CS3237 Final Project Report

AY22/23 Semester 1

Group 27

CycleSafe

Team Member	Student No.	Email
Low Jing Woon	A0200768E	e0407749@u.nus.edu
Ng Chi Sern	A0219866M	e0550524@u.nus.edu
Nur Hayati Bte Daud	A0206326M	e0426249@u.nus.edu
Pua Li Xue	A0219708X	e0550366@u.nus.edu
Tai Kah Kiang	A0219771X	e0550429@u.nus.edu

Table of Contents

1. Introduction

Amid a COVID-19-inspired boom in the popularity of cycling, the number of road traffic accidents involving bicycles has since increased.[1] An average of about 560 serious accidents involving cyclists on roads have been recorded yearly in the past five years.[2] Accidents usually happen due to negligent drivers and the inconvenience of the cyclists to constantly check the situation behind and in blind spots. These accidents are dangerous and possibly life-threatening, particularly for cyclists compared to drivers. Furthermore, some injured cyclists face difficulties in contacting for help, and this issue is magnified when there is no one around to help them.

Our proposed solution, hence, aims to minimise the negative consequences of cycling accidents and reduce the chances of these accidents by introducing a comprehensive Internet of Things (IoT) system. This system comprises three main components: vehicle detection, fall detection, and an alarm system.

Fall Detection

When cyclists fall, they might be left attended or the neighboring vehicles might not notice the cyclist lying on the ground, which may cause vehicles to run over them. To avoid this dangerous situation, we have attached a fall detection module to the bicycle to detect collisions and falls and alert the surroundings to help drivers notice the fallen cyclist more easily. At the same time, we have proposed a telegram bot to help notify the cyclist's family members and friends concerned with the cyclist's safety. Most recent recordings regarding the fall will also be uploaded to provide further confirmation.

From Fall Detection to Fall Prevention

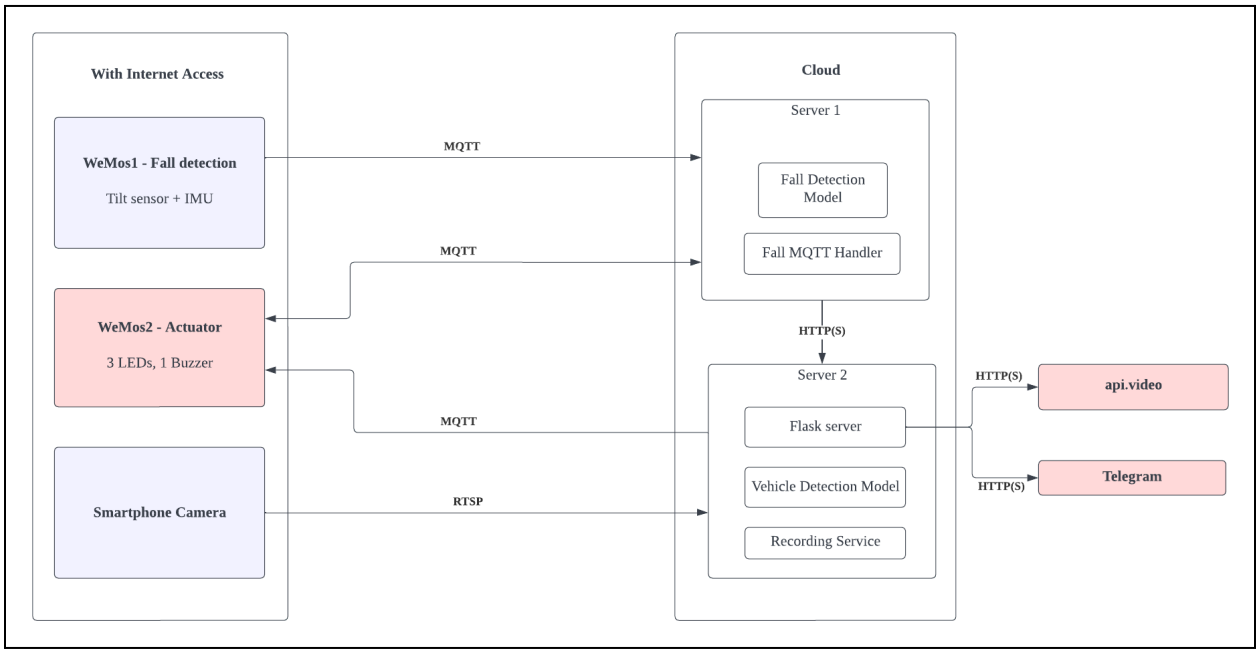
While cyclists can observe the road conditions most of the time, blindspots are usually neglected and leave cyclists unaware of approaching vehicles. Our IoT system plans to attach a smartphone camera to detect vehicles in blind spots using computer vision and keep the cyclists wary of the presence of their surrounding vehicles, the vehicles' speed and the proximity between the cyclist and these vehicles. The camera will also serve as a dashcam to record the vehicles behind in the case of a hit-and-run.

[1] https://www.channelnewsasia.com/singapore/bicycle-road-accidents-spike-amid-cycling-boom-1846736

[2] https://www.straitstimes.com/singapore/politics/average-of-560-yearly-serious-accidents-involving-cyclists-on-roads-iswaran

2. Techniques employed

2.1 System Architecture



Architecture Diagram of CycleSafe

As seen from the diagram, the system architecture is made up of two primary components: the edge devices and the cloud.

Our edge devices consist of the first WeMos which is used for fall detection, a smartphone camera and a second WeMos for the actuators. Fall detection requires the use of two tilt switches and an inertial measurement unit (IMU). Data from these sensors will subsequently be sent to the cloud via Message Queuing Telemetry Transport (MQTT). For vehicle detection, video data from the smartphone is sent to the cloud via Real Time Streaming Protocol (RTSP). The second WeMos is used for the actuators, which are the three LEDs and one buzzer. Data used to toggle the buzzer is both received from and transmitted to a server in the cloud (used for fall detection) whereas the data used for the three LEDs originates from another server in the cloud (used for vehicle detection). All these edge devices require Internet access for data transmission.

The cloud component comprises two servers; Server 1 is used for fall detection, and Server 2 is used for vehicle detection. The fall detection model and fall MQTT handler are found in Server 1, which will also relay information to Server 2 using HTTPS for Telegram notification purposes. The Vehicle Detection Model, Flask server and Recording Service are hosted on Server 2. The Flask server is used to output the Telegram messages and videos (api.video) from the point of view of the bicycle's rear to a Telegram channel, and this is executed via HTTP(S). Depending on the different events that occur, different Telegram messages and videos will be published in

the Telegram channel where individuals concerned with the safety of the cyclist can subscribe to.

2.2 Main Techniques

Communication

- Messaging Queueing Telemetry Transport (MQTT) with Transport Layer Security (TLS) Encryption for communication between the WeMoses, Cloud and Telegram server
- Real Time Streaming Protocol (RTSP) to transmit smartphone camera data to the cloud

Fall Detection

- The model is trained with Multilayer Perceptron (MLP) and saved
- Prediction is performed in the cloud with the saved model to detect falling events

Vehicle Detection

- A pre-trained Tiny - YOLOv3[3] model is used to detect vehicles
- An algorithm is used to detect the changes of size of vehicles between each frames to determine the safety status
- The video stream obtained is recorded and saved in 10 seconds chunk, and most recent 20 videos are kept

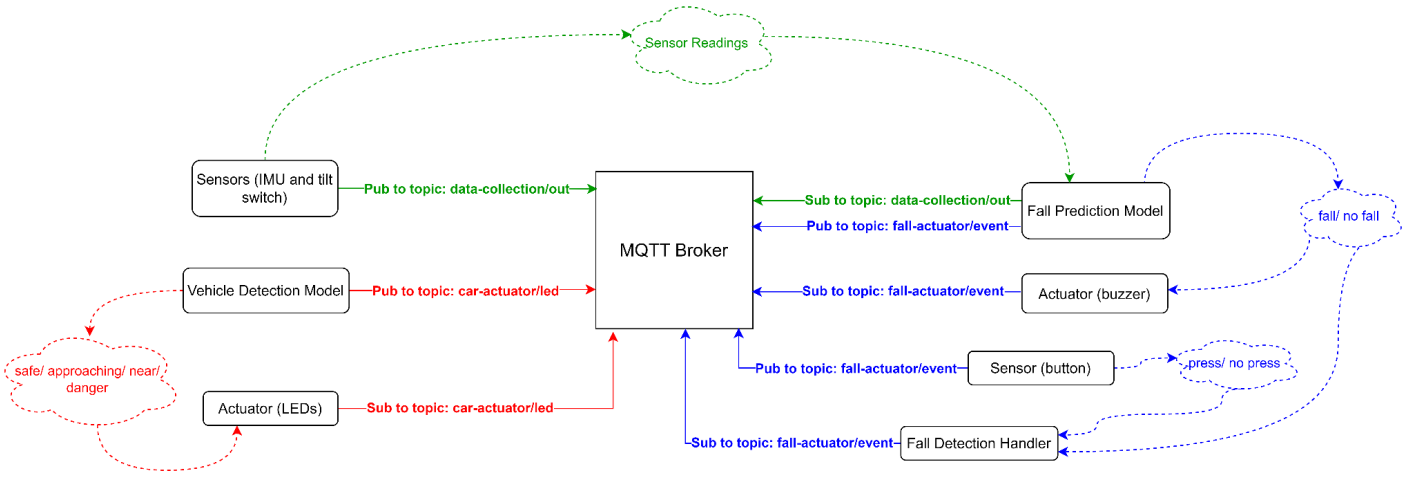
Miscellaneous

- 3D printed case to contain all the actuators and sensors
- Smartphone camera for obtaining video stream of incoming vehicles and for video recording
- Sensors: IMU (MPU-9250/6500), 2 tilt switches
- Actuators: 1 Buzzer, 3 LEDs, a telegram bot that notifies those who are concerned with the cyclist's most recent statuses

[3] https://github.com/eric612/Vehicle-Detection

3. Implementation Details

3.1 Communication Protocol



Communication Protocol of CycleSafe

We have implemented a lightweight protocol, MQTT, for transferring messages across different IoT devices. All the devices or clients are connected to the same broker, either the server on our laptop or the deployed service in HiveMQ. Since MQTT supports symmetric publisher-subscriber relationships, we have only employed three main MQTT topics, as shown in the figure above, where a publisher can be a subscriber simultaneously.

The first MQTT topic implemented is *data-collection/out,* where our sensors, including IMU and tilt switches, are the publisher. They will publish sensor readings such as acceleration and gyroscope readings which our fall prediction model will then consume. Next, the fall prediction model will publish the prediction result to another topic, *fall-actuator/event.* Our actuator (buzzer) and fall detection handler will consume this result. The buzzer will ring if a fall is detected to alert the surroundings, and the fall detection handler will send a message to the Telegram server to alert the rider's close contacts.

We have also utilized another sensor, a button for the users to press to indicate that they are safe and to stop the buzzer. The button state will also be published to the same topic, *fall-actuator/event,* which the fall detection handler will subscribe to inform the rider's safe condition to the Telegram server. In cases where the button is not pressed 30 seconds after a fall, an *Emergency* message will be automatically sent to the subscribers through the Telegram bot to notify them of the cyclist's safety.

Lastly, we have our final MQTT topic, *car-actuator/led,* where the vehicle detection model will publish the surrounding states such as *in a safe condition*, *car approaching*, *car is near* and *car approaching fast*. The LEDs will then consume the results and blink accordingly to alert the

cyclists. We have implemented three LEDs where the first LED will light up for *car approaching* event, the first and second LEDs will light up for *car is near* condition, all three LEDs will light up for *car approaching fast* event while no LEDs will light up for the *safe* state.

3.2 Fall Detection

Our fall detection development process consists of four main stages: data collection, model training, model testing, and real-world testing.

3.2.1 Data collection

Using the inertial measurement unit (IMU) and tilt switches, we collected data for acceleration and gyroscope values in all 3 axes. The resultant acceleration and whether the device is tilted to the left or right were also collected. Using this data, we were interested in detecting 8 different states of the bicycle: upright, moving forward, turning left, turning right, placing it down slowly, lying on the ground, lifting it back up, and falling. We collected the data using 3 approaches. All of these approaches involved clicking on a switch attached to our device to indicate a state change. This allowed us to label the states conveniently after collecting the data.

The first approach was moving forward then turning left, and repeating this process. The switch was clicked every time we transitioned from moving forward to turning left and vice versa. This resulted in all data for moving forward and turning left separated and distinguished easily. Similarly, we replaced turning left with turning right and repeated the whole process. Three states were collected for this approach: moving forward, turning left, turning right.

The second approach was moving forward and suddenly falling. After falling, we let the bicycle lie on the ground for some time before lifting it back up. Similarly, we clicked the switch whenever we transitioned between different states, allowing easy labeling of data after collection. Four states were collected for this approach: moving forward, falling, lying on the ground, lifting it back up.

Finally, the third approach was to start with an upright bicycle, then placing it down slowly until it lay on the ground. We then lifted it back up after some time. This approach aims to train the model to differentiate between a real fall and just putting the bicycle down. Four states were collected for this approach: upright, placing it down slowly, lying on the ground, lifting it back up.

All of the approaches described above were done with a sampling rate of 10Hz on the WeMos and repeated until we had tens of thousands of data instances.

3.2.2 Model training

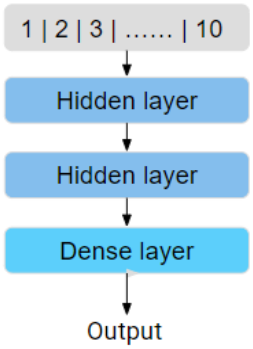
Prior to building our models, preprocessing of the raw data was done. As described in the data collection process, we mapped the corresponding states to integers 0 - 7. In the process, we

discovered that a small portion of the data all contained the same values. This was most likely caused by the impact of the fall, which caused some wirings to become loose. We removed these erroneous data and made sure the device was properly connected and working.

Next, we extracted additional features from the data, such as the maximum peak-to-peak acceleration amplitude, sum vector magnitude, angle between the z-axis and vertical, orientation change in the horizontal plane, etc. We now have two data sets, the raw data and the extracted features. These datasets were split into train and test sets, trained using a multilayer perceptron (MLP) model and long short-term memory (LSTM) model separately, resulting in four different models.

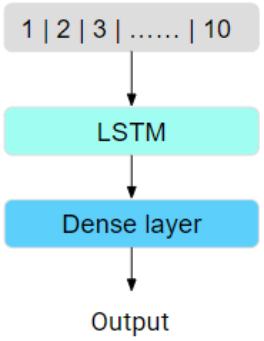
Training method for MLP

Each data instance was combined with its previous 9 samples to form a single row so that the model could take into account the previous events. These data are then fed into the model with two hidden layers. The output layer uses the softmax activation function to predict the final label.



Training method for LSTM

Similar to MLP, 10 samples were combined before feeding them into the model with a lookback of 10. The output from the LSTM layer was passed to a fully connected layer with a softmax activation function to predict the label.



3.2.3 Model testing

After training, the accuracy is higher for input data without feature extraction, so we just focused on the models trained with raw data collected. Between the two models, MLP had a higher

accuracy at around 78% and was significantly faster to train. The table below summarizes their performance.

	LSTM	MLP
Accuracy	71.0%	78.1%
Training time	60s / epoch Several hours for > 100 epochs	1s / epoch Few minutes for > 100 epochs
Prediction time	~50ms	~50ms

Since MLP offers a higher accuracy score, we have chosen it as our final fall detection model.

3.2.4 Real-world testing

Initially, we used HTTP to send data collected to the cloud for prediction. Each new data instance was combined with the previous 9 samples before prediction. This posed an issue of communication latency between the WeMos and the cloud. We experienced a 30-second delay between the actual fall and the response sent back by the cloud. Hence, we replaced HTTP with MQTT, which is a lightweight messaging protocol used widely for the Internet of Things (IoT), reducing communication delay. Besides, MQTT also supports authentication and data security mechanisms such as TLS encryption which provides safe communication. The table below shows the comparisons between HTTP and MQTT.[4]

No. messages in a connection cycle for MQTT	MQTT average response time per message (ms)	HTTP average response time per message (ms)
1	113	289
100	47	289
1000	43	289

Besides, data was sent to the cloud at a rate of 10Hz, resulting in high power consumption. After inspection of our dataset, we realised that the acceleration usually had a value of around 1.5g (gravitational acceleration) during falls. Thus, we modified the WeMos to only send data to the cloud when the acceleration is greater than 1.3. By reducing the number of communications,

[4] https://www.google.com/url?q=https://www.hivemq.com/blog/mqtt-vs-http-protocols-in-iot-iiot/%23:~:text%3DThey%2520both%2520run%2520over%2520TCP,respond%2520to%2520requests%2520from%2520clients&sa=D&source=docs&ust=1668610255584354&usg=AOvVaw0e3wNImvp1ksVpvz9iES5a

we lowered the power consumption, extending the lifetime of the WeMos. In addition, this further reduced the delay to less than 5 seconds since fewer predictions were made in the cloud.

3.3 Vehicle Detection and Alert

3.3.1 Obtaining Video Stream

RTSP is used to obtain a video stream from the smartphone camera attached to the back of the bicycle. The application IP Camera is used on the phone to start the stream.

Initially, at the server side program, there is a while loop in which we first obtain the frame from the video stream and process the frame accordingly before obtaining the next frame. There is an issue with this approach. Reading frames from the stream using cv2.VideoCapture.read() is a blocking operation, and the processing of the frames is blocking as well. To counter this, we used multithreading to obtain the frames in parallel with the processing of the frame, which significantly reduced the lag and enabled “pseudo-real-time” processing of the frames.

3.3.2 Detecting Vehicles

We have tried several methods for vehicle detection. One of which is the Haar feature-based cascade classifier. It is fast but lacks accuracy. We needed a way to detect incoming vehicles fast and accurately, which led us to use a pre-trained Tiny-YOLOv3 vehicle detection model.

		M-30	M-30-HD	Urban1
Haar Cascade	Time [s]	0.08–0.13	0.3–0.44	0.02–0.06
	Accuracy	43%	75%	40%
SSD	Time [s]	4–7	11–14	2.6–5.6
	Accuracy	22%	70%	69%
YOLO v3	Time [s]	1.0–1.8	1.0–1.8	1.0–1.8
	Accuracy	82%	86%	91%
Mask R-CNN	Time [s]	2.4–3.0	2.4–3.0	2.4–3.0
	Accuracy	89%	91%	46%

Performance comparison of different detectors[5]

3.3.3 Vehicle Detection Algorithm

The algorithm is as follows:

[5] https://www.researchgate.net/figure/Performance-comparison-according-to-the-processing-time-and-vehicle-detection-accuracy-of_tbl1_337617332

1. Detect vehicles and obtain the sizes of the vehicles by the area of the bounding box
2. For each frame, obtain the maximum size of the bounding boxes (*MAX_SIZE*)
3. Compare the current *MAX_SIZE* with the previous frame's MAX_SIZE in each frame. ratio = current *MAX_SIZE* / previous *MAX_SIZE*
4. Depending on the threshold, alert the cyclist about the incoming vehicles

- Ratio < 1.10: safe
- Ratio between 1.10 and 1.38: approaching
- Ratio > 1.39: danger (approaching fast)

5. If the *MAX_SIZE* is above *MAX_SIZE_THRESHOLD*, alert the cyclist that the vehicle is too near.

However, there is an issue with this implementation as this detection model takes the whole frame of the camera into account. The region of our interest is only the lane the bicycle is on and so we need a way to ignore the opposite lane where cars travel in the other way and other lanes the bicycle is not concerned about. In other words, we only care about the vehicles on the lane the bicycle is on.

Hence, we limit the region of interest to be in a conical shape and only care about the cars whose centroid lies within this conical shape area which resembles the current lane.

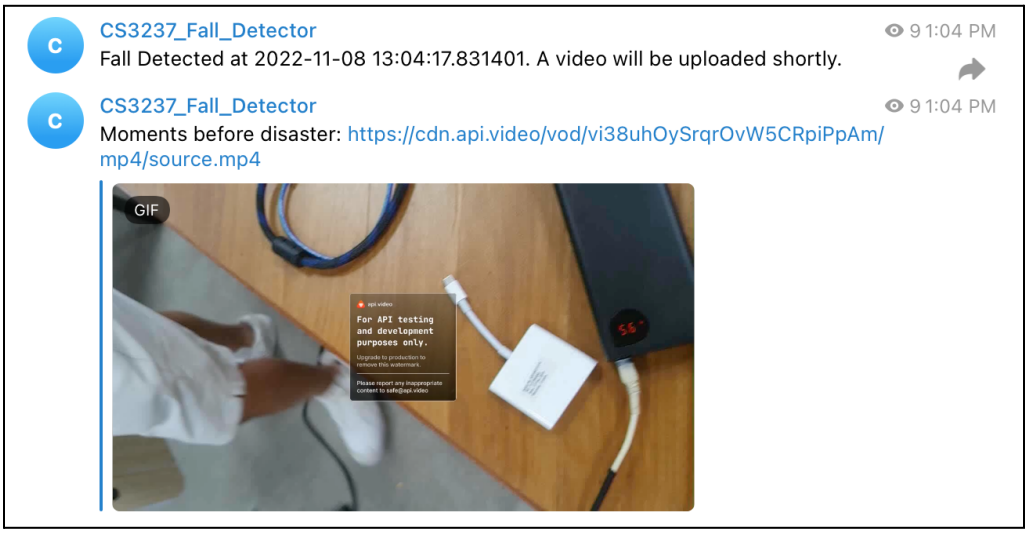
3.4 Video Recording

While the video stream is being read, it is also being recorded and saved to the server in 10 seconds chunks. It will only keep the most recent 20 videos at any point to save storage space.

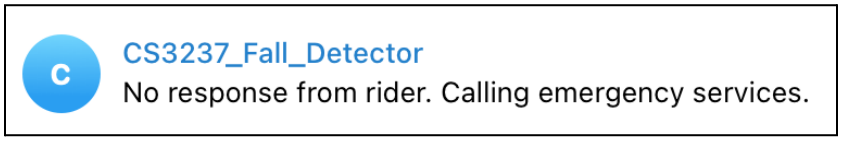
3.5 Telegram Bot

A companion telegram bot is also created for those concerned about the cyclist's safety. Users can subscribe to the telegram bot to get the latest updates about the cyclist's state if a fall is detected.

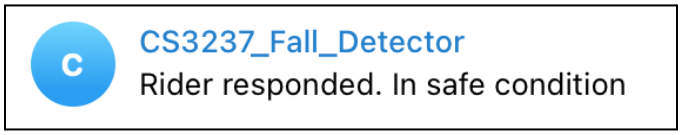
When a system detects a fall, the telegram bot will automatically send an alert to the subscribers, notifying them that a fall has been detected, along with a recent video recording (past 20 seconds). This recording aims to help the subscribers to monitor the cyclist's condition.



If there are no responses from the cyclists after 30 seconds, the telegram bot will automatically notify the subscribers to call emergency services or contact the cyclist.



Suppose a fall is wrongly detected or the cyclist has recovered from the fall safely, the cyclist would need to press on the recover button to indicate his/her safety. A message will then be automatically sent to the subscribers through the Telegram bot to notify the cyclist's safety.



3.5.1 Fall detection MQTT handler

A script is run on the server side. It subscribes to *fall-actuator/event* topic. Once it receives a *fall* event from the topic, it will notify a Flask server via REST to act accordingly.

3.5.2 Flask Server

The Flask server receives requests from the MQTT handler and acts as a middleman.
A telegram server is used to send messages, and the api.video server is used to upload video recordings which will be saved for 24 hours. The communication between these servers is done via REST.

This Flask server has three endpoints:

1. /fall - sends an alert through telegram, uploads the most recent recording to the api.video server and sends the link to the uploaded video through telegram
2. /emergency - sends an alert when 30 seconds has passed since the fall and no response is detected
3. /recovered - sends an alert when the cyclist has recovered from the fall

3.5 Summary of the Overall Flow

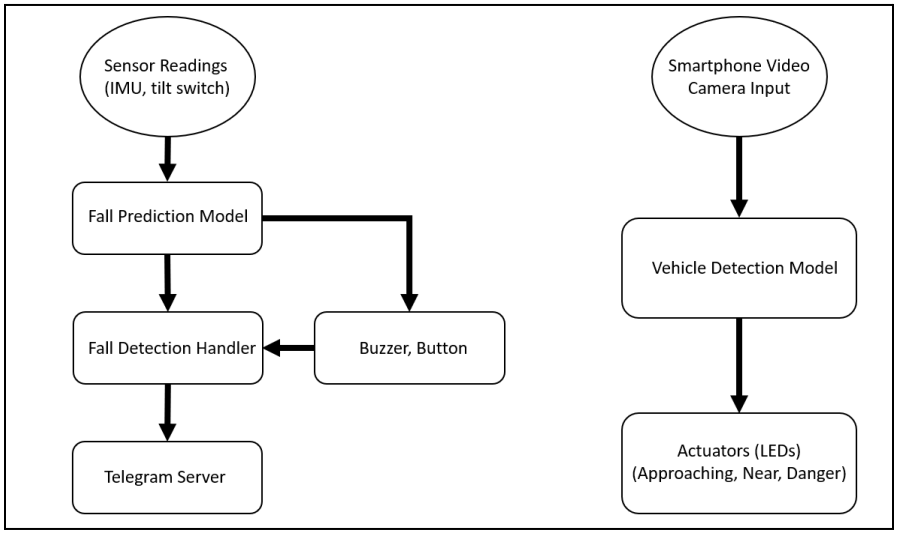


Diagram of Overall Flow

The flow chart above depicts the sequence of data transmission for both the fall detection model and the vehicle detection model.

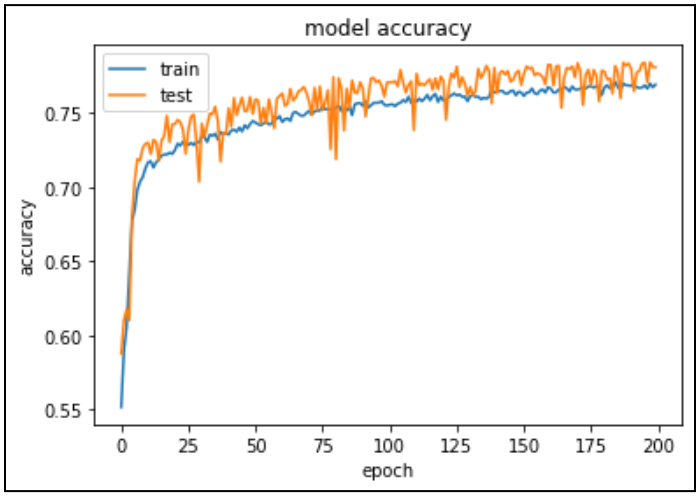
For fall detection, readings from the IMU and tilt switch are used as inputs for the fall prediction model which subsequently sends the data to the fall detection handler. In the event of a fall, the fall detection handler will send data to the telegram server, which outputs the message and video in the telegram channel. At the same time, the buzzer will also play an alarm to alert others in the surroundings. The siren from the buzzer can be stopped using a button and this will also send the data to the fall detection handler to output a different message in the Telegram channel, informing the channel subscribers that the cyclist has responded.

For vehicle detection, the video camera from the smartphone acts as the input for the vehicle detection model. Depending on the different events that occur, the actuators, which are the three LEDs, will light up accordingly to notify the cyclist if the vehicle behind is approaching, near or dangerous.

4. Experimental Evaluation

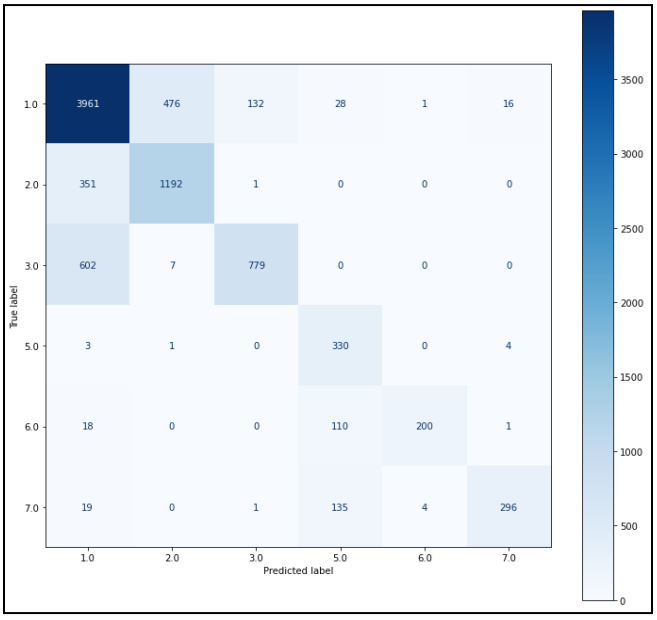
4.1 Fall Detection

We plotted the learning curve of this model and verified that underfitting did not happen because the model had a satisfactory accuracy score of 78.1%. Besides, there is also no evidence of overfitting as the train and test performance curves are close to each other.

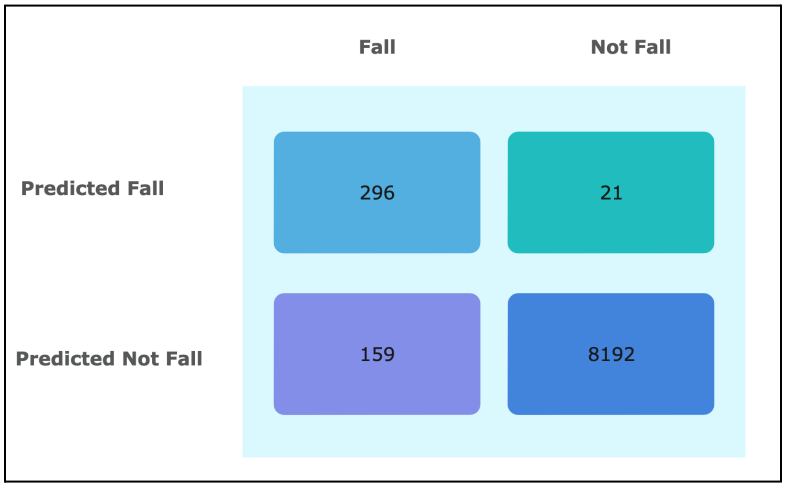


Learning curve of fall detection model

We utilized the confusion matrix to measure the performance of the model classification. Since we are more interested in the *fall* event (event 7), we generated a more specific confusion matrix based on the values below.



Confusion matrix of fall detection model



Confusion matrix of fall event

True Positive (TP) = 296 False Positive (FP) = 21
False Negative (FN) = 159 True Negative (TN) = 8192

False positives and false negatives are equally costly in our system. If the system keeps wrongly detecting a fall (false positive), it will affect the usability and cycling experience. Similarly, if the system does not react when there is a fall, then the system will not be able to serve its purpose. Hence, we should take both false positives and negatives into account. Since the data is highly imbalanced, f1 score will provide a better assessment of the model performance.

Precision = TP / (TP + FP) = 0.9338, Recall = TP / (TP + FN) = 0.6505
f1-score = 1 / (1 / precision + 1 / recall) = 0.7668

As f1-score is the harmonic mean of precision and recall, the model is performing decently well against false positives and false negatives.

4.2 Battery Life Estimation

On average, the current consumption for both of the WeMoses operating normally is around 220mA by experimental evaluation. With constant transmitting (without the 1.3g resultant acceleration check prior to transmission), the average current measured would be around 260mA instead.

Using a 5000 mAh power bank as a gauge, battery life would be:
T1 = 5000/220 = 20.727 hours = 20 hours 44 minutes
T2 = 5000/260 = 19.231 hours = 19 hours 14 minutes

Sampling method	Average Current (mA)	Battery Life
With 1.3g acceleration check	220	20 hours 44 minutes
Constant Transmission	260	19 hours 14 minutes

However, this also depends on the battery life of the smartphone as well as the smartphone needs to be transmitting the video stream. On average, the battery life measured for the smartphone used (Galaxy S10+) in our case is around 2.5 hours of continuous streaming. A more efficient low-power camera can be used here instead to maximize battery life.

4.3 Security

Security is an important aspect of our IoT system as we need to prevent cyber attacks and the leakage of information. Firstly, Transport Layer Security (TLS) is enabled to encrypt the communication between the broker and the clients, to reduce the risk of third parties reading or altering the content during transmission.

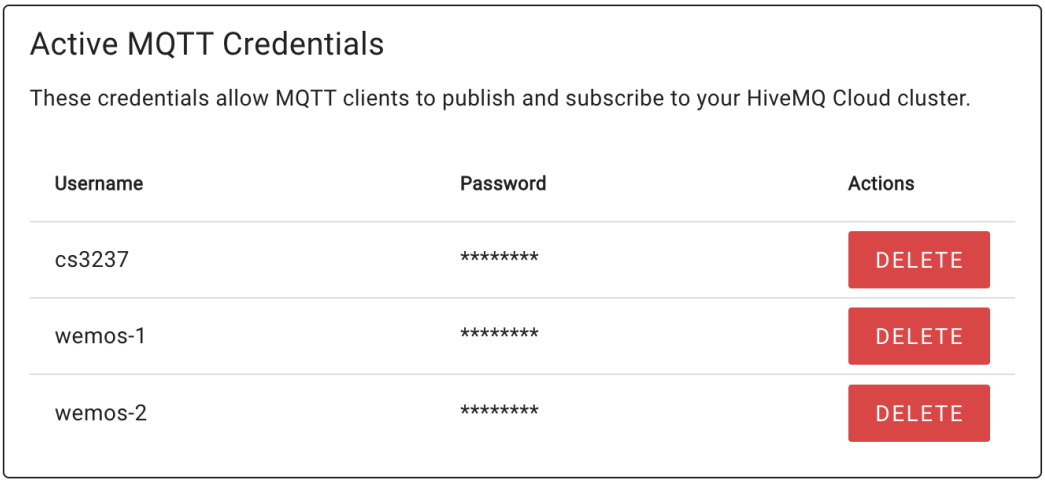
```
int numCerts = certStore.initCertStore(LittleFS, PSTR("/certs.idx"), PSTR("/certs.ar"));
Serial.printf("Number of CA certs read: %d\n", numCerts);
if (numCerts == 0) {
  Serial.printf("No certs found. Did you run certs-from-mozilla.py and upload the LittleFS d
  return; // Can't connect to anything w/o certs!
}

BearSSL::WiFiClientSecure *bear = new BearSSL::WiFiClientSecure();
// Integrate the cert store with this connection
bear->setCertStore(&certStore);

client = new PubSubClient(*bear);
```

Code to Initialize Secure Connections

At the application level, the MQTT server is also configured to request client authentication using a valid password and username before a connection is granted. Upon authentication, the MQTT broker can provide authorization to control which clients can be connected to the server and what topics a client can subscribe or publish to. This implementation helps to prevent authenticated users from sniffing data of other users.



Access Management on HiveMQ

5. Challenges and Limitations

5.1 Challenges

One of the challenges that we faced would be data collection. In order to develop a working fall detection model, we had to collect thousands of data points for each cycling event, which took more than 10 hours in total. Not only that, we still needed to experiment with various methods or parameters such as sampling rate, communication protocol, machine learning models, number of cycling events, etc, until we found the most successful one. For instance, we had to recollect the data when we replaced HTTP with MQTT as the communication protocol as their communication latency is dissimilar, which could negatively influence the result if we reuse the same dataset. The data collection alone required our group to meet physically weekly for about 4 hours.

Other than that, the source of inaccuracies of the model in the real-world setting is also difficult to pinpoint. It could be due to inconsistencies in data collection, uninformative features, poor performances of the machine learning models, unstable network connection during data collection or real-world testing, sensor fault, bugs in the model prediction code, or all of them at once. Each component had to be checked over for errors to detect the issue.

5.2 Limitations

One of the limitations is the fall detection model being unable to distinguish between some cycling events with similar motions. For example, there are instances where the model misclassified the *turning left* as the *turning right* event. We suspect this misclassification is due to the lack of variation when we collected the *turning left* event, causing it to be less robust and only able to detect a narrow range of cases. However, since this misclassification does not directly affect the model's ability to detect a fall, we did not trouble ourselves to look into the issue and focused working on the other main aspects instead.

Next, the vehicle detection part only works during the day where the image captured is clearer. A night vision camera or better low-light performance camera can be used to counteract this.
The algorithm for alerting danger is also fairly simple and only works properly when the approaching vehicle behind is fairly head-on. If the vehicle behind faces the camera to the side, the algorithm may not perform correctly.

6. Future Extension

6.1 Case Design

Although the current case design is functional, it is not ideal for real-life usage. The big size of the case might have poorer installability and low stability on the bicycle. Going forward, we plan to improve the case design so that it would fit better on the bicycle and be able to withstand harsh environments.

6.2 Battery Life

A dynamo can be attached to the wheels of the bicycle to recharge the power bank or rechargeable battery to prolong the battery life of the device. A dedicated smaller low-powered camera module can also be used to replace the smartphone to have better battery life.

6.3 Dedicated distance sensor

Due to project requirements constraints, we are unable to use external sensors. However, if there are no such constraints, an ultrasonic or LiDAR camera can be used to accurately detect the vehicles behind instead of solely relying on the smartphone camera. If depth or distance can be measured accurately, the detection algorithm used can be much more advanced and more accurate.

6.4 Integration with Emergency Medical Service

Another path forward is integrating the fall alert functionality with the emergency medical service. Should a fall be detected in our system, essential information such as user basic information and current location will be delivered to agencies like Singapore Civil Defence Force for immediate action.