University of Bahrain
College of Engineering
Electrical engineering Department

Date of publication January 08, 2022, date of current version January 05, 2021.

A Smart Blackbox for Vehicle Tracking and Monitoring

Ahmed A. Alsahlawi[1]

[1]Student at Department of Electrical Engineering, University of Bahrain, Kingdom of Bahrain

Corresponding author: Ahmed A. Alsahlawi (e-mail: 20092127@ stu.uob.edu.bh).

ABSTRACT This paper conducts a comprehensive study of applying the concept of airplanes Blackbox on vehicle tracking and monitoring at the time of accidents. The solution combines hardware tools like microcontrollers and sensors and software like MQTT and Node-Red. the data exchanged between Blackbox and the MQTT server is essential to build confidence in the severity of the accident. The huge leap in the Internet of things (IoT) field allowed for prototyping smart solutions with the minimum cost required. This paper highlights the drawbacks of the current systems, showing the missing parameters required to perform accurate analysis. Constructing the solution is done in steps, starting from connecting hardware until showing the results. The challenges of implementing such a solution are also highlighted besides the possible enhancements for future work. this device can also be applied in multiple areas such as insurance risk analysis and fleet management.

INDEX TERMS IoT, Blackbox, Vehicle Accidents, GPS Tracking

NOMENCLATURE

Abbreviations

AI	Artificial Intelligence
IoT	Internet of Things
BD	Big Data
ML	Machine Learning
DL	Deep Learning
5G	Fifth generation
ICT	Information and Communication Technology
Wi-Fi	Wireless Fidelity
MCU	Microcontroller
MQTT	MQ Telemetry Transport

I. INTRODUCTION

The concept of Blackbox is not something new. It was implemented first in airplanes, to capture the necessary information such as the speed, geographical location, attitude, and several other variables such as engine and fuel information. The same concept can be adapted in any transport method to provide useful and accurate information at the time of the accidents. With the emerge of the Internet of Things (IoT) field, the cost of implementing such devices is almost cheap and the complexity is acceptable. Of course, like any other technology, there are multiple challenges involved. This paper's main aim is to prototype the concept of Blackbox for vehicles, by implementing it using simple microcontrollers and a few sets of sensors like GPS, shock sensors to mimic the Blackbox prototype.

A. Motivation

Most countries nowadays are shifting to become smart cities and automate all their processes with the help of Artificial Intelligence (AI). Some of these cities for example are securing their utilities with regular or smart surveillance, mostly to reduce crimes, prevent illegal constructions or sometimes to just monitor the traffic to either detect traffic jams or to study some accidents. Most of the time, the process involved in these studies is manual, where the responsible staff replays the accident to study exactly what happened. There are multiple factors here that can help the staff to take specific decisions, which are the quality of captured video or the angle where the footage is taken from. Unfortunately, sometimes the angle is not properly aligned with the accident or sometimes it's too far and the quality of the footage is not that clear, so this can result in inaccuracy in judgments. In another scenario, some of the countries are already forcing drivers to install dash cameras in their vehicles, which is quite helpful but in some severe accidents it can be very hard to judge, or the recording can be damaged before it's being extracted for an investigation like in fire accidents. Especially since these dash-cams are

offline, with no utility to update the footage to the cloud in a frequent manner.

B. RELATED WORK

After searching thoroughly to my best effort in big journals for recent applications like IEEE and Elsevier, I only found one article that speaks particularly about the Blackbox prototyping for vehicles. This could be probably because most of the modern vehicles are having such technologies built within, but let's not forget the fact that these features are available in high-end vehicles with full options set. Some articles speak about vehicles in smart grids or traffic management. One article found by Ke Lu et. al. spoke about detecting driver sleepiness using consumer wearable devices in manual and partial automated real-world driving. The study is based on training Artificial Neural Networks (ANN) to analyze and study the reading of heartrate features to determine if the driver is sleeping or not [1]. The main challenge in this approach is that it depends on smart wearables to capture the data, which is dependent on the driver not on the vehicle. Another Study done by Abdallah Kassem [2], very related to this topic is speaking about prototyping a Blackbox for vehicles. The author used several sensors, like speed sensors, water sensors, switches, accident sensors, brake and belt sensors. The sensors were interfaced with a microcontroller. Although this research is containing more sensors than what we are suggesting here, the main lack of this implementation is that it is not communicating the data into the cloud. The software is designed with a Visual Basic application to pull the data whenever it is connected. The second drawback of this implementation is that it is outdated, the research is conducted in 2008 and there are a lot of gaps in technology between now and back then. This is an issue since modern cities started to adopt Electric Vehicles (EV) and Smart Power Grids (SG), and the main advantage of these two is that they are depending on data exchange between vehicles and SG.

C. Paper Outline

The rest of the paper is divided into the following sections: Section II will talk about the problem and the theoretical

part of the solution. Section III will illustrate the practical work done and the software interfacing the captured data. Challenges and possible enhancements are listed in Section VI. Final section, will provide the conclusion to this work.

II.BACKGROUND

The biggest issue involved in assessing those accidents is the lack of information. You will need an expert to analyze the footage and extract all the possible information. These experts can cost traffic departments, besides even with an expert eye there is always a possibility of human errors.

Another contributing factor is vehicle speed, which is absent at the time of judgment, and could be roughly estimated by those experts depending on braking marks. Moreover, the force of the impact, which is generated by transferring collision energy from car body to the collided an object is an important factor that can enhance damage assessment. Therefore, the existence of a cost-efficient and accurate Blackbox in a vehicle during an accident, can change the equation and unfold an unseen detail.

The unit will be combined with multiple modules besides the main microcontroller board. First, we will need a good board with reasonable processing power, which can handle the attached modules and perform additional machine learning operations such as TinyML. Then, we will need a GPS module, to get speed information. The location could be useful if we add it to the factors list when performing advanced analysis. Third, a Wi-Fi/GSM module is required to publish the sensed data into the cloud for further analysis. Last, we will need a Vibration/accelerometer sensor, to measure the force

of the impact during the collision. These modules attached with the microcontroller can make a very cost-effective and more importantly a great tool to capture useful information to assess these collisions and to provide additionally a forecast to prevent such accidents in the future, therefore more lives can be saved. See figure 1, which illustrates a high overview of the proposed components.

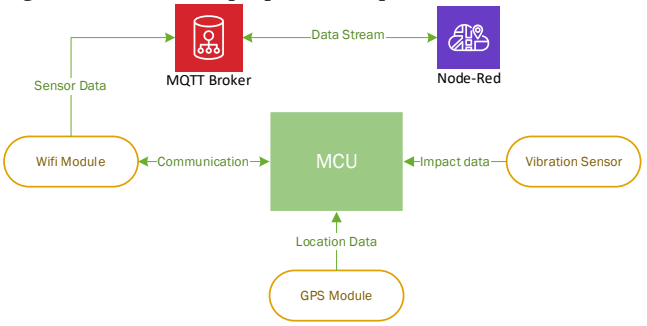

FIGURE 1. High level System overview

MQTT is a popular protocol for data transport, and it's widely used in the IoT. The main characteristic of MQTT is that it can publish and subscribe for data exchange between devices and brokers. The packet published contains multiple information such as topic name, payload message, QoS. On the other hand, the broker can also publish something for subscribers to give some instructions. This allows communication on many devices and reliably allows data exchange.

C. Hardware Parts

The selected hardware based on figure (1) is as follows:

Microcontroller

The selected microcontroller is ESP8266 board from ESPRESSIF community. The board is having multiple advantages, such as enhanced power saving architecture which makes it works on batter for very long time. It's highly durable, can work in hard conditions and it's compact and small which allows for smaller prototype design and fitting. The most important feature of this board that it contains Wi-Fi chip onboard, so this factor will reduce external components and will ease the setup of Wi-Fi communications [3].

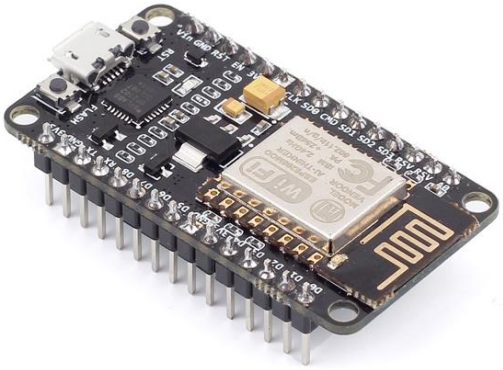

FIGURE 2. ESP8266 microcontroller

Vibration Sensor

For vibration, I have used SW-420 vibration sensor, it's widely popular. The way it works is that it opens when a vibration is detected and closes when there is no vibration. The sensor contains 3 wirings, 2 for power (in & out), and D0 to output vibration strength. There is also an LED connected to on the board, goes high and off if there is a vibration detected [4]. This sensor is operating on 3.3 to 5v, so this is well flexible to install on any board.

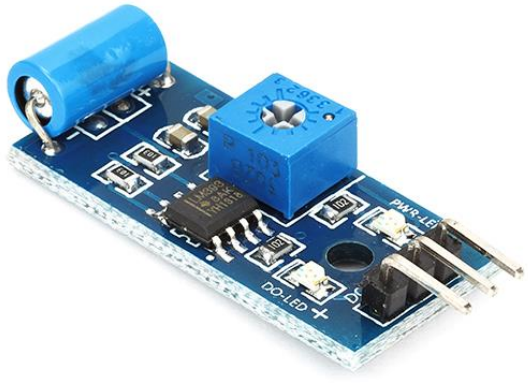

FIGURE 3. SW-420 VIBRATION SENSOR

GPS Sensor

The selected GPS sensor is by Goouuu Tech Company, the model is GT-U7. The sensor comes with several protocols which is UART, USB to interface it with USB supported MCU and it is also compatible with Serial Peripheral Interface (SPI). The sensor built with powerful power management, with echo Mode to save power when location information is not changed. The accuracy is very high, supported with A-GPS and Assist Now Autonomous feature [5].

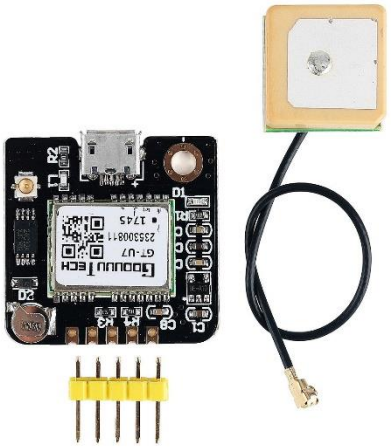

FIGURE 4. GT-U7 GPS NEO-6M SENSOR

For the component above are connected with regular breadboard to connect wires. Power source is from USB port (External 3.3v Power). The software used in this experiment is as the following:

1. Arduino IDE

The Arduino IDE is used with C language to develop the sketch containing the code required to connect the components and capture the data form the sensors, store them into variables then push them to the MQTT Broker. The figure below shows the interface of Arduino IDE.

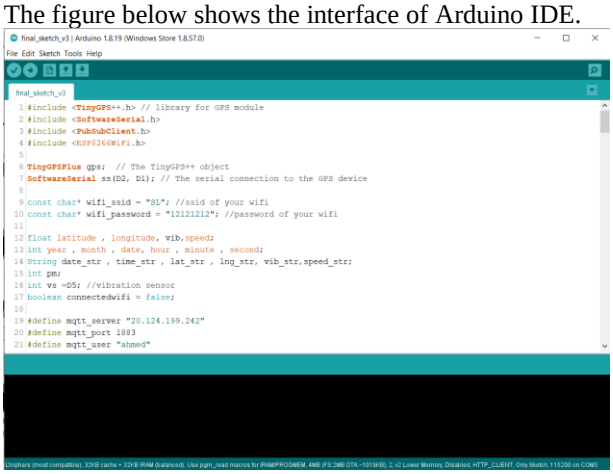

FIGURE 5. ARDUINO IDE INTERFACE

2. Mosquito MQTT Broker

Mosquito is open source MQTT broker, based on version 5.0 of the MQTT protocol. It's a lightweight application that can be installed in any environment and use in any low-power MCUs connectivity for edge computing. This will be installed in a cloud server on Azure, to allow it to distribute MQTT to both local and online hosts.

3. MQTT Explorer

Open-source application by Thomas Nordquist, to explore and see MQTT topics/ messages published to the broker. Below is a screenshot to the interface showing sample of our published topics through the MQTT:

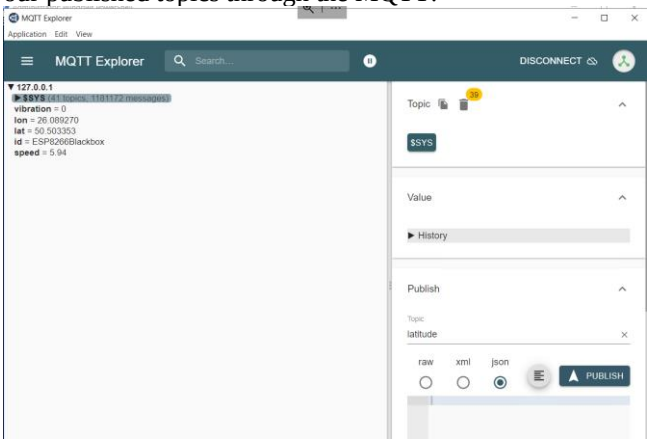

FIGURE 6. MQTT EXPLORER APPLICATION

4. Node-RED

Node red is a programming tool to connect hardware devices and enhance the flow of IoT processes by combining multiple features and libraries. Node-red is developed initially by IBM, with JavaScript library Node.js [6]. The node red can be installed anywhere, on local machine or on server.

III. LAB WORK

This section will carry the steps followed to connect the components, show the code and the end results.

1- First, we will need to connect the hardware together with the ESP8266 board. The connectivity of the components is shown in the schematic diagram below:

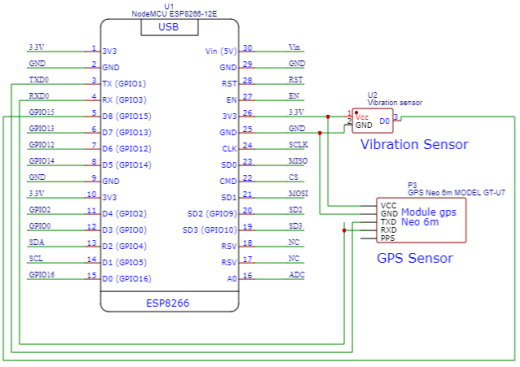

FIGURE 7. SCHEMATIC DIAGRAM OF THE BLACKBOX

2- Besides the diagram below, I will Add an LED with resistor and connect it to D5. This is optional step, I will just use it to blink if the MQTT broker connected as the device will not have any other outputs to indicate or debug. This step is just for debugging.

3- The final connections of the components are shown in the image below.

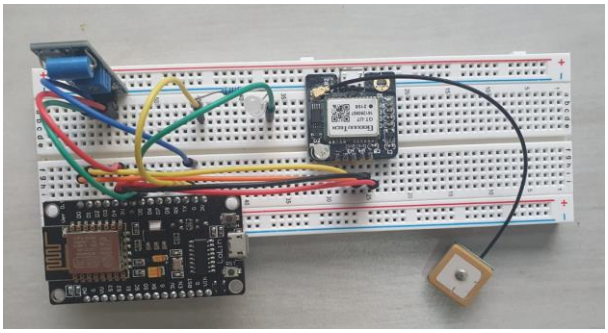

FIGURE 8. MCU AND SENSOR ARE CONNECTED TOGETHER

4- Now we need to connect the NodeMCU (ESP8266) to the computer to install the sketch needed to capture vehicle data. To perform this step, we will need to first install ESP8266 USB drivers from [3]. After installing the drivers, we need to add the library to Arduino IDE to install board drivers to program it

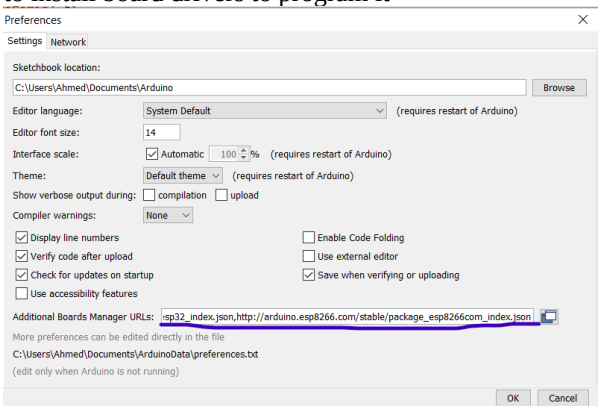

FIGURE 9. ARDUINO IDE PREFERENCE SCREEN

5- After adding the library to the preference section, you will be able to see the board in Board Manager

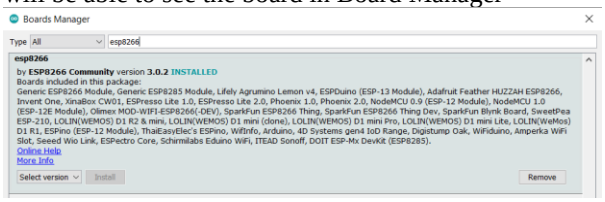

FIGURE 10. BOARD MANAGER IN ARDUINO IDE

6- now the board is added and therefore we can program it with any sketch we want. The next step will be to do the code needed (detailed code notes in appendix B)

7- Now for the code, we will include the needed libraries:

```
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
#include <PubSubClient.h>
#include <ESP8266WiFi.h>

TinyGPSPlus gps;
SoftwareSerial ss(D2, D1);
```

TinyGPS++ is the library for GPS module [7], it will provide a decode mechanism for GPS signals. Software serial is a library used to shift the usage of original serial ports (TX and RX) to other digital pins. In our case we selected D1 and D2. This step is needed to avoid the inturrption in TX and RX when the program is uploaded or resetted in the MCU. Then we setup wifi credentials to connect to the Wi-Fi.

```
const char* wifi_ssid = "xxx";
const char* wifi_password = "xxx";
```

Afterward, we do the setup for the MQTT broker server, which is hosted in the internet in Azure web server.

```
#define mqtt_server "20.124.199.242"
#define mqtt_port 1883
#define mqtt_user "ahmed"
#define mqtt_password "password"
#define out_topic "vibration"
#define location_long_topic "lon"
#define location_lat_topic "lat"
#define device_id "id"
#define location_speed_topic "speed"
```

8- After setting up the variable and initiating the MQTT broker, we then start to read GPS with vibration sensor Data from the sensors and publish them into the topics via MQTT client. The code below

```
client.publish(out_topic,
String(measurement).c_str(), true);

client.publish(location_long_topic,
lat_str.c_str(), true);

client.publish(location_lat_topic,
lng_str.c_str(), true);

client.publish(device_id,"ESP8266Blackbo
x", true);

client.publish(location_speed_topic,
speed_str.c_str(), true);
```

9- Now Once the GPS is connected, it takes up to 5 seconds to get signals and connect to GPS satellites. Only then, sensor LED will start to blink more frequently and gets the data. To test if it is working well, I have connected to Azure server via the Remote Desktop tool by Microsoft and logged in into MQTT explorer. See appendix A for more on Azure Setup.

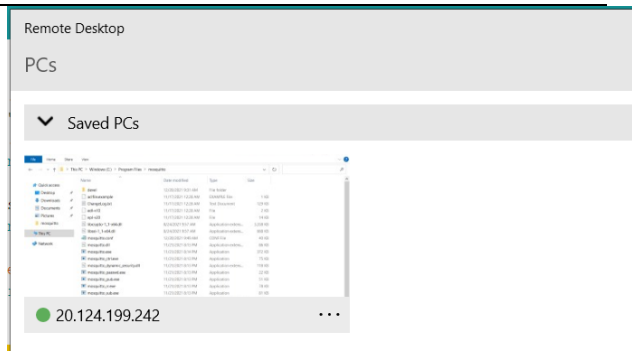

FIGURE 11. MICROSOFT REMOTE DESKTOP APPLICAITON

10- After checking the MQTT explorer, I started to get data for my current position.

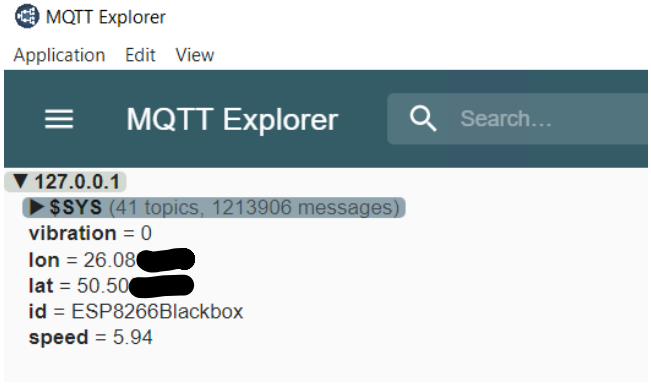

FIGURE 12. MQTT EXPLORDER, SHOWING MY CURRENT LOCATION

Vibration is given with integer value, Longitude and Latitude are given with Geo-Coordinates, ID is passed by to identify the device (in case we have more), Speed is with Kilometers per hour.

11- Now after connecting everything, we we are getting all what we need from the sensors. We will shift our attention to handle these data within Node-red.

12- Installing Node-Red in the PC will require Node.js to be installed as a pre-requisite. We download it from the original website at https://nodejs.org/en/ .

13- Afterward, we install node-red application from it's website https://nodered.org/ . Note that Node-Red is host application, which means it can be installed within my local computer, web server or even in an android device or Raspberry Pi x. As long as my MQTT broker is setup in an online server, it can serve Node-Red or any application capable of handling/receiving MQTT messages wherever it's installed.

14- Now the next step is to run the node red in the computer to start creating flows and handling the data received from my sensors. See the figure below of how we can start Node-red.

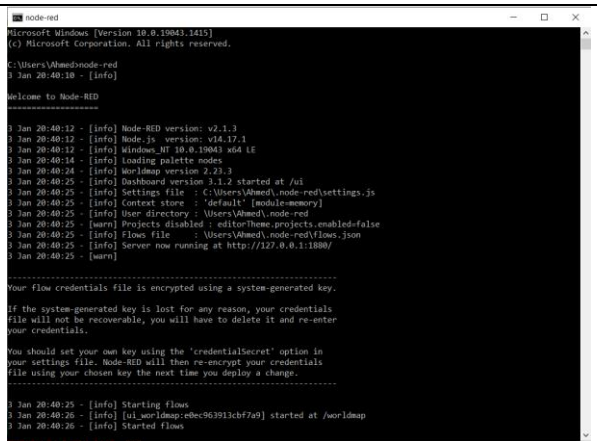

FIGURE 13. NODE-RED STARTUP

15- After the lunch of Node-RED, we visit the link displayed in the log as provided http://127.0.0.1:1880 to see the node-red interface, as illustrated in figure 13

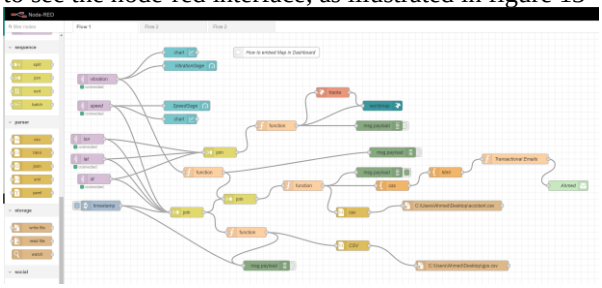

FIGURE 14. NODE-RED FLOW CONTROL SCREEN

To prototype the Blackbox, I have split our data processing into two parts. The first part will take care of the interface to display vehicle information within charts in a dashboard. The second part will store the location of the vehicle with its speed and vibration strength whenever the vibration exceeds a certain limit. This should represent the impact moment.

16- For the first part, we will display the live data into our dashboard to reflect sensor data in real-time.

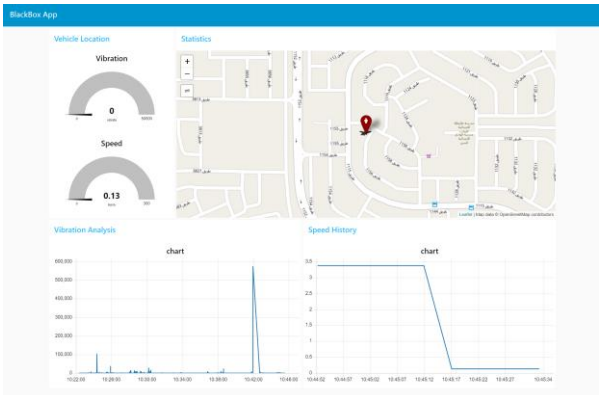

FIGURE 15. NODE-RED DASHBOARD SCREEN

17- Next, we will do the analysis part. And for this, we will start analyzing the variables. For our scope, we will consider vibration readings to determine the power of the impact. With Node-Red, we will create a function to decide if it is an impact or not. This can be easily done with the following code in node red:

```
if (msg.payload >= 1500)
{
    return msg;
}
```

For our experiment, I made an impact decision stands on the value of 1500. Anything beyond will consider an impact. Note here that this will need to be adjusted based on the environment of the vehicle, we don't want a speed bump to be considered as an impact.

18- Finally, after filtering the accidents, we will have to store the message payload in excel. This can be done easily in Node-Red with "csv" node, which accept the payload and store it directly in excel file. You can see clearly the speed of the vehicle, vibration strength, longitude and latitude of the vehicle, ID of the device and Linux based timestamp. See figure (16):

	speed	vibration	lon	lat	id	timestamp
33	13.5	52986	26.08946	50.503685	ESP8266Blackbox	1641246105478
34	13.5	52986	26.08946	50.503685	ESP8266Blackbox	1641246106055
35	13.5	52986	26.08946	50.503685	ESP8266Blackbox	1641246108106
36	13.5	1526	26.08946	50.503685	ESP8266Blackbox	1641246119815
37	13.5	1526	26.08946	50.503685	ESP8266Blackbox	1641246120822
38	16.5	9390	26.089298	50.503479	ESP8266Blackbox	1641246121846
39	16.5	9390	26.089298	50.503479	ESP8266Blackbox	1641246122836
40	16.5	9390	26.089298	50.503479	ESP8266Blackbox	1641246126336
41	16.5	14761	26.089298	50.503479	ESP8266Blackbox	1641246137467
42	16.5	14761	26.089298	50.503479	ESP8266Blackbox	1641246138446
43	16.5	14761	26.089298	50.503479	ESP8266Blackbox	1641246138868

FIGURE 16. IMPACT DETAILS RECORDED IN EXCEL

19- On the other side, I have made a notification to reach traffic department, to alert them with accidents. This could be useful if they want to send a policeman to the site. The image below shows sample of the email to be sent:

Dear Traffic Department,

An Accident just happened in with the following details:

Geo-Coordinates 26.089199,50.503368

Speed: 1.59 Km/H

IT Department

FIGURE 17. EMAIL SENT TO TRAFFIC DEPARTMENT

20- On the other hand, if there is a critical accident, the driver will be notified and the national ambulance system that there is a critical accident happened. Fast report will allow to save more lives as in such accidents every second counts. the driver will receive a message that an ambulance is reaching him, to calm him down.

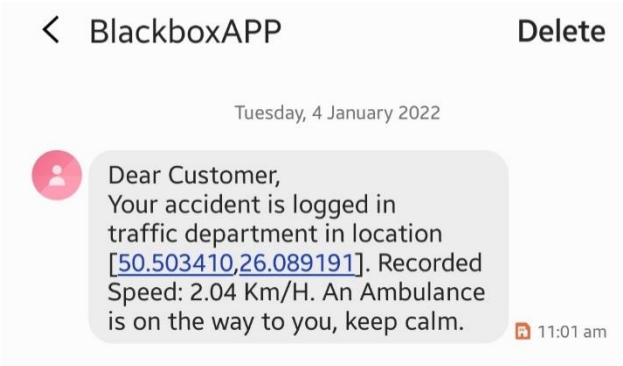

FIGURE 18. DRIVER MESSAGE FOR CRITICAL ACCIDENTS

The SMS below is configured using Twillio provider, we setup a Twilio account, and capture the information provided to start sending SMS to the client:

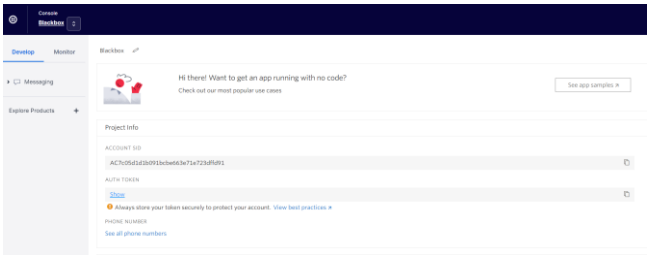

FIGURE 19. TWILLIO ACCOUNT MAIN CONFIG SCREEN

We insert the provided information to Twilio node in Node-Red, with configuring the name and the from string. This way the application is ready to send messages.

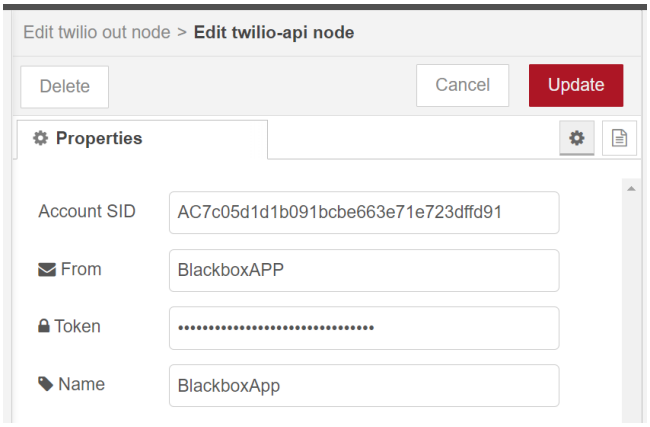

FIGURE 20. TWILLIO NODE SETUP IN NODE-RED

IV. RESULTS

I have taken the device into a test drive, into Bahrain main highway, to verify the accurateness of the data collected. The data gathered is accurate. The speed has 1 degree the actual vehicle, but most of the time it gives exact readings. The 1 degree of error could be because of communication delays, as I had quite slow connectivity to the internet. But even with this degree, the result is still almost accurate and can give 99% accuracy in collision analysis. see figure 21, it shows the data for actual trip I took to test the device GPS. You can clearly see the device on my hand (named it ESP8266Blackbox), changes in location and speed.

13242				
13243	26.166969	50.503677	109.68	ESP8266Blackbox
13244	26.166969	50.503677	109.68	ESP8266Blackbox
13245	26.166969	50.503677	109.68	ESP8266Blackbox
13246	26.166969	50.503677	109.68	ESP8266Blackbox
13247	26.166969	50.503677	109.68	ESP8266Blackbox
13248	26.166969	50.503677	109.68	ESP8266Blackbox
13249	26.166969	50.503677	109.68	ESP8266Blackbox
13250	26.166969	50.503677	109.68	ESP8266Blackbox
13251	26.166969	50.503677	109.68	ESP8266Blackbox
13252	26.166969	50.503677	109.68	ESP8266Blackbox
13253	26.166969	50.503677	109.68	ESP8266Blackbox
13254	26.166969	50.503677	109.68	ESP8266Blackbox
13255	26.166969	50.503677	109.68	ESP8266Blackbox
13256	26.166969	50.503677	109.68	ESP8266Blackbox
13257	26.166969	50.503677	109.68	ESP8266Blackbox
13258	26.140762	50.510647	115.56	ESP8266Blackbox
13259	26.140762	50.510647	115.56	ESP8266Blackbox
13260	26.139896	50.510662	114.6	ESP8266Blackbox
13261	26.139896	50.510662	114.6	ESP8266Blackbox
13262	26.139896	50.510662	114.6	ESP8266Blackbox
13263	26.139896	50.510662	114.6	ESP8266Blackbox
13264	26.139896	50.510662	114.6	ESP8266Blackbox
13265	26.139896	50.510662	114.6	ESP8266Blackbox
13266	26.139896	50.510662	114.6	ESP8266Blackbox
13267	26.139896	50.510662	114.6	ESP8266Blackbox
13268	26.139896	50.510662	114.6	ESP8266Blackbox
13269	26.139896	50.510662	114.6	ESP8266Blackbox
13270	26.135254	50.509907	116.14	ESP8266Blackbox
13271	26.135254	50.509907	116.14	ESP8266Blackbox
13272				

FIGURE 21. TRIP RECORDS ON BAHRAIN HIGHWAY, CAPTURED IN TRACKING.CSV

On the other hand, during my trip to simulate an accident, I have hit the device with my hands to get a strong vibration and see if the moment will be captured or not. you can see clearly from figure 22 that Node-Red recorded the "hit" moment at the speed of 115 and 114, with the time stamp for that moment.

347	115.55	16036	26.140762	50.510647	ESP8266Blackbox	1641297617642
348	114.6	16036	26.139896	50.510662	ESP8266Blackbox	1641297618660

FIGURE 21. ACCIDENT RECORDS ON BAHRAIN HIGHWAY, CAPTURED IN ACCIDENTS.CSV

You can also see the location captured at the time of the accident. The accuracy of this information can provide good help to police departments to assess the accident and how critical it is. This can even be enhanced more by adding more sensors or applying Artificial Neural Networks (ANN) to provide on-time accurate, AI-based decisions. This is a very good result but can be more enhanced by

adding more sensors to the unit, which we will talk about next.

V. CHALLENGES

A. USER PRIVACY

User privacy is one of the important challenges in this project. We are collecting user locations, which means we knows where the drivers are at any time. This is a sensitive issue and need to be studied very well to apply strict measures to control data availability. Moreover, we need to have a strong authorization system to allow limited staff to access these data. We need a tight control and regulation in term of distributing such devices in the market, to avoid any legal related issues.

B. SECURITY

In any application, security is always the first concern to arise. In our prototype, mostly need the security for data transmission since it contains sensitive data. This will ensure that there is no data leak due to an attack therefore we reduce the risk of data abuse.

C. COST OF IMPLEMENTATION

It cost almost 14 BHD for a single device component. This is fair for 2 or 10 devices, but for 700,000 vehicles, we will need around 9.8 million Dinars. This is a quite large budget to reserve, although this can be paid by the drivers themselves at the time of vehicle registration. Moreover, there are also the cost of infrastructure, like fast internet connectivity with 4G or 5G, and a strong database and application servers to handle the massive data traffic caused by these devices. consequently, this will require a thorough study to make sure the infrastructure is reliable and good enough to handle the load all of the time.

D. STORAGE

The data generated from this device is massive. So we will need to use Big Data (BD) concepts and analysis techniques to handle the transmission and manage the traffic from one side, and to store and retrieve the data in the optimum manner. Back to point C, we will need a large, optimized database to store and retrieve all the data faster.

E. INFRASTRUCTURE

Since these devices are used to report accidents, gather information and they will transfer a quite massive data amounts, we will need a strong network infrastructure. We will need fast internet connectivity to transfer sensors data in Realtime. This is also critical for severe accidents, since we will need to report it to the hospitals to send the nearest ambulance to save lives.

Besides having a good communication, we will need to apply strong security mechanisms on data transfer, data store, data gathering to prevent any possible attacks or data leakage.

VI. POSSIBLE ENHANCEMENTS AND POSSIBLE APPLIATIONS

The Blackbox is a starting prototype. It can have more accuracy and provide more information by adding more sensors. For example, we can add accelerometer to support the vibration sensors in accident decisions, where vehicle flipping is involved. This can enhance the decision of the criticality of the accident. We can add mics, with TinyML on the MCU to allow hot-word activation in case the user needs help if he is stuck or got disabled, or even in case of natural disasters like floods, especially when the driver is in isolated area. We can add cameras, with computer vision technology to identify more the circumstances of the accident using Deep Learning Methods, and to help identifying accident place when we lost the signals from GPS. The current application is lacking AI decisions. We can enhance this prototype by feeding a database with sensors data for further analysis instead of just saving them in excels and wait for staff do it. This could enhance the decisions and eliminate human errors as well.

POSSIBLE APPLICATIONS

A. Insurance Risk Analysis

As for the applications, we can not only use this for helping traffic accidents, but we can use it for insurance companies as well. The availability of such data can help an insurance company to better calculate the risks involved with insuring a client. Current insurance practices involve judging the client by his address, age, vehicle type, make model. we can use the information provided between Blackbox to analyze driving behavior, if he is abusing the car, or driving gently. Such information can give a better pricing scheme, where both insurance companies and the drivers can benefit from. It can provide "Live" risk analysis.

B. Fleet Management

The same device can be used to manage fleet of vehicles, say in rental car company. It can provide information about the vehicle location, speed, vibration which can determine if there is an abuse in the vehicle usage. Such information can also reduce costs and prices for those companies and their customers.

V. APPENDIX

A. AZURE SERVER SETUP

Azure is a service provided by a Microsoft, for cloud computing. It provides a very large set of services, such as Virtual Machines (VM), AI computational instances, hosting servers, database management and much more. Our interest for this project is to create a server to install MQTT broker within, to allows us to push the data into the cloud instead of working locally. Firsts I signed up in Azure in https://portal.azure.com/, then I went to VM section. There, we click on " + Create" in the top left, then select VM. The following screen will show

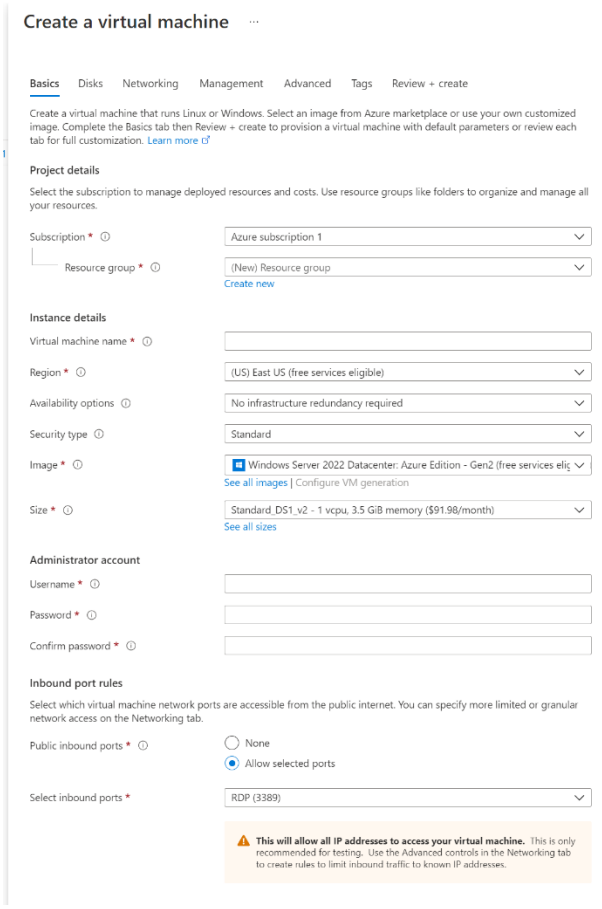

FIGURE A-1. CREATE VM WINDOW IN AZURE

You can see in figure (A-1) that you must fill the required information. Subscription is your subscription on Azure (Account), you have to name it, in my case I named the machine *MqttBlackbox* , assign the machine to data center (Region), select type of the machine as windows server 2022, and select the desired size which is the processing power and capacity.

Next will be to setup administrator account, which will allow you to access the VM through Remote Desktop (RD) app, provided by default on windows. Finally, we will hit on Review + create to review the summery of our selection and create the VM. This should take few minutes, after that the machine is ready to be used:

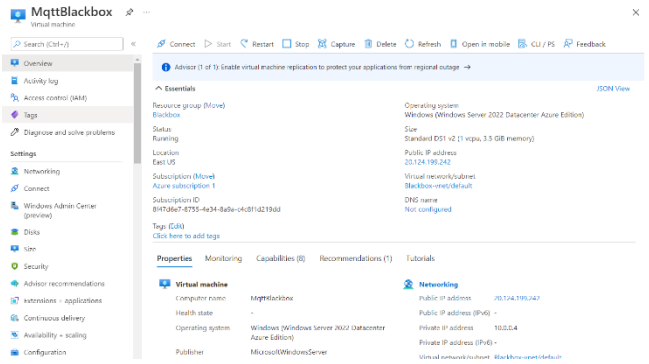

FIGURE A-2. VM DETAILS AFTER ITS DONE

On networking tab, we will need to setup firewall rules to allow the MQTT to communicate to the outside. We will first create rule, and allow port 1883 to be accessed from the outside. This way any device with MQTT username and password will be able to access the server and publish topics.

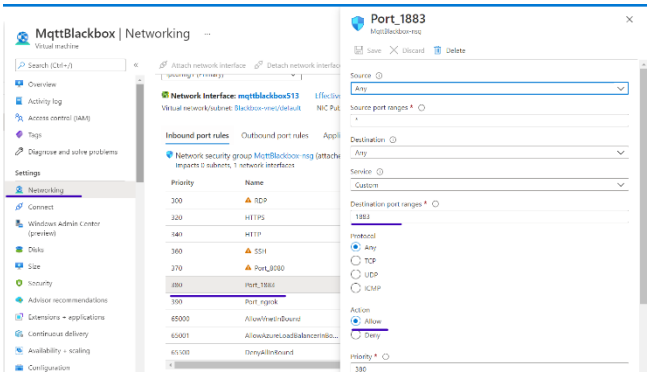

FIGURE A-3. CREATE INBOUND RULE IN NETWORKING

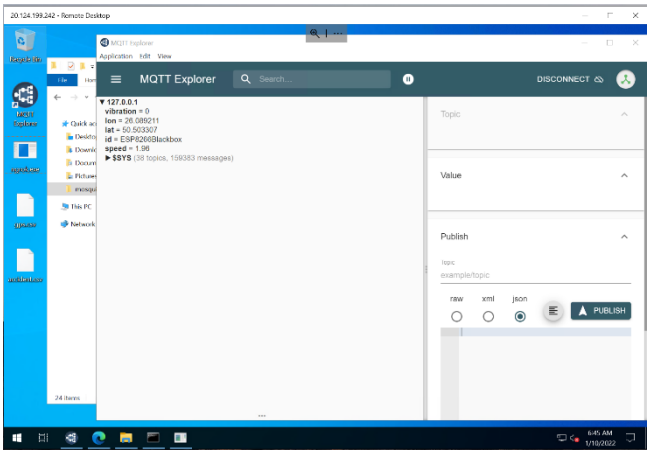

FIGURE A-4. FINAL VM ACCESSED FROM RD

B. CODE REPOSITORY

The code to conduct this experiment is uploaded to GitHub repositories, can be found with more detailed notes on the code at this link:
https://github.com/alsahlawi/blackbox/blob/main/blackbox.ino

ACKNOLEDGEMENTS

Special thanks for Prof. Mohab Mangood, from university of Bahrain for encouraging me to conduct such experiment, and to get more involved in Internet of things devices. the thanks are also for FABLAB Bahrain team for arranging practical course under the title of "Introduction in the IoT" where it give me practical knowledge on how to handle Arduino boards.

VI. CONCLUSION

Such IoT devices can enhance the quality of our lives. It can provide a great help to traffic departments to better assess accidents, with low costs. It also can help people in need, especially in large scale countries where there are deserts and isolated areas. In this report, we have used a cheap component to construct a prototype of the Blackbox. We started by introducing the concept and elaborating more in the problem. Afterward we went through the motivation to conduct this experiment. Passing by the background, we showed hardware and software components used in this experiment, and their features. The next step was to experiment, step by step under Lab Work section. The steps start with preparing the components and connect them, then prepare the code for uploading into the microcontroller. Then we prepared the server and installed MQTT broker, and installed Node-Red with the necessary flows to control the dashboards and data analysis. We then showed the result of the experiment, showing the data gathered from the sensors stored in files by the Node-Red application. Finally, I have expressed the challenges for applying such solution, and as a bonus, I put more idea applicable to such IoT device which can carried out in future research.

REFERENCES

[1] K. Lu, J. Karlsson, A. S. Dahlman, B. A. Sjoqvist, and S. Candefjord, “Detecting driver sleepiness using consumer wearable devices in manual and partial automated real-road driving,” *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–10, 2021.

[2] A. Kassem, R. Jabr, G. Salamouni, and Z. K. Maalouf, “Vehicle black box system,” 2008 2nd Annual *IEEE Systems Conference*, 2008.

[3] “ESP8266,” ESP8266v Espressif Systems. [Online]: https://www.espressif.com/en/products/socs/esp8266. [Accessed: 03-Jan-2022].

[4] “SW-420 Vibration Sensor Module,” Components101. [Online]. Available: https://components101.com/sensors/sw-420-vibration-sensor-module. [Accessed: 03-Jan-2022].

[5] “GT-U7 GPS modules.” [Online]. Available: https://images-na.ssl-images-amazon.com/images/I/91tuvtrO2jL.pdf. [Accessed: 03-Jan-2022].

[6] “Node-Red,” Node-Red Org. [Online]. Available: https://nodered.org/. [Accessed: 03-Jan-2022].

[7] “TinyGPS++: arduiniana,”.[Online]. Available: http://arduiniana.org/libraries/tinygpsplus/. [Accessed: 10-Jan-2022].