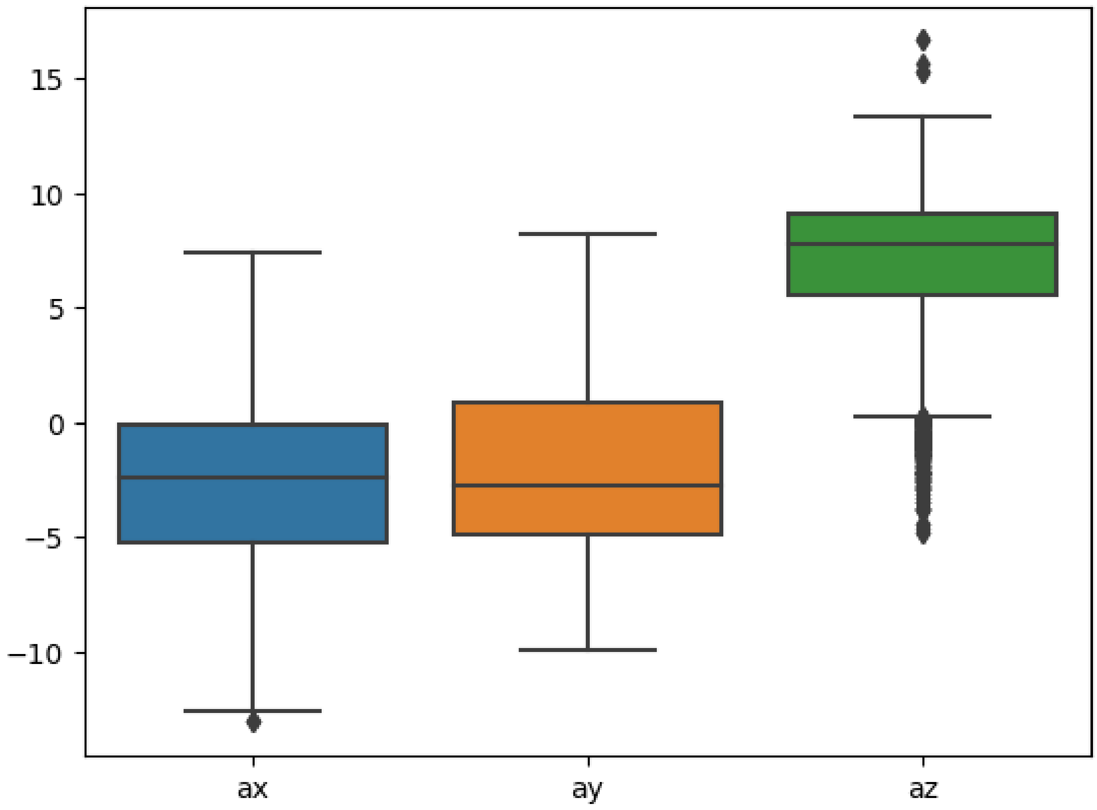
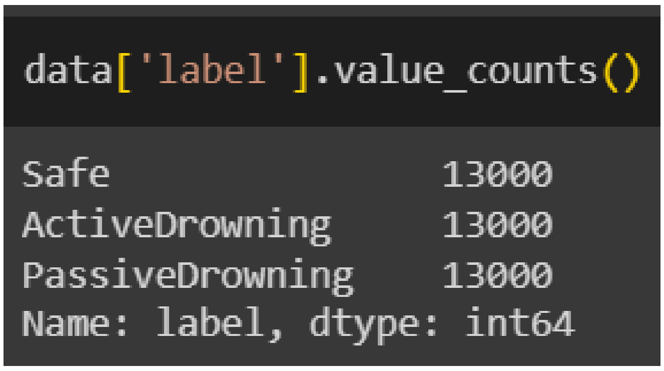


Figure 36 Passive Drowning Data

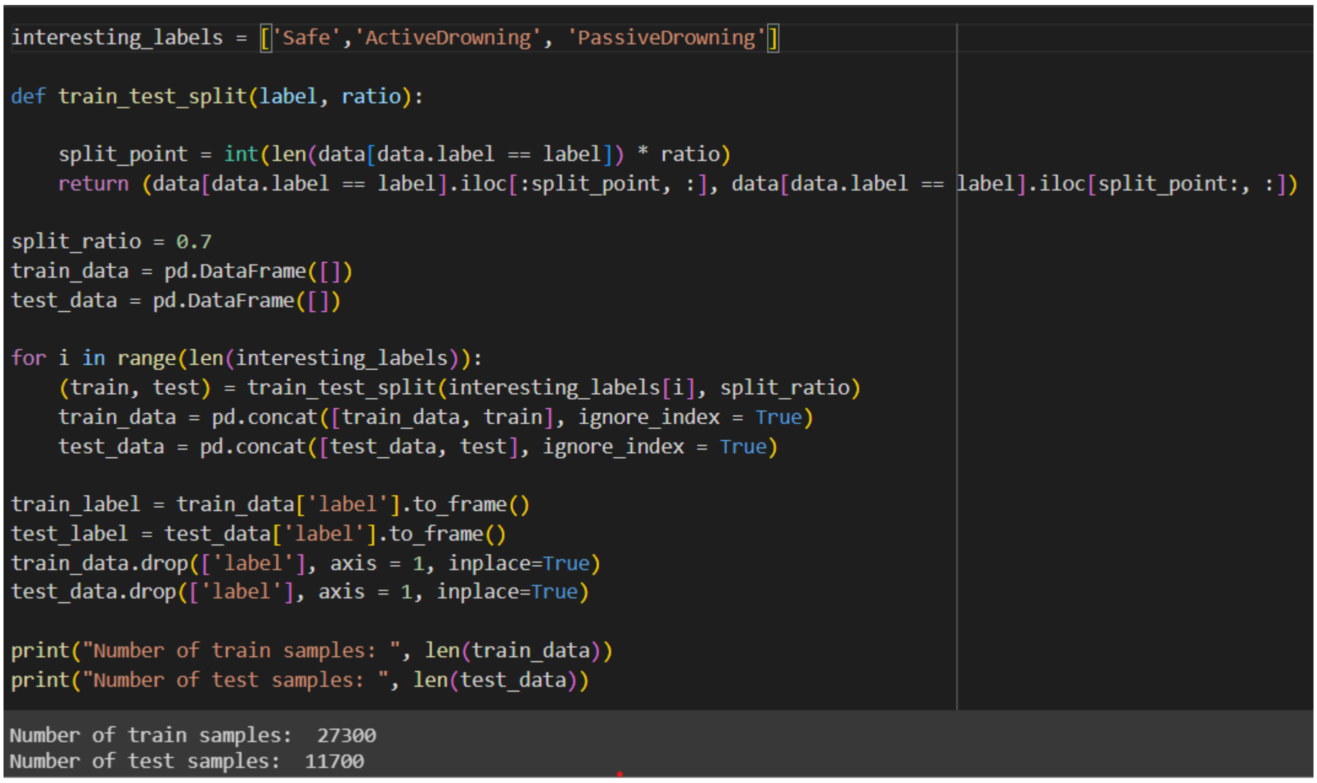
Data Labels Counts:



```
data['label'].value_counts()
```

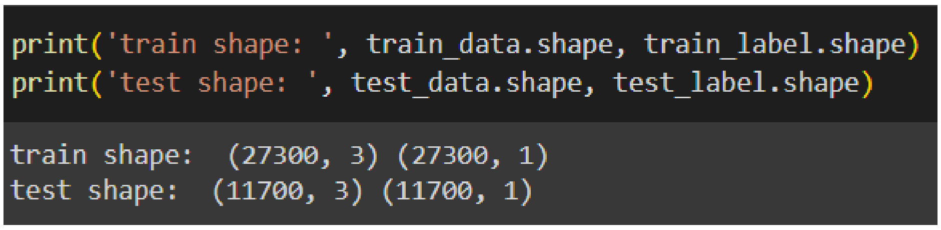
```
Safe                13000
ActiveDrowning      13000
PassiveDrowning     13000
Name: label, dtype: int64
```

Data Splitting:



```
interesting_labels = ['Safe','ActiveDrowning', 'PassiveDrowning']

def train_test_split(label, ratio):

    split_point = int(len(data[data.label == label]) * ratio)
    return (data[data.label == label].iloc[:split_point, :], data[data.label == label].iloc[split_point:, :])

split_ratio = 0.7
train_data = pd.DataFrame([])
test_data = pd.DataFrame([])

for i in range(len(interesting_labels)):
    (train, test) = train_test_split(interesting_labels[i], split_ratio)
    train_data = pd.concat([train_data, train], ignore_index = True)
    test_data = pd.concat([test_data, test], ignore_index = True)

train_label = train_data['label'].to_frame()
test_label = test_data['label'].to_frame()
train_data.drop(['label'], axis = 1, inplace=True)
test_data.drop(['label'], axis = 1, inplace=True)

print("Number of train samples: ", len(train_data))
print("Number of test samples: ", len(test_data))
```

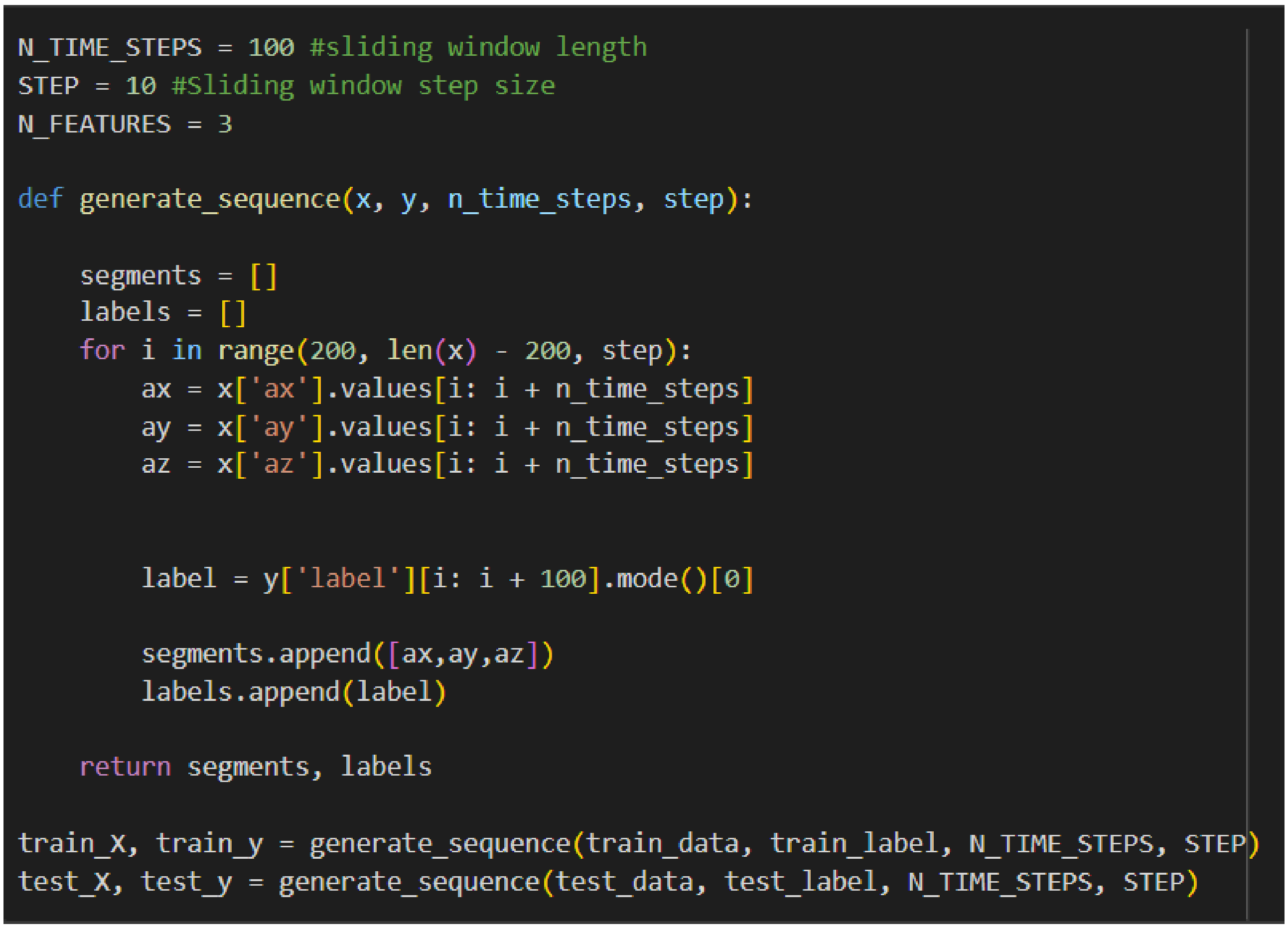
```
Number of train samples:  27300
Number of test samples:  11700
```



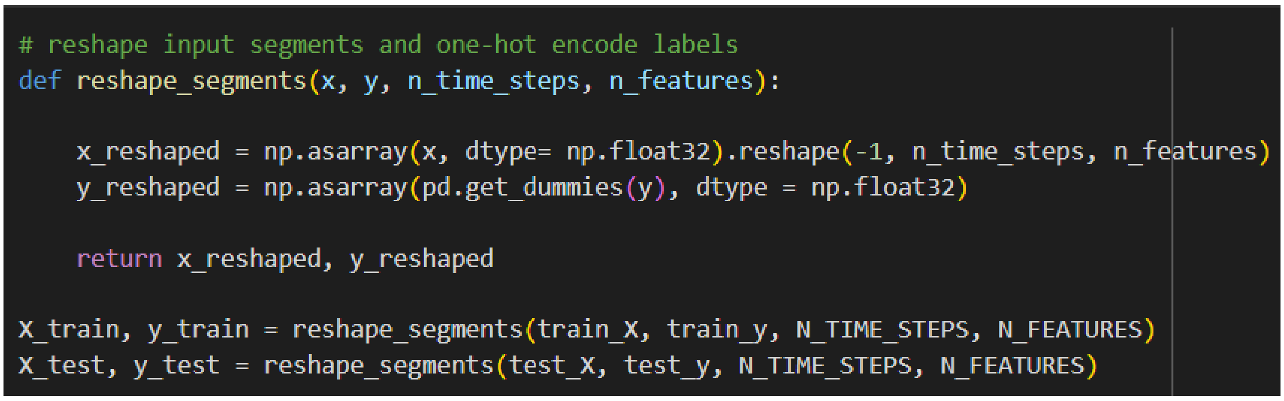
```
print('train shape: ', train_data.shape, train_label.shape)
print('test shape: ', test_data.shape, test_label.shape)
```

```
train shape:  (27300, 3) (27300, 1)
test shape:  (11700, 3) (11700, 1)
```

Data Segmentation:

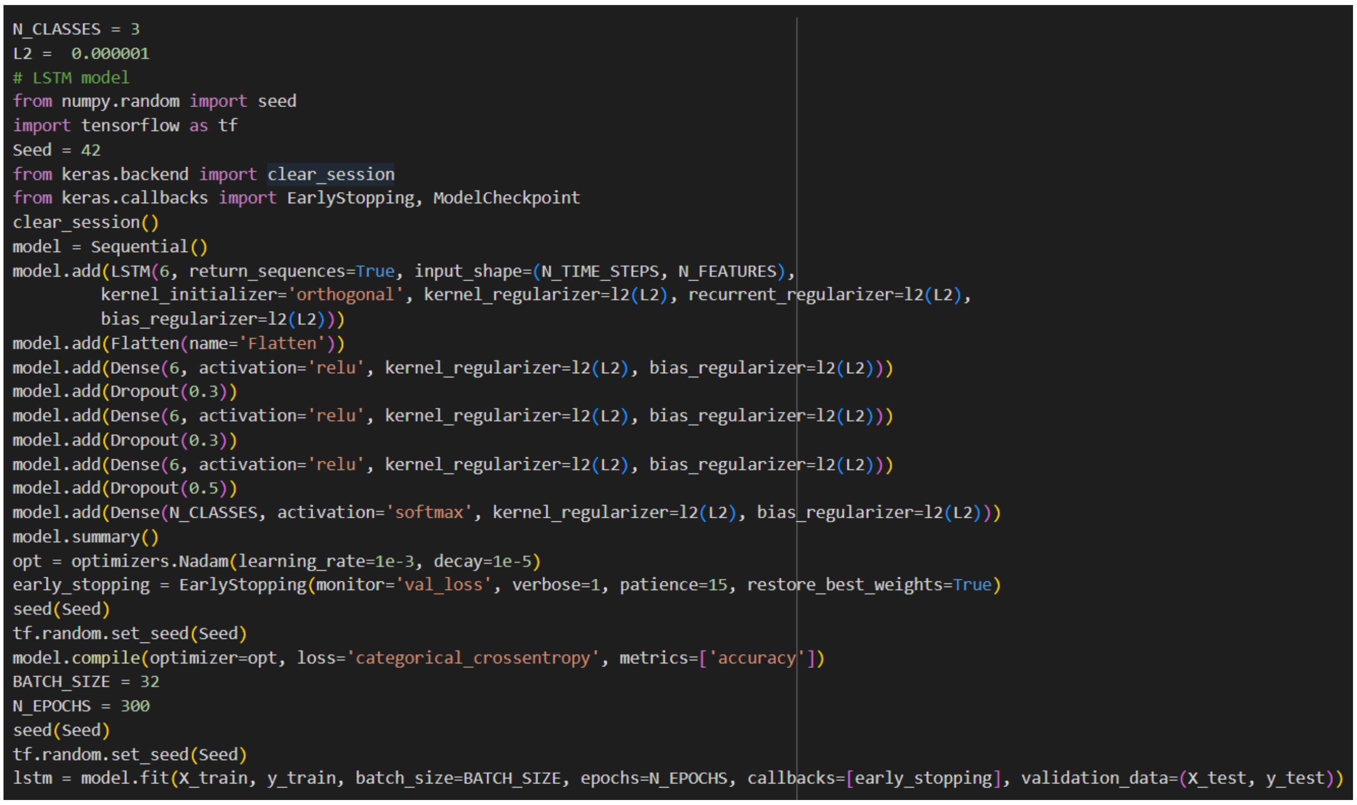


```
N_TIME_STEPS = 100 #sliding window length
STEP = 10 #Sliding window step size
N_FEATURES = 3

def generate_sequence(x, y, n_time_steps, step):

    segments = []
    labels = []
    for i in range(200, len(x) - 200, step):
        ax = x['ax'].values[i: i + n_time_steps]
        ay = x['ay'].values[i: i + n_time_steps]
        az = x['az'].values[i: i + n_time_steps]


        label = y['label'][i: i + 100].mode()[0]

        segments.append([ax,ay,az])
        labels.append(label)

    return segments, labels

train_X, train_y = generate_sequence(train_data, train_label, N_TIME_STEPS, STEP)
test_X, test_y = generate_sequence(test_data, test_label, N_TIME_STEPS, STEP)
```



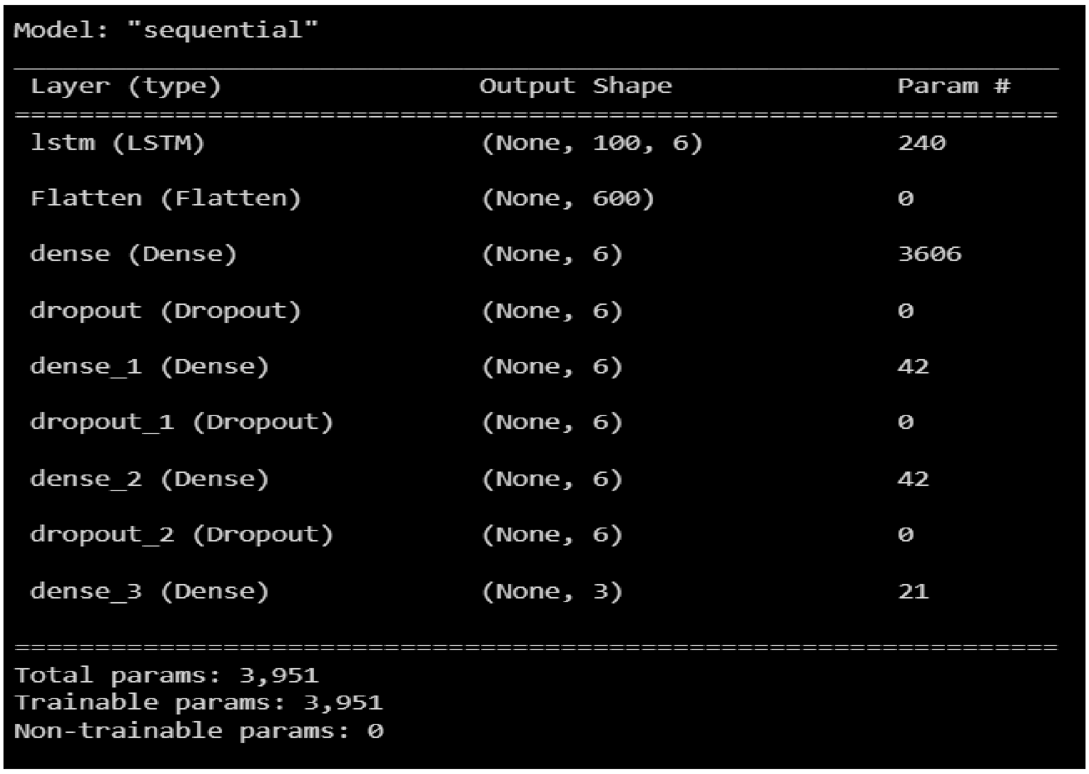
```
# reshape input segments and one-hot encode labels
def reshape_segments(x, y, n_time_steps, n_features):

    x_reshaped = np.asarray(x, dtype= np.float32).reshape(-1, n_time_steps, n_features)
    y_reshaped = np.asarray(pd.get_dummies(y), dtype = np.float32)

    return x_reshaped, y_reshaped

X_train, y_train = reshape_segments(train_X, train_y, N_TIME_STEPS, N_FEATURES)
X_test, y_test = reshape_segments(test_X, test_y, N_TIME_STEPS, N_FEATURES)
```

Model Architecture:

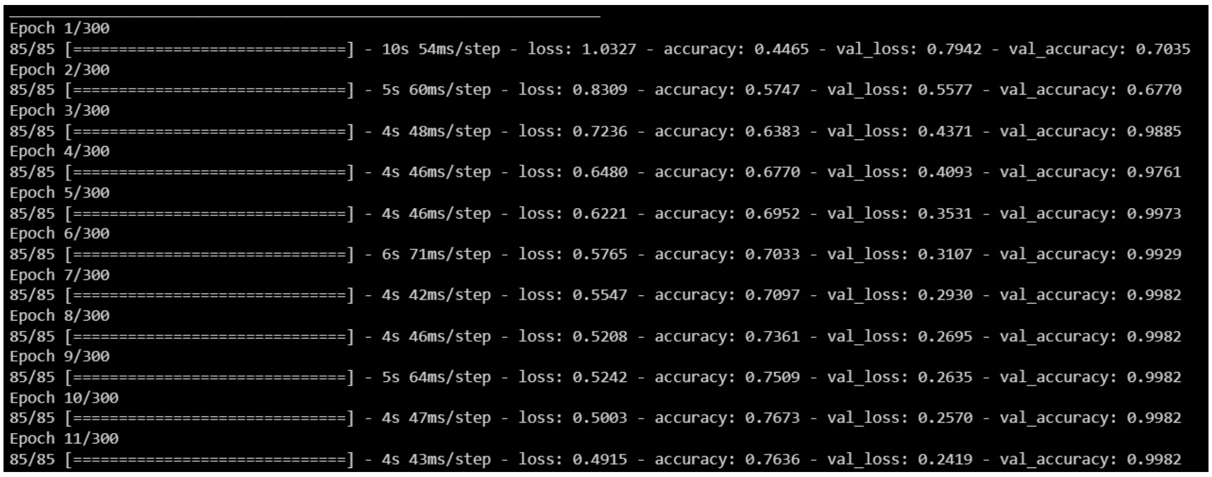


```
N_CLASSES = 3
L2 =  0.000001
# LSTM model
from numpy.random import seed
import tensorflow as tf
Seed = 42
from keras.backend import clear_session
from keras.callbacks import EarlyStopping, ModelCheckpoint
clear_session()
model = Sequential()
model.add(LSTM(6, return_sequences=True, input_shape=(N_TIME_STEPS, N_FEATURES),
         kernel_initializer='orthogonal', kernel_regularizer=l2(L2), recurrent_regularizer=l2(L2),
         bias_regularizer=l2(L2)))
model.add(Flatten(name='Flatten'))
model.add(Dense(6, activation='relu', kernel_regularizer=l2(L2), bias_regularizer=l2(L2)))
model.add(Dropout(0.3))
model.add(Dense(6, activation='relu', kernel_regularizer=l2(L2), bias_regularizer=l2(L2)))
model.add(Dropout(0.3))
model.add(Dense(6, activation='relu', kernel_regularizer=l2(L2), bias_regularizer=l2(L2)))
model.add(Dropout(0.5))
model.add(Dense(N_CLASSES, activation='softmax', kernel_regularizer=l2(L2), bias_regularizer=l2(L2)))
model.summary()
opt = optimizers.Nadam(learning_rate=1e-3, decay=1e-5)
early_stopping = EarlyStopping(monitor='val_loss', verbose=1, patience=15, restore_best_weights=True)
seed(Seed)
tf.random.set_seed(Seed)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
BATCH_SIZE = 32
N_EPOCHS = 300
seed(Seed)
tf.random.set_seed(Seed)
lstm = model.fit(X_train, y_train, batch_size=BATCH_SIZE, epochs=N_EPOCHS, callbacks=[early_stopping], validation_data=(X_test, y_test))
```

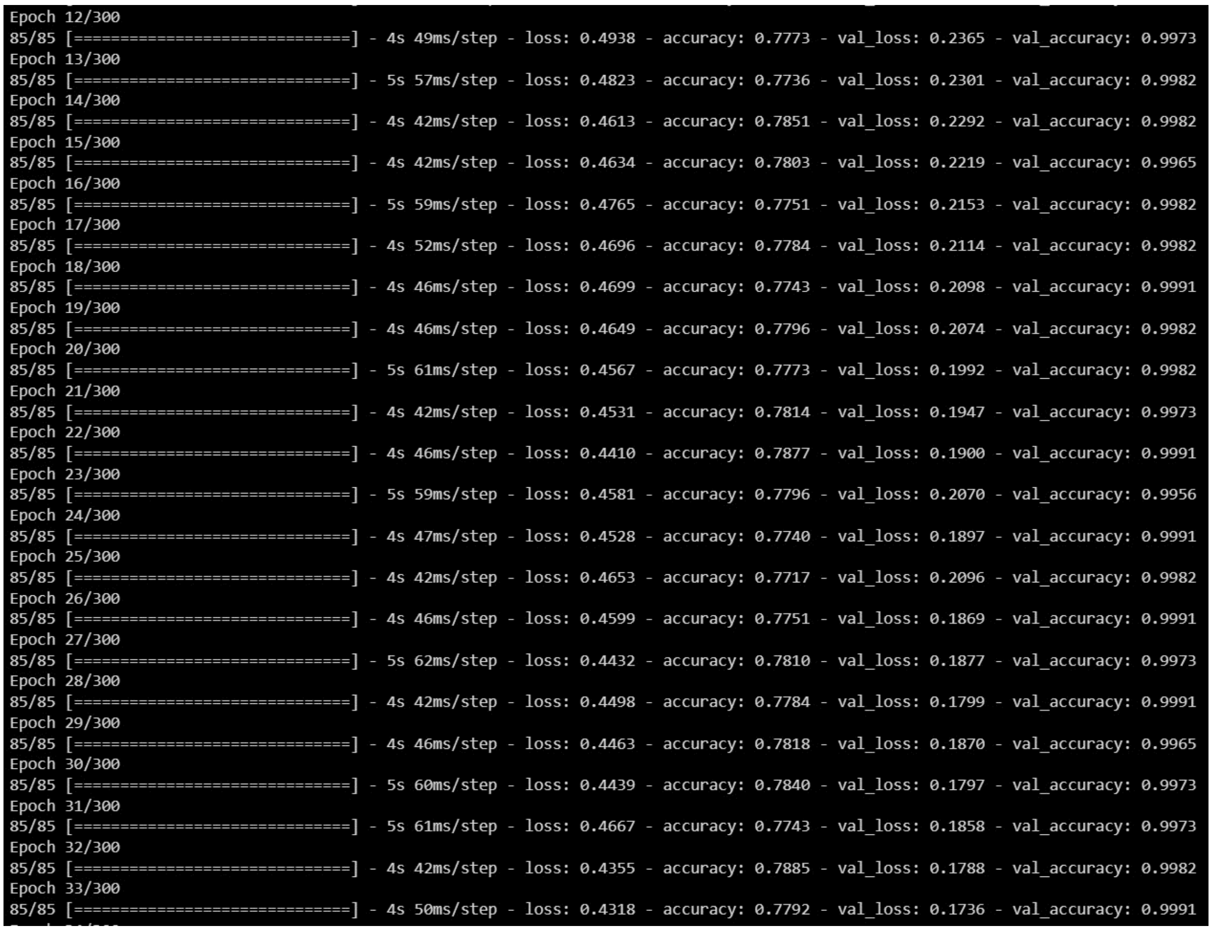
Model Results:



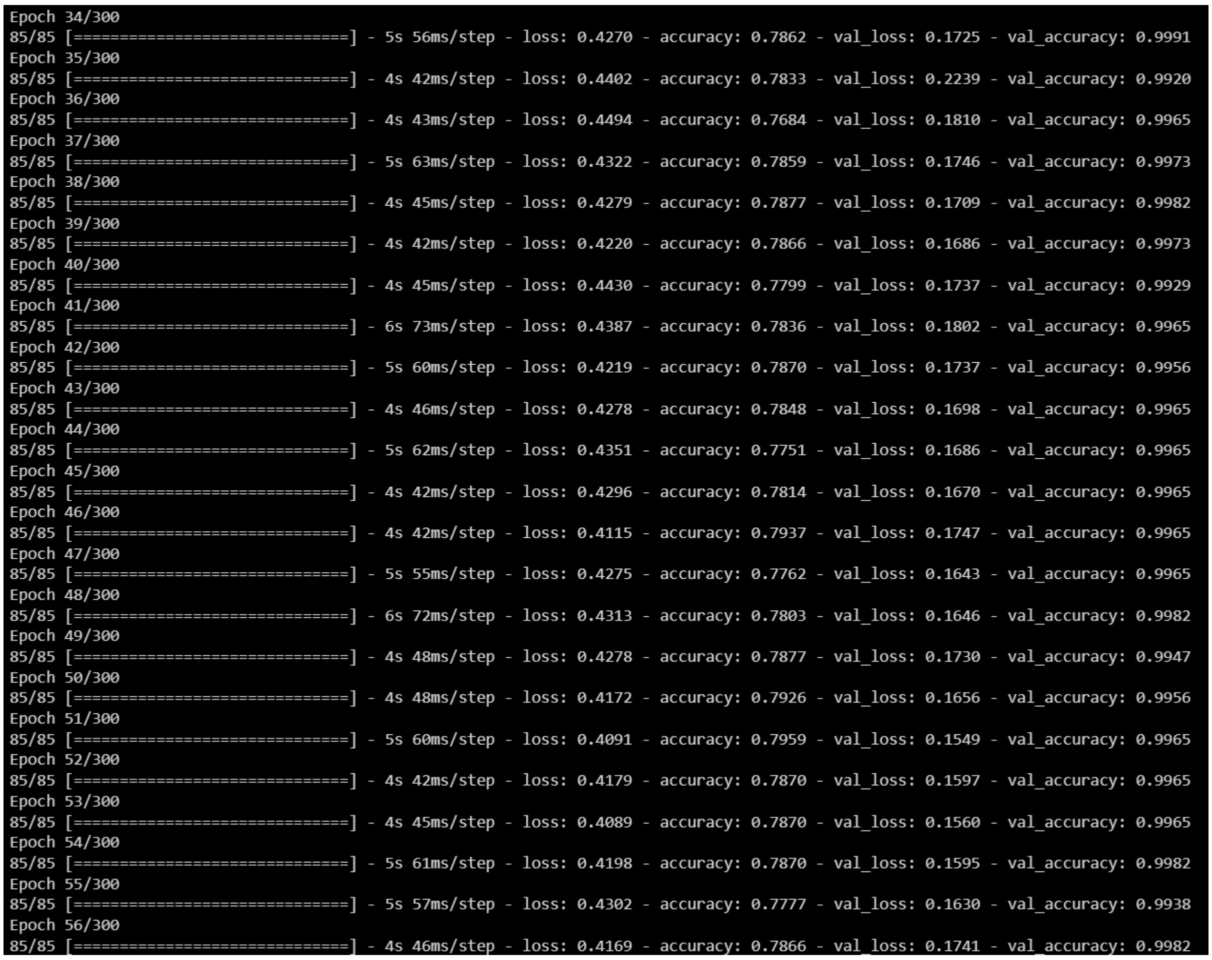
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 6)            240

 Flatten (Flatten)           (None, 600)               0

 dense (Dense)               (None, 6)                 3606

 dropout (Dropout)           (None, 6)                 0

 dense_1 (Dense)             (None, 6)                 42

 dropout_1 (Dropout)         (None, 6)                 0

 dense_2 (Dense)             (None, 6)                 42

 dropout_2 (Dropout)         (None, 6)                 0

 dense_3 (Dense)             (None, 3)                 21

=================================================================
Total params: 3,951
Trainable params: 3,951
Non-trainable params: 0
```



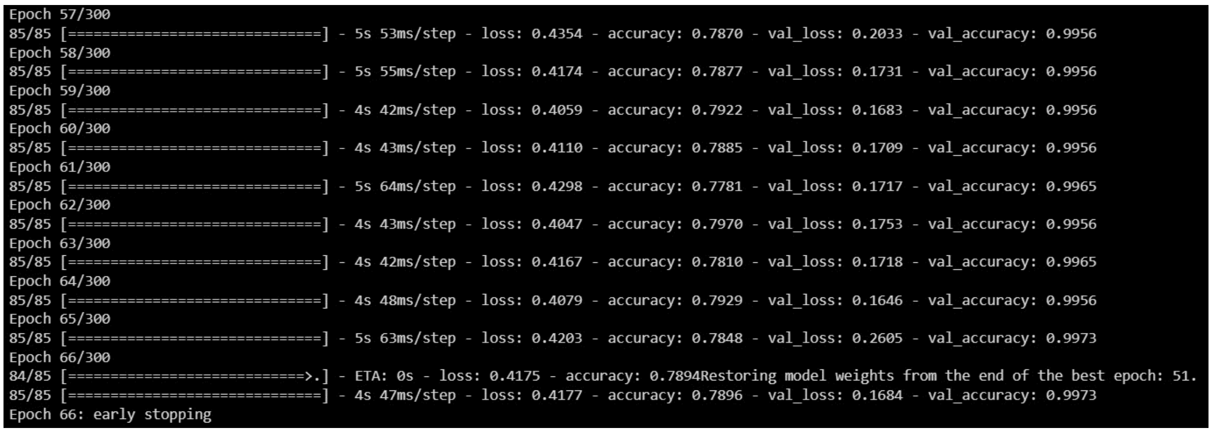
```
Epoch 1/300
85/85 [==============================] - 10s 54ms/step - loss: 1.0327 - accuracy: 0.4465 - val_loss: 0.7942 - val_accuracy: 0.7035
Epoch 2/300
85/85 [==============================] - 5s 60ms/step - loss: 0.8309 - accuracy: 0.5747 - val_loss: 0.5577 - val_accuracy: 0.6770
Epoch 3/300
85/85 [==============================] - 4s 48ms/step - loss: 0.7236 - accuracy: 0.6383 - val_loss: 0.4371 - val_accuracy: 0.9885
Epoch 4/300
85/85 [==============================] - 4s 46ms/step - loss: 0.6480 - accuracy: 0.6770 - val_loss: 0.4093 - val_accuracy: 0.9761
Epoch 5/300
85/85 [==============================] - 4s 46ms/step - loss: 0.6221 - accuracy: 0.6952 - val_loss: 0.3531 - val_accuracy: 0.9973
Epoch 6/300
85/85 [==============================] - 6s 71ms/step - loss: 0.5765 - accuracy: 0.7033 - val_loss: 0.3107 - val_accuracy: 0.9929
Epoch 7/300
85/85 [==============================] - 4s 42ms/step - loss: 0.5547 - accuracy: 0.7097 - val_loss: 0.2930 - val_accuracy: 0.9982
Epoch 8/300
85/85 [==============================] - 4s 46ms/step - loss: 0.5208 - accuracy: 0.7361 - val_loss: 0.2695 - val_accuracy: 0.9982
Epoch 9/300
85/85 [==============================] - 5s 64ms/step - loss: 0.5242 - accuracy: 0.7509 - val_loss: 0.2635 - val_accuracy: 0.9982
Epoch 10/300
85/85 [==============================] - 4s 47ms/step - loss: 0.5003 - accuracy: 0.7673 - val_loss: 0.2570 - val_accuracy: 0.9982
Epoch 11/300
85/85 [==============================] - 4s 43ms/step - loss: 0.4915 - accuracy: 0.7636 - val_loss: 0.2419 - val_accuracy: 0.9982
```



```
Epoch 12/300
85/85 [==============================] - 4s 49ms/step - loss: 0.4938 - accuracy: 0.7773 - val_loss: 0.2365 - val_accuracy: 0.9973
Epoch 13/300
85/85 [==============================] - 5s 57ms/step - loss: 0.4823 - accuracy: 0.7736 - val_loss: 0.2301 - val_accuracy: 0.9982
Epoch 14/300
85/85 [==============================] - 4s 42ms/step - loss: 0.4613 - accuracy: 0.7851 - val_loss: 0.2292 - val_accuracy: 0.9982
Epoch 15/300
85/85 [==============================] - 4s 42ms/step - loss: 0.4634 - accuracy: 0.7803 - val_loss: 0.2219 - val_accuracy: 0.9965
Epoch 16/300
85/85 [==============================] - 5s 59ms/step - loss: 0.4765 - accuracy: 0.7751 - val_loss: 0.2153 - val_accuracy: 0.9982
Epoch 17/300
85/85 [==============================] - 4s 52ms/step - loss: 0.4696 - accuracy: 0.7784 - val_loss: 0.2114 - val_accuracy: 0.9982
Epoch 18/300
85/85 [==============================] - 4s 46ms/step - loss: 0.4699 - accuracy: 0.7743 - val_loss: 0.2098 - val_accuracy: 0.9991
Epoch 19/300
85/85 [==============================] - 4s 46ms/step - loss: 0.4649 - accuracy: 0.7796 - val_loss: 0.2074 - val_accuracy: 0.9982
Epoch 20/300
85/85 [==============================] - 5s 61ms/step - loss: 0.4567 - accuracy: 0.7773 - val_loss: 0.1992 - val_accuracy: 0.9982
Epoch 21/300
85/85 [==============================] - 4s 42ms/step - loss: 0.4531 - accuracy: 0.7814 - val_loss: 0.1947 - val_accuracy: 0.9973
Epoch 22/300
85/85 [==============================] - 4s 46ms/step - loss: 0.4410 - accuracy: 0.7877 - val_loss: 0.1900 - val_accuracy: 0.9991
Epoch 23/300
85/85 [==============================] - 5s 59ms/step - loss: 0.4581 - accuracy: 0.7796 - val_loss: 0.2070 - val_accuracy: 0.9956
Epoch 24/300
85/85 [==============================] - 4s 47ms/step - loss: 0.4528 - accuracy: 0.7740 - val_loss: 0.1897 - val_accuracy: 0.9991
Epoch 25/300
85/85 [==============================] - 4s 42ms/step - loss: 0.4653 - accuracy: 0.7717 - val_loss: 0.2096 - val_accuracy: 0.9982
Epoch 26/300
85/85 [==============================] - 4s 46ms/step - loss: 0.4599 - accuracy: 0.7751 - val_loss: 0.1869 - val_accuracy: 0.9991
Epoch 27/300
85/85 [==============================] - 5s 62ms/step - loss: 0.4432 - accuracy: 0.7810 - val_loss: 0.1877 - val_accuracy: 0.9973
Epoch 28/300
85/85 [==============================] - 4s 42ms/step - loss: 0.4498 - accuracy: 0.7784 - val_loss: 0.1799 - val_accuracy: 0.9991
Epoch 29/300
85/85 [==============================] - 4s 46ms/step - loss: 0.4463 - accuracy: 0.7818 - val_loss: 0.1870 - val_accuracy: 0.9965
Epoch 30/300
85/85 [==============================] - 5s 60ms/step - loss: 0.4439 - accuracy: 0.7840 - val_loss: 0.1797 - val_accuracy: 0.9973
Epoch 31/300
85/85 [==============================] - 5s 61ms/step - loss: 0.4667 - accuracy: 0.7743 - val_loss: 0.1858 - val_accuracy: 0.9973
Epoch 32/300
85/85 [==============================] - 4s 42ms/step - loss: 0.4355 - accuracy: 0.7885 - val_loss: 0.1788 - val_accuracy: 0.9982
Epoch 33/300
85/85 [==============================] - 4s 50ms/step - loss: 0.4318 - accuracy: 0.7792 - val_loss: 0.1736 - val_accuracy: 0.9991
```

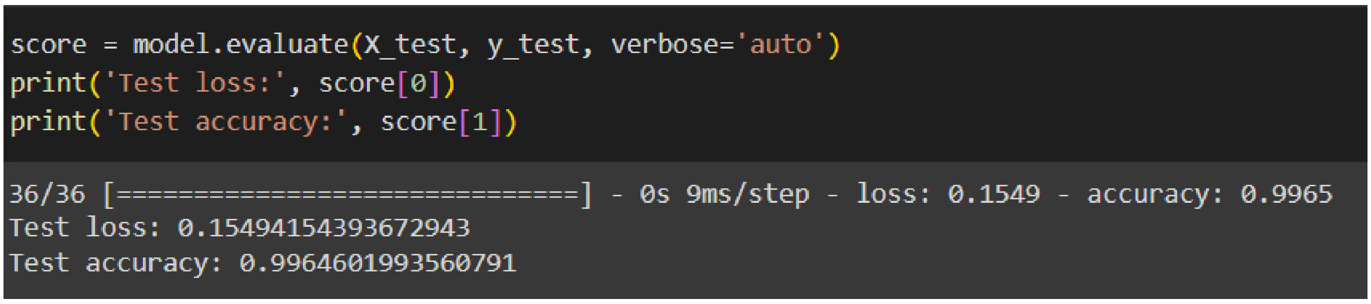


```
Epoch 34/300
85/85 [==============================] - 5s 56ms/step - loss: 0.4270 - accuracy: 0.7862 - val_loss: 0.1725 - val_accuracy: 0.9991
Epoch 35/300
85/85 [==============================] - 4s 42ms/step - loss: 0.4402 - accuracy: 0.7833 - val_loss: 0.2239 - val_accuracy: 0.9920
Epoch 36/300
85/85 [==============================] - 4s 43ms/step - loss: 0.4494 - accuracy: 0.7684 - val_loss: 0.1810 - val_accuracy: 0.9965
Epoch 37/300
85/85 [==============================] - 5s 63ms/step - loss: 0.4322 - accuracy: 0.7859 - val_loss: 0.1746 - val_accuracy: 0.9973
Epoch 38/300
85/85 [==============================] - 4s 45ms/step - loss: 0.4279 - accuracy: 0.7877 - val_loss: 0.1709 - val_accuracy: 0.9982
Epoch 39/300
85/85 [==============================] - 4s 42ms/step - loss: 0.4220 - accuracy: 0.7866 - val_loss: 0.1686 - val_accuracy: 0.9973
Epoch 40/300
85/85 [==============================] - 4s 45ms/step - loss: 0.4430 - accuracy: 0.7799 - val_loss: 0.1737 - val_accuracy: 0.9929
Epoch 41/300
85/85 [==============================] - 6s 73ms/step - loss: 0.4387 - accuracy: 0.7836 - val_loss: 0.1802 - val_accuracy: 0.9965
Epoch 42/300
85/85 [==============================] - 5s 60ms/step - loss: 0.4219 - accuracy: 0.7870 - val_loss: 0.1737 - val_accuracy: 0.9956
Epoch 43/300
85/85 [==============================] - 4s 46ms/step - loss: 0.4278 - accuracy: 0.7848 - val_loss: 0.1698 - val_accuracy: 0.9965
Epoch 44/300
85/85 [==============================] - 5s 62ms/step - loss: 0.4351 - accuracy: 0.7751 - val_loss: 0.1686 - val_accuracy: 0.9965
Epoch 45/300
85/85 [==============================] - 4s 42ms/step - loss: 0.4296 - accuracy: 0.7814 - val_loss: 0.1670 - val_accuracy: 0.9965
Epoch 46/300
85/85 [==============================] - 4s 42ms/step - loss: 0.4115 - accuracy: 0.7937 - val_loss: 0.1747 - val_accuracy: 0.9965
Epoch 47/300
85/85 [==============================] - 5s 55ms/step - loss: 0.4275 - accuracy: 0.7762 - val_loss: 0.1643 - val_accuracy: 0.9965
Epoch 48/300
85/85 [==============================] - 6s 72ms/step - loss: 0.4313 - accuracy: 0.7803 - val_loss: 0.1646 - val_accuracy: 0.9982
Epoch 49/300
85/85 [==============================] - 4s 48ms/step - loss: 0.4278 - accuracy: 0.7877 - val_loss: 0.1730 - val_accuracy: 0.9947
Epoch 50/300
85/85 [==============================] - 4s 48ms/step - loss: 0.4172 - accuracy: 0.7926 - val_loss: 0.1656 - val_accuracy: 0.9956
Epoch 51/300
85/85 [==============================] - 5s 60ms/step - loss: 0.4091 - accuracy: 0.7959 - val_loss: 0.1549 - val_accuracy: 0.9965
Epoch 52/300
85/85 [==============================] - 4s 42ms/step - loss: 0.4179 - accuracy: 0.7870 - val_loss: 0.1597 - val_accuracy: 0.9965
Epoch 53/300
85/85 [==============================] - 4s 45ms/step - loss: 0.4089 - accuracy: 0.7870 - val_loss: 0.1560 - val_accuracy: 0.9965
Epoch 54/300
85/85 [==============================] - 5s 61ms/step - loss: 0.4198 - accuracy: 0.7870 - val_loss: 0.1595 - val_accuracy: 0.9982
Epoch 55/300
85/85 [==============================] - 5s 57ms/step - loss: 0.4302 - accuracy: 0.7777 - val_loss: 0.1630 - val_accuracy: 0.9938
Epoch 56/300
85/85 [==============================] - 4s 46ms/step - loss: 0.4169 - accuracy: 0.7866 - val_loss: 0.1741 - val_accuracy: 0.9982
```



```
Epoch 57/300
85/85 [==============================] - 5s 53ms/step - loss: 0.4354 - accuracy: 0.7870 - val_loss: 0.2033 - val_accuracy: 0.9956
Epoch 58/300
85/85 [==============================] - 5s 55ms/step - loss: 0.4174 - accuracy: 0.7877 - val_loss: 0.1731 - val_accuracy: 0.9956
Epoch 59/300
85/85 [==============================] - 4s 42ms/step - loss: 0.4059 - accuracy: 0.7922 - val_loss: 0.1683 - val_accuracy: 0.9956
Epoch 60/300
85/85 [==============================] - 4s 43ms/step - loss: 0.4110 - accuracy: 0.7885 - val_loss: 0.1709 - val_accuracy: 0.9956
Epoch 61/300
85/85 [==============================] - 5s 64ms/step - loss: 0.4298 - accuracy: 0.7781 - val_loss: 0.1717 - val_accuracy: 0.9965
Epoch 62/300
85/85 [==============================] - 4s 43ms/step - loss: 0.4047 - accuracy: 0.7970 - val_loss: 0.1753 - val_accuracy: 0.9956
Epoch 63/300
85/85 [==============================] - 4s 42ms/step - loss: 0.4167 - accuracy: 0.7810 - val_loss: 0.1718 - val_accuracy: 0.9965
Epoch 64/300
85/85 [==============================] - 4s 48ms/step - loss: 0.4079 - accuracy: 0.7929 - val_loss: 0.1646 - val_accuracy: 0.9956
Epoch 65/300
85/85 [==============================] - 5s 63ms/step - loss: 0.4203 - accuracy: 0.7848 - val_loss: 0.2605 - val_accuracy: 0.9973
Epoch 66/300
84/85 [============================>.] - ETA: 0s - loss: 0.4175 - accuracy: 0.7894Restoring model weights from the end of the best epoch: 51.
85/85 [==============================] - 4s 47ms/step - loss: 0.4177 - accuracy: 0.7896 - val_loss: 0.1684 - val_accuracy: 0.9973
Epoch 66: early stopping
```

Model Evaluation:



```
score = model.evaluate(X_test, y_test, verbose='auto')
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
36/36 [==============================] - 0s 9ms/step - loss: 0.1549 - accuracy: 0.9965
Test loss: 0.15494154393672943
Test accuracy: 0.9964601993560791
```

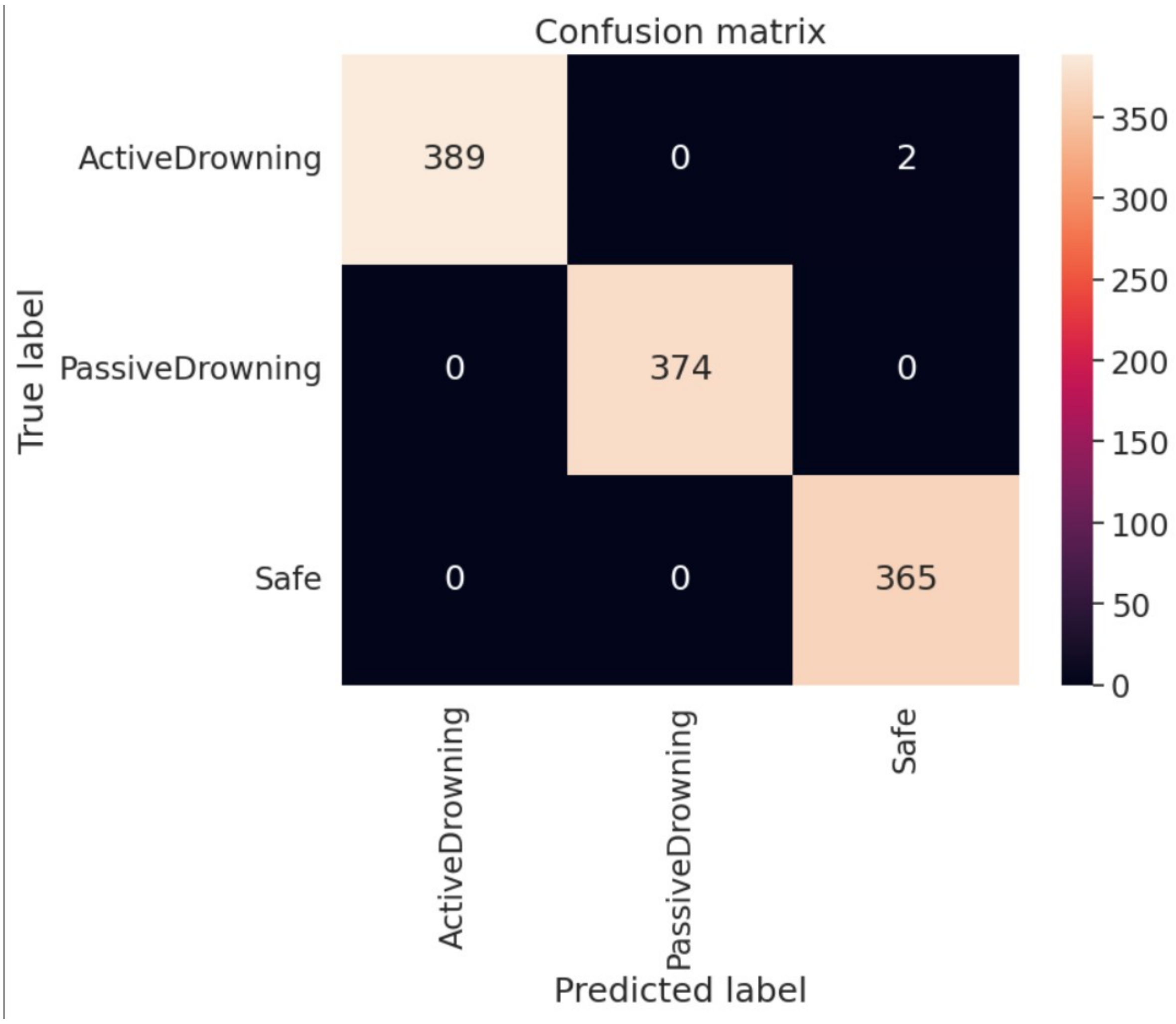


Figure 37 True labels vs Predicted Labels

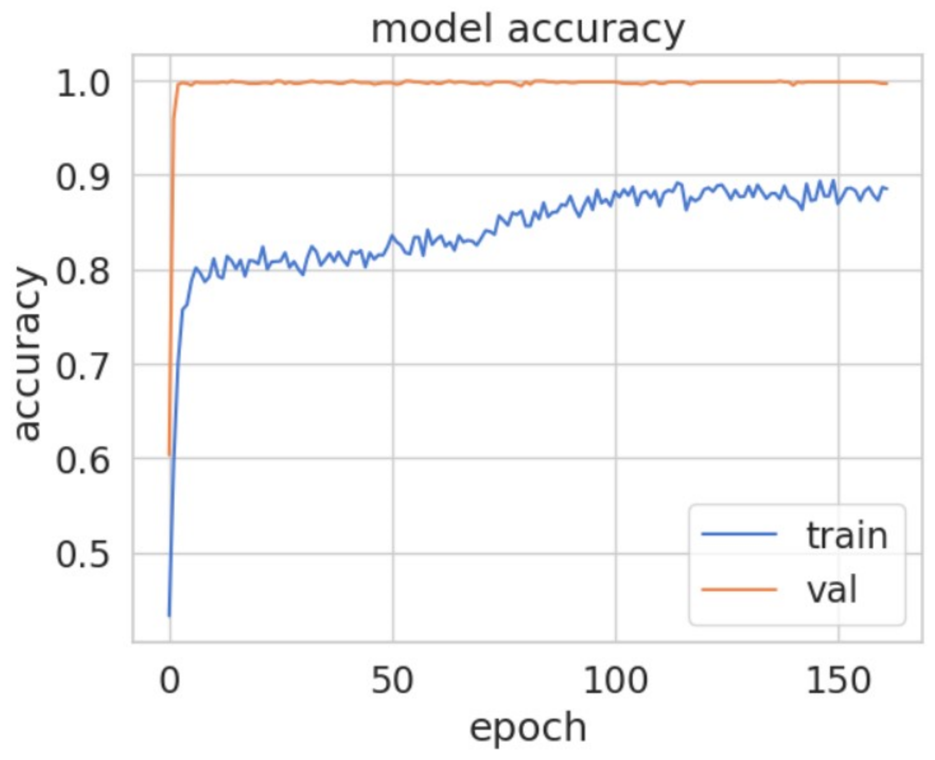


Figure 38 Model Accuracy

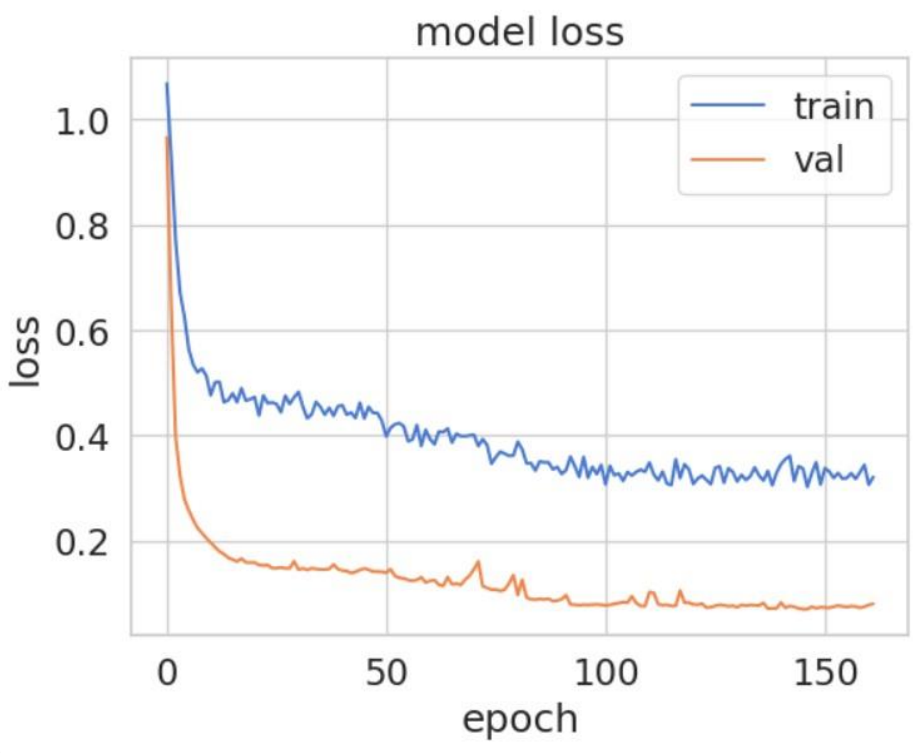


Figure 39 Model Loss

Heart Rate & Oxygen Saturation Percentage Sensor:

Dataset Description:

Graph identifying heart rate data:

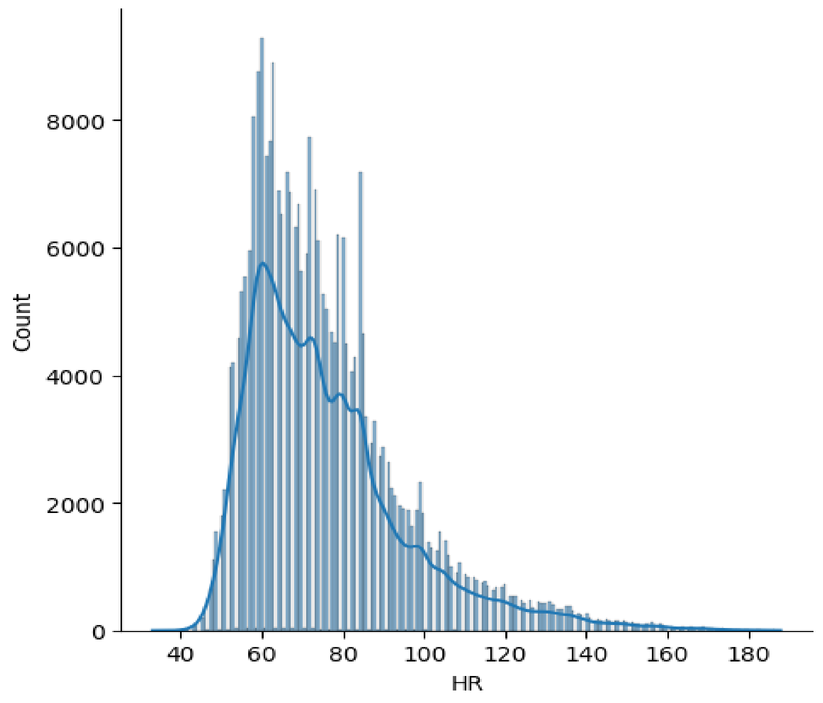
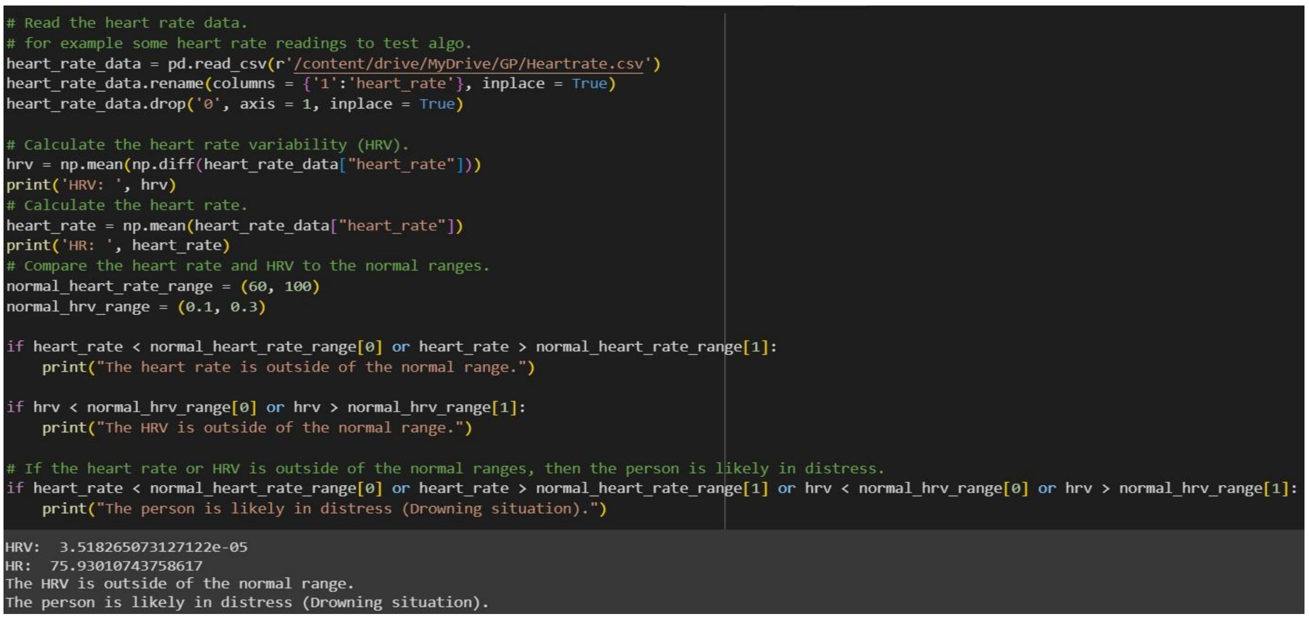


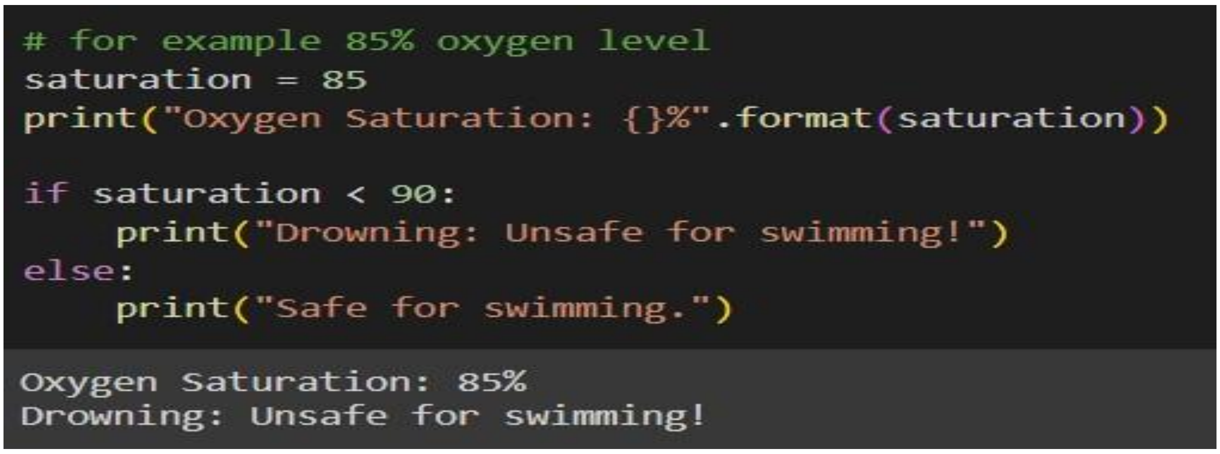
Figure 40 Heart Rate Data

Algorithm:



```
# Read the heart rate data.
# for example some heart rate readings to test algo.
heart_rate_data = pd.read_csv(r'/content/drive/MyDrive/GP/Heartrate.csv')
heart_rate_data.rename(columns = {'1':'heart_rate'}, inplace = True)
heart_rate_data.drop('0', axis = 1, inplace = True)

# Calculate the heart rate variability (HRV).
hrv = np.mean(np.diff(heart_rate_data["heart_rate"]))
print('HRV: ', hrv)
# Calculate the heart rate.
heart_rate = np.mean(heart_rate_data["heart_rate"])
print('HR: ', heart_rate)
# Compare the heart rate and HRV to the normal ranges.
normal_heart_rate_range = (60, 100)
normal_hrv_range = (0.1, 0.3)

if heart_rate < normal_heart_rate_range[0] or heart_rate > normal_heart_rate_range[1]:
    print("The heart rate is outside of the normal range.")

if hrv < normal_hrv_range[0] or hrv > normal_hrv_range[1]:
    print("The HRV is outside of the normal range.")

# If the heart rate or HRV is outside of the normal ranges, then the person is likely in distress.
if heart_rate < normal_heart_rate_range[0] or heart_rate > normal_heart_rate_range[1] or hrv < normal_hrv_range[0] or hrv > normal_hrv_range[1]:
    print("The person is likely in distress (Drowning situation).")
```

```
HRV:  3.518265073127122e-05
HR:  75.93010743758617
The HRV is outside of the normal range.
The person is likely in distress (Drowning situation).
```



```
# for example 85% oxygen level
saturation = 85
print("Oxygen Saturation: {}%".format(saturation))

if saturation < 90:
    print("Drowning: Unsafe for swimming!")
else:
    print("Safe for swimming.")
```

```
Oxygen Saturation: 85%
Drowning: Unsafe for swimming!
```

6.6.3 **Libraries:**

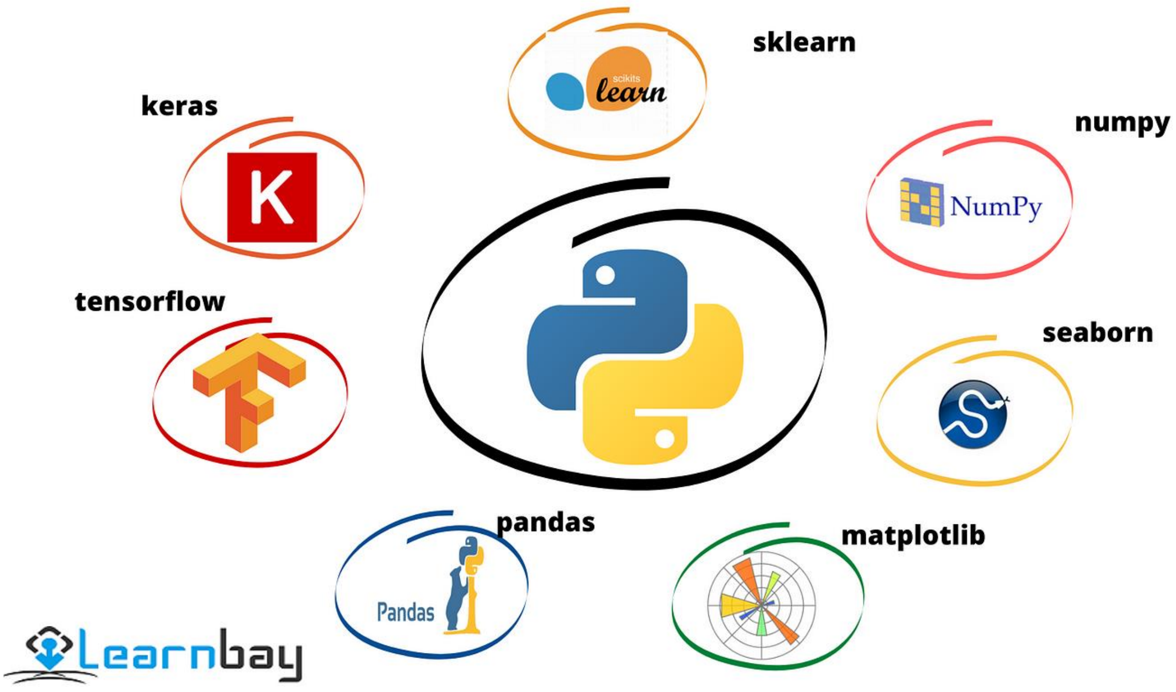


Figure 41 Python Libraries used in project

7 Conclusion and future work

After the long way through designing such a big system, we got great results. which includes:

- Designing the SWIDRO band that work about 4 hours of use.
- Designing the SWIDRO gateway that work about 5 hours of use.
- We have developed a fully integrated web application that can manage the organizations' pools and beaches such as hotels.
- We used the best possible solutions in communication for a water environment which is challenging.
- We developed a unique algorithm in detecting early drowning situations.
- We have two backup emergency systems which include a timeout for being underwater for long periods of 16 seconds and can be adjusted and an emergency button.
- We didn't rely on simulations instead we made several field tests in pools (10 tests in pool)
- The system needs a little tuning to be industry ready as we developed the system using portable software for different microcontrollers in the industry.

For future work we have a great vision for our system to be integrated in different organizations.

- Integrate the system with different emergency systems within the organizations.
- Make the product industry ready with industrial sensors and materials.
- Use secure communication to protect organizations' data.

8 References

1. Kałamajska, E.;Misiurewicz, J.; Weremczuk, J.Wearable Pulse Oximeter for Swimming Pool Safety. Sensors 2022,22, 3823. https://doi.org/10.3390/s22103823
2. Maher, Salma & Ali, Ziad & Mahmoud, Haitham & Abdellatif, Sameh & Abdellatif, Mohammad. (2019). Performance of RF underwater communications operating at 433 MHz and 2.4 GHz. 334-339. 10.1109/ITCE.2019.8646491.
3. Ramdhan, M.S. & Ali, Muhammad & Paulson, Eberechukwu & N.Effiyana, Ghazali & Ali, Samura & Kamaludin, M.Y.. (2018). An Early Drowning Detection System for Internet of Things (IoT) Applications. Telkomnika (Telecommunication Computing Electronics and Control). 16. 1870-1876. 10.12928/TELKOMNIKA.v16i4.9046.
4. Jalalifar, Salman & Kashizadeh, Afsaneh & Mahmood, Ishmam & Belford, Andrew & Drake, Nicolle & Razmjou, Amir & Asadnia, Mohsen. (2022). A Smart Multi-Sensor Device to Detect Distress in Swimmers. Sensors. 22. 1059. 10.3390/s22031059.
5. https://www.sealswimsafe.com/
6. https://coralmylo.com/
7. M. O. Santos, S. M. M. Faria and T. R. Fernandcs, "Real Time Underwater Radio Communications in Swimming Training Using Antenna Diversity," 2021 Telecoms Conference (ConfTELE), 2021, pp. 1-5, doi: 10.1109/ConfTELE50222.2021.9435592.
8. https://github.com/farhantandia/Smartwatch-SwimStyle-and-Drowning-detection-model