MANCHESTER 1824
The University of Manchester

Electric Bike Performance Enhancement – Security System

Third Year Individual Project – Final Report

April 2022

Tong Wu

10497665

Supervisor: Laith Danoon

School of Electrical and Electronic Engineering

Contents

Total word count: 8439

Abstract

Electric bikes are experiencing rapid growth as a result of technological advances. While bicycle sales have risen, the annual bicycle theft rate has also increased. Today, many bicycles have anti-theft solutions with a single chain, which does not provide adequate protection for the many valuable electronic components of an e-bike. This project focuses on building a new generation of an anti-theft system that uses multiple sensors to collect movement data and object distances from various bicycle parts, uses low-power Bluetooth (BLE) technology to communicate within the system and send any suspicious events to the bike owner via SMS. In addition, this project investigates a fingerprint-based solution for unlocking the bike and a display showing important information about the bike's movement to ensure the best riding experience for the user.

This report will show some testing and draw conclusions on the successful implementation of the security system as well as highlight the area where the project faced limitations. At the end of the report, some areas where future works can be further concentrated to improve the developed system.

1. Introduction

1.1. Background

Since the bicycle's invention in the 19th century, the bike has become an indispensable mode of transportation. Over the past few decades, the growth of technology and cities has driven the demand for transportation and commuting. There is a need for faster and longer-range transportation and improvements to the vehicle's safety and the safety of passengers. With the increase in car use, people began to look for cheaper and more efficient methods of travelling short and medium distances. The bicycle, a classic mode of transport, is now being retrofitted with batteries and motors to increase its speed and range, which consumers worldwide have come to love, as this is the definition of an e-bike.

E-bikes are becoming increasingly popular as a means of transportation. E-bike sales have increased dramatically in the past five years. For example, there were 32.57 million e-bikes sold in China last year. In 2020, sales increased by 48.34 million units, an increase of 48%.[1]

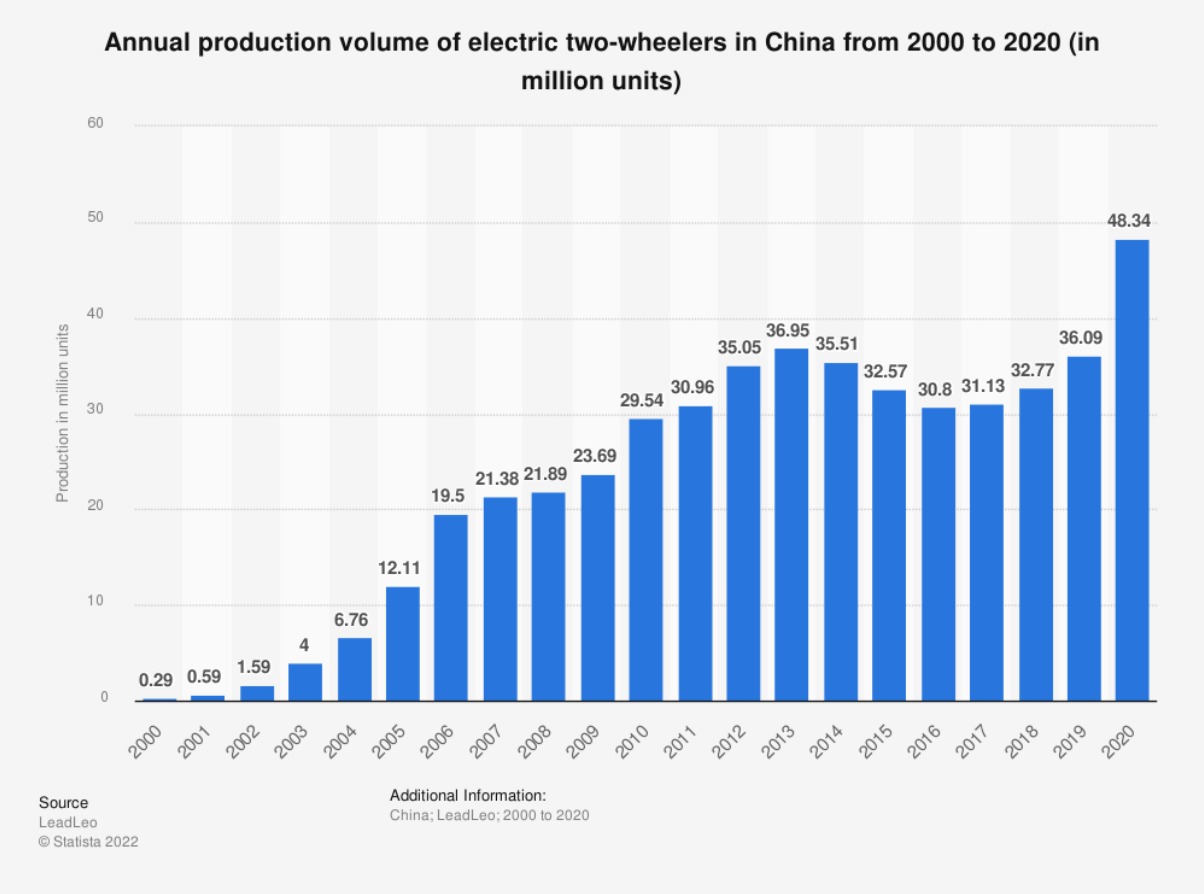

Fig. 1 Annual production volume of electric two-wheelers in China from 2000 to 2020[1]

The same upward trend in sales is also happening in Europe: rising from about 5920 million dollars in 2018 to 7789 million in 2020, growing nearly 31% [2]. Figure 2 indicates that the e-bike market is becoming increasingly popular.

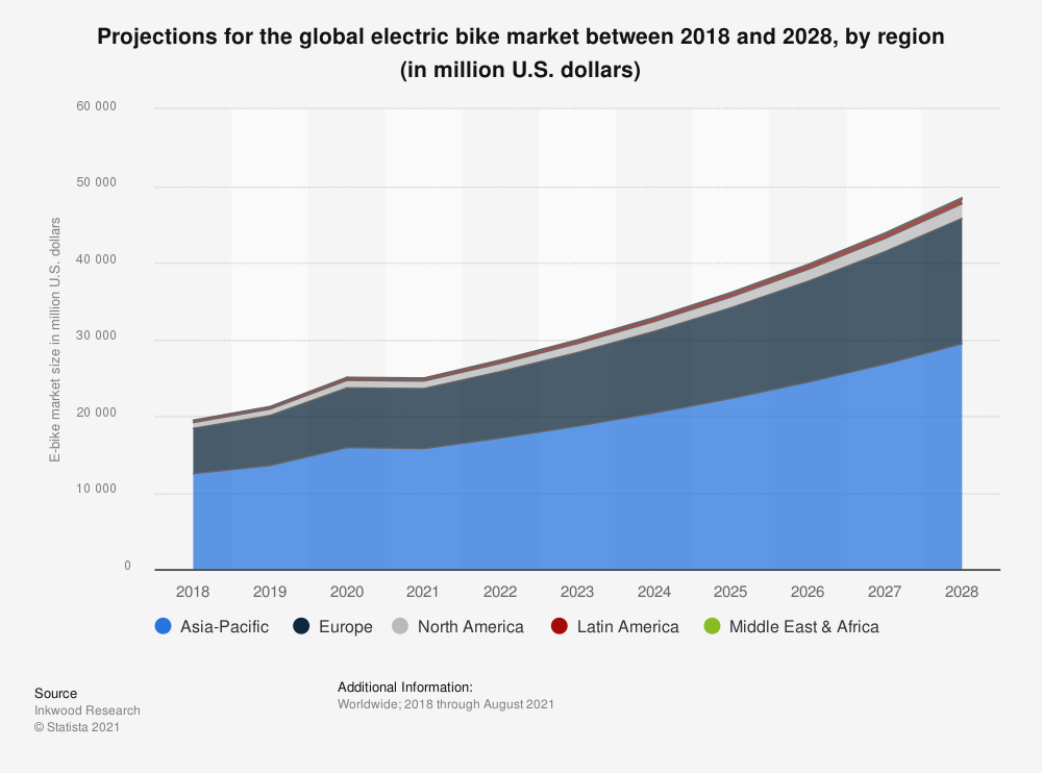

Fig. 2 Projections for the global electric bike market between 2018 and 2028, by region[2]

The peripheral systems for e-bikes have not kept pace with the rapid growth of the E-bike market. The most apparent aspect of e-bikes to consider is their security system, which should be considered before purchasing. The theft of nearly 90,000 bicycles in England and Wales in 2019[3] indicates the importance of bicycle security. Current e-bike security solutions rely on chains, but this does not provide sufficient protection to e-bikes. Other valuable components of an e-bike include the display, batteries, motors, and even the tires. It is not possible to lock these components according to traditional methods. Therefore, creating an intelligent, multi-sensor security system becomes essential to protect the e-bike itself and its accessories.

1.2. Motivations

In response to the market and theft data mentioned above, there is an obvious need to improve the security system on e-bikes to protect the bike itself and the valuable components on the bike, such as the display, batteries, and motor. The most significant advantages of an e-bike over traditional bikes are that the controller and motor can control the input and output power of the

e-bike. Additionally, an e-bike is equipped with a battery capable of powering the sensors and other modules needed for an aftermarket security system. Building an intelligent security system has been prompted by these advantages. Nevertheless, the ultimate goal of the entire project is to create a universal security system on bicycles, whether or not the bike comes with its battery – however, certain features can only be implemented on e-bikes and will not carry over to a conventional bicycle.

1.3. Literature Review

From 2015 to 2020, the number of bicycle thefts in England and Wales fluctuated but not significantly, averaging 90,000 per year[3].

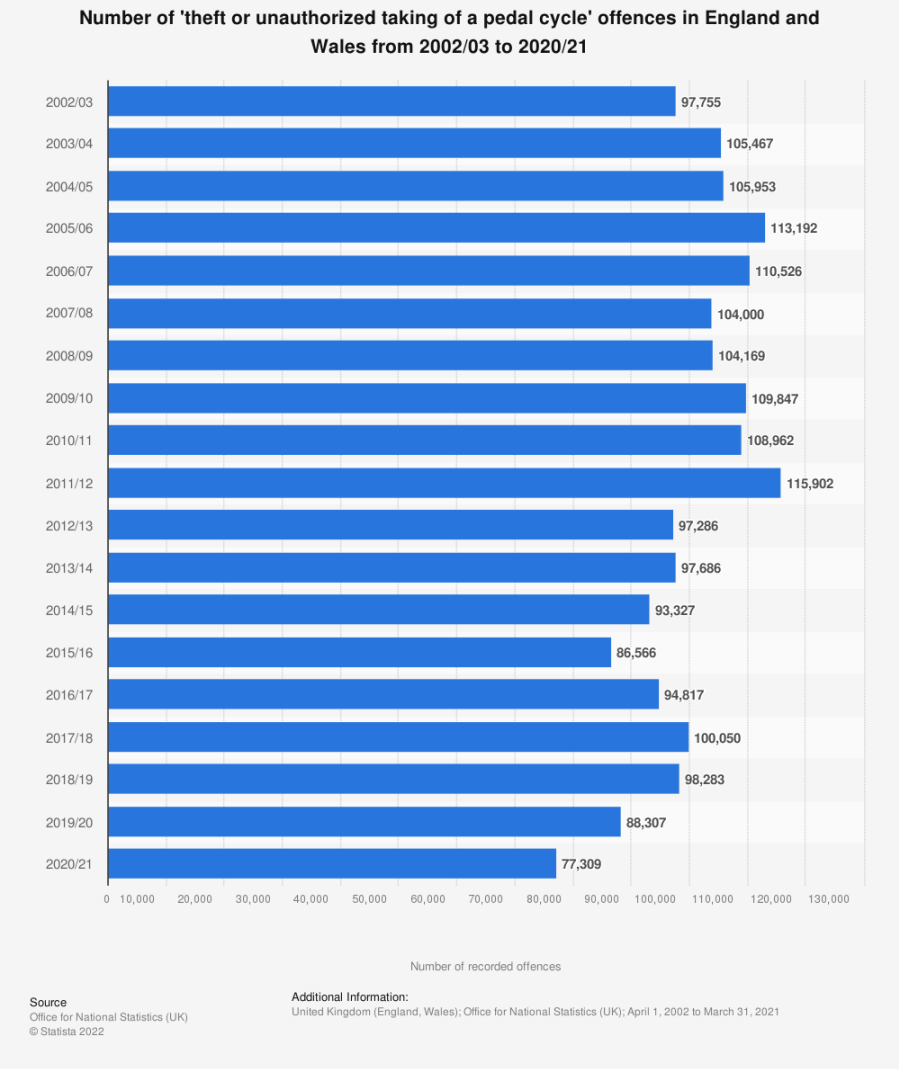

Fig. 3 Number of 'theft or unauthorised taking of a pedal cycle' offences in England and Wales from 2002/03 to 2020/21[3]

The same trend is playing out in France: the number of annual bicycle thefts has increased rather than decreased over the past 15 years, from nearly 330,000 in 2006 to 400,000 in 2017 and 360,000 in 2018 [4]. Such numbers of thefts had also contributed to the public's attitude towards bicycle theft, with nearly 72% of those surveyed not considering "bike theft is a rare thing", with 23% of surveyed people disagree with this opinion, according to the survey published in 2019 by Federation of Bicycle Users (Fédération des usagers de la Bicyclette) [5].

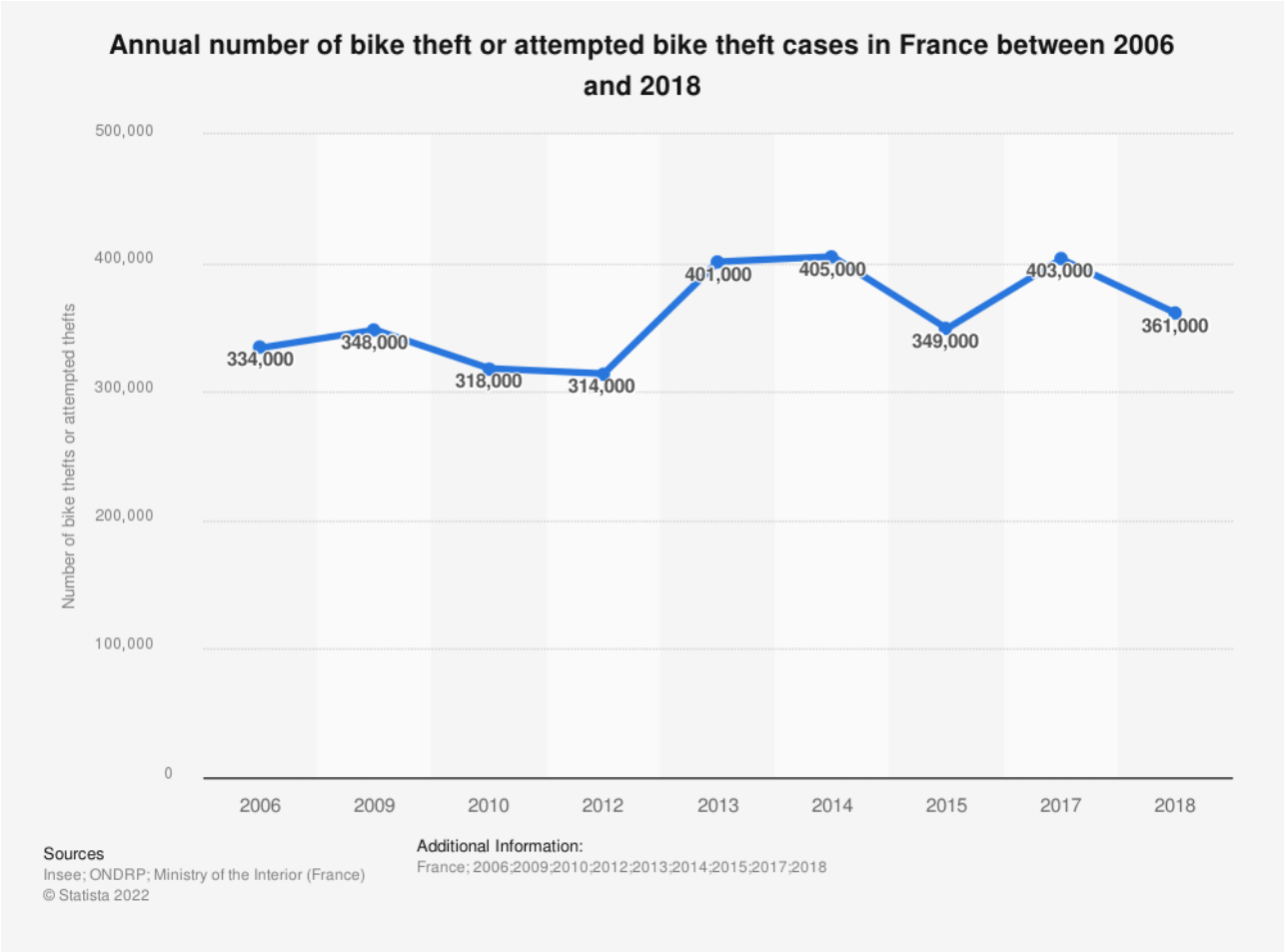

Fig. 4 Number of bike thefts or attempted bike thefts in France 2006-2018 [4]

Although there has not been a significant rise with the rapid development of technology, the lack of reduction in bicycle theft is not a result of the slowdown in the development of anti-theft technology but the lack of companies applying advanced anti-theft technology to bicycles. After researching the previous papers, a result was summarised that most of the research in this area was based only on the response to the theft but not on monitoring the bike's status. Dhruvi K. Zala and Argade Geetanjali Arjun, for example, mention in the paper the use of GPS systems to locate bicycles and send information via GSM modules[6] [7]. Veerandi Maleesha Kulasekara implemented a smart key system for the bike based on the ZigBee topology, a wireless

communication protocol running under the personal area network [8].

In summary, although previous researchers use some security methods, they are only effective after the actual theft has taken place, not at or before the theft. Thus, theft prevention is still an unexplored area. Building on the previous work, various sensors to monitor changes in the bicycle itself and the surrounding objects will be used to prevent theft.

1.4. Aims and Objectives

This project aims to create a universal smart, multi-sensor anti-theft and security system that can be used on e-bikes. Such an intelligent system will provide a fully functional system to help users monitor and protect their bikes. Provided functions contain monitoring the motion and movement of bikes; monitoring whether the wheel has been rotated; giving additional protection for valuable components of the bike, blocking bike use after a theft; notify users of bike status and GPS location in advance.

To achieve this, numerous objectives need to be completed. These objectives are listed below:

- Software and hardware development to measure the throttle output
- Reading and detecting hall sensor changes within the motor
- Implementation of proximity sensors
- Implementation of accelerometer and gyroscope
- Implementation of board-to-board Bluetooth communication
- Implementation of buzzer
- Implementation of GPS/SIM module
- Implementation of Fingerprint lock/unlock method
- Integration of the multiple sensors into a single system

2. System and Hardware Overview

The entire security system consists of one main Arduino board (Arduino Uno Wi-Fi Rev.2) and two supplementary Arduino boards (Arduino Nano 33 IoT). The three Arduino boards are placed in different locations on the bike to ensure that all parts that are vulnerable to theft are covered. Each Arduino runs its decision tree program to analyse its own IMU and ToF sensor. All the theft conditions should be transmitted to the main Arduino since the main Arduino monitors the throttle, hall sensor, fingerprint scanner, and control the buzzer and the GSM & GNSS module to

communicate with the outside. This chapter discusses the main parameters and hardware components needed to achieve the desired outcomes for the proposed security system.

2.1. Hall Sensor

An e-bike for testing was equipped with a brushless DC motor (BLDC), which replaced a mechanical commutator with a Hall sensor, which is better than a mechanical commutator. When the rotor of a brushless motor passes through a Hall sensor, the sensor will generate a high or low voltage level to indicate which rotor pole has passed. Essentially, the Hall sensor is responsible for capturing the wheel's rotation in this project. The Hall sensor will be able to provide an output voltage due to the rotor pole passing. In contrast, if the wheel rotates under external forces, the Hall sensor will also give a voltage value. When the e-bike is locked, the Hall sensor monitoring is performed to prevent the e-bike from being stolen. The Hall sensor will return a series of voltage values if the wheel is moved. Hall sensors are connected to the main controller of the e-bike using the power, earth, and data wires, so the method of interacting Arduino with them is similar, i.e., by connecting the Arduino board between the main controller and the throttle and Hall sensor, with Arduino acting as an intermediate bridge to read and transmit the data wires. This allows the Arduino to read the voltage output of the throttle and the Hall sensor, do specific actions for these voltage outputs, and transmit them to the main controller in real-time.

2.2. Throttle and LPF circuit

The power instruction given to the controller of the e-bike is sent from the throttle. Under the actual condition, the user will turn the throttle to control the e-bike's BLDC motor to have a specific speed. When building a security system, some of the instructions from the user need to be taken into account, as they may not be from the authorised user. The throttle, an essential component that can drive the e-bike, needs to be monitored and controlled. Therefore, using the Arduino to read the throttle voltage value and output the original voltage value from the other port if the bike is authorised to boot is the desired feature. However, the throttle voltage is in the form of DC, which is not the same as the Arduino's voltage capable of outputting (AC), so a low-pass filter (LPF) circuit needs to be added between the Arduino's output port and the e-bike controller. The LPF circuit serves to pass signals with a frequency lower than a selected cut-off frequency and attenuates signals with frequencies higher than the cut-off frequency [2]. When designing the LPF circuit, since one of the goals of this feature is to output the throttle voltage to the controller in DC without loss, it is necessary to ensure that the output AC voltage looks like the

DC voltage. The AC voltage is characterised by flow to a periodically varying voltage, a sine wave voltage with positive and negative output voltage values as its peak. The cut-off frequency must be small enough; 1hz is the superior value: a 330-ohm resistor 'R' and a 470 microfarad capacitor 'C' are applied to the LPF circuit. The cut-off frequency can be calculated at about 1.026 Hz according to equations 1 and 2.

$$f_c = \frac{1}{2\pi RC} \tag{1}$$

$$f_c = \frac{1}{2\pi \times 330 \times 470 \times 10^{-6}} = 1.026Hz \tag{2}$$

2.3. Time of flight (ToF) distance sensor

The time-of-flight (ToF) sensor VL53L1X was integrated into the system, which uses a laser-ranging sensor to measure the distance between the reflective surface of an object and the sensor. The ToF sensor emits a laser with a wavelength of 940nm, which is invisible light. Comparing the laser-ranging performance of the VL53L1X with a traditional infrared ranging sensor (e.g. VCNL4040), the laser-ranging method is more accurate and has a much longer valid testing range of about 4000 mm compared to infrared ranging 200 mm. A measuring range of 200 mm is not sufficient for the security system to monitor the theft action. After careful consideration, the ToF sensor has been used in the security system. The sensor is connected to the Arduino using the IIC protocol, and the device address is 0x29.

Fig. 5 VL53L1X ToF distance sensor

2.4. Inertial measurement unit (IMU)

For the safety of electric bicycles, the movement state of the bicycle needs to be monitored, such as the degree of tilt, the intensity of shaking, and acceleration, to determine the bicycle's status. A 9-axis IMU with accelerometer, gyroscope and magnetometer is incorporated into the security system. The 9 degree of freedom sensor is the LSM6DS33+LIS3MDL, where the 3-axis accelerometer tells the position of the object about the ground (i.e., which direction it is facing),

and the 3-axis gyroscope provides information on the sway and rotation of the bike and the magnetometer measures the orientation of the bike. The manufacturer of this sensor, Adafruit, provides code libraries to facilitate programming, which allows the range and output frequency of the accelerometer, gyroscope, and magnetometer to be adjusted. The sensor and Arduino can be connected via the IIC protocol at 0x6A and 0x1D, corresponding to LSM6DS33 and LIS3MDL.

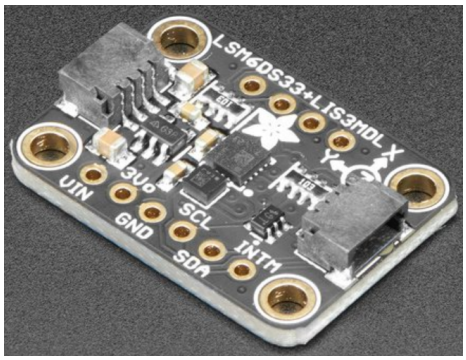

Fig. 6 Adafruit LSM6DS33 + LIS3MDL - 9 DoF IMU

2.5. Bluetooth Module with Bluetooth Low Energy (BLE) technology

The Arduino board is already integrated with a Bluetooth 4.0 chip supporting Blue Low Energy (BLE) technology. Unlike standard Bluetooth communication, based on an asynchronous serial connection (UART), a Bluetooth LE radio acts as a community bulletin board.[1] The computers that connect to it are like community members that read the bulletin board. The BLE device is defined as a 'central' or 'peripheral' device by the General Access Profile (GAP).[9] GAP controls connections and advertising in Bluetooth. GAP is what makes the device visible to the outside world and determines how two devices can (or cannot) interact with each other. Peripheral devices are small, low power, resource-constrained devices that can connect to a much more powerful central device.[10] The peripheral device will continuously advertise services to the public, which can contain multiple characteristics. The characteristic contains the data that the peripheral device needs to transmit, for example, the value of the accelerometer and the ToF sensor, and each of the characteristics can send data within 31 bytes. Figure 7 below shows the principle of advertising. The peripheral device will set a specific advertising interval, and whenever this interval is exceeded, the peripheral device will rebroadcast the packet.[10] Extending the advertising interval will save power, while it will have a longer response time, which appears to the central device to make the peripheral more sluggish.

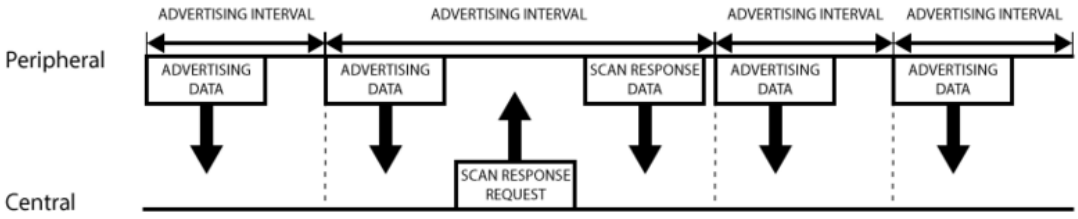

Fig. 7 Advertising Process[10]

2.6. The Global Navigation Satellite System (GNSS) and Global System for Mobile communication (GSM) module

The Global Navigation Satellite System (GNSS) & Global System for Mobile communication (GSM) module contains two chips to send a message and get the position. The GNSS chip is used to acquire latitude and longitude information about the module, and the GSM module is used to connect the LTE network and send specific messages or data packages to the authorised user and website. For a security system, it is essential to notify the authorised user and the surrounding when a suspected theft action is detected. Due to the outdoor environment, connecting to Wi-Fi is reduced, so a GNSS & GSM module that can be plugged into a SIM card and supports positioning is used in the system. This module uses a UART connection to communicate with the Arduino board, which occupies Digital Output Pin 0 & 1, the Rx and Tx portal.

Fig. 8 GNSS & GSM module SIM7600CE[11]

2.7. Fingerprint Module

The standard locking solution is limited to the vehicle lock for the bicycle, which lacks some portability and versatility. Fingers, the most commonly used part of the human body, are also used by engineers to investigate ways to improve safety. As one of the human biometric features, fingerprints are unique and challenging to change, so using them as a key can significantly increase security. Fingerprint unlocking is used on many electronic devices, such as mobile phones and computers, and now it is considered to be integrated into the bicycle security system. A fingerprint

sensor ID809 has been used, which uses a capacitive fingerprint sensor that supports fingerprint capture, storage and comparison [12]. It contains a separate processor on which all fingerprint recognition and storage algorithms can be performed. The fingerprint module can store up to 200 different fingerprints and add, test, and delete fingerprints independently of the computer. The sensor supports the IIC protocol for connection to the Arduino.

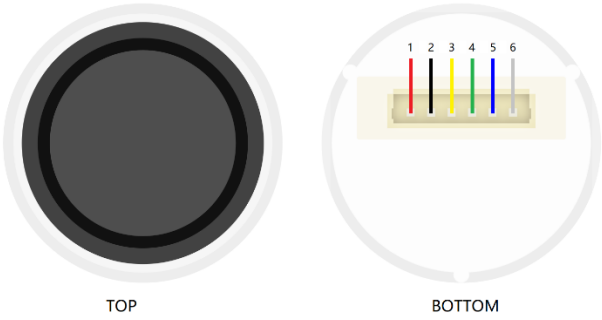

Fig. 9 Top and button view of the fingerprint sensor[12]

2.8. Display Module

Displays are often used in bicycles to show various states of the bicycle, and this is also the case in this project. Some necessary safety parameters of the bike are shown in a small display that consumes less power to indicate the security system and the actions of the authorised user. Adafruit's SH110X OLED display module is used in the system, which, due to its display technology, allows for purer blacks, not emitting light at all, which saves power when displaying less text. This display is connected to the Arduino via the IIC protocol at address 0x3C.

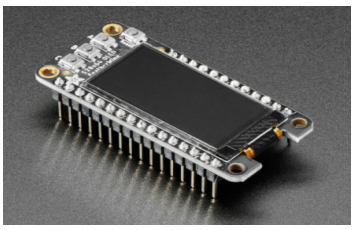

Fig. 10 Display Module[1]

3. Software Development

3.1. Bluetooth with BLE

The Bluetooth BLE technology is used in this project to cope with the problem of communicating among Arduino boards when reading sensor data from separate locations. The chosen board,

Arduino Uno Wi-Fi Rev.2 and Arduino Nano 33 IoT, both have Bluetooth module support with BLE technology. As mentioned earlier, the security system consists of one main Arduino and two supplementary Arduinos. When considering the communication profile of these Arduino boards, it was decided to let the main Arduino as the 'central device', which is mainly responsible for the data receiving and computing; and let two supplementary Arduinos as 'peripheral devices ', which is mainly responsible for the data acquisition from sensors, and transmit them by advertising service with its characterisations, and updating the sensor data regularly. A communication flow chart corresponding to the main Arduino and the complementary Arduinos can be drawn with this basic framework in place.

The flow chart is shown below in Figure 11, which illustrates the supplementary Arduino Board (peripheral device). In the initialisation part, known as the 'setup' function, the supplementary Arduino board needs to initialise the service and characteristics. In the Arduino BLE library, the service can be created by calling an object from the class 'BLEService', and the characteristics can be created by calling an object from the class 'BLECharacteristic'. Service and characteristics need a unique UUID, a 128-bit long identifier for each activity. After initialising the service and characterisation, use the sentence to set an advertised service and then add a service characteristic. One service can contain more than one characteristic and advertise at once. After setup, all the characteristics start advertising the service by calling the function contained in the class.

The central device can connect to the peripheral device by searching the UUID of the service. Then the supplementary board go to the 'loop' function, which has an 'if' statement to determine whether it is connected to the main Arduino board (central device). If the return value is false, the supplementary board will continuously update the characteristics to write the latest sensor value to the characteristic; if the return value is true, the supplementary Arduino will stop advertising the service to save the battery and wait for the main board successfully receive the current sensor value. As it is up to the main control board to connect and disconnect devices, the peripheral device is designed to wait until its BLE status changes to disconnected, after which the board will pause for 2 seconds. This pause is because multiple supplementary devices in the security system need to send data to the main controller, and the Arduino Uno and Arduino Nano series control board chips do not support concurrent programming, i.e. two communication functions can not run at the same time. Also, the Bluetooth protocol used by Arduino is BLE 4.0, which does not support a central device. This protocol does not support the connection of multiple edge devices

simultaneously. Therefore, multiple Arduino can not be connected simultaneously nor transfer data at the same time. When the Arduino connection is disconnected, a pause of 2 seconds allows another Arduino control board to have enough time to connect to the main control board.

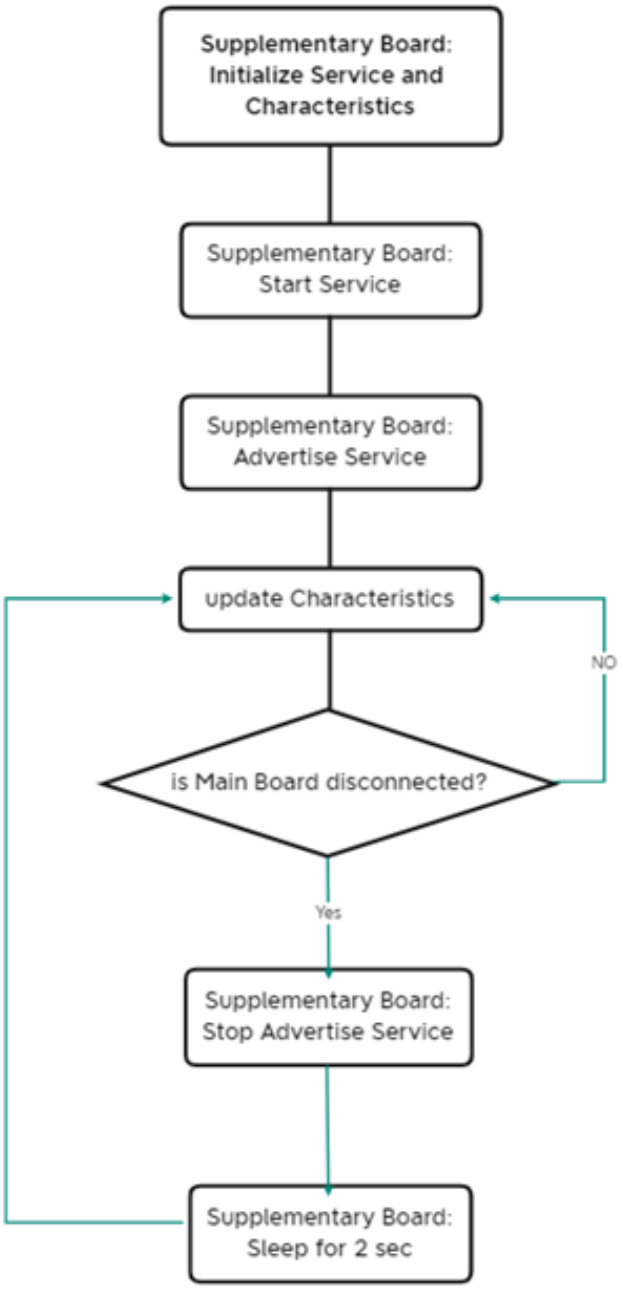

Fig. 11 Flow chart for Supplementary Arduino Board

For the main Arduino board (central device), it needs to search for a specific peripheral device by its UUID. After connecting to the peripheral device, the characteristic's UUID will be stored in the object 'peripheral'. Then, the central board can start to retrieve the characteristic. After the reading is complete, the central board will disconnect and proceeds to the next cycle.

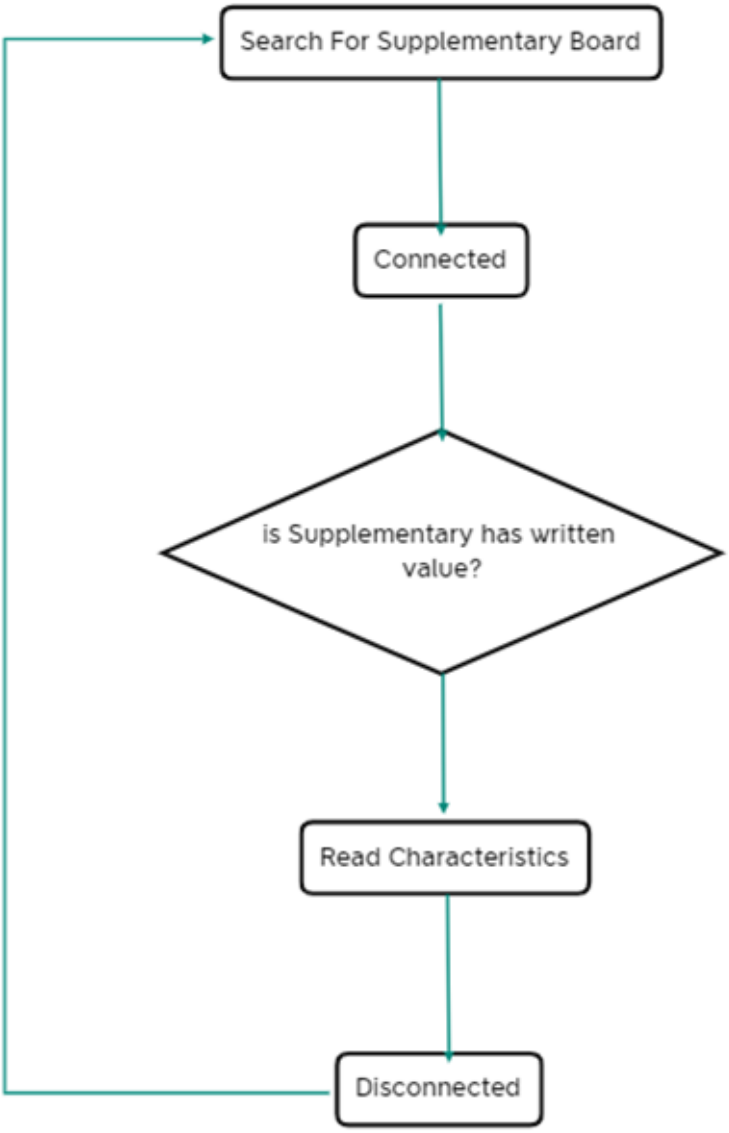

Fig. 12 Flow chart for Main Arduino Board

3.2. GNSS and GSM module with AT commands

Unlike the common C/C++ code for Arduino, the GNSS & GSM module SIM7600CE uses the Hayes command set, also called AT command. AT command set consists of a series of short text strings that can be combined to produce commands for operations such as dialling, hanging up, and changing the connection parameters.[13] AT commands can usually be divided into four main categories. The basic command set refers to an uppercase character followed by a number; the extended command set is preceded by an "&"; the proprietary command set is used in modems, usually with a "\" or "%"; the register command set consists of the number and value of the period register.[13] The SIM7600CE/T module and the Arduino are connected using Tx/Rx, so the module

also has a specific baud rate (for SIM7600CE is 115200) to communicate with the Arduino. Normally, the module is debugged in the IDE by entering AT commands to the module using the command prompt. However, in a PC-independent environment, the Arduino controls the module by entering commands to the Rx port, which need to be included in the function "SIM7600CE.print("Char ");". The commands to send SMS and obtain GPS information are shown in Table 1 below.

Table 1 AT Command list for GNSS and GSM module

AT command	Feature
AT+CPOWD=1	Power up the module
AT+CMGF=1	Enable the GSM module
AT+CMGS=\”0044xxx xxxxxxxx”\r	Send a message to the specific phone number
AT+CGNSINF	Retrieve position data by GNSS module
AT+CBC	Retrieve battery level

It is worth noting that after the SMS command is entered, the text content of the SMS needs to be entered following, ending with '1A' in HEX, which is '26' in char type value in Arduino code.

3.3. ToF distance sensor

The manufacturer of the ToF sensor, Adafruit, provides a code library for the sensor that is suitable for use within the Arduino. The functions in the library header file can be called to acquire data from the ToF sensor. As the ToF sensor has a clear purpose of measuring the distance, the data acquisition for the ToF sensor is called the 'distance' function. The 'distance' function returns a float value which is the distance in millimetres. It is important to note that when the distance function returns a value of -1, the sensor cannot obtain a distance value, possibly because the object is too far or too close to the sensor. After collecting data from the sensor, the data needs to be analysed to determine a threshold value for the ToF sensor, beyond which a flag needs to be returned to indicate that the ToF sensor is reading an abnormal value. Figure 13 shows the ToF sensor workflow.

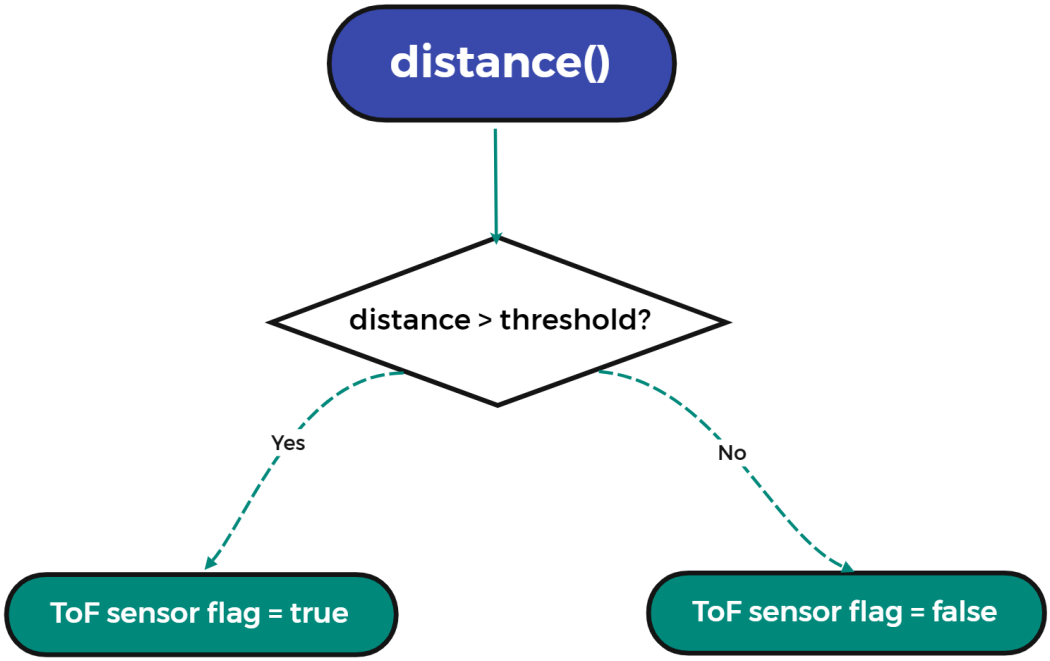

Fig. 13 Flow Chart for ToF sensor

3.4. IMU sensor fusion by Complementary Filter

The IMU specification used in this project was mentioned in the previous hardware section and used a 3-axis accelerometer, gyroscope, and magnetometer. In the early stages of the project, 3-axis data from the accelerometer was used to calculate the difference between the current and previous, then summed it and compared it to the sum of the 3-axis data from the gyroscope. If both data were more significant than a specific threshold, it was determined that there was human activity causing the bike to wobble. However, a simple fusion analysis method of the 9-axis data called a Complementary Filter is used to find the angles, which consist of the roll (θ), pitch (Φ) and yaw (ψ) corresponding to the flip angle in the x, y and z axes respectively. Figure 14 below indicates the relationship between the axis and roll-pitch-yaw angle. With these angles, the current attitude of the chip (bike) can be precisely defined.

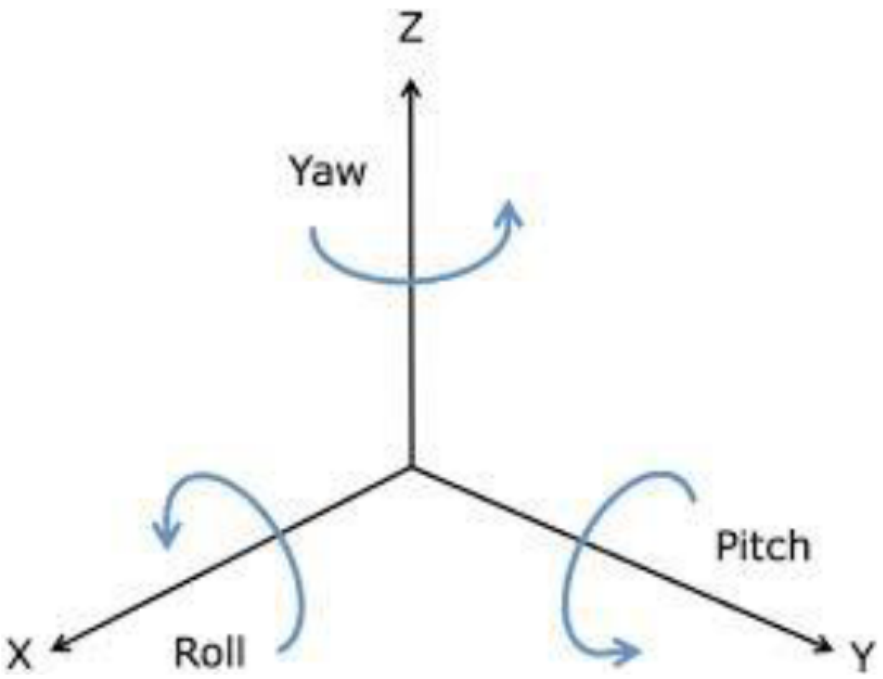

Fig. 14 Definition of roll, pitch, and yaw[14]

3.4.1. Accelerometer

According to the definition and the principle of the accelerometer, the value of the z-axis is always gravity when it is at rest towards the ground, i.e., approximating 1g. It is not precisely 1g because every sensor is subject to systematic or random errors, which are unavoidable. When the accelerometer is rotated according to the x-axis, the value of the y and z axis will change so that the roll can be calculated by the equation 3 below:

$$\theta_{Acc} = tan^{-1}\left(\frac{Acc_y}{Acc_z}\right) \tag{3}$$

When the accelerometer is rotated according to the y-axis, the value of the x and z-axis changes and pitch can be calculated by the equation 4 below:

$$\phi_{Acc} = tan^{-1}\left(-\frac{Acc_x}{\sqrt[2]{Acc_y^2 + Acc_z^2}}\right) \tag{4}$$

While for the yaw, the accelerometer cannot determine the yaw change since when the accelerometer rotates around the z-axis, all axis value does not change since the 'ball' inside the sensor is already touched on the z-axis plane. So, it is unable to use the accelerometer to calculate the yaw. Note that all the reading from the accelerometer is meter per square second, and the

value of roll and pitch is in radians.

The accelerometer provides a precise roll and pitch when the object is stationary. However, while the bike is in motion, the accelerometer reading may not be as accurate, as the accelerometer is moving, and the bike's acceleration will influence the results [14]. So, the fusion method is essential to use other sensors to cover the accelerometer's noise.

3.4.2. Gyroscope

The gyroscope provides the angular speed on each axis, so it is easy to get the roll, pitch and yaw value from the gyroscope reading by integrating it according to equations 5, 6 and 7.

$$\theta_{Gyro} = \omega_x * \Delta t \tag{5}$$

$$\phi_{Gyro} = \omega_y * \Delta t \tag{6}$$

$$\psi_{Gyro} = \omega_z * \Delta t \tag{7}$$

The problem of the gyroscope measurement is related to the time lagging, which is the time the controller takes to request and receive the data from the gyroscope. The IMU slightly delays measuring the gyroscope value and cannot precisely predict the measurement time. As Equations 5, 6 and 7 mentioned, angles computed by gyroscope value need the measuring time of the gyroscope since it is an integration function, the error of measurement time will be enlarged and will finally cause the drift over time. Also, the gyroscope value will not come reset to zero when it is stationary because of the inertia. According to this, the gyroscope itself will have a drift for an extended period. These issues have been considered and mitigated within the following integration calculation.

3.4.3. Magnetometer

The magnetometer is designed to show the strength of the earth's magnetic field, so it can only retrieve the yaw angle when placing the magnetometer on a smooth surface (e.g. the ground). Equation 8 shows how to calculate the yaw angle by using the y and x-axis of the magnetometer

$$\psi_{Mag} = tan^{-1}\left(\frac{m_y}{m_x}\right) \tag{8}$$

3.4.4. Fusion method and Complementary Filter

After retrieving all the values from sensors, a fusion of the accelerometer and the magnetometer need to be done. A step where using the roll and pitch calculated by the accelerometer to calculate the compensation value for yaw [15]. The new roll and pitch can be calculated by the Equations 9 and 10, and the new yaw angle is calculated by the Equation 11

$$\boldsymbol{\theta_{CF}} = \boldsymbol{m_x}\cos(\boldsymbol{\phi_{Acc}}) + \boldsymbol{m_y}\sin(\boldsymbol{\phi_{Acc}}) - \boldsymbol{m_z}\cos(\boldsymbol{\theta_{Acc}})\sin(\boldsymbol{\theta_{Acc}}) \quad (9)$$

$$\boldsymbol{\phi_{CF}} = \boldsymbol{m_y}\cos(\boldsymbol{\theta_{Acc}}) + \boldsymbol{m_z}\sin(\boldsymbol{\theta_{Acc}}) \quad (10)$$

$$\psi_{Mag_{New}} = tan^{-1}\left(\frac{\boldsymbol{Y_{com}}}{\boldsymbol{X_{com}}}\right) \quad (11)$$

The complementary filter uses high pass weights and low pass weights to fuse the accelerometer and gyroscope data, and as far as possible, the shortcomings of each sensor are eliminated [14]. As mentioned previously, the gyroscope data will drift a certain amount in the long-term case, so the complementary filter will reduce the weight of the gyroscope in the long term and use the accelerometer data. The equation for the complementary filter is expressed in Equations 12 and 13.

$$\boldsymbol{\theta_{CF}} = \boldsymbol{HP}\left(\boldsymbol{\theta_{CF}} + \boldsymbol{\theta_{Gyro}}\right) + \boldsymbol{LP}\theta_{Acc} \quad (12)$$

$$\boldsymbol{\phi_{CF}} = \boldsymbol{HP}\left(\boldsymbol{\phi_{CF}} + \phi_{Gyro}\right) + \boldsymbol{LP}\phi_{Acc} \quad (13)$$

Where according to A. Noordin et al., the best combination coefficients are known to be HP = 0.98 and LP = 0.2 [15].

3.5. System Flow chart and Event Flags

After obtaining the sensor data and calculating the roll, pitch and yaw, an algorithm is used in order to determine the security. When the value of each angle exceeds the threshold, a true flag is returned. A flag in this context signifies a possible theft event, and the flowchart is shown below in figure 15.

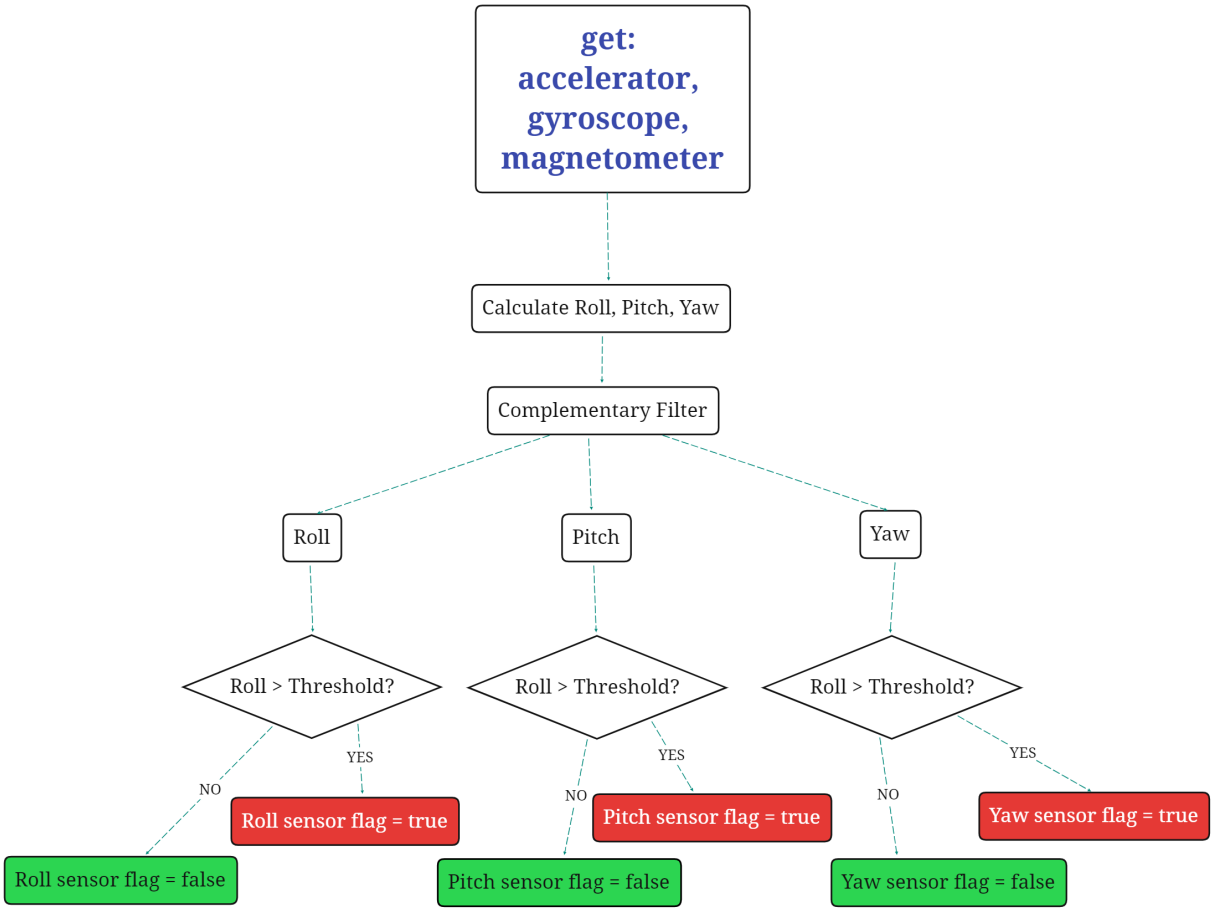

Fig. 15 Flowchart for IMU

3.6. Fingerprint lock/unlock method

As detailed in the 'hardware overview' section, the fingerprint module was to be used to lock/unlock the e-bike. As this fingerprint module has LED, the addition of LED lighting to indicate the current mode when entering different functions has been implemented—the sample code for the fingerprint module given the essential functions for fingerprint adding and testing. An integrated flow has been made to let the fingerprint module enter different modes according to the time the finger presses on the sensor. After the finger presses on the sensor for 3 seconds, the module will enter the testing mode, which reads the fingerprint on the sensor and matches it with the recorded fingerprint data, and returns the fingerprint ID if it is matched with the stored fingerprint, or a false indicator if the match is failed. After pressing about 8 seconds, the module enters the adding mode, which allows the authorised user to add a fingerprint to the sensor's register. By repeatedly pressing the sensor three times, the function will return a fingerprint ID if it is successfully registered. By pressing 13 seconds or more, the module will enter the deleting

mode, in which the program will delete the fingerprint that matches the finger on the sensor and returns the deleted fingerprint ID upon successful deletion.

The additional challenge presented by the fingerprint recognition module was the implementation of an 'interrupt-likely' function without interrupting the program as the ESP32 platform does. A count is used to ensure that the main loop will not be interrupted during the first 3 seconds when the finger is pressed on the sensor. After 3 seconds, an interrupt-likely function is called, which is implemented via a 'while' loop within the function.

3.7. Display module

Many statuses like the fingerprint reading, adding, and deleting, and the Euler angle are expected to be shown somewhere outside the computer screen because it is an essential message for the authorised user to show the e-bike status. With this information, an authorised user can become more familiar with their bike. Adafruit, the screen manufacturer, provides a library to help with implementation. A class 'Adafruit_SH1107' needs to be called and create an object of this class to call the display. The 'display' function must be called after all the content is input into the display's register to show the content on the screen.

4. System Integration and Testing

4.1. Strategic Sensor Placement

As previously stated, multiple Arduinos are used for the overall security of the bike, as the parts that need to be protected are not in the same place but scattered. Such an attribute makes it essential to give extra consideration to the range and conditions that the sensors can monitor when placing. The placement of the main Arduino and two supplementary Arduinos are shown in Figure 20.

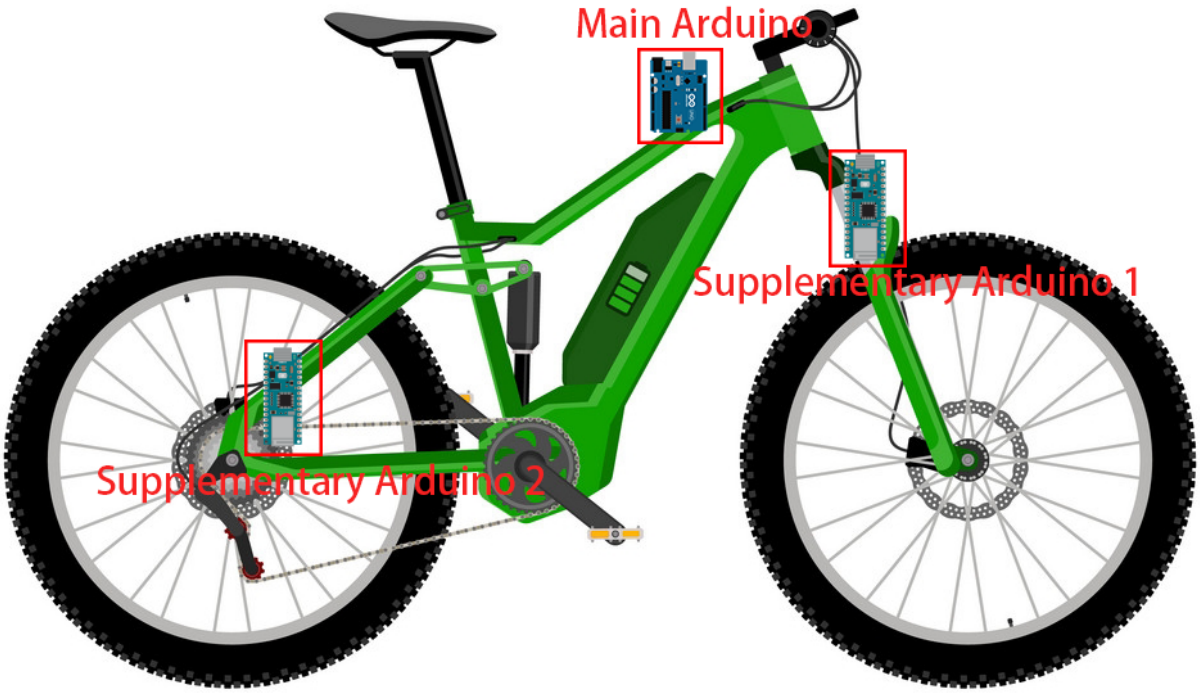

Fig. 16 Arduino and Sensor Placement

The main Arduino needs to be placed in the bike's centre since it needs to monitor and control additional components, including the fingerprint scanner, GSM & GNSS module and buzzer that interact with the authorised user, and the throttle and hall sensors that interact with the bike. We, therefore, choose to place the main Arduino in the top tube, which allows the subsystem of the main Arduino (containing the ToF and IMU sensor) to protect the security of the bike frame, and central bike controller and the battery.

Apart from the main body of the bike, two wheels of the bike are the most vulnerable to theft. Therefore, two supplementary Arduinos are placed here. The first supplementary Arduino is placed near the fork, which monitors the front wheel's steering and proximity. The second supplementary Arduino is placed near the seat tube, where the sensors monitor the rear wheel and the chain wheel.

4.2. Hardware connections

Several hardware devices need to be connected by a wire on the main Arduino board, including the ToF sensor, accelerometer and gyroscope and magnetometer (9-DOF IMU), fingerprint sensor, buzzer, and display module, GNSS & GSM module, throttle voltage and hall sensor data line. The hall sensor and throttle voltage data lines are connected to the Arduino's Analog IN A3 and A4

port, and an output port is also needed to output the throttle voltage to the bike controller, so the A5 port is used. The ToF sensor, IMU and display module all support IIC connection, and they have QWIIC compatible STEMMA QT connectors, which enable the connection of these sensors in series. A breadboard connects all the components, and the final connection is shown in Figure 16 below.

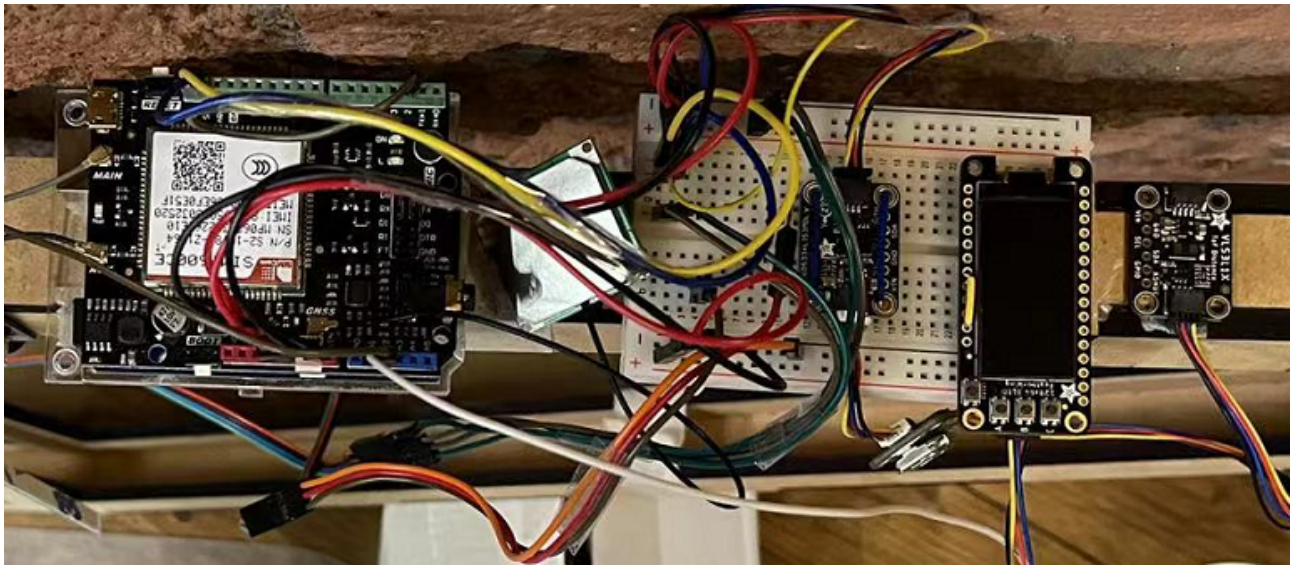

Fig. 17 Connection of Main Arduino

The supplementary board need to connect with the ToF sensor and IMU, and the connection is shown in figure 17 below.

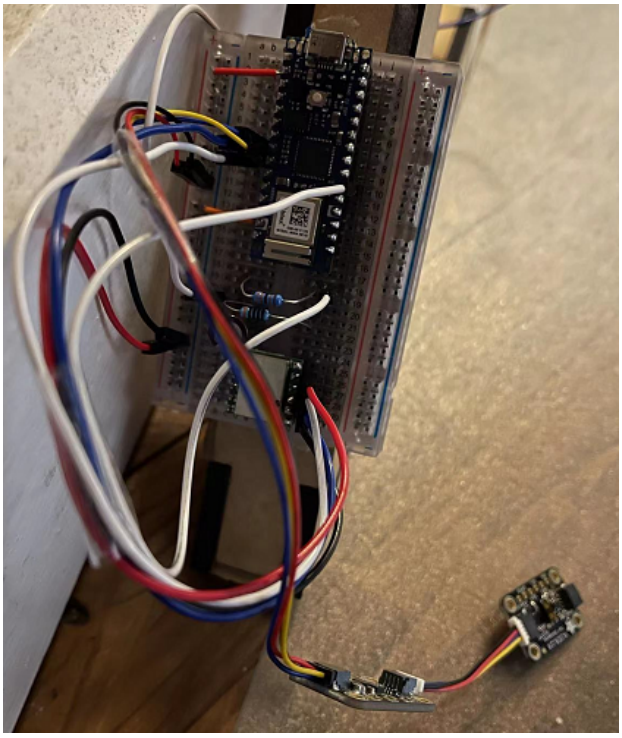

Fig. 18 Connection of Supplementary Arduino

4.3. Decision Analysis

Once the values from all the sensor and hardware devices have been read, the program then quantifies the values from each sensor according to the flow chart described earlier to output a flag that indicates which sensor or device is returning which data is abnormal. For the other sensors, the flag will only be adjusted to true if the reading from the sensor exceeds a threshold value, which means the sensor returns an unsafe signal. Otherwise, it will be false. Rising a flag to true for the fingerprint sensor means that the bike is locked, and false means the bike is not locked, i.e., the authorised user is riding it. In the Arduino program's setup function, the status of the fingerprint sensor flag is initialled to 'true' to make a default status that the bike is locked.

After all the flags have been generated, an analysis of the flags needs to be done by the program to determine if the bike is under a theft action. The decision tree is an algorithm in decision analysis, and it is often used in the machine learning area. The decision tree algorithm is often coming with a flowchart-like figure in which each internal node represents a "test" on an attribute (e.g., whether a coin flip comes up with heads or tails), and each branch of the tree represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). [16] The number of attributes depends on the number of flags that need to be analysed. According to the previous section, each sensor needs at least one attribute, so a decision tree algorithm for both main and supplementary Arduino has been written based on all the flags and the corresponding importance and thresholds. First, the main Arduino board's visual decision tree diagram is placed in Figure 18.

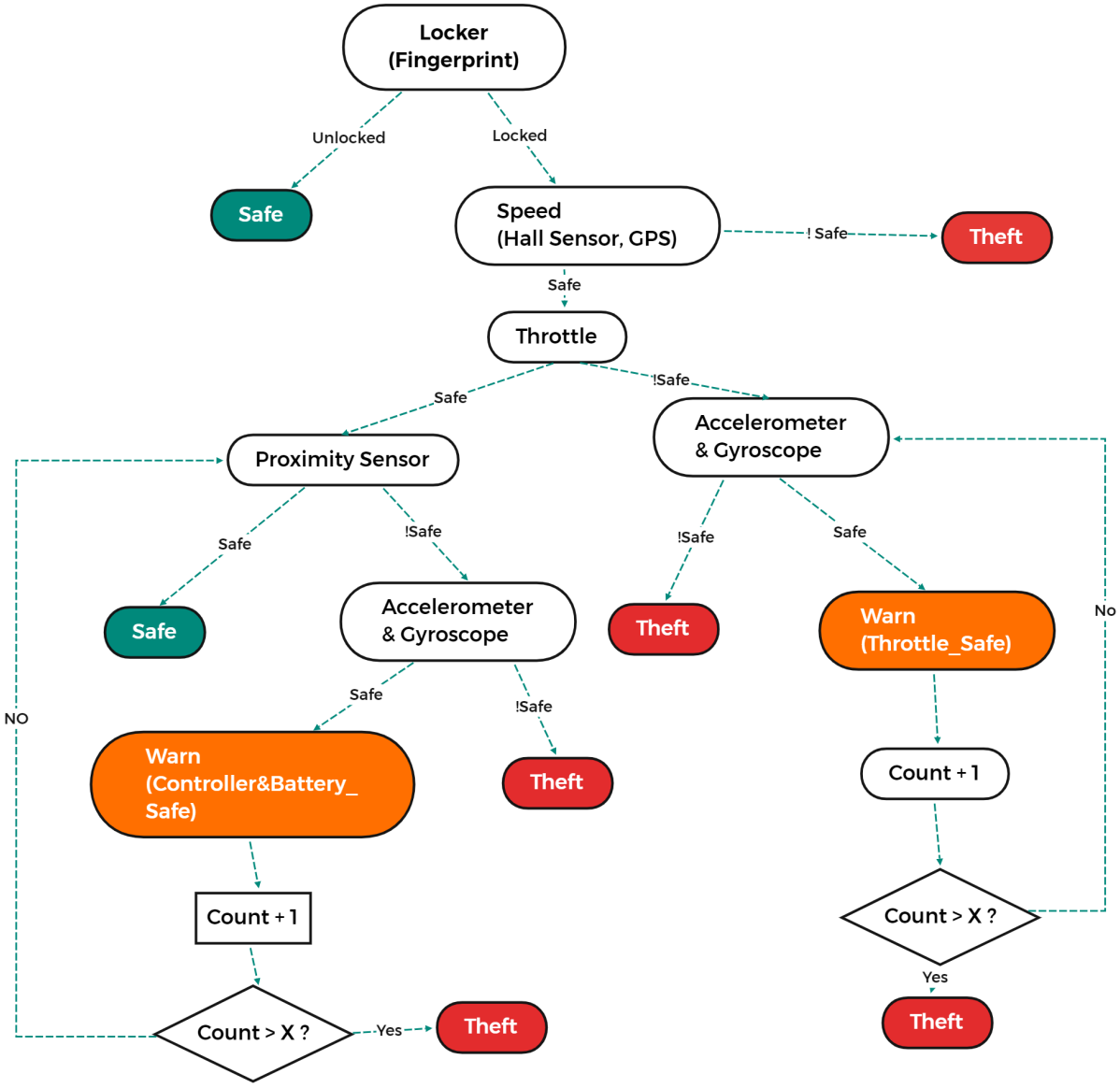

Fig. 19 Decision Tree of Main Arduino Board

The Main Arduino has more sensors and devices than the Supplementary Arduino, mainly for detecting the wheels and throttle and some devices that interact with the outside world. Therefore, the decision tree of the Main Arduino is more complex. At the beginning of the tree, the first node is used to read the flag of the fingerprint sensor. As the hardware devices detect the voltage of the Hall sensors on the wheels and the throttle, if these devices return anomalies, theft is likely to take place. The reason is that these sensors are embedded and therefore not easily damaged and have a low error rate. So, on balance, the flags of the two hardware devices are placed in the front node. After that, the decision tree will check the flags of all sensors and determine a specific consideration for each possible anomalous event flag and finally output

whether the theft action was taken place or not.

For the supplementary Arduino, the decision tree is more straightforward as fewer hardware devices are to be judged. The decision tree for the supplementary Arduino is shown in Figure 19.

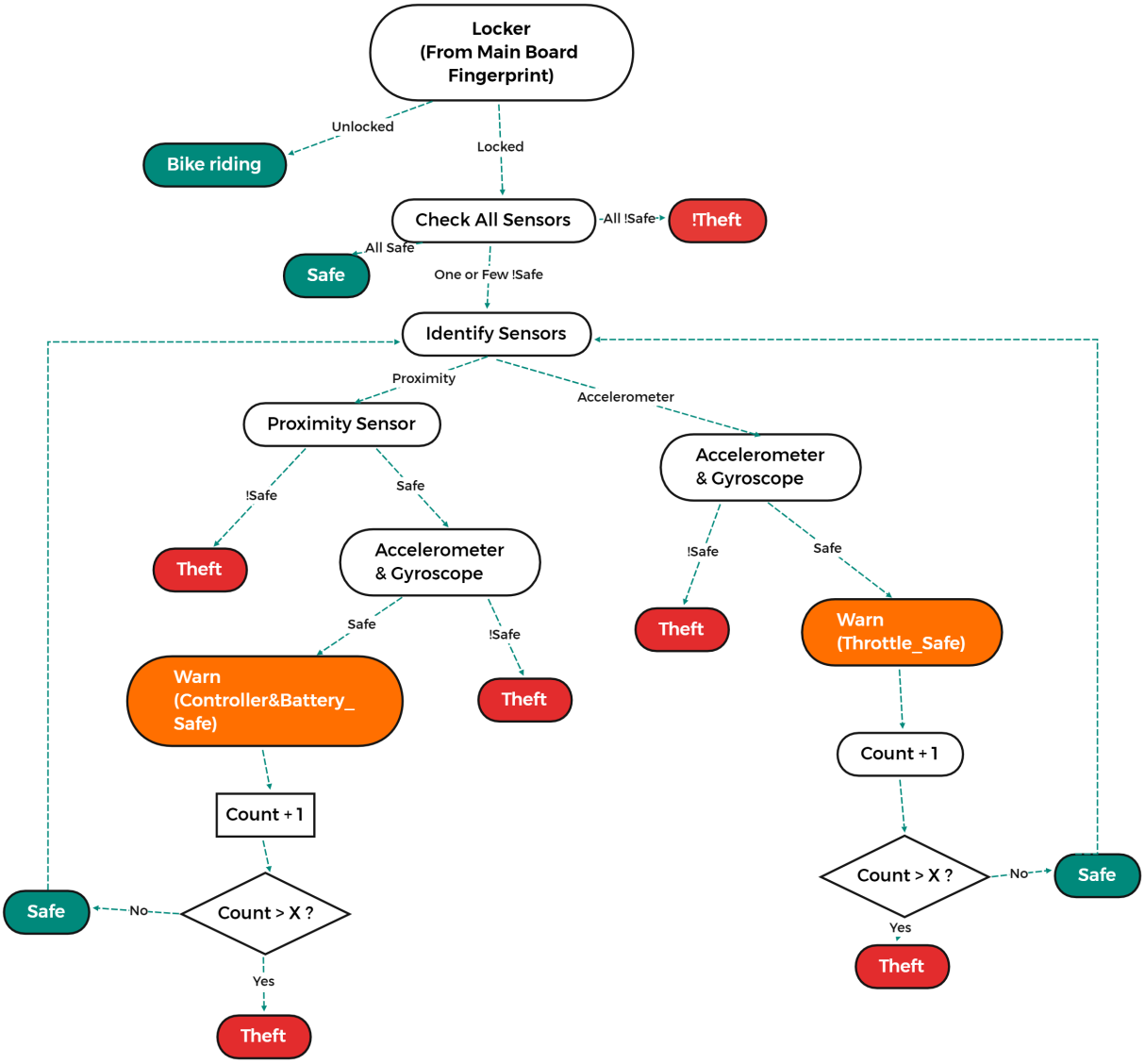

Fig. 20 Decision Tree for Supplementary Arduino Board

The very first decision node is still the fingerprint sensor, as the fingerprint sensor determines whether the user is riding or not. The flag value of this node will be transmitted by the Main Arduino using the Arduino BLE. If the flag value is false, the following nodes will not run. Conversely, if the flag is false, the following decision process will be similar to the main Arduino mentioned earlier, determining the flags of the two sensors and making a decision.

4.4. Response before/during/after the theft

If the decision tree determines that a theft has occurred, the 'theft()' function will be called to invoke the GNSS & GSM module and buzzer to interact with the outside world. The buzzer is first switched on to make an alarm, and then the latitude and longitude of the current location are obtained via the GNSS module and included in a text message to the authorised user's mobile phone, which also contains the specific sensor or device that flagged the anomaly, along with the current time. After a text message has been sent, the program will assign a 'SIM_TextMessage = 200' value and decrees its value each time it enters the 'theft()' function. This is to avoid repeatedly sending multiple messages to the authorised user's phone in a short period to consume excessive power consumption and fees.

4.5. Sensor Calibration and Baseline Data Acquisition for ToF and IMU

The ToF sensor uses a laser ranging method, which has much better accuracy and measuring range than infrared and ultrasonic ranging. Although the measuring error of ToF can be described as small, it is still considered and mitigated. In the program, the function ‘getSensorData’ aims to retrieve sensor data continuous. The global macro, ‘SENSOR_ROLLING,’ determines how many times the program needs to read the sensor data in succession, which is set as 15. After the value of the TOF sensor has been read, it needs to be analysed. A threshold value was set at 5cm, which was decided through repeated experiments and reference to previous papers [17]. When the value of the ToF sensor is less than this threshold, a flag will be triggered to indicate that the value of this sensor is abnormal.

The IMU calibration is more complex, as it is mentioned in section 'Gyroscope' that the gyroscope has drifted on-time problem, which has been considered and mitigated its effect. The function 'calibrate' in the header file 'EulerAngle' calibrates the gyroscope by calculating the average readings of the gyroscope in stationary and dividing by the number of calibrations to give the average drift for each reading. The average value of drift is integrated into the complementary filter algorithm and effectively reduces gyroscope error. A threshold value for each angle is applied as 5 degrees. Any changes of angle above 5 degrees will be flagged as abnormal.

4.6. Test under multiple conditions

A complete security system is built on a bicycle frame, as shown in Figure 21. The main Arduino is placed on the top of the frame, with a buzzer, fingerprint reader and a display module. These components should be placed close to the user as they are frequently used and accessed by the

user. The first supplementary Arduino is placed on the right side of the frame. This subsystem is used to monitor the steering of the front wheel and the proximity of objects. As the frame is not equipped with wheels, the two sensors used are freely hanging in this set-up - however, this should be fixed to the wheels in real applications to the e-bike. Similarly, a second supplementary Arduino is placed on the left side, which is used to protect the rear wheel.

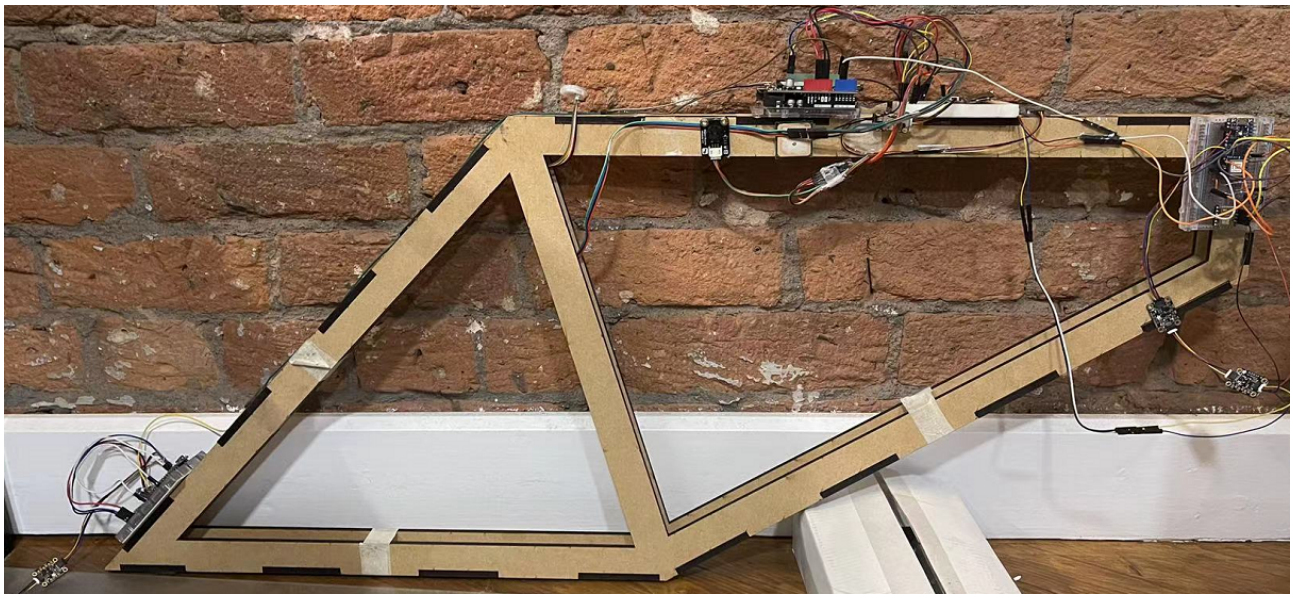

Fig. 21 Whole picture of the system

4.6.1 Component

In the test, all sensors were operating correctly and were able to calculate the complementary filter and output more accurate values. The IMU had an error within 2° of the resting state, and after experiencing movement, the error value is magnified to about 3° due to the gyroscope drift mentioned earlier. Figure 22 shows error values from IMU and ToF in stationary and stationary after experiencing movement, respectively. An additional IMU is used on the supplementary board to display accurate Euler angles and accommodate measurements at different angles to reduce the computational stress and increase the accuracy of reading on low-performance Arduino. The additional IMU has the Kalman filter algorithm onboard, which has been used. This algorithm can provide more accurate calculations of the Euler angles than the complementary filter.

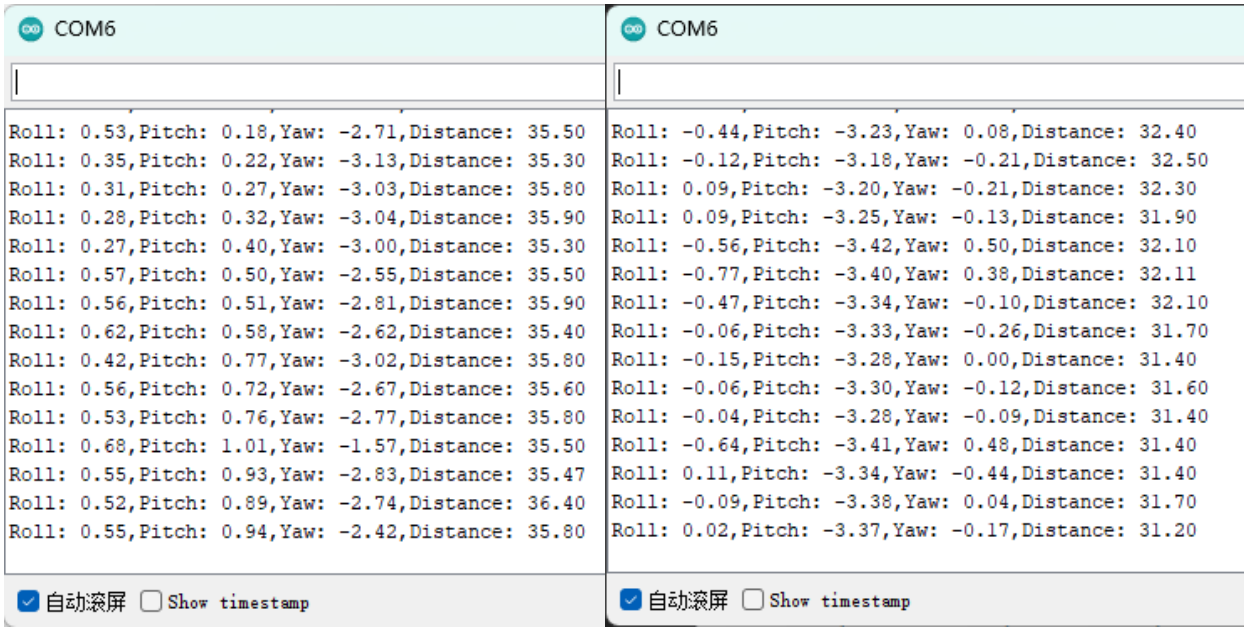

Fig. 22 IMU testing

The fingerprint sensor is positioned close to the seat to make it easier for the authorised user to unlock the e-bike. The fingerprint sensor has an integrated LED indicating the current mode and status, as shown in figure 23. When the finger is placed on the sensor for more than three seconds, the blue LED flashes quickly for three seconds to indicate that the sensor is in matching mode (figure 23.1); after the finger is removed, the sensor lights up green as success (figure 23.2) or red as failure (figure 23.3) depending on the result of the matching.

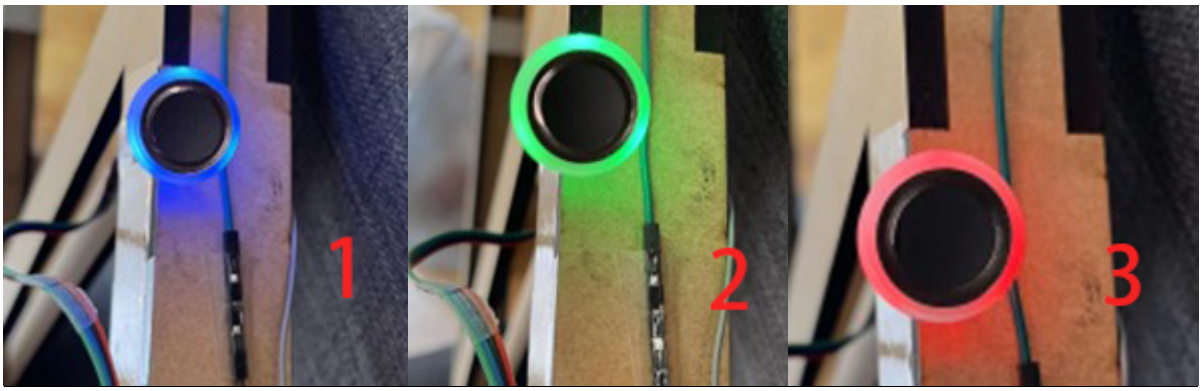

Fig. 23 Fingerprint module and matching mode

The display module, also placed above the frame, shows the current Euler angle and ToF readings, as shown in figure 24. This provides an easy access to the data which makes the testing procedure of the security system easier and more efficient.

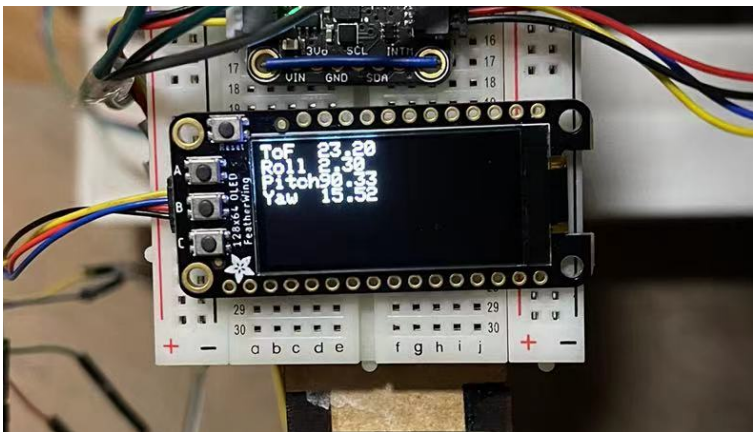

Fig. 24 Display Module

4.6.2 Decision Algorithm

The decision tree algorithm worked successfully and accurately during testing, reflecting the theft clearly and minimising the error rate. Figure 25 shows that the Arduino recognises when a ToF sensor is obscured or when an IMU detects a violent shake during the test.

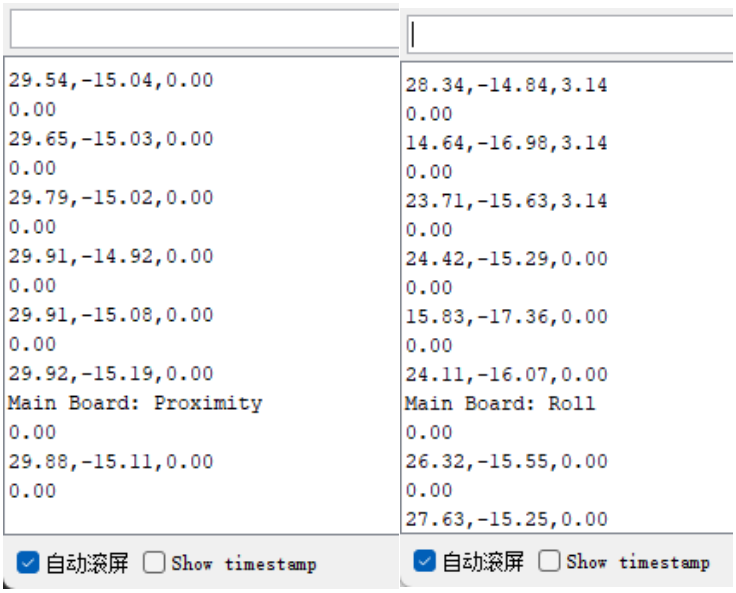

Fig. 25 Decision Algorithm Output

When a theft occurs in one of the subsystems, the main Arduino logs it and simultaneously prints in the serial monitor and sends an SMS from the SIM module to the user's mobile phone. Figure 26 shows the messages sent from the Arduino in the different scenarios tested, including the board number and which sensor caused it and when it occurred.

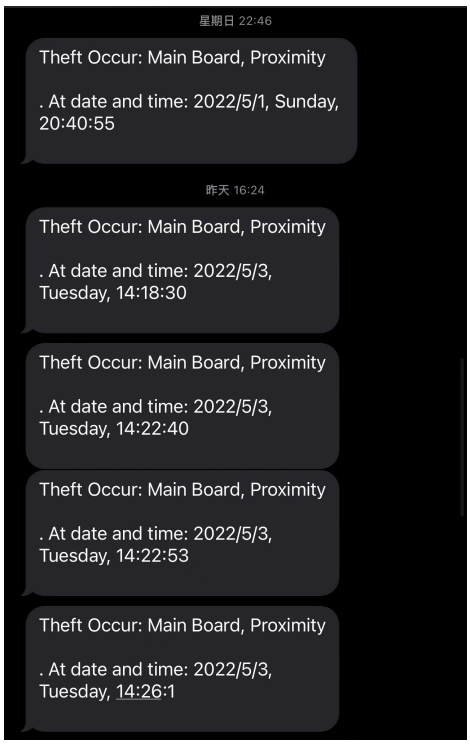

Fig. 26 Theft action message that alerts the user of a suspicious event

5. Result & Discussion

5.1. Result

The security system has been built and tested on a bicycle frame in multiple environments and scenarios, including shaking the bike and having the body close to the bike's chassis. The testing procedures show that the security system worked well in detecting theft events.

Also, the other functions in the security system worked well. For example, the fingerprint can return the correct status, the display module can show the essential data about the bike, and the SIM module can send a message to notify the occurrence of theft with sensor data. The accuracy of the sensors was excellent. ToF sensors have a negligible ranging error, and the IMU will have 2-3 degrees of angle measuring error in stationary, and it will have about 5-20 degrees of error when the IMU is kept in continuous motion.

5.2. Critical Analysis

Although almost all objectives set at the beginning of the project were achieved in this project, some shortcomings and deficiencies still need to be reflected upon. When considering the integration of the Arduino BLE, it was not possible to add the Arduino BLE library to the Main

Arduino due to the overflow of the flash memory caused by the inclusion of other code libraries for sensors and GNSS & GSM module in the mainboard program. This was caused by choosing an Arduino board with smaller storage at the very early stages of implementation without sufficient upfront knowledge about the required functionality and performance. This issue increased the time cost and detracted from the performance of the overall program. To overcome this issue, two solutions were considered:

1. Connecting an additional Arduino Nano 33 IoT to the main Arduino using a wired connection, allowing the Nano 33 IoT to communicate with the supplementary Arduino for BLE transfer data via the port with the main Arduino;
2. Connecting three Arduinos via the wired method and transmitting the voltage.

Finally, the wired way was chosen. A digital output pin has been used in supplementary Arduino to output the voltage, indicating the specific event that occurred in the Arduino. Due to the PWM output of the digital pin, an LPF circuit has been used to convert the voltage to a flat form. However, the wired connection cannot be used when the supplementary Arduino is powered on by external power (power bank, batteries). The reasons for this remain unknown, and the supplementary Arduino was powered via a computer. It is thought that this issue might be due to the current draw required by the supplementary Arduino, which the power bank can not supply. Future testing is needed to establish the cause of this issue.

Wired communication enhances the response time between Arduinos, which is much faster than using the BLE communication. However, the false alarming rate increased. The reason is that the LPF circuit cannot raise the voltage to the expected level instantly. So when the Arduino reads a voltage that is rising or falling, a false alarm occurs.

Overall, the IMU represents a high accuracy reading in the project testing. When writing the IMU fusion algorithm, it was not possible to calibrate the gyroscope in real-time, so only a one-time drift value was compensated for each time the Arduino was powered on, which resulted in a certain amount of drift and error in the pitch angle value when calculating the angles. Although the high-pass and low-pass weighting of the complementary filter had been optimised, drift and error are still unavoidable. The drift and error problem may be mitigated using a more complex filter algorithm such as the Kalman filter, mentioned in the 'Future Work' section.

In testing the decision tree, it was found that the decision tree that was written caused delays to

the Arduino program due to excessive judgement conditions and loops, which indirectly led to a bias in the calculation of angles in the IMU fusion algorithm and an overall degradation in the performance of the Arduino program.

The above issues were noted, and many of them are possible to address and solve in future work, where time and resources to produce a system prototype are not very limited, as was the case for this project. The ‘Future Work’ section will explore these areas and comment on optimising the security system's features.

6. Conclusion and Future Work

6.1. Conclusion

The expansion of the e-bike market has stimulated the development of e-bikes. At the same time, anti-theft solutions for bicycles deserve attention. However, according to data from the past few years, the development of the anti-theft system is at a standstill. Developing advanced, intelligent security for bicycles has become necessary in this context. This project focuses on collecting and analysing bicycle movement and distance data and notifying the user via SMS. Additional, a proven fingerprint unlocking solution is also integrated to ensure optimal security of the bicycle.

Overall, this project has developed and tested the bicycle security system. All the sensors and devices included in the security system are working correctly. The decision tree algorithm and IMU fusion algorithm can produce accurate results. The solution for communication between multiple Arduino is also available via BLE, allowing multiple Arduinos to connect and transfer data. After testing, all features work as expected. There were some minor errors during the project, and deficiencies were identified during the project and testing, summarised for reflection and improvement in the 'critical analysis' section. Due to time and budget factors, some features were left in the conceptual stage, and these ideas are included in the 'Future Work' section.

6.2. Future Work

Looking back at the stage project is at, some flaws and shortcomings are summarised in the 'critical analysis' section, and these shortcomings are expected to be addressed in other ways.

When choosing the main controller board, better expandability and more storage controller board should be chosen, such as Arduino Mega 2560, Arduino Due or the Raspberry Pi 4. These controllers provide much higher performance than Arduino Uno to include more libraries and

codes to host more functions in the project.

During computing the Euler angle on the IMU, a lack of accuracy and gyroscope drift was encountered. During the research progress, a complex, accurate gauss distribution and the two-dimensional matrix-based algorithm, Kalman filter, were considered for the project. Kalman filter deals effectively with the uncertainty due to noisy sensor data and, to some extent, with random external factors [18], which provides an accurate way for sensor fusion. Based on the 9-DOF IMU and GPS module data, the Kalman filter can be used to build an accurate inertial guidance system to monitor the bike's position, direction, and speed to provide a more comprehensive security system.

When optimising the performance of the decision tree algorithm, an inevitably think raised of training the decision tree model by using a sizeable labelled dataset in order to make a more accurate algorithm. The decision tree is also a basic machine learning algorithm. The decision tree is a supervised learning algorithm which does not require computer insight into additional unknown problems. The difference between the decision tree in machine learning and the one that used in this project is that it uses the calculation and ranking of the entropy and the calculation of the information gain of each node. Entropy is a measure of the uncertainty of random variables, which in this case can be interpreted as the measurement of the relationship between sensor and theft action. Equation 14 calculates the entropy by Shannon's entropy measure [19], where p_i is the probability that event i occurs.

$$E(S) = \sum_{i=1}^{n} p_i \log_2 p_i \tag{14}$$

Higher entropy represents a more significant uncertainty of the variable, i.e. a more negligible correlation with the occurrence of the theft. On the other hand, the information gain represents entropy reduction. Decision tree algorithms in machine learning use the increase of the entropy and information gain to select the parent node, which increases the algorithm's stability and performance. Furthermore, the random forest algorithm, which contains multiple decision trees, can be used instead of the decision tree algorithm. Alternatively, the results of all sensors measurement can be considered a whole part and use the multi-dimensional k-Nearest Neighbor (KNN) algorithm to cluster the multiple attributes referencing the classification tag. However, a high dimensionally set by the KNN will exponentially increase training time, so a dimension reduction needs to be used in advance [20].

Our current unlocking solution for bikes is the fingerprint, which is a very convenient way to lock and unlock the bike. NFC was considered a relatively safe and convenient way to unlock the bike but was not integrated into the system due to time issues. NFC detectors recognise the signal for a specific NFC tag and can take locking/unlocking actions. In the past, NFC was usually used for building access and people needed to carry a card with them, but nowadays, thanks to the technology, most mobile phones are equipped with NFC and open access so that authorised users can copy the NFC card tag to their smartphone and store it in a mobile wallet such as Apple Wallet. The use of NFC is a more innovative, more convenient way to unlock bicycles.

7. References

[1] LeadLeo, 'Annual production volume of electric two-wheelers in China from 2000 to 2020 (in million units).', *Statista*. https://www.statista.com/statistics/1208623/china-annual-production-of-e-bikes/ (accessed Apr. 04, 2022).

[2] 'Global e-bike market forecast by region 2028', *Statista*. https://www.statista.com/statistics/1260524/global-e-bike-market-forecast-by-region/ (accessed Apr. 04, 2022).

[3] 'Bicycle theft England and Wales 2021', *Statista*. https://www.statista.com/statistics/303562/bicycle-theft-in-england-and-wales-uk-y-on-y/ (accessed Apr. 02, 2022).

[4] 'France: reported bike thefts or attempted thefts 2006-2018', *Statista*. https://www.statista.com/statistics/949345/bike-thefts-or-attempted-thefts-france/ (accessed May 01, 2022).

[5] 'Opinion of cyclists on bike theft in France', *Statista*. https://www.statista.com/statistics/1218353/cyclist-opinion-bike-theft-france/ (accessed May 01, 2022).

[6] D. K. Zala, 'Bike Security with Theft Prevention', in *2018 3rd International Conference on Inventive Computation Technologies (ICICT)*, 2018, pp. 640–643. doi: 10.1109/ICICT43934.2018.9034252.

[7] A. Arjun and M. Mukhedkar, 'Advance Bike Security System', 2014. https://www.semanticscholar.org/paper/Advance-Bike-Security-System-Arjun-Mukhedkar/a42e05727ef35d8e2165e203065403414e87569d (accessed Apr. 04, 2022).

[8] V. Maleesha Kulasekara, I. Kavalchuk, and A. Smith, 'Smart Key System Design for Electric Bike for Vietnam Environment', in *2019 International Conference on System Science and Engineering (ICSSE)*, Jul. 2019, pp. 451–455. doi: 10.1109/ICSSE.2019.8823367.

[9] 'Arduino BLE library'. Arduino. [Online]. Available: https://www.arduino.cc/en/Reference/ArduinoBLE

[10] 'Introduction to Bluetooth Low Energy - GAP'. Adafruit. [Online]. Available: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap

[11] 'SIM7600CE-T_4G(LTE)_Shield_V1.0_SKU_TEL0124-DFRobot'. https://wiki.dfrobot.com/SIM7600CE-T_4G(LTE)_Shield_V1.0_SKU_TEL0124 (accessed Apr. 04, 2022).

[12] 'ID809 high-performance processor and semiconductor fingerprint sensor'. DFROBOT. [Online]. Available: https://wiki.dfrobot.com/Capacitive_Fingerprint_Sensor_SKU_SEN0348

[13] 'Hayes command set', *Wikipedia*. Oct. 31, 2021. Accessed: Apr. 03, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Hayes_command_set&oldid=1052901861

[14] L. D. Tran, 'Data Fusion with 9 degrees of freedom Inertial Measurement Unit to determine object's orientation', 2017.

[15] A. Noordin, M. A. M. Basri, and Z. Mohamed, 'Sensor fusion algorithm by complementary filter for attitude estimation of quadrotor with low-cost IMU', *Telkomnika*, vol. 16, no. 2, pp. 868–875, 2018.

[16] 'Decision tree', *Wikipedia*. Mar. 28, 2022. Accessed: Apr. 04, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Decision_tree&oldid=1079851424

[17] N. L. Buck and R. A. Aherin, 'Human presence detection by a capacitive proximity sensor', *Applied Engineering in Agriculture*, vol. 7, no. 1, pp. 55–60, Jan. 1991, doi: 10.13031/2013.26191.

[18] 'Kalman filter', *Wikipedia*. Apr. 15, 2022. Accessed: Apr. 27, 2022. [Online].

Available: https://en.wikipedia.org/w/index.php?title=Kalman_filter&oldid=1082910871
[19] Q. R. Wang and C. Y. Suen, ‘Analysis and Design of a Decision Tree Based on Entropy Reduction and Its Application to Large Character Set Recognition’, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 4, pp. 406–417, Jul. 1984, doi: 10.1109/TPAMI.1984.4767546.
[20] P. Ray, S. S. Reddy, and T. Banerjee, ‘Various dimension reduction techniques for high dimensional data analysis: a review’, *Artif Intell Rev*, vol. 54, no. 5, pp. 3473–3515, Jun. 2021, doi: 10.1007/s10462-020-09928-0.

8. Appendices

8.1. Appendix 1 Main Board Source Code

Main.cpp

```
#include <Arduino.h>
#include "DecisionAlgorithm.h"

void setup() {
   Serial.begin(9600);
   pinMode(A1, INPUT);
   pinMode(A2, INPUT);
   while (!Serial)
      delay(1);
   // BLE.begin();
   SIM7600CE_Setup();  // GSM & GPRS startup
   FingerprintSetup();
   // BLERecieverSetup();
   digitalWrite(BUZZER_PIN, LOW);
   vl53_Setup();
   IMUsetup();
   //displaySetup();
   RTCSetup();
   //GPSsetup();
}

void loop() {
   Algorithm();
   /*Locker();
   getSensorData();    // Get sensor data from the main board
   // getHardwareData();  // Get hardware data from the main board
   setFlag();          // set flags according to the sensor and
hardware data
   StopBuzzer();
   decisionTree();
   Clear();*/
}
```

BLEReciever.h

```
//
// Created by TedWu on 2022/3/19.
```

```
//

#ifndef MAINBOARD_BLERECIEVER_H
#define MAINBOARD_BLERECIEVER_H

#include "Arduino.h"
#include <ArduinoBLE.h>

BLEService SupplementaryFrame("19B10000-E8F2-537E-4F6C-
D104768A1214"); // Bluetooth® Low Energy LED Service
BLEService SupplementaryBack("19B20000-E8F2-537E-4F6C-
D104768A1214");
BLEService SupplementaryFront("19B30000-E8F2-537E-4F6C-
D104768A1214");

BLEByteCharacteristic proximityCharacteristic("19B10001-E8F2-537E-
4F6C-D104768A1214", BLERead | BLEWrite);
BLEByteCharacteristic accCharacteristic("19B10002-E8F2-537E-4F6C-
D104768A1214", BLERead | BLEWrite);
BLEByteCharacteristic gyroCharacteristic("19B10003-E8F2-537E-4F6C-
D104768A1214", BLERead | BLEWrite);
// BLEDevice peripheral;

bool Supplementary1Event[3] = {
      false, // 1: proximity
      false, // 1: acc
      false, // 1: gyro
};

void BLERecieverSetup() {
   // begin initialization
   if (!BLE.begin()) {
      Serial.println("starting Bluetooth® Low Energy module
failed!");

      while (1);
   }

   // set advertised local name and service UUID:
   BLE.setLocalName("Central");
   BLE.setAdvertisedService(SupplementaryFrame);

   // add the characteristic to the service
   SupplementaryFrame.addCharacteristic(proximityCharacteristic);
   SupplementaryFrame.addCharacteristic(accCharacteristic);
   SupplementaryFrame.addCharacteristic(gyroCharacteristic);
```

```
    // add service
    BLE.addService(SupplementaryFrame);

    // set the initial value for the characeristic:
    proximityCharacteristic.writeValue(0);
    accCharacteristic.writeValue(0);
    gyroCharacteristic.writeValue(0);

    // start advertising
    BLE.advertise();

    Serial.println("BLE Central");
}

void BLERecieve() {
    // listen for Bluetooth® Low Energy peripherals to connect:
    BLEDevice peripheral = BLE.central();

    // if a central is connected to peripheral:
    if (peripheral) {
        Serial.print("Connected to central: ");
        // print the central's MAC address:
        Serial.println(peripheral.address());

        // while the central is still connected to peripheral:
        if (peripheral.connected()) {
            // if the remote device wrote to the characteristic,
            // use the value to control the LED:
            if (proximityCharacteristic.written()) {
                if (proximityCharacteristic.value()) {  // any value
other than 0
                    Serial.println("Proximity");
                    Supplementary1Event[1] = true;
                }
            }
            if (accCharacteristic.written()) {
                if (accCharacteristic.value()) {
                    Serial.println("Accelerometer");
                    Supplementary1Event[2] = true;
                }
            }
            if (gyroCharacteristic.written()) {
                if (gyroCharacteristic.value()) {
                    Serial.println("Gyro");
                    Supplementary1Event[3] = true;
                }
            }
```

```
      }

      // when the central disconnects, print it out:
      // Serial.print(F("Disconnected from peripheral: "));
      // Serial.println(peripheral.address());
   }
}

#endif //MAINBOARD_BLERECIEVER_H
```

Buzzer.h

```
//
// Created by TedWu on 2022/3/13.
//

#ifndef MAINBOARD_BUZZER_H
#define MAINBOARD_BUZZER_H

#include "Arduino.h"

const int BUZZER_PIN = 2;

void StartBuzzer() {
   digitalWrite(BUZZER_PIN, HIGH);
}

void StopBuzzer() {
   digitalWrite(BUZZER_PIN, LOW);
}

#endif //MAINBOARD_BUZZER_H
```

DecisionAlgorithm.h

```
//
// Created by TedWu on 2022/3/30.
//

#ifndef MAINBOARDWIRED_DECISIONALGORITHM_H
#define MAINBOARDWIRED_DECISIONALGORITHM_H

#include "Arduino.h"
#include "buzzer.h"
#include "Fingerprint.h"
```

```
#include "VL53L1X.h"
#include "HallSensor.h"
#include "Throttle.h"
#include "SIM7600CE.h"
//#include "SIM7000C.h"
#include "RTC.h"
#include "EulerAngle.h"
#include "GPS.h"
#include "Display.h"


#define ROLL_THRESHOLD 3
#define PITCH_THRESHOLD 3
#define TOF_THRESHOLD 5
#define SENSOR_ROLLING 15
#define SIM_LIMIT_TIME 200
#define DECISION_ROLLING 10

bool eventFlag[7] = {false, false, false, false, false, false,
false};
/*
 * Table of eventFlag
 * 0. Fingerprint
 * 1. Hall Sensor
 * 2. Throttle Voltage
 * 3. ToF value
 * 4. Roll
 * 5. Pitch
 */
int SIM_TextTime = 0;
int ToF_Count = 0;
int rollCount = 0;
int pitchCount = 0;
int displayCount = 0;
float distance = 0;

/*
 * getSensorData();
 * input -> void
 * output-> void
 * Function:
 * Get the time-of-flight sensor data
 * Get the IMU data, convert to Euler Angle
 */
void getSensorData()
{
   //LSM6DS33();
```

```
    // setPrevious();
    /*for (int i = 0; i < 15; ++i) {
      LSM6DS33();
      distance += getDistance() / 10.0;
      accValue += AccelCalculateDifference();
      gyroValue += getGyro();
    }
    gyroValue = gyroValue / 15;
    distance = distance / 15;
    accValue = accValue / 15;*/

    /*Get distance*/
    for (int i = 0; i < SENSOR_ROLLING; ++i) {
       distance += getDistance() / 10.0;
    }
    distance = distance / SENSOR_ROLLING;
    /*Get Euler Angle*/
    getEulerAngle();
    /*Print them on the IDE*/
    /*
    Serial.print("ToF: ");
    Serial.println(distance);
    Serial.print("Roll: ");
    Serial.println(complementaryFilter_roll);
    Serial.print("Pitch: ");
    Serial.println(complementaryFilter_pitch);
    */
    /*Print them on the display*/
    /*if (displayCount == 0) {
       display.clearDisplay();
       display.setCursor(0,0);
       display.print("ToF\nRoll\nPitch");
      display.setCursor(30,0);
      display.print(distance);
      display.setCursor(30,8);
      display.print(complementaryRoll);
      display.setCursor(30,16);
      display.print(complementaryPitch);
      display.display();
       displayCount = 5;
    } else {displayCount--;}*/
}

/*
 * setFlag();
 * input -> void
 * output-> void
```

```
 * Function:
 *
 */
void setFlag() {
   /*Fingerprint*/
   if (isLocked)
      eventFlag[0] = true;
   else
      eventFlag[0] = false;

   /*Hall Sensor*/
   /*
   bool CurrentHall, PrevHall;
   if (HardwareData[1] <= 2)
     CurrentHall = false;
   else if (HardwareData[1] >= 4)
     CurrentHall = true;
   if (HardwareData[2] <= 2)
     PrevHall = false;
   else if (HardwareData[2] >=4)
     PrevHall = true;
   if (CurrentHall != PrevHall) {
     if (HallCount >= 0)
      HallCount++;
   }
   else {
     if (HallCount > 0)
      HallCount -= 0.1;
   }
   if (HallCount >= 15) {
     eventFlag[1] = true;
     HallCount = 0;
   }
   */

   /*Throttle*/
   /*
   if (HardwareData[0] >= 1.8)
     eventFlag[2] = true;
   */

   /*Proximity*/
   if (distance <= TOF_THRESHOLD && distance >= 0)
      eventFlag[3] = true;
   else if (distance > TOF_THRESHOLD)
      eventFlag[3] = false;
```

```
    /*Roll*/
    if (abs((int)(diff_complementaryRoll)) >= ROLL_THRESHOLD) {
        eventFlag[4] = true;
    } else {
        eventFlag[4] = false;
    }

    /*Pitch*/
    if (abs((int)(diff_complementaryPitch)) >= PITCH_THRESHOLD)
        eventFlag[5] = true;
    else {
        eventFlag[5] = false;
    }
}

/*
 * MainBoardTheft();
 * input -> void
 * output-> void
 * Function:
 *
 */
void MainBoardTheft(int charEventFlag) {
    StartBuzzer();
    //getGPS();
    // Serial.println(AlarmFlag[pointer]);
    Serial.print("Main Board: ");
    switch (charEventFlag) {
        case 1:
            Serial.println("Hall Sensor");
            SendMessage(0, 1);
            break;
        case 2:
            Serial.println("Throttle");
            SendMessage(0, 2);
            break;
        case 3:
            Serial.println("Hall Sensor & Throttle");
            SendMessage(0, 3);
            break;
        case 4:
            Serial.println("Proximity & Roll & Pitch");
            SendMessage(0, 4);
            break;
        case 5:
            Serial.println("Proximity & Roll");
            SendMessage(0, 5);
```

```
            break;
        case 6:
            Serial.println("Proximity & Pitch");
            SendMessage(0, 6);
            break;
        case 7:
            Serial.println("Roll & Pitch");
            SendMessage(0, 7);
            break;
        case 8:
            Serial.println("Proximity");
            SendMessage(0, 8);
            break;
        case 9:
            Serial.println("Roll");
            SendMessage(0, 9);
            break;
        case 10:
            Serial.println("Pitch");
            SendMessage(0, 10);
            break;
    }
    //printGPS();
    // Serial.println(charEventFlag);
    if (SIM_TextTime == 0) {
        SIM_TextTime = SIM_LIMIT_TIME;
        // SendMessage('Main Board', charEventFlag);
    }
}

void decisionTree() {
    if (eventFlag[0]/*Fingerprint*/) {
        if (eventFlag[1]/*Hall Sensor*/) {
            if (eventFlag[2]/*Throttle*/) {
                //Serial.println("Hall sensor & Throttle");
                MainBoardTheft(3);
            } else {
                //Serial.println("Hall sensor");
                MainBoardTheft(1);
            }
        } else if (eventFlag[2]/*Throttle*/) {
            if (eventFlag[1]/*Hall Sensor*/) {
                //Serial.println("Hall sensor & Throttle");
                MainBoardTheft(3);
            } else {
                //Serial.println("Throttle");
                MainBoardTheft(2);
```

```
        }
      } else { /*Check all sensor*/
        if (eventFlag[3] /*ToF*/ && eventFlag[4] /*Roll*/ &&
eventFlag[5]/*Pitch*/) {
          //Serial.println("tof & Roll & Pitch");
          MainBoardTheft(4);
        }
          // Check each sensor
          // check ToF starts
        else if(eventFlag[3]/*ToF*/) { //check each sensor
          if (eventFlag[4]/*Roll*/) {
            if (eventFlag[5]/*Pitch*/) {
              //Serial.println("tof & Roll & Pitch");
              MainBoardTheft(4);
            }
            else {
              //Serial.println("tof & Roll");
              MainBoardTheft(5);
            }
          } else if (eventFlag[5]/*Pitch*/) {
            if (eventFlag[4]/*Roll*/) {
              //Serial.println("tof & Roll & Pitch");
              MainBoardTheft(4);
            } else {
              //Serial.println("tof & Pitch");
              MainBoardTheft(6);
            }
          } else {
            ToF_Count++;
            if (ToF_Count >= DECISION_ROLLING) {
              //Serial.println("tof");
              MainBoardTheft(8);
              ToF_Count = 0;
            }
          }
        }
          // ToF part ends
          // check Roll starts
        else if(eventFlag[4]/*Roll*/) {
          if (eventFlag[3]/*ToF*/) {
            //Serial.println("tof & Roll");
            MainBoardTheft(5);
          } else if (eventFlag[5]/*Pitch*/) {
            //Serial.println("Roll & Pitch");
            MainBoardTheft(7);
          } else {
            rollCount++;
```

```
                if (rollCount >= DECISION_ROLLING) {
                    //Serial.println("Roll");
                    MainBoardTheft(9);
                    rollCount = 0;
                }
            }
        }
            // acc part ends
            // check Pitch starts
        else if(eventFlag[5]/*Pitch*/) {
            if (eventFlag[3]/*ToF*/) {
                //Serial.println("tof & Pitch");
                MainBoardTheft(6);
            } else if (eventFlag[4]/*Roll*/) {
                //Serial.println("Roll & Pitch");
                MainBoardTheft(7);
            } else {
                pitchCount++;
                if (pitchCount >= DECISION_ROLLING) {
                    //Serial.println("Pitch");
                    MainBoardTheft(10);
                    pitchCount = 0;
                }
            }
        }
    }
    } else {
        Serial.println("Bike riding");
        /*if (displayCount == 0) {
          display.setCursor(40,32);
          display.print("Bike riding");
          display.display();
        }*/
    }
}

void Clear() {
    // StopBuzzer();
    if(SIM_TextTime > 0)
        SIM_TextTime--;
    // Clear Event Flag
    for (bool & i : eventFlag) {
        i = false;
    }
    /*for (int i = 0; i < 3; ++i) {
      Supplementary1Event[i] = false;
      Supplementary2Event[i] = false;
```

```
      Supplementary3Event[i] = false;
    }*/
    distance = 0;
}

void readSupplementaryBoardTheft(int supplementaryBoard) {
    float supplementaryVoltage = 0;
    switch (supplementaryBoard) {
        case 1:
            supplementaryVoltage = analogRead(A1) / 4;
            break;
        case 2:
            supplementaryVoltage = analogRead(A2) / 4;
            break;
    }
    int supplementaryEvent = 0;
    //Serial.println(supplementaryVoltage);
    if (supplementaryVoltage >= 15) { /*If the supplementary voltage
is non-zero*/

        /*Start classify the voltage*/
        if ( supplementaryVoltage >= 20  && supplementaryVoltage <=
25  ) {
            supplementaryEvent = 1;
        }
        else if(supplementaryVoltage >= 30  && supplementaryVoltage
<= 35) {
            supplementaryEvent = 2;
        }
        else if(supplementaryVoltage >= 41  && supplementaryVoltage
<= 49) {
            supplementaryEvent = 3;
        }
        else if(supplementaryVoltage >= 55  && supplementaryVoltage
<= 60) {
            supplementaryEvent = 4;
        }
        else if(supplementaryVoltage >= 69  && supplementaryVoltage
<= 74) {
            supplementaryEvent = 5;
        }
        else if(supplementaryVoltage >= 82  && supplementaryVoltage
<= 87) {
            supplementaryEvent = 6;
        }
        else if(supplementaryVoltage >= 95  && supplementaryVoltage
<= 100) {
```

```
            supplementaryEvent = 7;
        }

        /*Classify the Event and send message*/
        Serial.print("Supplementary board ");
        Serial.print(supplementaryBoard);
        switch (supplementaryEvent) {
            case 1:
                Serial.println("Proximity & Roll & Pitch");
                break;
            case 2:
                Serial.println("Proximity & Roll");
                break;
            case 3:
                Serial.println("Proximity & Pitch");
                break;
            case 4:
                Serial.println("Roll & Pitch");
                break;
            case 5:
                Serial.println("Proximity");
                break;
            case 6:
                Serial.println("Roll");
                break;
            case 7:
                Serial.println("Pitch");
                break;
        }
    }
}

void Algorithm() {
     //int start = millis();
     Locker();
    getSensorData();    // Get sensor data from the main board
     printCalculations();
     Serial.print("Distance: ");
     Serial.println(distance);
    // getHardwareData();  // Get hardware data from the main board
    setFlag();          // set flags according to the sensor and
hardware data
    StopBuzzer();
    decisionTree();
    readSupplementaryBoardTheft(1);
    Clear();
     //int gap = millis() - start;
```

```
    //Serial.println(gap);
   //getGPS();
}

#endif //MAINBOARDWIRED_DECISIONALGORITHM_H
```

Display.h

```
//
// Created by TedWu on 2022/3/24.
//

#ifndef MAINBOARD_DISPLAY_H
#define MAINBOARD_DISPLAY_H
#include "Arduino.h"
#include "SPI.h"
#include "Wire.h"
#include "Adafruit_GFX.h"
#include "Adafruit_SH110X.h"
Adafruit_SH1107 display = Adafruit_SH1107(64, 128, &Wire);

void displaySetup() {
   Serial.println("128x64 OLED FeatherWing test");
   delay(250); // wait for the OLED to power up
   display.begin(0x3C, true);
   Serial.println("OLED begun");
   display.display();
   delay(1000);
   display.clearDisplay();
   display.display();
   display.setRotation(1);
   Serial.println("Button test");
   display.setTextSize(1);
   display.setTextColor(SH110X_WHITE);
   //display.setCursor(0,0);
   //display.print("ToF\nRoll\nPitch");
   //display.display(); // actually display all of the above
}


#endif //MAINBOARD_DISPLAY_H
```

EulerAngle.h

```
      //
// Created by TedWu on 2022/4/16.
//

#ifndef MAINBOARDWIRED_EULERANGLE_H
#define MAINBOARDWIRED_EULERANGLE_H

#include "Arduino.h"
#include "Adafruit_LSM6DS33.h"
#include "Adafruit_LIS3MDL.h"
#define HP 0.6
#define LP 0.4
#define RADIAN_DEGREE 180/M_PI

Adafruit_LIS3MDL Mag;
Adafruit_LSM6DS33 AccGyro;
sensors_event_t accel, gyro, temp;

float accelX, accelY, accelZ;
float gyroscope_x, gyroscope_y, gyroscope_z;
float drift_x, drift_y, drift_z;
float gyroscope_roll, gyroscope_pitch, gyroscope_yaw;
float gyroscope_afterDrift_Roll, gyroscope_afterDrift_Pitch,
gyroscope_afterDrift_Yaw;
float accelerometer_roll, accelerometer_pitch;
float complementaryFilter_roll, complementaryFilter_pitch,
complementaryFilter_yaw;
float magnetometer_x, magnetometer_y, magnetometer_z;
float acc_mag_CF_x, acc_mag_CF_y, magnetometer_yaw;
float pre_complementaryRoll, pre_complementaryPitch,
pre_complementaryYaw;
float diff_complementaryRoll, diff_complementaryPitch,
diff_complementaryYaw;

long lastTime;
long lastInterval;

bool readIMU() {
   AccGyro.getEvent(&accel, &gyro, &temp);
   Mag.read();
   accelX = accel.acceleration.x;
   accelY = accel.acceleration.y;
   accelZ = accel.acceleration.z;
   gyroscope_x = gyro.gyro.x;
   gyroscope_y = gyro.gyro.y;
   gyroscope_z = gyro.gyro.z;
   magnetometer_x = Mag.x;
```

```
   magnetometer_y = Mag.y;
   magnetometer_z = Mag.z;
   return true;
}

void calibrate(int delayTime, int calibrationTime) {
   int calibrationCount = 0;
   delay(delayTime);
   float x, y, z;
   int startTime = millis();
   while (millis() < startTime + calibrationTime) {
      if (readIMU()) {
         x += gyroscope_x;
         y += gyroscope_y;
         z += gyroscope_z;
         calibrationCount++;
      }
   }

   if (calibrationCount == 0) {
      Serial.println("Failed to calibrate");
   }

   drift_x = x / calibrationCount;
   drift_y = y / calibrationCount;
   drift_z = z / calibrationCount;

}

void doCalculations() {
   accelerometer_roll = atan2(accelY, accelZ) * RADIAN_DEGREE;
   accelerometer_pitch = atan2(-accelX, sqrt(accelY * accelY +
accelZ * accelZ)) * RADIAN_DEGREE;

   float lastFrequency = (float) 1000000.0 / lastInterval;
   gyroscope_roll = gyroscope_roll + (gyroscope_x / lastFrequency);
   gyroscope_pitch = gyroscope_pitch + (gyroscope_y /
lastFrequency);
   gyroscope_yaw = gyroscope_yaw + (gyroscope_z / lastFrequency);

   gyroscope_afterDrift_Roll = gyroscope_afterDrift_Roll +
((gyroscope_x - drift_x) / lastFrequency);
   gyroscope_afterDrift_Pitch = gyroscope_afterDrift_Pitch +
((gyroscope_y - drift_y) / lastFrequency);
   // gyroscope_afterDrift_Yaw = gyroscope_afterDrift_Yaw +
((gyroscope_z - drift_z) / lastFrequency);
```

```
   acc_mag_CF_x = magnetometer_x * cos(accelerometer_pitch) +
magnetometer_y * sin(accelerometer_pitch) - magnetometer_z *
cos(accelerometer_roll) * sin(accelerometer_roll);
   acc_mag_CF_y = magnetometer_y * cos(accelerometer_roll) +
magnetometer_z * sin(accelerometer_roll);
   magnetometer_yaw = atan2(acc_mag_CF_y, acc_mag_CF_x);
   complementaryFilter_yaw = complementaryFilter_yaw +
((gyroscope_z - drift_z) / lastFrequency);

   complementaryFilter_roll = HP * (complementaryFilter_roll +
gyroscope_afterDrift_Roll) + LP * accelerometer_roll;
   complementaryFilter_pitch = HP * (complementaryFilter_pitch +
gyroscope_afterDrift_Pitch) + LP * accelerometer_pitch;
   //complementaryFilter_yaw = magnetometer_yaw;

   diff_complementaryRoll = complementaryFilter_roll -
pre_complementaryRoll;
   diff_complementaryPitch = complementaryFilter_pitch -
pre_complementaryPitch;
   diff_complementaryYaw = complementaryFilter_yaw -
pre_complementaryYaw;
}

void printCalculations() {
  Serial.print("Roll: ");
   Serial.print(complementaryFilter_roll);
   Serial.print(',');
  Serial.print("Pitch: ");
   Serial.print(complementaryFilter_pitch);
   Serial.print(',');
  Serial.print("Yaw: ");
   Serial.print(magnetometer_yaw);
  Serial.print(',');
   //Serial.println("");
}

void setPrevious() {
   pre_complementaryRoll = complementaryFilter_roll;
   pre_complementaryPitch = complementaryFilter_pitch;
   pre_complementaryYaw = complementaryFilter_yaw;
}

void IMUsetup() {
   while (!AccGyro.begin_I2C()) {
      Serial.println("Cannot find Accgyro chip");
   }
 while (!Mag.begin_I2C()) {
```

```
  Serial.println("Cannot find Mag chip");
 }
   calibrate(250, 250);
   lastTime = micros();
}

void getEulerAngle() {
   if (readIMU()) {
      setPrevious();
      long currentTime = micros();
      lastInterval = currentTime - lastTime;
      lastTime = currentTime;
      doCalculations();
      //printCalculations();
   }
}

#endif //MAINBOARDWIRED_EULERANGLE_H
```

Fingerprint.h

```
//
// Created by TedWu on 2022/3/15.
//

#ifndef MAINBOARD_FINGERPRINT_H
#define MAINBOARD_FINGERPRINT_H

#include <DFRobot_ID809_I2C.h>
#define COLLECT_NUMBER 3  //Fingerprint sampling times, can be set
to 1-3
#define IRQ         6  //IRQ pin
DFRobot_ID809_I2C fingerprint;
bool isLocked = true;
//Compare fingerprints
void fingerprintMatching(){
   /*Compare the captured fingerprint with all fingerprints in the
fingerprint library
     Return fingerprint ID number(1-80) if succeed, return 0 when
failed
    */
   uint8_t ret = fingerprint.search();
   if(ret != 0){
      /*Set fingerprint LED ring to always ON in green*/
      fingerprint.ctrlLED(/*LEDMode = */fingerprint.eKeepsOn,
/*LEDColor = */fingerprint.eLEDGreen, /*blinkCount = */0);
```

```
      Serial.print("Successfully matched,ID=");
      Serial.println(ret);
      isLocked = !isLocked;
   }else{
      /*Set fingerprint LED Ring to always ON in red*/
      fingerprint.ctrlLED(/*LEDMode = */fingerprint.eKeepsOn,
/*LEDColor = */fingerprint.eLEDRed, /*blinkCount = */0);
      Serial.println("Matching failed");
   }
   delay(1000);
   /*Turn off fingerprint LED Ring*/
   fingerprint.ctrlLED(/*LEDMode = */fingerprint.eNormalClose,
/*LEDColor = */fingerprint.eLEDBlue, /*blinkCount = */0);
   Serial.println("-----------------------------");
}

//Fingerprint Registration
void fingerprintRegistration(){
   uint8_t ID,i;
   /*Compare the captured fingerprint with all fingerprints in the
fingerprint library
     Return fingerprint ID number(1-80) if succeed, return 0 when
failed
     Function: clear the last captured fingerprint image
    */
   fingerprint.search();       //Can add "if else" statement to
judge whether the fingerprint has been registered.
   /*Get a unregistered ID for saving fingerprint
     Return ID number when succeed
     Return ERR_ID809 if failed
    */
   if((ID = fingerprint.getEmptyID()) == ERR_ID809){
      while(1){
         /*Get error code imformation*/
         //desc = fingerprint.getErrorDescription();
         //Serial.println(desc);
         delay(1000);
      }
   }
   Serial.print("Unregistered ID,ID=");
   Serial.println(ID);
   i = 0;   //Clear sampling times
   /*Fingerprint Sampling 3 times */
   while(i < COLLECT_NUMBER){
      /*Set fingerprint LED ring to breathing lighting in blue*/
      fingerprint.ctrlLED(/*LEDMode = */fingerprint.eBreathing,
/*LEDColor = */fingerprint.eLEDBlue, /*blinkCount = */0);
```

```
      Serial.print("The fingerprint sampling of the");
      Serial.print(i+1);
      Serial.println("(th) time is being taken");
      Serial.println("Please press down your finger");
      /*Capture fingerprint image, 10s idle timeout
        If succeed return 0, otherwise return ERR_ID809
       */
      if((fingerprint.collectionFingerprint(/*timeout = */10)) !=
ERR_ID809){
         /*Set fingerprint LED ring to quick blink in yellow 3
times*/
         fingerprint.ctrlLED(/*LEDMode = */fingerprint.eFastBlink,
/*LEDColor = */fingerprint.eLEDYellow, /*blinkCount = */3);
         Serial.println("Capturing succeeds");
         i++;   //Sampling times +1
      }else{
         Serial.println("Capturing fails");
         /*Get error code information*/
         //desc = fingerprint.getErrorDescription();
         //Serial.println(desc);
      }
      Serial.println("Please release your finger");
      /*Wait for finger to release
        Return 1 when finger is detected, otherwise return 0
       */
      while(fingerprint.detectFinger());
   }

   /*Save fingerprint information into an unregistered ID*/
   if(fingerprint.storeFingerprint(/*Empty ID = */ID) !=
ERR_ID809){
      Serial.print("Saving succeed,ID=");
      Serial.println(ID);
      /*Set fingerprint LED ring to always ON in green*/
      fingerprint.ctrlLED(/*LEDMode = */fingerprint.eKeepsOn,
/*LEDColor = */fingerprint.eLEDGreen, /*blinkCount = */0);
      delay(1000);
      /*Turn off fingerprint LED ring */
      fingerprint.ctrlLED(/*LEDMode = */fingerprint.eNormalClose,
/*LEDColor = */fingerprint.eLEDBlue, /*blinkCount = */0);
   }else{
      Serial.println("Saving failed");
      /*Get error code information*/
      //desc = fingerprint.getErrorDescription();
      //Serial.println(desc);
   }
   Serial.println("----------------------------");
```

```
}

//Fingerprint deletion
void fingerprintDeletion(){
   uint8_t ret;
   /*Compare the captured fingerprint with all the fingerprints in
the fingerprint library
     Return fingerprint ID(1-80) if succeed, return 0 when failed
    */
   ret = fingerprint.search();
   if(ret){
      /*Set fingerprint LED ring to always ON in green*/
      fingerprint.ctrlLED(/*LEDMode = */fingerprint.eKeepsOn,
/*LEDColor = */fingerprint.eLEDGreen, /*blinkCount = */0);
      fingerprint.delFingerprint(ret);
      Serial.print("deleted fingerprint,ID=");
      Serial.println(ret);
   }else{
      /*Set fingerprint LED ring to always ON in red*/
      fingerprint.ctrlLED(/*LEDMode = */fingerprint.eKeepsOn,
/*LEDColor = */fingerprint.eLEDRed, /*blinkCount = */0);
      Serial.println("Matching failed or the fingerprint hasn't
been registered");
   }
   delay(1000);
   /*Turn off fingerprint LED ring*/
   fingerprint.ctrlLED(/*LEDMode = */fingerprint.eNormalClose,
/*LEDColor = */fingerprint.eLEDBlue, /*blinkCount = */0);
   Serial.println("-----------------------------");
}

void FingerprintSetup() {
   fingerprint.begin();
   while(!fingerprint.isConnected()){
      Serial.println("Communication with device failed, please
check connection");
      /*Get error code information*/
      //desc = fingerprint.getErrorDescription();
      //Serial.println(desc);
      delay(1000);
   }
}

void Fingerprint() {
   if(digitalRead(IRQ)){
      uint16_t i = 0;
      /*Capture fingerprint image, 5s idle timeout
```

```
        Return 0 if succeed, otherwise return ERR_ID809
       */
      if((fingerprint.collectionFingerprint(/*timeout=*/5)) !=
ERR_ID809){
         /*Get the time finger pressed down*/
         /*Set fingerprint LED ring mode, color, and number of
blinks
           Can be set as follows:
           Parameter 1:<LEDMode>
           eBreathing   eFastBlink   eKeepsOn    eNormalClose
           eFadeIn      eFadeOut     eSlowBlink
           Paramerer 2:<LEDColor>
           eLEDGreen  eLEDRed      eLEDYellow   eLEDBlue
           eLEDCyan   eLEDMagenta  eLEDWhite
           Parameter 3:<number of blinks> 0 represents blinking all
the time
           This parameter will only be valid in mode eBreathing,
eFastBlink, eSlowBlink
          */
         fingerprint.ctrlLED(/*LEDMode = */fingerprint.eFastBlink,
/*LEDColor = */fingerprint.eLEDBlue, /*blinkCount = */3);  //blue
LED blinks quickly 3 times, means it's in fingerprint comparison
mode now
         /*Wait for finger to relase */
         while(fingerprint.detectFinger()){
            delay(50);
            i++;
            if(i == 15){             //Yellow LED blinks quickly 3
times, means it's in fingerprint regisrtation mode now
               /*Set fingerprint LED ring to always ON in yellow*/
               fingerprint.ctrlLED(/*LEDMode =
*/fingerprint.eFastBlink, /*LEDColor = */fingerprint.eLEDYellow,
/*blinkCount = */3);
            }else if(i == 30){      //Red LED blinks quickly 3
times, means it's in fingerprint deletion mode now
               /*Set fingerprint LED ring to always ON in red*/
               fingerprint.ctrlLED(/*LEDMode =
*/fingerprint.eFastBlink, /*LEDColor = */fingerprint.eLEDRed,
/*blinkCount = */3);
            }
         }
      }
      if(i == 0){
         /*Fingerprint capturing failed*/
      }else if(i > 0 && i < 15){
         Serial.println("Enter fingerprint comparison mode");
         /*Compare fingerprints*/
```

```
            fingerprintMatching();
        }else if(i >= 15 && i < 30){
            Serial.println("Enter fingerprint registration mode");
            /*Registrate fingerprint*/
            fingerprintRegistration();
        }else{
            Serial.println("Enter fingerprint deletion mode");
            /*Delete this fingerprint*/
            fingerprintDeletion();
        }
    }
}

uint16_t k = 0;
void Locker() {
    uint16_t j = 0;
    if (fingerprint.detectFinger()) {
        Serial.println("Finger detected");
        k++;
        if (k == 3) {
            /*Capture fingerprint image, 5s idle timeout
          Return 0 if succeed, otherwise return ERR_ID809
         */
            if((fingerprint.collectionFingerprint(/*timeout=*/3)) !=
ERR_ID809){
              /*Get the time finger pressed down*/
              /*Set fingerprint LED ring mode, color, and number of
blinks
                Can be set as follows:
                Parameter 1:<LEDMode>
                eBreathing   eFastBlink   eKeepsOn    eNormalClose
                eFadeIn      eFadeOut     eSlowBlink
                Paramerer 2:<LEDColor>
                eLEDGreen  eLEDRed      eLEDYellow   eLEDBlue
                eLEDCyan   eLEDMagenta  eLEDWhite
                Parameter 3:<number of blinks> 0 represents blinking
all the time
                This parameter will only be valid in mode eBreathing,
eFastBlink, eSlowBlink
               */
              fingerprint.ctrlLED(/*LEDMode =
*/fingerprint.eFastBlink, /*LEDColor = */fingerprint.eLEDBlue,
/*blinkCount = */3);  //blue LED blinks quickly 3 times, means it's
in fingerprint comparison mode now
              /*Wait for finger to relase */
              while(fingerprint.detectFinger()){
                 delay(50);
```

```
                j++;
                if(j == 15){              //Yellow LED blinks quickly
3 times, means it's in fingerprint regisrtation mode now
                   /*Set fingerprint LED ring to always ON in
yellow*/
                   fingerprint.ctrlLED(/*LEDMode =
*/fingerprint.eFastBlink, /*LEDColor = */fingerprint.eLEDYellow,
/*blinkCount = */3);
                }else if(j == 30){      //Red LED blinks quickly 3
times, means it's in fingerprint deletion mode now
                   /*Set fingerprint LED ring to always ON in red*/
                   fingerprint.ctrlLED(/*LEDMode =
*/fingerprint.eFastBlink, /*LEDColor = */fingerprint.eLEDRed,
/*blinkCount = */3);
                }
             }
          }
          if(j == 0){
             /*Fingerprint capturing failed*/
          }else if(j > 0 && j < 15){
             Serial.println("Enter fingerprint comparison mode");
             /*Compare fingerprints*/
              fingerprintMatching();
          }else if(j >= 15 && j < 30){
             Serial.println("Enter fingerprint registration mode");
             /*Registrate fingerprint*/
             fingerprintRegistration();
          }else{
             Serial.println("Enter fingerprint deletion mode");
             /*Delete this fingerprint*/
             fingerprintDeletion();
          }
          k = 0;
       }
    }
}
#endif //MAINBOARD_FINGERPRINT_H
```

HallSensor.h

```
//
// Created by TedWu on 2022/3/13.
//

#ifndef MAINBOARD_HALL_SENSOR_H
#define MAINBOARD_HALL_SENSOR_H
```

```
#include "Arduino.h"
float HallSafeCount = 10;
bool Prev_State = false;
double getHallSensorVoltage() {
   double voltage = analogRead(A1) * (5.0 / 1023.0);
   return voltage;
}

double HallSensorMonitor() {
   /*
   // Monitor the Hall sensor voltage
    * Check the voltage change with this sequence:
     * LOW -> HIGH -> LOW -> HIGH -> LOW -> HIGH -> LOW -> HIGH ->
LOW -> HIGH
    *
   while(ReadHallSensorVoltage() <= 2) {
      while (ReadHallSensorVoltage() >= 4)
         while (ReadHallSensorVoltage() <= 2)
            while (ReadHallSensorVoltage() >= 4)
               while (ReadHallSensorVoltage() <= 2)
                  while (ReadHallSensorVoltage() >= 4)
                     while (ReadHallSensorVoltage() <= 2)
                        while (ReadHallSensorVoltage() >= 4)
                           while (ReadHallSensorVoltage() <= 2)
                              while (ReadHallSensorVoltage() >= 4)
{
                                 Serial.println("Wheel Moved");
                                 break;
                              }
   }
   */
   // Determine the motion state
   bool CurrentState;
   if(getHallSensorVoltage() <= 2) {
      CurrentState = false; // False for down state for Hall Sensor
   }
   else {
      CurrentState = true;
   }

   if(CurrentState & Prev_State) {
      if (HallSafeCount >= 0.5) {
         HallSafeCount -= 0.5;
      }
   }
   else {
```

```
      if (HallSafeCount <= 9) {
         HallSafeCount ++;
      }
   }
   return HallSafeCount;
}

#endif //MAINBOARD_HALL_SENSOR_H
```

RTC.h

```
//
// Created by TedWu on 2022/3/30.
//

#ifndef MAINBOARDWIRED_RTC_H
#define MAINBOARDWIRED_RTC_H
#include "Arduino.h"
#include "RTClib.h"

RTC_PCF8523 rtc;
char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday",
"Wednesday", "Thursday", "Friday", "Saturday"};

void RTCSetup() {
   while (! rtc.begin()) {
      Serial.println("Couldn't find RTC");
      Serial.flush();
   }
   if (! rtc.initialized() || rtc.lostPower()) {
      Serial.println("RTC is NOT initialized, let's set the time!");
      // When time needs to be set on a new device, or after a
power loss, the
      // following line sets the RTC to the date & time this sketch
was compiled
      rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
      // This line sets the RTC with an explicit date & time, for
example to set
      // January 21, 2014 at 3am you would call:
      // rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0));
      //
      // Note: allow 2 seconds after inserting battery or applying
external power
      // without battery before calling adjust(). This gives the
PCF8523's
      // crystal oscillator time to stabilize. If you call adjust()
```

```
very quickly
      // after the RTC is powered, lostPower() may still return
true.
   }
   rtc.start();
   float drift = 43; // seconds plus or minus over oservation
period - set to 0 to cancel previous calibration.
   float period_sec = (7 * 86400);  // total obsevation period in
seconds (86400 = seconds in 1 day:  7 days = (7 * 86400) seconds )
   float deviation_ppm = (drift / period_sec * 1000000); //
deviation in parts per million (µs)
   float drift_unit = 4.34; // use with offset mode
PCF8523_TwoHours
   // float drift_unit = 4.069; //For corrections every min the
drift_unit is 4.069 ppm (use with offset mode PCF8523_OneMinute)
   int offset = round(deviation_ppm / drift_unit);
   // rtc.calibrate(PCF8523_TwoHours, offset); // Un-comment to
perform calibration once drift (seconds) and observation period
(seconds) are correct
   // rtc.calibrate(PCF8523_TwoHours, 0); // Un-comment to cancel
previous calibration
   Serial.print("Offset is "); Serial.println(offset); // Print to
control offset
   DateTime now = rtc.now();
}

#endif //MAINBOARDWIRED_RTC_H
```

SIM7000C.h

```
//
// Created by TedWu on 2022/5/4.
//

#ifndef MAINBOARDWIRED_SIM7000C_H
#define MAINBOARDWIRED_SIM7000C_H
#include <DFRobot_SIM7000.h>
#include "RTC.h"
#define PIN_TX     7
#define PIN_RX     8
SoftwareSerial     mySerial(PIN_RX,PIN_TX);
DFRobot_SIM7000    sim7000(&mySerial);

void SIM7000C_setup() {
   mySerial.begin(19200);
   Serial.println("Turn ON SIM7000......");
```

```
if(sim7000.turnON()){                                          //Tu
rn ON SIM7000
      Serial.println("Turn ON !");
   }
   Serial.println("Set baud rate......");
   while(1){

if(sim7000.setBaudRate(19200)){                                //Set
SIM7000 baud rate from 115200 to 19200 reduce the baud rate to
avoid distortion
         Serial.println("Set baud rate:19200");
         break;
      }else{
         Serial.println("Faile to set baud rate");
         delay(1000);
      }
   }
   Serial.println("Check SIM card......");

if(sim7000.checkSIMStatus()){                                  //Ch
eck SIM card
      Serial.println("SIM card READY");
   }else{
      Serial.println("SIM card ERROR, Check if you have insert SIM
card and restart SIM7000");
      while(1);
   }
   Serial.println("Init positioning function......");
   while(1){
      if(sim7000.initPos()){
         Serial.println("Positioning function initialized");
         break;
      }else{
         Serial.println("Fail to init positioning function");
         delay(1000);
      }
   }
}

void getPosition() {
   Serial.println("Getting position......");

if(sim7000.getPosition()){                                     //Ge
t the current position
      Serial.print(" Longitude : ");
      Serial.println(sim7000.getLongitude());
```

```
//Get longitude
      Serial.print(" Latitude : ");
      Serial.println(sim7000.getLatitude());
//Get latitude
   }else{
      Serial.println("Wrong data try again");
   }
}

void SendMessage(int board, int event) {
   char boardInfo;
   char eventInfo;
   DateTime now = rtc.now();
   mySerial.println("AT+CMGF=1");
   delay(1000);
   mySerial.println("AT+CMGS=\"00447738425764\"\r");
   delay(1000);
   mySerial.print("Theft Occur: ");
   delay(100);
   //mySerial.print(board);
   switch (board) {
      case 0:
         mySerial.print("Main Board");
         break;
      case 1:
         mySerial.print("Supplementary Board 1");
         break;
      case 2:
         mySerial.print("Supplementary Board 2");
         break;
   }
   delay(100);
   mySerial.print(", ");
   delay(100);
   //mySerial.print(event);
   switch (event) {
      case 1:
         mySerial.print("Hall Sensor");
         break;
      case 2:
         mySerial.print("Throttle");
         break;
      case 3:
         mySerial.print("Hall Sensor & Throttle");
         break;
      case 4:
         mySerial.print("Proximity & Roll & Pitch");
```

```
            break;
        case 5:
            mySerial.print("Proximity & Roll");
            break;
        case 6:
            mySerial.print("Proximity & Pitch");
            break;
        case 7:
            mySerial.print("Roll & Pitch");
            break;
        case 8:
            mySerial.print("Proximity");
            break;
        case 9:
            mySerial.print("Roll");
            break;
        case 10:
            mySerial.print("Pitch");
            break;
    }
    mySerial.println();
    delay(100);
    mySerial.print(". ");
    delay(100);
    mySerial.print("At date and time: ");
    delay(100);
    mySerial.print("2022/");
    delay(100);
    mySerial.print(now.month(), DEC);
    delay(100);
    mySerial.print("/");
    delay(100);
    mySerial.print(now.day(), DEC);
    delay(100);
    mySerial.print(", ");
    delay(100);
    mySerial.print(daysOfTheWeek[now.dayOfTheWeek()]);
    delay(100);
    mySerial.print(", ");
    delay(100);
    mySerial.print(now.hour(), DEC);
    delay(100);
    mySerial.print(":");
    delay(100);
    mySerial.print(now.minute(), DEC);
    delay(100);
    mySerial.print(":");
```

```
    delay(100);
    mySerial.println(now.second(), DEC);
    delay(100);
    mySerial.println((char)26);
    delay(1000);
}


#endif //MAINBOARDWIRED_SIM7000C_H
```

SIM7600CE.h

```
//
// Created by TedWu on 2022/3/13.
//

#ifndef MAINBOARD_SIM7600CE_H
#define MAINBOARD_SIM7600CE_H

#include "Arduino.h"
#include "SoftwareSerial.h"
#include "RTC.h"
SoftwareSerial SIM7600CE(0, 1);

void SendMessage(int board, int event) {
    char boardInfo;
    char eventInfo;
    DateTime now = rtc.now();
    SIM7600CE.println("AT+CMGF=1");
    delay(500);
    SIM7600CE.println("AT+CMGS=\"00447738425764\"\r");
    delay(500);
    SIM7600CE.print("Theft Occur: ");
    delay(100);
    //SIM7600CE.print(board);
    switch (board) {
        case 0:
            SIM7600CE.print("Main Board");
            break;
        case 1:
            SIM7600CE.print("Supplementary Board 1");
            break;
        case 2:
            SIM7600CE.print("Supplementary Board 2");
            break;
    }
```

```
delay(50);
SIM7600CE.print(", ");
delay(50);
//SIM7600CE.print(event);
switch (event) {
   case 1:
      SIM7600CE.print("Hall Sensor");
      break;
   case 2:
      SIM7600CE.print("Throttle");
      break;
   case 3:
      SIM7600CE.print("Hall Sensor & Throttle");
      break;
   case 4:
      SIM7600CE.print("Proximity & Roll & Pitch");
      break;
   case 5:
      SIM7600CE.print("Proximity & Roll");
      break;
   case 6:
      SIM7600CE.print("Proximity & Pitch");
      break;
   case 7:
      SIM7600CE.print("Roll & Pitch");
      break;
   case 8:
      SIM7600CE.print("Proximity");
      break;
   case 9:
      SIM7600CE.print("Roll");
      break;
   case 10:
      SIM7600CE.print("Pitch");
      break;
}
SIM7600CE.println();
delay(50);
SIM7600CE.print(". ");
delay(50);
SIM7600CE.print("At date and time: ");
delay(50);
SIM7600CE.print("2022/");
delay(50);
SIM7600CE.print(now.month(), DEC);
delay(50);
SIM7600CE.print("/");
```

```
    delay(50);
    SIM7600CE.print(now.day(), DEC);
    delay(50);
    SIM7600CE.print(", ");
    delay(50);
    SIM7600CE.print(daysOfTheWeek[now.dayOfTheWeek()]);
    delay(50);
    SIM7600CE.print(", ");
    delay(50);
    SIM7600CE.print(now.hour(), DEC);
    delay(50);
    SIM7600CE.print(":");
    delay(50);
    SIM7600CE.print(now.minute(), DEC);
    delay(50);
    SIM7600CE.print(":");
    delay(50);
    SIM7600CE.println(now.second(), DEC);
    delay(50);
    SIM7600CE.println((char)26);
    delay(500);
}

void SIM7600CE_Setup() {
    SIM7600CE.begin(115200);
}



#endif //MAINBOARD_SIM7600CE_H
```

Throttle.h

```
//
// Created by TedWu on 2022/3/13.
//

#ifndef MAINBOARD_THROTTLE_H
#define MAINBOARD_THROTTLE_H

#include "Arduino.h"
#include "DecisionAlgorithm.h"
#include "Throttle.h"

#define THROTTLE_PIN 11
```

```
double getThrottleVoltage() {
   return analogRead(A0) / 4.0;
}

double ThrottleMonitor () {
   //int throttle_voltage = analogRead(A0) ;
   // Serial.println(throttle_voltage );
   if (eventFlag[0]) {
      analogWrite(THROTTLE_PIN, 0);
   } else {
      analogWrite(THROTTLE_PIN, (int)getThrottleVoltage() );
   }
   return getThrottleVoltage();
}

#endif //MAINBOARD_THROTTLE_H
```

VL53L1X.h

```
//
// Created by TedWu on 2022/3/15.
//

#ifndef MAINBOARD_VL53L1X_H
#define MAINBOARD_VL53L1X_H

#include "Arduino.h"
#include "Adafruit_VL53L1X.h"

#define IRQ_PIN 2
#define XSHUT_PIN 3

Adafruit_VL53L1X vl53 = Adafruit_VL53L1X(XSHUT_PIN, IRQ_PIN);

void vl53_Setup() {
   Wire.begin();
   Serial.println("***Initialize the ToF Sensor***");
   while (! vl53.begin(0x29, &Wire)) {
      Serial.print(F("Error on init of VL sensor: "));
      Serial.println(vl53.vl_status);
      delay(10);
   }
   Serial.println(F("VL53L1X sensor OK!"));

   Serial.print(F("VL53L1X Sensor ID: 0x"));
   Serial.println(vl53.sensorID(), HEX);
```

```
    while (! vl53.startRanging()) {
        Serial.print(F("Couldn't start ranging: "));
        Serial.println(vl53.vl_status);
        delay(10);
    }
    Serial.println(F("Ranging started"));

    // Valid timing budgets: 15, 20, 33, 50, 100, 200 and 500ms!
    vl53.setTimingBudget(50);
    Serial.print(F("Timing budget (ms): "));
    Serial.println(vl53.getTimingBudget());

    /*
    vl.VL53L1X_SetDistanceThreshold(100, 300, 3, 1);
    vl.VL53L1X_SetInterruptPolarity(0);
    */

    while(! vl53.dataReady());
    Serial.println("ToF Sensor Ready");

    Serial.println("***End of ToF Sensor***");
}

int16_t getDistance() {
    int16_t distance = vl53.distance();
    // Distance in mm
    if (distance == -1) {
        // something went wrong!
        Serial.print(F("Couldn't get distance: "));
        Serial.println(vl53.vl_status);
    }
    return distance;
}


#endif //MAINBOARD_VL53L1X_H
```

8.2. Appendix 2 Supplementary Arduino Source Code

Main.cpp

```
#include "DecisionAlgorithm.h"
void setup() {
```

```
    pinMode(OUTPUT_PIN, OUTPUT);
    Serial.begin(9600);
    while (!Serial)
        delay(1);
    vl53_Setup();
    IMUsetup();
    displaySetup();
}

void loop() {
    Algorithm();
    /*
    getSensorData();    // Get sensor data from the main board
    // getHardwareData();  // Get hardware data from the main board
    SetFlag();          // set flags according to the sensor and
hardware data
    decisionTree();
    Clear();
    */
}
```

BLEReciever.h

```
//
// Created by TedWu on 2022/3/19.
//

#ifndef MAINBOARD_BLERECIEVER_H
#define MAINBOARD_BLERECIEVER_H

#include "Arduino.h"
#include <ArduinoBLE.h>

BLEService SupplementaryFrame("19B10000-E8F2-537E-4F6C-
D104768A1214"); // Bluetooth® Low Energy LED Service
BLEService SupplementaryBack("19B20000-E8F2-537E-4F6C-
D104768A1214");
BLEService SupplementaryFront("19B30000-E8F2-537E-4F6C-
D104768A1214");

// Bluetooth® Low Energy LED Switch Characteristic - custom 128-bit
UUID, read and writable by central
BLEByteCharacteristic proximityCharacteristic("19B10001-E8F2-537E-
4F6C-D104768A1214", BLERead | BLEWrite);
BLEByteCharacteristic accCharacteristic("19B10002-E8F2-537E-4F6C-
D104768A1214", BLERead | BLEWrite);
```

```
BLEByteCharacteristic gyroCharacteristic("19B10003-E8F2-537E-4F6C-
D104768A1214", BLERead | BLEWrite);
// BLEDevice peripheral;

bool Supplementary1Event[3] = {
      false, // 1: proximity
      false, // 1: acc
      false, // 1: gyro
};

void BLERecieverSetup() {
   // begin initialization
   if (!BLE.begin()) {
      Serial.println("starting Bluetooth® Low Energy module
failed!");

      while (1);
   }

   // set advertised local name and service UUID:
   BLE.setLocalName("Central");
   BLE.setAdvertisedService(SupplementaryFrame);

   // add the characteristic to the service
   SupplementaryFrame.addCharacteristic(proximityCharacteristic);
   SupplementaryFrame.addCharacteristic(accCharacteristic);
   SupplementaryFrame.addCharacteristic(gyroCharacteristic);

   // add service
   BLE.addService(SupplementaryFrame);

   // set the initial value for the characeristic:
   proximityCharacteristic.writeValue(0);
   accCharacteristic.writeValue(0);
   gyroCharacteristic.writeValue(0);

   // start advertising
   BLE.advertise();

   Serial.println("BLE Central");
}

void BLERecieve() {
   // listen for Bluetooth® Low Energy peripherals to connect:
   BLEDevice peripheral = BLE.central();

   // if a central is connected to peripheral:
```

```
   if (peripheral) {
      Serial.print("Connected to central: ");
      // print the central's MAC address:
      Serial.println(peripheral.address());

      // while the central is still connected to peripheral:
      if (peripheral.connected()) {
         // if the remote device wrote to the characteristic,
         // use the value to control the LED:
         if (proximityCharacteristic.written()) {
            if (proximityCharacteristic.value()) {  // any value
other than 0
               Serial.println("Proximity");
               Supplementary1Event[1] = true;
            }
         }
         if (accCharacteristic.written()) {
            if (accCharacteristic.value()) {
               Serial.println("Accelerometer");
               Supplementary1Event[2] = true;
            }
         }
         if (gyroCharacteristic.written()) {
            if (gyroCharacteristic.value()) {
               Serial.println("Gyro");
               Supplementary1Event[3] = true;
            }
         }
      }

      // when the central disconnects, print it out:
      // Serial.print(F("Disconnected from peripheral: "));
      // Serial.println(peripheral.address());
   }
}

#endif //MAINBOARD_BLERECIEVER_H
```

DecisionAlgorithm.h

```
//
// Created by TedWu on 2022/3/30.
//

#include "Arduino.h"
#include "VL53L1X.h"
```

```
#include "EulerAngle.h"
//#include "Display.h"
#include "Wire.h"
#include "JY901.h"

#define ROLL_THRESHOLD 8
#define PITCH_THRESHOLD 8
#define TOF_THRESHOLD 5
#define SENSOR_ROLLING 15
#define DECISION_ROLLING 5
#define OUTPUT_PIN 3

bool eventFlag[3] = {false, false, false};
/*
 * Table of eventFlag
 * 0. ToF value
 * 1. Roll
 * 2. Pitch
 */
int ToF_Count = 0;
int rollCount = 0;
int pitchCount = 0;
int displayCount = 0;
long startTime = 0;
int KFroll_diff = 0;
int KFpitch_diff = 0;
int KFyaw_diff = 0;
int KFroll_pre = 0;
int KFpitch_pre = 0;
int KFyaw_pre = 0;
float distance = 0;

/*Get the sensor data and convert the data from IMU to Euler
Angle*/
void getSensorData()
{
   /*Get distance*/
   for (int i = 0; i < SENSOR_ROLLING; ++i) {
      distance += getDistance() / 10.0;
   }
  distance /= SENSOR_ROLLING;
   /*Get Euler Angle*/
   getEulerAngle();
  JY901.GetAngle();
   /*Print them on the IDE*/
  /*
   Serial.print(distance);
```

```
  Serial.print(" ");
   Serial.print(complementaryRoll);
  Serial.print(" ");
   Serial.println(complementaryPitch);
 */
 /*Print angle on the display unit*/

Serial.print("Angle:");Serial.print((float)JY901.stcAngle.Angle[0]/
32768*180);Serial.print("
");Serial.print((float)JY901.stcAngle.Angle[1]/32768*180);Serial.pr
int(" ");Serial.println((float)JY901.stcAngle.Angle[2]/32768*180);
 float KF_pitch = (float)JY901.stcAngle.Angle[0]/32768*180;
 float KF_roll = (float)JY901.stcAngle.Angle[1]/32768*180;
 float KF_yaw = (float)JY901.stcAngle.Angle[2]/32768*180;
 KFpitch_diff = KF_pitch - KFpitch_pre;
 KFroll_diff = KF_roll - KFroll_pre;
 KFyaw_diff = KF_yaw - KFyaw_pre;
 KFpitch_pre = KF_pitch;
 KFroll_pre = KF_roll;
 KFyaw_pre = KF_yaw;
 /*
   display.clearDisplay();
    display.setCursor(0,0);
    display.print("ToF\nRoll\nPitch\nYaw");
    display.setCursor(30,0);
    display.print(distance);
    display.setCursor(30,8);
    display.print(KF_roll);
    display.setCursor(30,16);
    display.print(KF_pitch);
    display.setCursor(30,24);
    display.print(KF_yaw);
    display.display();
    displayCount = 5;
    */
}

void SetFlag() {
   /*Proximity*/
   if (distance <= TOF_THRESHOLD && distance >= 0)
      eventFlag[0] = true;
   else if (distance > TOF_THRESHOLD)
      eventFlag[0] = false;

   /*Roll*/
   if (abs(/*diff_complementaryRoll*/KFroll_diff) >= ROLL_THRESHOLD)
{
```

```
      eventFlag[1] = true;
   } else {
      eventFlag[1] = false;
   }

   /*Pitch*/
   if (abs(/*diff_complementaryPitch*/KFpitch_diff) >=
PITCH_THRESHOLD)
      eventFlag[2] = true;
   else {
      eventFlag[2] = false;
   }
}

void SupplementaryBoardTheft(int EventFlag) {
    //analogWrite(OUTPUT_PIN, 0);
   switch (EventFlag) {
      case 1:
         Serial.println("Proximity & Roll & Pitch");
         analogWrite(OUTPUT_PIN, 40);
         break;
      case 2:
         Serial.println("Proximity & Roll");
         analogWrite(OUTPUT_PIN, 60);
         break;
      case 3:
         Serial.println("Proximity & Pitch");
         analogWrite(OUTPUT_PIN, 80);
         break;
      case 4:
         Serial.println("Roll & Pitch");
         analogWrite(OUTPUT_PIN, 100);
         break;
      case 5:
         Serial.println("Proximity");
         analogWrite(OUTPUT_PIN, 120);
         break;
      case 6:
         Serial.println("Roll");
         analogWrite(OUTPUT_PIN, 140);
         break;
      case 7:
         Serial.println("Pitch");
         analogWrite(OUTPUT_PIN, 160);
         break;
   }
   startTime = millis();
```

```
}

void decisionTree() {
   if (eventFlag[0] /*ToF*/ && eventFlag[1] /*Roll*/ &&
eventFlag[2]/*Pitch*/) {
      //Serial.println("tof & Roll & Pitch");
      SupplementaryBoardTheft(1);
   }
      // Check each sensor
      // check ToF starts
   else if(eventFlag[0]/*ToF*/) { //check each sensor
      if (eventFlag[1]/*Roll*/) {
         if (eventFlag[2]/*Pitch*/) {
            //Serial.println("tof & Roll & Pitch");
            SupplementaryBoardTheft(1);
         }
         else {
            //Serial.println("tof & Roll");
            SupplementaryBoardTheft(2);
         }
      } else if (eventFlag[2]/*Pitch*/) {
         if (eventFlag[1]/*Roll*/) {
            //Serial.println("tof & Roll & Pitch");
            SupplementaryBoardTheft(1);
         } else {
            //Serial.println("tof & Pitch");
            SupplementaryBoardTheft(3);
         }
      } else {
         ToF_Count++;
         if (ToF_Count >= DECISION_ROLLING) {
            //Serial.println("tof");
            SupplementaryBoardTheft(5);
            ToF_Count = 0;
         } else {
            Serial.println("Safe");
            //analogWrite(OUTPUT_PIN, 0);
         }
      }
   }
      // ToF part ends
      // check Roll starts
   else if(eventFlag[1]/*Roll*/) {
      if (eventFlag[0]/*ToF*/) {
         //Serial.println("tof & Roll");
         SupplementaryBoardTheft(2);
      } else if (eventFlag[2]/*Pitch*/) {
```

```
            //Serial.println("Roll & Pitch");
            SupplementaryBoardTheft(3);
        } else {
            rollCount++;
            if (rollCount >= 1) {
                //Serial.println("Roll");
                SupplementaryBoardTheft(6);
                rollCount = 0;
            } else {
                Serial.println("Safe");
                //analogWrite(OUTPUT_PIN, 0);
            }
        }
    }
        // acc part ends
        // check Pitch starts
    else if(eventFlag[2]/*Pitch*/) {
        if (eventFlag[0]/*ToF*/) {
            //Serial.println("tof & Pitch");
            SupplementaryBoardTheft(3);
        } else if (eventFlag[1]/*Roll*/) {
            //Serial.println("Roll & Pitch");
            SupplementaryBoardTheft(4);
        } else {
            pitchCount++;
            if (pitchCount >= 1) {
                //Serial.println("Pitch");
                SupplementaryBoardTheft(7);
                pitchCount = 0;
            } else {
                Serial.println("Safe");
                //analogWrite(OUTPUT_PIN, 0);
            }
        }
    }
    else {
        Serial.println("Safe");
        //analogWrite(OUTPUT_PIN, 0);
    }
}

void Clear() {
    // Clear Event Flag
    for (bool & i : eventFlag) {
        i = false;
    }
    distance = 0;
```

```
    if ((millis() - startTime) >= 3000) {
        analogWrite(OUTPUT_PIN, 0);
    }
}

void Algorithm() {
    getSensorData();    // Get sensor data from the main board
   printCalculations();
   delay(260);
    // getHardwareData();  // Get hardware data from the main board
    SetFlag();          // set flags according to the sensor and
hardware data
    decisionTree();
    Clear();
}
```

Display.h

```
//
// Created by TedWu on 2022/3/24.
//

#ifndef MAINBOARD_DISPLAY_H
#define MAINBOARD_DISPLAY_H
#include "Arduino.h"
#include "SPI.h"
#include "Wire.h"
#include "Adafruit_SH110X.h"
#include "Adafruit_GFX.h"
Adafruit_SH1107 display = Adafruit_SH1107(64, 128, &Wire);

void displaySetup() {
    Serial.println("128x64 OLED FeatherWing test");
    display.begin(0x3C, true); // Address 0x3C default
    Serial.println("OLED begun");
    display.display();
    delay(1000);
    display.clearDisplay();
    display.display();
    display.setRotation(1);
    display.setTextSize(1);
    display.setTextColor(SH110X_WHITE);
    display.setCursor(0,0);
    display.print("ToF\nRoll\nPitch");
    display.display(); // actually display all of the above
}
```

```
#endif //MAINBOARD_DISPLAY_H
```

EulerAngle.h

```
//
// Created by TedWu on 2022/4/16.
//

#ifndef MAINBOARDWIRED_EULERANGLE_H
#define MAINBOARDWIRED_EULERANGLE_H

#include "Arduino.h"
#include "Adafruit_LSM6DS33.h"
#include "Adafruit_LIS3MDL.h"
#define HP 0.6
#define LP 0.4
#define RADIAN_DEGREE 180/M_PI

Adafruit_LIS3MDL Mag;
Adafruit_LSM6DS33 AccGyro;
sensors_event_t accel, gyro, temp;

float accelX, accelY, accelZ;
float gyroscope_x, gyroscope_y, gyroscope_z;
float drift_x, drift_y, drift_z;
float gyroscope_roll, gyroscope_pitch, gyroscope_yaw;
float gyroscope_afterDrift_Roll, gyroscope_afterDrift_Pitch,
gyroscope_afterDrift_Yaw;
float accelerometer_roll, accelerometer_pitch;
float complementaryFilter_roll, complementaryFilter_pitch,
complementaryFilter_yaw;
float magnetometer_x, magnetometer_y, magnetometer_z;
float acc_mag_CF_x, acc_mag_CF_y, magnetometer_yaw;
float pre_complementaryRoll, pre_complementaryPitch,
pre_complementaryYaw;
float diff_complementaryRoll, diff_complementaryPitch,
diff_complementaryYaw;

long lastTime;
long lastInterval;

bool readIMU() {
   AccGyro.getEvent(&accel, &gyro, &temp);
   Mag.read();
```

```
    accelX = accel.acceleration.x;
    accelY = accel.acceleration.y;
    accelZ = accel.acceleration.z;
    gyroscope_x = gyro.gyro.x;
    gyroscope_y = gyro.gyro.y;
    gyroscope_z = gyro.gyro.z;
    magnetometer_x = Mag.x;
    magnetometer_y = Mag.y;
    magnetometer_z = Mag.z;
    return true;
}

void calibrate(int delayTime, int calibrationTime) {
    int calibrationCount = 0;
    delay(delayTime);
    float x, y, z;
    int startTime = millis();
    while (millis() < startTime + calibrationTime) {
        if (readIMU()) {
            x += gyroscope_x;
            y += gyroscope_y;
            z += gyroscope_z;
            calibrationCount++;
        }
    }

    if (calibrationCount == 0) {
        Serial.println("Failed to calibrate");
    }

    drift_x = x / calibrationCount;
    drift_y = y / calibrationCount;
    drift_z = z / calibrationCount;

}

void doCalculations() {
    accelerometer_roll = atan2(accelY, accelZ) * RADIAN_DEGREE;
    accelerometer_pitch = atan2(-accelX, sqrt(accelY * accelY +
accelZ * accelZ)) * RADIAN_DEGREE;

    float lastFrequency = (float) 1000000.0 / lastInterval;
    gyroscope_roll = gyroscope_roll + (gyroscope_x / lastFrequency);
    gyroscope_pitch = gyroscope_pitch + (gyroscope_y /
lastFrequency);
    gyroscope_yaw = gyroscope_yaw + (gyroscope_z / lastFrequency);
```

```
   gyroscope_afterDrift_Roll = gyroscope_afterDrift_Roll +
((gyroscope_x - drift_x) / lastFrequency);
   gyroscope_afterDrift_Pitch = gyroscope_afterDrift_Pitch +
((gyroscope_y - drift_y) / lastFrequency);
   // gyroscope_afterDrift_Yaw = gyroscope_afterDrift_Yaw +
((gyroscope_z - drift_z) / lastFrequency);

   acc_mag_CF_x = magnetometer_x * cos(accelerometer_pitch) +
magnetometer_y * sin(accelerometer_pitch) - magnetometer_z *
cos(accelerometer_roll) * sin(accelerometer_roll);
   acc_mag_CF_y = magnetometer_y * cos(accelerometer_roll) +
magnetometer_z * sin(accelerometer_roll);
   magnetometer_yaw = atan2(acc_mag_CF_y, acc_mag_CF_x);
   complementaryFilter_yaw = complementaryFilter_yaw +
((gyroscope_z - drift_z) / lastFrequency);

   complementaryFilter_roll = HP * (complementaryFilter_roll +
gyroscope_afterDrift_Roll) + LP * accelerometer_roll;
   complementaryFilter_pitch = HP * (complementaryFilter_pitch +
gyroscope_afterDrift_Pitch) + LP * accelerometer_pitch;
   //complementaryFilter_yaw = magnetometer_yaw;

   diff_complementaryRoll = complementaryFilter_roll -
pre_complementaryRoll;
   diff_complementaryPitch = complementaryFilter_pitch -
pre_complementaryPitch;
   diff_complementaryYaw = complementaryFilter_yaw -
pre_complementaryYaw;
}

void printCalculations() {
  Serial.print("Roll: ");
   Serial.print(complementaryFilter_roll);
   Serial.print(',');
  Serial.print("Pitch: ");
   Serial.print(complementaryFilter_pitch);
   Serial.print(',');
  Serial.print("Yaw: ");
   Serial.print(magnetometer_yaw);
  Serial.print(',');
   //Serial.println("");
}

void setPrevious() {
   pre_complementaryRoll = complementaryFilter_roll;
   pre_complementaryPitch = complementaryFilter_pitch;
   pre_complementaryYaw = complementaryFilter_yaw;
```

```
}

void IMUsetup() {
   while (!AccGyro.begin_I2C()) {
      Serial.println("Cannot find Accgyro chip");
   }
 while (!Mag.begin_I2C()) {
  Serial.println("Cannot find Mag chip");
 }
   calibrate(250, 250);
   lastTime = micros();
}

void getEulerAngle() {
   if (readIMU()) {
      setPrevious();
      long currentTime = micros();
      lastInterval = currentTime - lastTime;
      lastTime = currentTime;
      doCalculations();
      //printCalculations();
   }
}

#endif //MAINBOARDWIRED_EULERANGLE_H
```

VL53L1X.h

```
//
// Created by TedWu on 2022/3/15.
//

#ifndef MAINBOARD_VL53L1X_H
#define MAINBOARD_VL53L1X_H

#include "Arduino.h"
#include "Adafruit_VL53L1X.h"

#define IRQ_PIN 2
#define XSHUT_PIN 3

Adafruit_VL53L1X vl53 = Adafruit_VL53L1X(XSHUT_PIN, IRQ_PIN);

void vl53_Setup() {
   Wire.begin();
   Serial.println("***Initialize the ToF Sensor***");
```

```
    while (! vl53.begin(0x29, &Wire)) {
        Serial.print(F("Error on init of VL sensor: "));
        Serial.println(vl53.vl_status);
        delay(10);
    }
    Serial.println(F("VL53L1X sensor OK!"));

    Serial.print(F("VL53L1X Sensor ID: 0x"));
    Serial.println(vl53.sensorID(), HEX);

    while (! vl53.startRanging()) {
        Serial.print(F("Couldn't start ranging: "));
        Serial.println(vl53.vl_status);
        delay(10);
    }
    Serial.println(F("Ranging started"));

    // Valid timing budgets: 15, 20, 33, 50, 100, 200 and 500ms!
    vl53.setTimingBudget(50);
    Serial.print(F("Timing budget (ms): "));
    Serial.println(vl53.getTimingBudget());

    /*
    vl.VL53L1X_SetDistanceThreshold(100, 300, 3, 1);
    vl.VL53L1X_SetInterruptPolarity(0);
    */

    while(! vl53.dataReady());
    Serial.println("ToF Sensor Ready");

    Serial.println("***End of ToF Sensor***");
}

int16_t getDistance() {
    int16_t distance = vl53.distance();
    // Distance in mm
    if (distance == -1) {
        // something went wrong!
        Serial.print(F("Couldn't get distance: "));
        Serial.println(vl53.vl_status);
    }
    return distance;
}


#endif //MAINBOARD_VL53L1X_H
```