**ACCIDENT PREVENTION AND MITIGATION SYSTEM FOR TWO WHEELERS**

**ABSTRACT**

Numerous road accidents take place every year mainly due to potholes. Nearly 4000 people have lost their lives in 2019 due to such incidents in India alone. Roads being an integral part of our daily lives has made this issue a pressing concern that needs to be addressed.

The system that we propose to build consists of two parts namely; Bike-Fall Detection System and Pothole Detection System. The bike fall detection system comprises of the usage of a MPU6050 Accelerometer and Gyroscope Module to detect the orientation of the bike, a GSM900A Module to establish cellular connection to make calls, these peripherals are connected to Arduino UNO R3. The microcontroller constantly monitors the orientation of the bike and compares the real time value with the predefined safe limits to detect a fall, this is done over a 12-15 second time period to confirm whether the two-wheeler is actually in a toppled position by incrementing a flag variable every 3 second. The system is designed in a way to enable the user/driver to turn off the transmission in the case of a false alarm via a button. However, when the user fails to do so, it implies that the driver is actually in danger and the bike is in a toppled position. The GSM900A Module calls the number that is given by the user. (Usually a trusted contact/family member) along with the SOS number (Police, Ambulance etc.)

In the second subsystem we have developed a Pothole Detection System using Machine Learning and Deep Learning techniques. Using photos of various potholes of different shapes and sizes as dataset, the training of the detection model was developed. Using Android Studio, we made an application that detects potholes and warns the rider with a voice notification that says “Pothole Detected”. The code was done in Python using TensorFlow package and a pre-trained SSD Mobile net v2 as 300x300. This model is a single-stage object detection model that goes straight from image pixels to bounding box coordinates and class probabilities. The model architecture is based on inverted residual structure where the input and output of the residual block are thin bottleneck layers as opposed to traditional residual models.

With the help of the proposed model, monitoring and persevering road conditions becomes less strenuous, thereby preventing numerous road mishaps and lowering maintenance charges of vehicles.

**ACKNOWLEDGEMENT**

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. K Chitra,** Associate Professor, School of Electronics Engineering, for her consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to, Dr. Sivasubramanian. A Dean of the School of Electronics Engineering, Vellore Institute of Technology, Chennai, for extending the facilities of the School towards our project and for her unstinting support.

We express our thanks to our Head of the Department Dr. Vetrivelan. P for his support throughout the course of this project. We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| **Sharath Akaash P** | **Amshuman G** | **RK Arjun** | **J Kiron** |
|  |  |  |  |
| **B Visweshwaran** | **R Kathir** | **E Gautamgovan** | **S Prasanna Kumar** |

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | | **SUBTOPIC** | **PAGE NO.** |
|  |  | ABSTRACT | III |
|  |  | ACKNOWLEDGEMENT | IV |
|  |  | LIST OF FIGURES | VII |
|  |  | LIST OF TABLES | IX |
| 1 |  | INTRODUCTION | 1 |
|  | 1.1 | PURPOSE | 1 |
|  | 1.2 | SCOPE | 1 |
| 2 |  | LITERATURE REVIEW | 2 |
| 3 |  | PROPOSED WORK | 6 |
|  | 3.1 | POTHOLE DETECTION SYSTEM | 6 |
|  | 3.2 | BIKE FALL DETECTION SYSTEM | 7 |
|  | 3.3 | PROPOSED WORK AS A SOLUTION THAT HAS REAL TIME SOCIETAL IMPACT | 8 |
| 4 |  | IMPLEMENTATION | 10 |
|  | 4.1 | HARDWARE IMPLEMENTATION | 10 |
|  | 4.2 | SOFTWARE IMPLEMENTATION | 15 |
| 5 |  | STANDARDS USED | 22 |
| 6 |  | RESULTS AND DISCUSSION | 25 |
|  | 6.1 | BIKE FALL DETECTION | 25 |
|  | 6.2 | POTHOLE DETECTION | 28 |
| 7 |  | OUTCOMES ACHIEVED (SOCIETAL PROBLEM SOLVED) | 32 |
|  | 7.1 | SURVEY FINDINGS | 32 |
|  | 7.2 | SURVEY INFERENCE | 34 |
| 8 |  | COST ANALYSIS | 36 |
| 9 |  | TESTING AND QUALITY ASSURANCE | 37 |
|  | 9.1 | TESTING OF THE TRAINED MODEL | 37 |
|  | 9.2 | CALIBRATION OF MPU6050 | 38 |
| 10 |  | CONCLUSION AND FUTURE WORK | 41 |
|  | 10.1 | CONCLUSION | 41 |
|  | 10.2 | FUTURE WORKS | 41 |
| 11 |  | REFERENCES | 43 |
| 12 |  | APPENDIX | 45 |
|  |  | BIODATA | 68 |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE**  **NO.** | **TITLE** | **PAGE NO.** |
| 3.1 | Flow diagram of Pothole Detection System | 6 |
| 3.2 | Flow diagram of the process | 7 |
| 3.3 | Flow diagram of Bike Fall Detection System | 8 |
| 4.1 | Block Diagram of Bike Fall Detection System | 10 |
| 4.2 | Circuit using Tilt sensor | 11 |
| 4.3 | Circuit using Potentiometer | 11 |
| 4.4 | a. Circuit implementation using Tinkercad; b. Real-time hardware implementation | 12 |
| 4.5 | Buzzer | 13 |
| 4.6 | GSM SIM900A Module | 13 |
| 4.7 | Pin diagram of GSM SIM900A | 14 |
| 4.8 | MPU6050 gyroscope sensor | 15 |
| 4.9 | Pin diagram of MPU6050 | 15 |
| 4.10 | Neural Networks | 18 |
| 4.11 | Basic structure for app building | 21 |
| 6.1 | X and Y Axes orientation condition | 25 |
| 6.2 | Flag condition | 25 |
| 6.3 | Tilting the tilt sensor above permissible limits | 26 |
| 6.4 | Flag value incrementations | 27 |
| 6.5 | Case where push button is pressed before flag value goes over 5 | 28 |
| 6.6 | a. Case where push button is not pressed before flag value goes over 5; b. Call being made to emergency contact via SIM900A module | 28 |
| 6.7 | Input images given for image processing | 29 |
| 6.8 | Processed Images | 29 |
| 6.9 | Intermediate Result | 30 |
| 6.10 | Pothole Detection App installed on a smartphone | 30 |
| 6.11 | Address and Co-ordinates of the potholes | 31 |
| 6.12 | Mapped coordinates of Potholes on a map | 31 |
| 7.1 | Pie chart showing Road Transportation used | 32 |
| 7.2 | Graph showing Safety of Indian roads | 32 |
| 7.3 | Pie chart Road accident frequency | 33 |
| 7.4 | Pie chart showing Helmet wearers | 33 |
| 7.5 | Graph showing causes of road accidents | 33 |
| 7.6 | Pie chart showing people who value additional safety measures | 34 |
| 7.7 | Graph showing Road safety measures that can be implemented | 34 |
| 9.1 | Pothole detection app scanning a. a crosswalk; b. a collection of potholes; c. lane markings with zebra crossing; d. median strip | 38 |
| 9.2 | Storing inclination in degrees from accelerometer readings | 39 |
| 9.3 | Outputs for various angle values | 40 |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **TABLE**  **NO.** | **TITLE** | **PAGE NO.** |
| 5.1 | Standards for Arduino Uno | 22 |
| 5.2 | Standards for SIM900A | 22 |
| 8.1 | Cost Analysis | 36 |

**CHAPTER 1**

**INTRODUCTION**

* 1. **Purpose**

Reckless, fast driving or other driver blunders are primary causes of accidents around the world; however, poor road conditions also play a crucial role. Road condition assessment involves identifying and analyzing distinct types of road surface distress, like potholes, cracks or texture changes as being maintenance-relevant features. Pothole is a particular case of road distress. Potholes have become a pressing issue in developing countries due its growing population thereby the increase in vehicles. There is also a lack of a proper road monitoring-maintenance system which has led to pathetic road conditions. Due to the bad road condition the passengers have to face unnecessary body fatigue. Indian roads are full of potholes and numerous accidents occur due to improper roads. Many accidents occur due to inconsistencies in our roadways. Upon a survey conducted using Google Forms, the surveyors said that there was an immediate need to address the issue of potholes. Many people are unable to control their vehicle when they encounter a pothole suddenly, resulting in an accident. Also, the participants of the survey wanted an automatic bike fall detection system that calls an emergency number of their choice when their vehicle has been toppled. This creates a necessity for a Pothole Detection System and a Bike Fall Detection system, both integrated as a single unit that can be placed on the vehicle

* 1. **Scope**

The project contains two sub-systems, Pothole Detection System and Bike-fall Detection System. The main goal of the project has been practically implemented. It can be implemented in various places. In India, the majority of people travel using two-wheelers. Hence, the prototype can be a part of the vehicle when the customer buys it from the retailer. In most universities, students and even some faculties travel using bicycle. The universities can spend on smart bicycles which have the prototype fixed on the front of the bicycles. The prototype is successfully built but there is always room for improvements which can be in the hardware as well as software part. The following ideas can be considered for the future expansion of the model.

* Pothole Detection System:
* Better training can be done to increase the accuracy of the model.
* Maps can be included in the android application for showing the potholes graphically.
* There can be voice notifications when a rider is going to encounter a pothole.
* A more complex prototype can be developed to control the speed of the vehicle when there is pothole ahead.
* A different algorithm can be used to find the distance of the pothole from the vehicle.
* Bike-fall Detection System:
* A more complex prototype can be built such that, when the bike tilts by an angle that can topple it, a closed control loop system that prevents the bike from toppling can be implemented.

**CHAPTER 2**

**LITERATURE REVIEW**

This paper describes accelerometer data-based pothole detection algorithms for deployment on devices with limited hardware/software resources and their evaluation on real world data acquired using different Android OS based smart-phones. Algorithm tests resulted in optimal setup for each selected algorithm and the performance analysis in context of different road irregularity classes show true positive rates as high as 90%. The future work includes experiments with combinations of algorithms and development of self-calibration functionality. [1]

This study proposes a real-time pothole detection method based on the mobile sensing techniques. This method uses Euler angle computation to normalize the accelerometer data obtained from mobile device with free angle establishment. Moreover, a pothole detection approach is proposed to be combined with Z-THRESH and G-ZERO approaches for reducing the false-positives of pothole detection. Furthermore, the spatial interpolation method is adopted to precisely obtain the location of pothole. In experiments, the results show that the proposed approach can precisely detect potholes without false-positives and the accuracy of the proposed approach is 100%. Therefore, the proposed Realtime pothole detection approach can be used to improve the safety of traffic for ITS. [2]

Accidents owing to potholes has become an alarming problem in today’s life. The first step to solve this problem requires, designing a device embedded on the vehicle which can continuously scan the road surface for identifying potholes, alerting the driver in time and enable the driver to avoid the pothole. The second step is to introduce a technique to enable the device to locate the position of the pothole via GPS (Global Positioning System). The GPS data can be uploaded via a GPRS (General Packet Radio Service) module or Bluetooth module onto a data base which is stored locally. This database can then be transferred to the cloud using Wi-Fi or 4G technology by connecting the system. The third aspect is to link the database to a network system incorporating mapping software such as Google Maps or Open-Street Map. The data in the system can be made available to the public as well as municipalities and road maintenance agencies. Awareness of the location of potholes will help drivers to avoid those roads and being more careful while driving on the same roads. This paper focuses on the pothole detection task based on image processing algorithms and the data captured from ultrasonic sensor placed on the vehicle. The later steps were implemented through Bluetooth interface available in smartphones. [3]

Potholes can generate damage such as flat tire and wheel damage, impact and damage of lower vehicle, vehicle collision, and major accidents. Thus, accurately, and quickly detecting potholes is one of the important tasks for determining proper strategies in ITS (Intelligent Transportation System) service and road management system. Several efforts have been made for developing a technology which can automatically detect and recognize potholes. In this study, a pothole detection method based on two dimensional (2D) images is proposed for improving the existing method and designing a pothole detection system to be applied to ITS service and road management system. For experiments, 2D road images that were collected by a survey vehicle in Korea were used and the performance of the proposed method was compared with that of the existing method for several conditions such as road, recording, and brightness. The results are promising, and the information extracted using the proposed method can be used, not only in determining the preliminary maintenance for a road management system and in taking immediate action for their repair and maintenance, but also in providing alert information of potholes to drivers as one of ITS services. [4]

Asphalt-surfaced pavements are subjected to a broad spectrum of traffic levels, from two-lane rural routes to multi-lane interstate highways. They age and deteriorate; thus, they require corrective measures to restore safety and ride-ability. The most common forms of distress on asphalt-surfaced pavements are potholes – small, bowl-shaped depressions in the pavement surface. Pothole repair is necessary in those situations where potholes compromise safety and pavement rideability. Pothole detection and estimation is one of the important tasks for the proper planning of reparation and rehabilitation of the asphalt-surfaced pavement. Road maintaining companies need many technicians for manual collection of data, and many working hours for rough estimation of damage on the road. There are many factors which influence decisions for pothole patching, such as the level of traffic, the time until scheduled rehabilitation or overlay, the availability of personnel, equipment, and materials, and the tolerance of the traveling public. The cost-effectiveness of the overall patching operation is affected by material, labor, and equipment costs. The key of decision making for future reconstruction is estimation of damage from collected information. In this paper, a new unsupervised method is proposed, which is based on image analysis and spectral clustering. It is vision-based method, which does not require additional filtering and training phase. Data is collected by using in-expensive and omnipresent equipment mounted on passenger vehicles and off the shelf digital cameras for video acquisition. [5]

The goal of this research paper is to develop a pothole detector using common devices that are used by many drivers over a wide area. Moreover, the devices should provide high detection accuracy at low cost. In this paper, a novel pothole-detection system is proposed using a commercial black-box camera. The proposed system is mounted on the front windshield of a vehicle and can detect a pothole in real-time. A pothole-detection algorithm is installed on an embedded board in the black-box camera. This algorithm collects information regarding the size of potholes and their location, and this information is stored in the black box and then transmitted to a pothole-management server. The proposed pothole-detection algorithm is uniquely designed in consideration of the embedded boards in black-box cameras. [6]

In this paper they have determined a two-way identification step to determine whether the accident has occurred or not. First is through accelerometer which will identify any sudden tilt of the vehicle in case of any accident. Then the heartbeat sensor will sense the heartbeat rate of the user and determine the seriousness of the accidents or fall, based on the changes in the heartbeat, a message is sent along with the location map to the control room and emergency contacts. The android application will send a text message to the nearest medical help center along with the location with the help of GPS which helps in saving crucial time. The heartbeat sensor very effectively eliminates the false alarms like in the case of a standing fall of the two-wheeler which the accelerometer would assume as an accident. [7]

In this paper they have designed a model to record informational data like plotting the vehicle and receiving alert message about the accident by using GPS as well as GSM using Raspberry Pi. The entire set up is called Black box. The Black box contains a Tilt sensor that sense the tilt angle or movement of the bike in case of accident and activate the framework, send the message to specific server. The webcam records the video at that instant. The recorded video in Black box is especially helpful in identifying the cause of accident. The android application will send an alert message to the nearest medical help center along with the location with the help of GPS which helps in saving crucial time. This data is also stored in the black box. This stored data is beneficial for the police investigation, hence finding the real culprit. [8]

The main objective of this paper is to propose a system which can effectively help in preventing any kind of mishaps and if such conditions occur then how it detects and informs the concerned authorities and people, so that the situation can be taken care of immediately. This system detects accidents by vibration sensors, accelerometers. For detection, GPS and GSM modules are used, which locates the site of the accident and correspondingly informs the person’s near ones and nearby hospitals through a text message. Only sending a text message to nearby hospitals won’t be enough because it cannot avoid secondary accidents and hence, this system caters this requirement too. The system provides an idiosyncratic prevention and detection system that dispenses the ultimate panacea for drivers which ensures safety and prevents loss of life by taking appropriate measures in right time. It also checks whether the driver is drowsy or in an unstable state which can lead to pedal mix-up and in some cases unintended acceleration or turning of the steering wheel to the wrong direction which can lead to crashing of the vehicle with other vehicles or concrete road barrier. This system also provides a mechanism by which it identifies whether the person that will be riding the bike has a valid driving license or a driving license at all by already embedded RFID on driving license. The RFID reader on the bike will have at most 10 registered users, so that the family members or his/her friends can also ride the bike. Hence, this mechanism also handles theft related issues. [9]

In Bangladesh, an accidental incident cannot be traced unless someone has noticed it. Sometime the situation gets worse when the spot cannot be traced down. Studies shows that swift rescue and medication can lower down the fatality count in many cases. Therefore, in this paper, an automatic, efficient, low-priced, and advance system has been developed that can perform this job. This technology will prevent the accident and notify the emergency services and the owner of the vehicles. The system has three parts. First one is accident prevention part, second one is accident detection and rescuing part and the last one is black-box part. [10]

Although there is a continuous improvement in road and vehicle safety, as well as improvements in IoT, the road traffic accidents have been increasing over the last decades. Therefore, it is necessary to find an effective way to reduce the frequency and severity of traffic accidents. This paper presents an intelligent traffic accident detection system in which vehicles exchange their microscopic vehicle variables with each other. The proposed system uses simulated data collected from vehicular ad-hoc networks (VANETs) based on the speeds and coordinates of the vehicles and then, it sends traffic alerts to the drivers. Furthermore, it shows how machine learning methods can be exploited to detect accidents on freeways in ITS. It is shown that if position and velocity values of every vehicle are given, vehicles’ behavior could be analyzed and accidents can be detected easily. Supervised machine learning algorithms such as Artificial Neural Networks (ANN), Support Vector Machine (SVM), and Random Forests (RF) are implemented on traffic data to develop a model to distinguish accident cases from normal cases. [11]

Major accidents on highways, freeways and local roads can lead to huge social and economic impacts. Minor accidents may be resolved by the passengers themselves and do not require escorting to hospitals whereas major accidents where airbags are deployed require immediate attention of authorities. Automatic Smart Accident Detection (ASAD) is an auto-detection unit system that immediately notifies an Emergency Contact through a text message when an instant change in acceleration, rotation and an impact force in an end of the vehicle is detected by the system, detailing the location and time of the accident. The idea is that as soon as an accident is detected by the system, the authorities should immediately be notified to prevent further car congestion as well as allow the passengers to be escorted to the hospital in a timely fashion. The system involves the use of fuzzy logic as a decision support built into the smartphone application that analyzes the incoming data from the sensors and makes a decision based on a set of rules. [12]

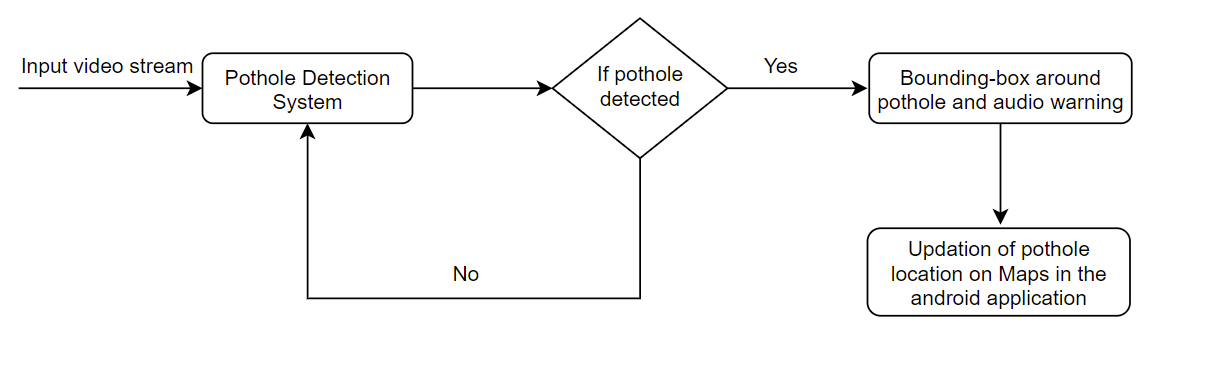
Several pothole detection systems use accelerometers to detect potholes as seen in the papers from Artis [1] and Wang [2]. While accelerometer is a good way to detect potholes, our detection system detects potholes with the help of Machine Learning algorithm, which is more efficient and accurate compared to an accelerometer. The system is also simple, compact and easy to setup compared to accelerometers. Papers by Nanda [3] and Jo [6] focuses on collecting and storing the location of potholes rather than instantaneously alerting the user. Our pothole detection focuses on instantaneous audio warning that alerts user of an upcoming pothole as this ensures a safe and smooth ride. When we look at accident prevention papers, GPS location system is used in almost all the proposed systems. Our project also sends the location of the accident along with the SMS using the GSM module. The paper by Shrinidhi [8] uses a tilt sensor similar to ours, but the problem with this paper as well as several other papers we went through is the safety check mechanism that we use. Our system plays an alarm for 8 seconds and waits for the driver’s response. If the driver is conscious and hit the safe button, the system recognizes that the driver is not in need of any immediate help and that the driver is conscious. This prevents any false alarms in case of minor accidents. The previously mentioned flaws in other papers were the absence of this safety mechanism which might lead to false alarms and unnecessary waste of time and labor.

**CHAPTER 3**

**PROPOSED WORK**

**3.1 Pothole Detection System**

The flow of the system is as follows, image is fed to CNN-based Mobile Net-SSD model to detect the pothole and plot a bounding box. When an input image or video is given to the system, the object detection algorithm, using Deep Learning detects objects that were trained previously. In our program, the objects that will be detected is pothole. Whenever the program detects an object prescribed, the confidence of the detection is calculated and if, and only if the confidence is greater than 0.5 will the detection system draw a bounding box around the detected object. The bounding box is drawn by using an inbuilt function from the OpenCV library, which is a computer vision library. When an image containing a pothole is given as input, the system bounds the entire pothole It does so by partitioning the bounding box into 4 parts along its height and bounding the top most part. Now, if the probability of pothole detection is above 0.90 then there is no need for license place detection. A basic flow diagram of pothole detection can be seen in Figure 3.1.



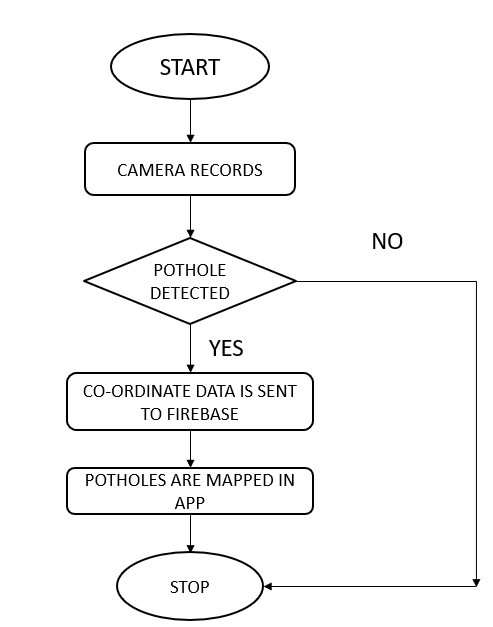
*Figure 3.1: Flow diagram of Pothole Detection System*

The work consists of two parts, developing the object detecting model and developing the API to detect potholes in real time using deep learning.

Model Development: For the pothole detection system program, first we need to collect a proper dataset that encompasses all the possible combinations where a pothole can be present. Such a dataset is carefully selected from the internet. Using various tutorials on how to train an object detection model, we are training the model with our dataset using TensorFlow model and a pre-trained Mobilenet SSD V2 300x300 model. After the training is done, we test the model using some more images of potholes and other exceptions that may be identified as pothole to make sure that the result is satisfactory.

App Development: Designed a simple implementation of pothole detection application with Java showing detection accuracy and bounding box. Used Android studio as the development environment for the Android app. Performed further tests on different android phones to make the app compatible with different android versions.

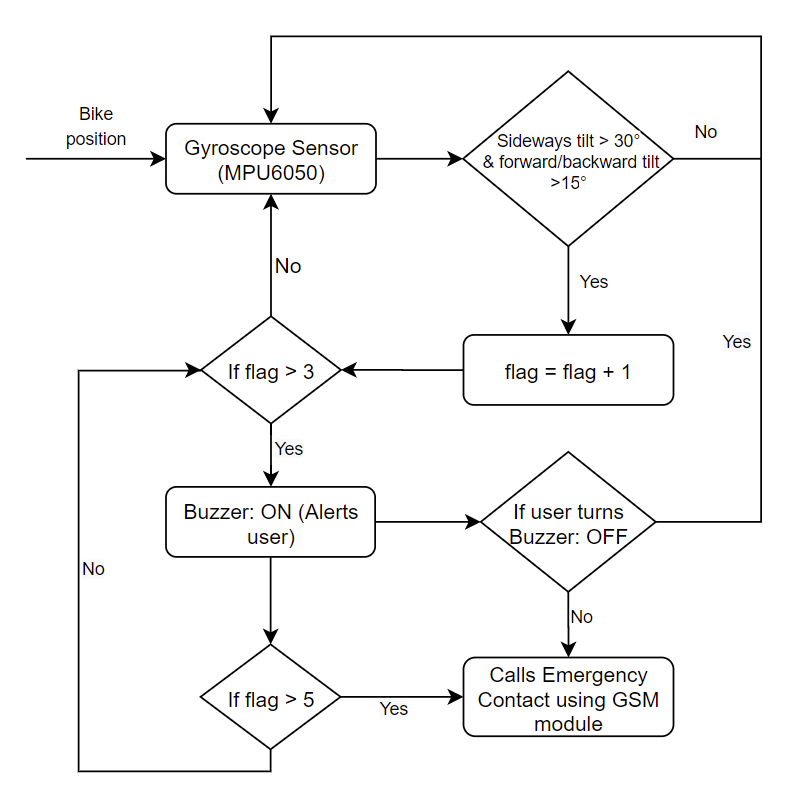
The API model used is a pretrained model on which we can upload our object detecting model. The API uses the mobile phone camera to see the surroundings and then when it detects one, a voice output is given that a pothole is detected. This location is updated in the database later. In Figure 3.2, the flow diagram explains the process mentioned above.

****

*Figure 3.2: Flow diagram of the process*

**3.2 Bike Fall Detection System**

The flow of the proposed system is as follows, the gyroscope sensor’s angle/orientation as the input, the MPU6050 sensor is mounted so that the three zero-degree axes is aligned with regular position of the bike i.e., 90 degrees upright from the ground. The goal is to identify potential toppled position of the bike based on the orientation of the sensor. For this we set a safe limit based on calibration. The safe range of angle is fixed to be as 30 degrees either ways in the sideways direction and 15 degrees in the forward and backward direction. Now we have to understand that some roads are banked and there is elevation in bridges. So, it is important to make sure that we don’t raise any false alarms. So, to avoid instant false alarms. A flag variable is used to check the status of the two-wheeler. The frequency at which the position is checked is once every 4 seconds. As given in Figure 3.3, when the flag value goes past 3 i.e., after 16 seconds the user will be informed first that the microcontroller senses that the bike is in a toppled position. Without any user interruption, this will be treated as an emergency and when the flag value goes past 5 i.e., at the 24th second a call is established to the number mentioned in the code. But however, another variable push is used to enable the user to have control over this transmission. The process mentioned above can be seen in the flow diagram of Bike Fall Detection System in Figure 3.3.



*Figure 3.3 Flow diagram of Bike Fall Detection System*

So, if a false alarm was raised, the user can stop the transmission with a simple stroke of the button when he hears the sound of the buzzer. This terminates the establishment of call immediately. The coding is done on Arduino IDE, microcontroller used is Arduino UNO R3 which monitors the orientation of the bike constantly.

**3.3 Proposed work as a solution that has real time societal impact**

India is one of the most dangerous countries to commute. With over 47,000 accidents, potholes were one of the major accident-causing factors in 2019. Though the number of accidents due to potholes decreased, its share in the total causes of accidents due to road features had increased. The number of deaths due to potholes increased significantly in that year. Accidents caused by potholes can’t be blamed on drivers but we can help drivers by alerting about potholes. Drivers can’t focus on both traffic and potholes. Therefore, if drivers get help in detecting potholes they can drive better as they focus on traffic. More than a third (37%) of those killed in road accidents in 2019 were two-wheeler riders, noted a Ministry of Road Transport and Highways’ report published in October this year.

India witnessed a rapid growth in per capita incomes in the last decade, which also led to people buying more vehicles – particularly two-wheelers – according to a study by the Institute for Social and Economic Change, Bangalore. India’s per capita income grew by 28% between 2013 and 2017 while two-wheeler registrations increased by 46% (compared to 44% overall new vehicle registrations) over the same period. Another main problem is after accidents happen there are no one near to help the victim and this has cost people’s lives. So, a system which can alert people if there is an accident so that the victim can receive help adequately. The system must not be a separate device as people tend to be lazy and do not use it. The System is integrated with the vehicle itself.

Even though there has been a decrease in the accidents due to potholes, the statistics provided by the Ministry of Road Transport and Highways (India) states that over 9300 deaths, 25000 injured in the last three years (2018-2020) due to potholes. According to a research conducted by the United Nations Economic and Social Commission for Asia and Pacific (UNESCAP), road traffic crashes cost the Indian economy almost 3% of its GDP, amounting to Rs.55,000 crore ($8.2 billion), every year from road accidents. By using these systems one can save lives and improve the standard of living in our country. The above points are stated to demonstrate the importance of this problem. Government authorities sanction large sums of money to fill these potholes, but due ineffective detection techniques it becomes hard to overcome this problem, thus paving way for more accidents and damages.

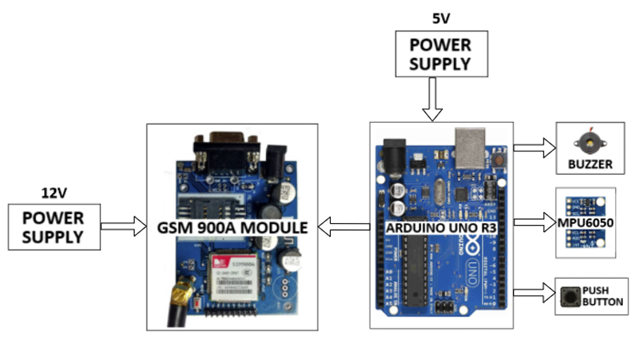
**CHAPTER 4**

**IMPLEMENTATION**

**4.1 Hardware Implementation**

**4.1.1 Block Schematic**

The heart of the circuit is the microcontroller Arduino UNO R3, it is powered by a USB port of a computer or a power bank. The GSM900A is powered by a 12V 2A Adapter. The MPU6050 module acts as the sensor which gains data from the world. It collects the sensor’s orientation information on 3 axes (which is attached to the bike), with the help of the code stored in the Arduino UNO, we are able to constantly monitor the readings to find whether the bike is in a safe position. The Push button is present to stop the establishment of a call to emergency contact in the case of a false alarm. The buzzer is present to alert the user for a period of 8 seconds, before it makes a call via the GSM module. We can see the block diagram of Bike Fall detection System with the components in Figure 4.1.

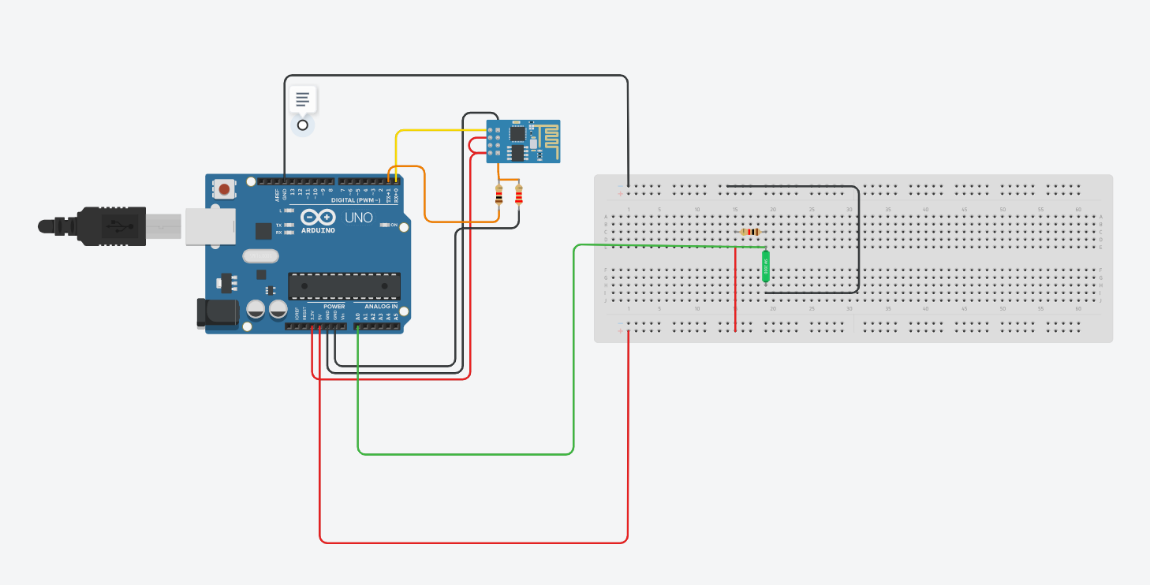


*Figure 4.1: Block Diagram of Bike Fall Detection System*

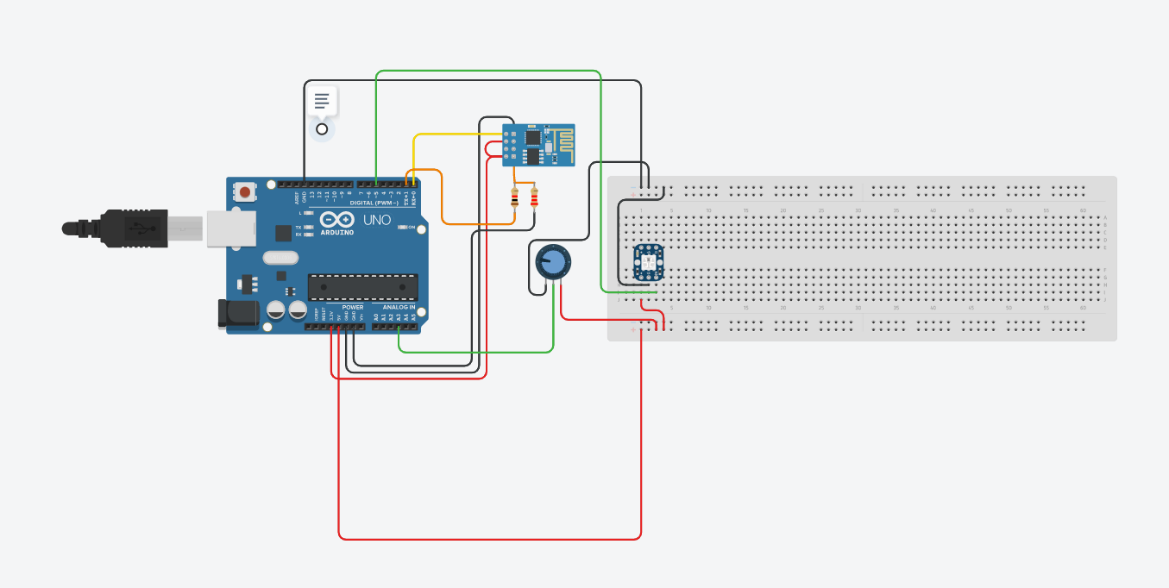
**4.1.2 Design Procedure**

For the design of the hardware prototype, first, many simulations of the circuit were tried out, to make the circuit as compact and efficient as possible.

The circuit was designed using Tinkercad. Although MPU6050 (gyroscope sensor) was not available, there was a 4-pin tilt sensor. We used that in the simulation instead. The output of the tilt sensor was recorded and was sent to ThingSpeak server using ESP8266 Wi-Fi module for simulation purpose. The circuit implementation with tilt sensor can be found in Figure 4.2 while an alternative circuit implementation with Potentiometer can be found in Figure 4.3.



*Figure 4.2 Circuit using Tilt sensor.*



*Figure 4.3 Circuit using Potentiometer.*

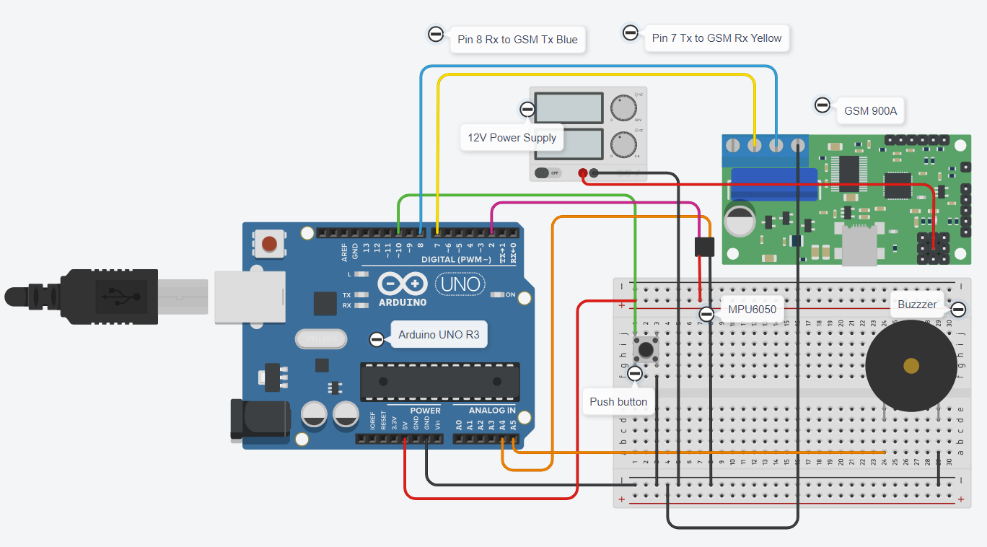
* ThingSpeak Data Visualization

Using the simulation, the prototype was built using the following hardware components:

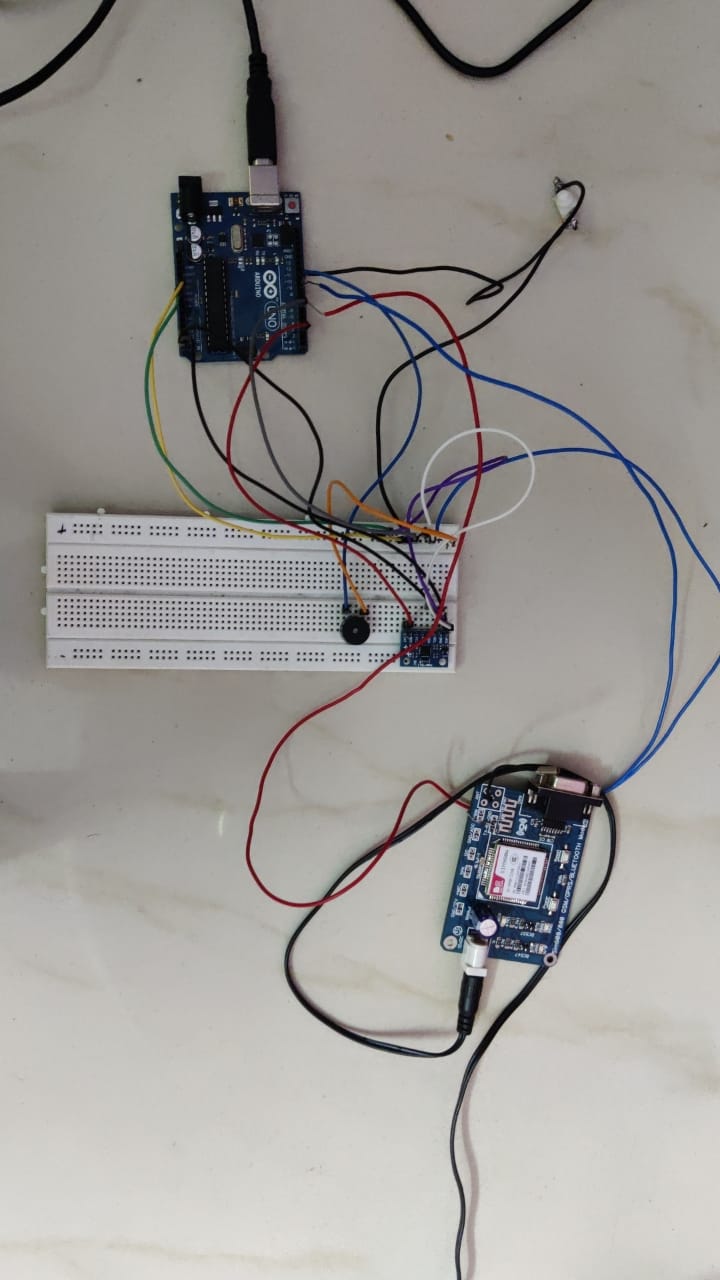
* + - 1. Arduino UNO R3: The prototype and the final product that we are manufacturing must be affordable enough for it to be implemented on the vehicle. Arduino UNO is the most cost-efficient microcontroller that can be used to get results that are at par with other expensive microcontrollers. Also, there was no necessity to use other microcontrollers since the number of pins needed for the project was enough.
      2. MPU6050 (gyroscope sensor): The project requires a gyroscope sensor that produces precise output. Hence, we need a standardized sensor. The MPU6050 is one of the top ten motion sensors that is also accurate and cost-effective. The sensor can also be easily used with Arduino UNO and has a sampling rate of 100 samples per second.
      3. GSM 900A module: The 900A is the most popular variant of GSM module available in the market. It is a dual band variant which is sufficient for our usage. There is also a 800 variant of the GSM shield module which contains a Bluetooth stack and an FM (frequency modulator). But for our purpose, we do not need these features, since we only need to call the emergency number.
      4. Buzzer: The buzzer used is a standard industry buzzer that is available widely.
      5. Push button: The push button is a standard industry manufactured component that is widely available.

**4.1.3 Circuit Diagram**

The Tinkercad circuit implementation of Bike Fall detection System and the real-time implementation of the hardware can be seen in Figure 4.4a and Figure 4.4b respectively.



*Figure 4.4a Circuit implementation using Tinkercad*



*Figure 4.4b Real-time hardware implementation*

**4.1.4 Hardware Specifications**

* Buzzer:

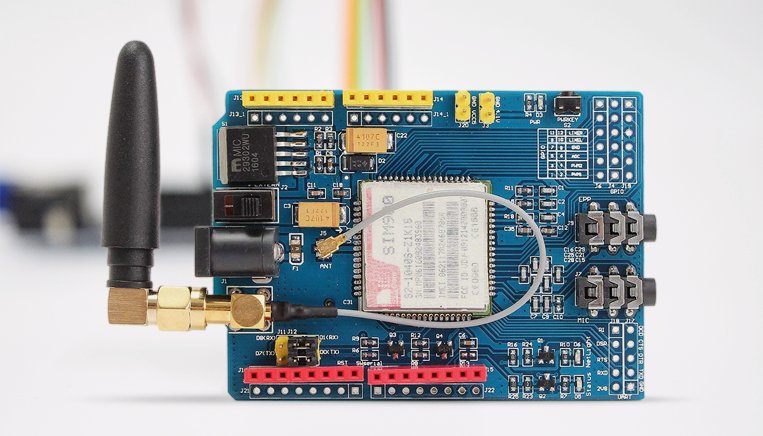
A buzzer or beeper is an audio signaling device which may be mechanical, electromechanical, or piezoelectric. The buzzer consists of an outside case with two pins to attach it to power and ground which is shown in Figure 4.5. Inside is a piezo element, which consists of a central ceramic disc surrounded by a metal (often bronze) vibration disc. When current is applied to the buzzer it causes the ceramic disk to contract or expand. Changing the This then causes the surrounding disc to vibrate. That’s the sound that you hear. By changing the frequency of the buzzer, the speed of the vibration’s changes, which changes the pitch of the resulting sound.



*Figure 4.5: Buzzer*

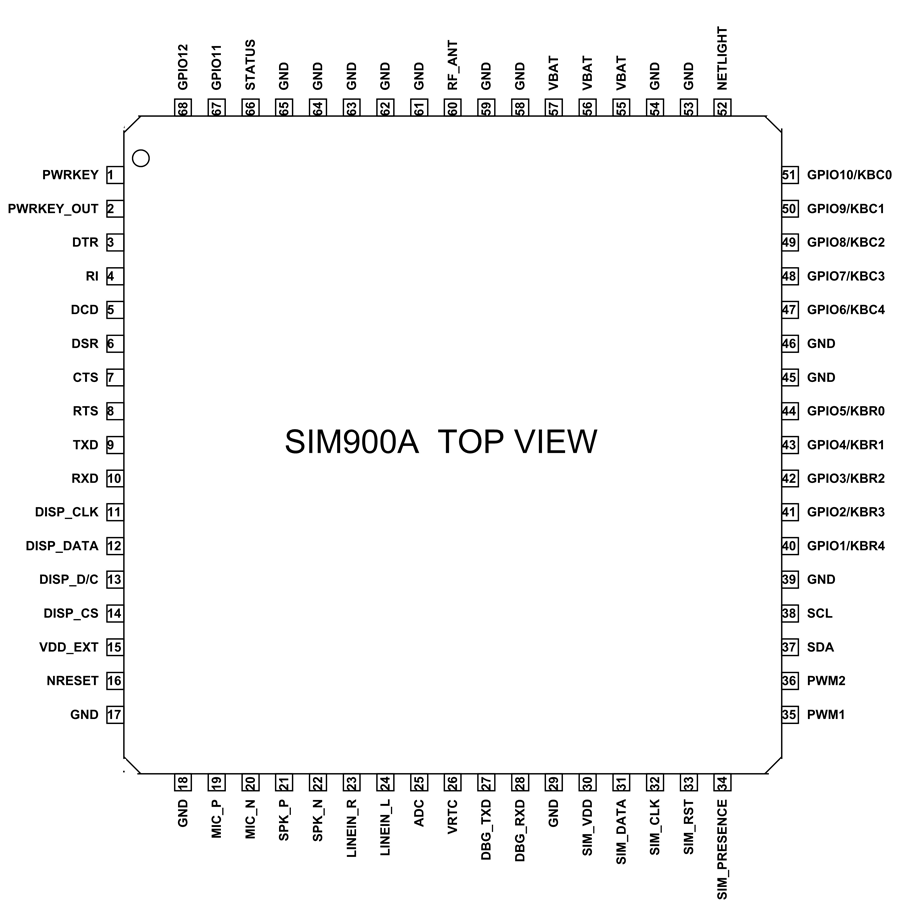
* GSM 900A MODULE:

The **SIM900A**is a readily available **GSM/GPRS module**, used in many mobile phones and PDA which is shown in Figure 4.6. The module can also be used for developing IOT (Internet of Things) and Embedded Applications. SIM900A Modem is built with Dual Band GSM based SIM900A modem from SIMCOM. It works on frequencies 900MHz. SIM900A can search these two bands automatically. The frequency bands can also be set by AT Commands. The baud rate is configurable from 1200-115200 through AT command. SIM900A is an ultra-compact and wireless module. The Modem is coming interface, which allows you connect PC as well as microcontroller with RS232 Chip (MAX232). It is suitable for SMS, Voice as well as DATA transfer application in M2M interface. The onboard Regulated Power supply allows you to connect wide range unregulated power supply. Using this modem, you can make audio calls, SMS, Read SMS, attend the incoming calls and ect. Through simple AT commands. This is a complete GSM module in a SMT type and made with a very powerful single-chip, allowing you to benefit from small dimensions. SIM 900A GSM Modem with serial and TTL outputs.[17]



*Figure 4.6: GSM SIM900A Module*

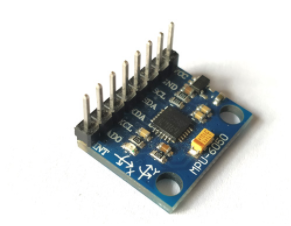
The pin diagram of the GSM 900A module is shown in Figure 4.7.



*Figure 4.7 Pin diagram of GSM SIM900A*

* MPU6050

The MPU6050 is a Micro Electro-Mechanical Systems (MEMS), shown in Figure 4.8 which consists of a 3-axis Accelerometer and 3-axis Gyroscope inside it. This helps us to measure acceleration, velocity, orientation, displacement and many other motions related parameter of a system or object. This module also has a (DMP) Digital Motion Processor inside it which is powerful enough to perform complex calculation and thus free up the work for Microcontroller. The module also has two auxiliary pins which can be used to interface external IIC modules like a magnetometer, however it is optional. Since the IIC address of the module is configurable more than one MPU6050 sensor can be interfaced to a Microcontroller using the AD0 pin. This module also has well documented and revised libraries available hence it’s very easy to use with famous platforms like Arduino.[16]



*Figure 4.8: MPU6050 gyroscope sensor*

#### Pin Diagram: The pin specifications of MPU6050 are shown in Figure 4.9.

#### 

#### *Figure 4.9 Pin diagram of MPU6050*

**4.2 Software Implementation**

This section discusses the steps involved in the object detection model, API and intends to provide a basic idea of implementation processes involved and the system flow.

**4.2.1 Object Detection Model**

The model development comprises of performing object detection on custom pothole images using TensorFlow Object Detection API, Using Google Colab’s free GPU for training and google drive to keep everything synced.[14]

* Gathering Images and Labels

The first step in the process would be to collect stock images as per our use case. Via Kaggle, we obtained a dataset consisting of over 1000 images of potholes. These images are labeled, using tools such as LabelImg. But this doesn’t have to be manually done by us since we are use Mobile net SSD model which automatically labels the images by classifying them.

* Setting up the environment

In a new google colab notebook, mount the gdrive folder containing the uploaded images and labels

* Splitting the images into training & testing:

The train-test split is a technique for evaluating the performance of a machine learning algorithm. It can be used for classification or regression problems and can be used for any supervised learning algorithm. The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

* Importing and Installing Required Packages

Google Colab has most of the packages pre-installed already; Python, TensorFlow, pandas, etc. Therefore, we only have to import them.

* Preprocessing images and Labels

We need to create two csv files for the .xml files in each train\_labels/ and test\_labels/. Other than the CSVs, we will need to create a pbtxt file that will contain the label map for each class. This file will tell the model what each object is by defining a mapping of class names to class ID numbers.

* Downloading TensorFlow model

TensorFlow model contains the object detection API we are interested in. We can download it using the following line:

!git clone –q <https://github.com/tensorflow/models.git>

It can then be tested if the model builder works correctly using:

!python3 object\_detection/builders/model\_builder\_test.py

* Generating TFRecords

TensorFlow accepts the data as TFRecords data. Record. TFRecord is a binary file that runs fast with low memory usage. It contains all the images and labels in one file. In our case, we will have two TFRecords; one for testing and another for training

* Selecting and Downloading a Pre-trained Model

A pre-trained model simply means that it has been trained on another dataset. That model has seen thousands or millions of images and objects. COCO (Common Objects in Context) is a dataset of 330,000 images that contains 1.5 million objects for 80 different classes. Such as, dogs, cats, cars, bananas, etc. Training a model from scratch is extremely time consuming; it may take days or weeks to finish training. A pre-trained model has already seen tons of objects and knows how to classify each one of them. So, we will use it. Because our interest is to interfere on a real time video, we will be choosing a model that has a low ms inference speed with a relatively high mAP on COCO. The model used for this project is ssd\_mobilenet\_v2\_coco. Then it can be downloaded by navigating to models/research/. DEST\_DIR is where the model will be downloaded.

* Configuring the Training Pipeline

TensorFlow Object Detection API model we downloaded comes with many sample config files. For each model, there is a config file that is ‘almost’ ready to be used. Ssd\_mobilenet\_v2\_coco.config is the config file for the pretrained model we are using. Required edits to the config file:

* model {} > ssd {}:
* train\_config {}: change fine\_tune\_checkpoint to the checkpoint file path.
* train\_input\_reader {}: set the path to the train\_labels.record and the label map pbtxt file.
* eval\_input\_reader {}: set the path to the test\_labels.record and the label map pbtxt file.
* Training the model:

The next step is training the model, the following line of codes are required

model\_main.py which runs the training process

pipeline\_config\_path=Path/to/config/file/model.config

model\_dir= Path/to/training/

* Export the trained model:

By default, the model will save a checkpoint every 600 seconds while training up to 5 checkpoints. Then, as new files are created, older files are deleted. By executing export\_inference\_graph.py to convert the model to a frozen model frozen\_inference\_graph.pb that we can use for inference. This frozen model can’t be used to resume training. However, saved\_model.pb gets exported as well which can be used to resume training as it has all the weights.

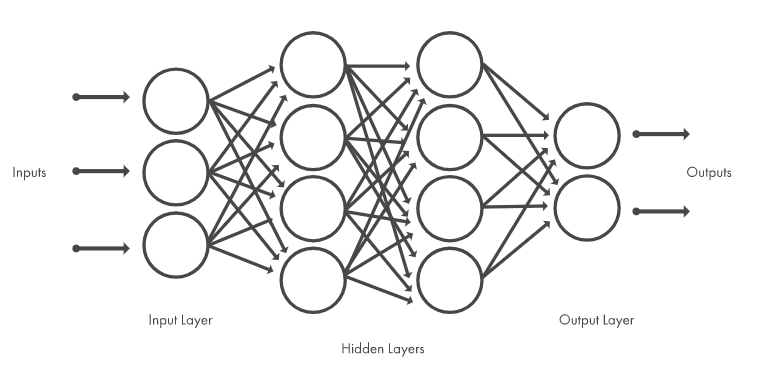
**4.2.2** **Deep Learning**

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labeled data and neural network architectures that contain many layers.

Deep learning achieves recognition accuracy at higher levels than ever before. Recent advances in deep learning have improved to the point where deep learning outperforms humans in some tasks like classifying objects in images. Most deep learning methods use neural network architectures, which is why deep learning models are often referred to as deep neural networks. A neural network (also called an artificial neural network) is an adaptive system that learns by using interconnected nodes or neurons in a layered structure that resembles a human brain. A neural network can learn from data—so it can be trained to recognize patterns, classify data, and forecast future events. A very simple Neural Network can be found in Figure 4.10.

The term “deep” usually refers to the number of hidden layers in the neural network. Traditional neural networks only contain 2-3 hidden layers, while deep networks can have as many as 150.

Deep learning models are trained by using large sets of labeled data and neural network architectures that learn features directly from the data without the need for manual feature extraction.[15]



*Figure 4.10 Neural Networks*

The three most common ways people use deep learning to perform object classification are:

* **Training from Scratch**

To train a deep network from scratch, you gather a very large labeled data set and design a network architecture that will learn the features and model. This is good for new applications, or applications that will have a large number of output categories. This is a less common approach because with the large amount of data and rate of learning, these networks typically take days or weeks to train.

* **Transfer Learning**

Most deep learning applications use the transfer learning approach, a process that involves fine-tuning a pretrained model. You start with an existing network, such as AlexNet or GoogLeNet, and feed in new data containing previously unknown classes. After making some tweaks to the network, you can now perform a new task, such as categorizing only dogs or cats instead of 1000 different objects. This also has the advantage of needing much less data (processing thousands of images, rather than millions), so computation time drops to minutes or hours. Transfer learning requires an interface to the internals of the pre-existing network, so it can be surgically modified and enhanced for the new task.

* **Feature Extraction**

A slightly less common, more specialized approach to deep learning is to use the network as a feature extractor. Since all the layers are tasked with learning certain features from images, we can pull these features out of the network at any time during the training process. These features can then be used as input to a machine learning model such as support vector machines (SVM). Transfer learning technique was used to build the model as we needed to extract the pothole regions. MobileNet V2 was used, which is an existing pre-trained quantized neural network model that extracts the required features.

Training dataset was collected from aggle.com which contains about 665 images of potholes along with xml files used as labels. Labels contain the coordinates of the bounding boxes of where the potholes are located, along with file name. We chose a pre trained SSD MobilenetV2 COCO quantized 300\*300 models.

A pre-trained model has been trained on another dataset and has seen a large number of images and objects. COCO (Common Objects in Context) is a dataset of 330,000 images that contains 1.5 million objects Sfor 80 different classes. Training a model from scratch is highly tedious and time consuming because it may take days or weeks to complete the training process. A pre-trained model has thousands of objects and knows how to classify every one of them.

**4.2.3 Android Application Development**

**4.2.3.1 Steps involved**

*Step 1*: The python version of Pothole detection is made into an application using Android studio.

*Step 2*: The language used for the application is Java.

*Step 3*: A new project is created titled Pothole detection.

*Step 4*: Connect your virtual device with Android Virtual Device Manager (AVD).

*Step 5*: Open fragmentfirst.xml and input the required dimensions and attributes.

*Step 6*: Redesign the component tree.

*Step 7:* Open text\_view and button in constraintLayout.

*Step 8:* Change property values in text\_view.

*Step 9*: Now go back to fragmentfirst.xml and open declared attributes of text\_view.

*Step 10*: Change display properties in text\_view under common attributes.

*Step 11:* Now select the text colour in the textColor field.

*Step 12:* Now close the window and open Text\_view.xml to see that the new properties have been added.

*Step 13:* Go to All attributes to check all Attributes and their properties listed.

*Step 14*: Go to colors.xml file and add the desired colours needed. For the following app Change the property android:textColor and give it a value of @android:color/white.

*Step 15*: Create a new colour resource called screenBackground and input the desired colour.

*Step 16*: Go back to fragment\_first.xml and select ConstraintLayout.

*Step 17*: Now set layout\_width and layout\_height to match with the parent.

*Step 18:* Set both the width and height of the TextView and the Button back to wrap\_content.

*Step 19:* Now add the second fragment.

*Step 20*: Open fragment\_second.xml (app > res > layout > fragment\_second.xml) and switch to

*Step 21*: Design View if needed. Notice that it has a ConstraintLayout that contains a TextView and a Button.

*Step 22:* Set the id to @+id/textview\_random (textview\_random in the Attributes panel.)

*Step 23:* Now implement the python code in Java under the following names Detector.java, Detection.java, CameraDetection.java, Fieldview.jva.

*Step 24*: Select Run app.

*Step 25*: Wait till the following instructions to run Gradle build running, installing apk and launching activity.

*Step 26:* Now export the application and install it in the desired phone.

**4.2.3.2 Design**

To confine and identify the potholes, an android application was constructed utilizing android studio. Once the latitude and longitude of the spot from the firebase is received, the information base now has all the whereabouts of the pothole. After detecting a pothole, the location of that pothole is sent to firebase’s Realtime Database. This helps in knowing where the potholes are present in an area and make the process of repairing the potholes easier. To make it more visually convincing, the data received are used for plotting the locations from the firebase into a map.

Android App is a software intended to run on an Android device or emulator. The term additionally alludes to an APK file which represents Android package. This document is a Zip folder containing application code, resources, and meta data.

Android apps can be written in Kotlin, Java, and C++ and are run inside Virtual Machine. The official development environment is Android Studio. Location-based service apps or location-aware apps are applications that offer various services depending on the user’s location. The user’s location and whereabouts are identified using GPS or data from the cell tower which are facilitated by Wi-Fi or cellular connectivity

Planning remains an integral part of this phase in the mobile app development process. Before actual development/programming efforts start, you will have to:

* + Technical architecture
  + Selecting an apt technology stack
  + Development

A typical mobile app project is made up of three integral parts: back-end/server technology, API(s) and the mobile app front-end.

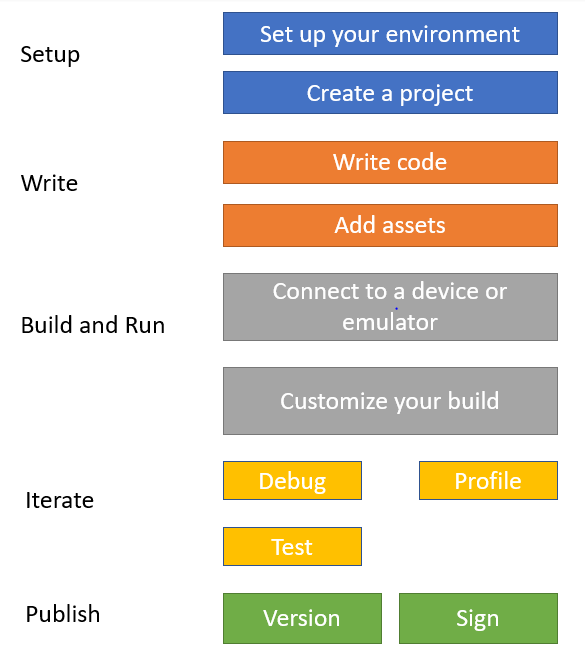
* **Back-End/Server Technology**

It includes database and server-side objects that are necessary in order to support functions that are designed for the mobile application. When using an existing back-end platform, modifications may be required to support the desired mobile functionality.

* **Mobile App Front-End**

The front-end is the native mobile app an end-user will use. Usually for the front end, mobile apps consist of interactive user experiences that uses the APIs and the back-end is for managing data. In some cases, when an app needs to allow users to work without internet access, the app may utilize local data storage.

You can use almost any web programming language and databases for the back-end. For native mobile apps, you must choose a technology stack depending on each mobile OS platform requirement. The basic structure for developing an app with Android Studio can be seen in Figure 4.11.



*Figure 4.11 Basic structure for app building*

For app development, Android studios is used. Android apps are built as a combination of components that can be invoked individually. Android allows to provide different resources for different devices. Here location based android application is developed and the location coordinates received from the firebase are plotted on the map. [13]

**CHAPTER 5**

**STANDARDS USED**

The codes were all written in C++ programming language and were written in Arduino IDE which is an inbuilt IDE in Arduino Uno. For all the respective processes, we created separate functions in separate files. This way the codes are written in a clean way and can be viewed and understood easily. All these codes were executed using the terminal inbuilt in the Arduino Uno. Table 5.1 and 5.2 enlists the prescribed standards for various hardware components used

|  |  |
| --- | --- |
| Microcontroller | Atmega328 |
| Operating Voltage | 5V |
| Input Voltage (Recommended) | 7-12V |
| Input Voltage (Limit) | 6-20V |
| Digital I/O Pins | 14 (of which 6 provide PWM output) |
| Analog Input Pins | 6 |
| DC Current per I/O Pin | 40 mA |
| DC Current for 3.3V Pin | 50 mA |
| Flash Memory | 32 KB (Atmega328) of which 0.5 KB used by bootloader |
| SRAM | 2 KB (Atmega328) |
| EEPROM | 1 KB (Atmega328) |
| Clock Speed | 16 MHz |

*Table**5.1: Standards for Arduino Uno*

The UART communication protocol is used to encode GSM data to a Universal Subscriber Identity Module (USIM). USIM uses a longer authentication to give greater security. It also mutually authenticates the user and the network. The Global System for Mobile Communications (GSM) is a standard developed by the European Telecommunications Standards Institute (ETSI) to describe the protocols for second-generation (2G) digital cellular networks used by mobile devices such as mobile phones and tablets.

|  |  |
| --- | --- |
| Microcontroller | Atmega328 |
| Operating Voltage | 5V |
| Input Voltage (Recommended) | 7-12V |
| Input Voltage (Limit) | 6-20V |
| Digital I/O Pins | 14 (of which 6 provide PWM output) |
| Analog Input Pins | 6 |
| DC Current per I/O Pin | 40 mA |
| DC Current for 3.3V Pin | 50 mA |
| Flash Memory | 32 KB (Atmega328) of which 0.5 KB used by bootloader |
| SRAM | 2 KB (Atmega328) |
| EEPROM | 1 KB (Atmega328) |
| Clock Speed | 16 MHz |

*Table 5.2: Standards for SIM900A*

Features and Support:

* Supports CSD, USSD, SMS, FAX
* Supports MIC and Audio Input
* Speaker Input
* Features keypad interface
* Features display interface
* Features Real Time Clock
* Supports UART interface
* Supports single SIM card
* Firmware upgrade by debug port
* Communication by using AT commands

**Standards for MPU6050**

**Gyroscope Features**

The triple-axis MEMS gyroscope in the MPU-60X0 includes a wide range of features:

* Digital-output X-, Y-, and Z-Axis angular rate sensors (gyroscopes) with a user-programmable full-scale range of ±250, ±500, ±1000, and ±2000°/sec
* External sync signal connected to the FSYNC pin supports image, video and GPS synchronization
* Integrated 16-bit ADCs enable simultaneous sampling of gyros
* Enhanced bias and sensitivity temperature stability reduces the need for user calibration
* Improved low-frequency noise performance
* Digitally-programmable low-pass filter
* Gyroscope operating current: 3.6mA
* Standby current: 5µA
* Factory calibrated sensitivity scale factor
* User self-test

**Accelerometer Features**

The triple-axis MEMS accelerometer in MPU-60X0 includes a wide range of features:

* + - Digital-output triple-axis accelerometer with a programmable full-scale range of ±2g, ±4g, ±8g and ±16g
    - Integrated 16-bit ADCs enable simultaneous sampling of accelerometers while requiring no external multiplexer
    - Accelerometer normal operating current: 500µA
    - Low power accelerometer mode current: 10µA at 1.25Hz, 20µA at 5Hz, 60µA at 20Hz, 110µA at 40Hz
    - Orientation detection and signaling
    - Tap detection
    - User-programmable interrupts
    - High-G interrupt
    - User self-test

We also use Wi-Fi connection to communicate back and forth with the Google APIs and to store and retrieve files from the cloud. Wi-Fi is an IEEE 802.11 standard.

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA. Each project in Android Studio contains one or more modules with source code files and resource files. Types of modules include: Android app modules, Library modules and Google App Engine modules. Each app module contains the following folders

* manifests: Contains the AndroidManifest.xml file.
* java: Contains the Java source code files, including Junit test code.
* res: Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

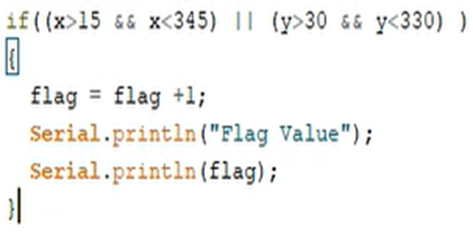
**\**

**CHAPTER 6**

**RESULTS AND DISCUSSION**

**6.1 Bike Fall Detection**

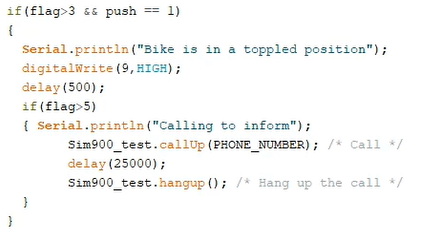
As we mentioned earlier the forward and backward tilt of 15° is permissible and a sideway tilt (right and left) of 30° is permissible. The coding block for the condition mentioned can be seen in Figure 6.1. The permissible state represents the position of a bike when it is not at a toppled position. When the value of the tilt sensor goes above the permissible limit, it indicates that the bike is in a toppled position.

****

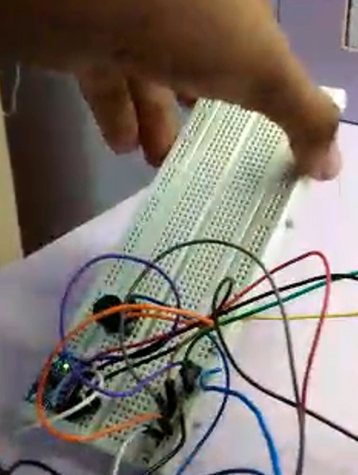
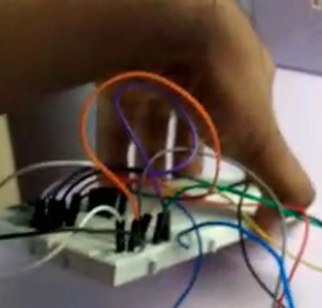
*Figure 6.1: X and Y Axes orientation condition*

Permissible limit of 15° for forward and backward tilt and sideway tilt of 30°. The flag value is increased if the bike tilt is more than permissible limit

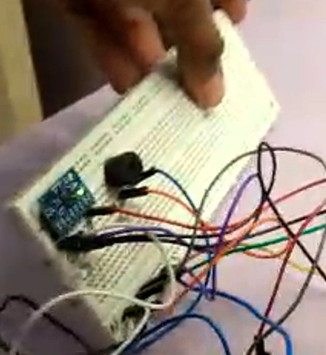
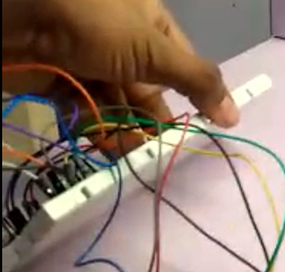
If the flag value exceeds 3 the buzzer buzzes to indicate the driver that the microcontroller senses that the bike is in a toppled position. Upon no response the flag value will continue to increase more than 5 (more than 20 second) at which point an SOS call is made. The coding block for the SOS condition can be seen in Figure 6.2 and the tilting of the prototype in different orientations is shown in Figure 6.3.



*Figure 6.2: Flag condition*

* *

1. *b)*

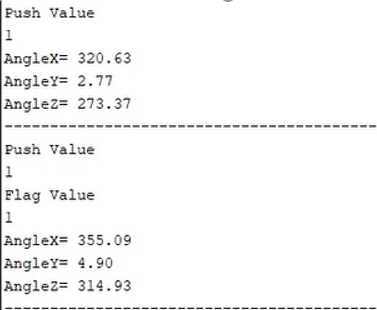
* *

*c) d)*

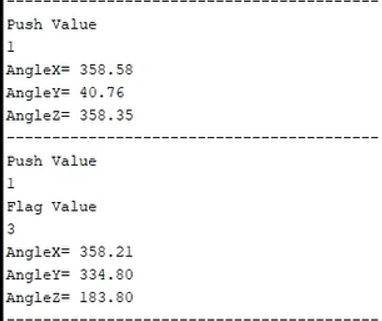
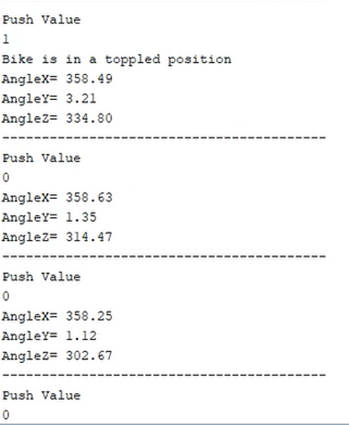
*Figure 6.3: Tilting the tilt sensor above permissible limits in a. Backward X direction;*

*b. Forward X direction, c.Backward Y Direction and d. Forward Y Direction respectively*

The Figure 6.4 consists of Arduino Serial Monitor screenshots where each sub figure corresponds to each sub figure in Figure 6.3, where we tilt the breadboard with MPU6050 above the permissible limit to increment the flag values.

* *

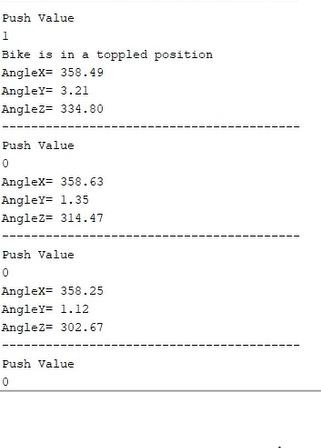
1. *b)*

* *

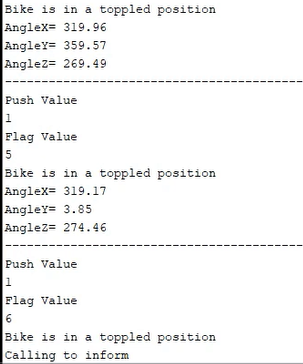
*c) d)*

*Figure 6.4: a. Flag value incremented to 1; b. Flag value incremented to 2; c. Flag value incremented to 3; d. Flag value incremented to 4.*

In Figure 6.5, we can see the case where the button is pressed declaring it’s not an emergency. When the Flag value goes over 5 and the button is not pressed a call is made which can be seen in Figure 6.6a and 6.6b.



*Figure 6.5: Case where push button is pressed before flag value goes over 5*

1. *b)*

*Figure 6.6: a. Case where push button is not pressed before flag value goes over 5; b. Call being made to emergency contact via SIM900A module*

**6.2 Pothole Detection System**

The system was tested by placing a smartphone with our app open in front of a footage of potholes. All the potholes from the footage were accurately detected and displayed on the app with audio warning that says “Pothole Detected”. The working of the module was tested several times successfully and precise outputs were observed. The images in Figure 6.6 are given as input for image processing and we get the images in Figure 6.7 as the result.

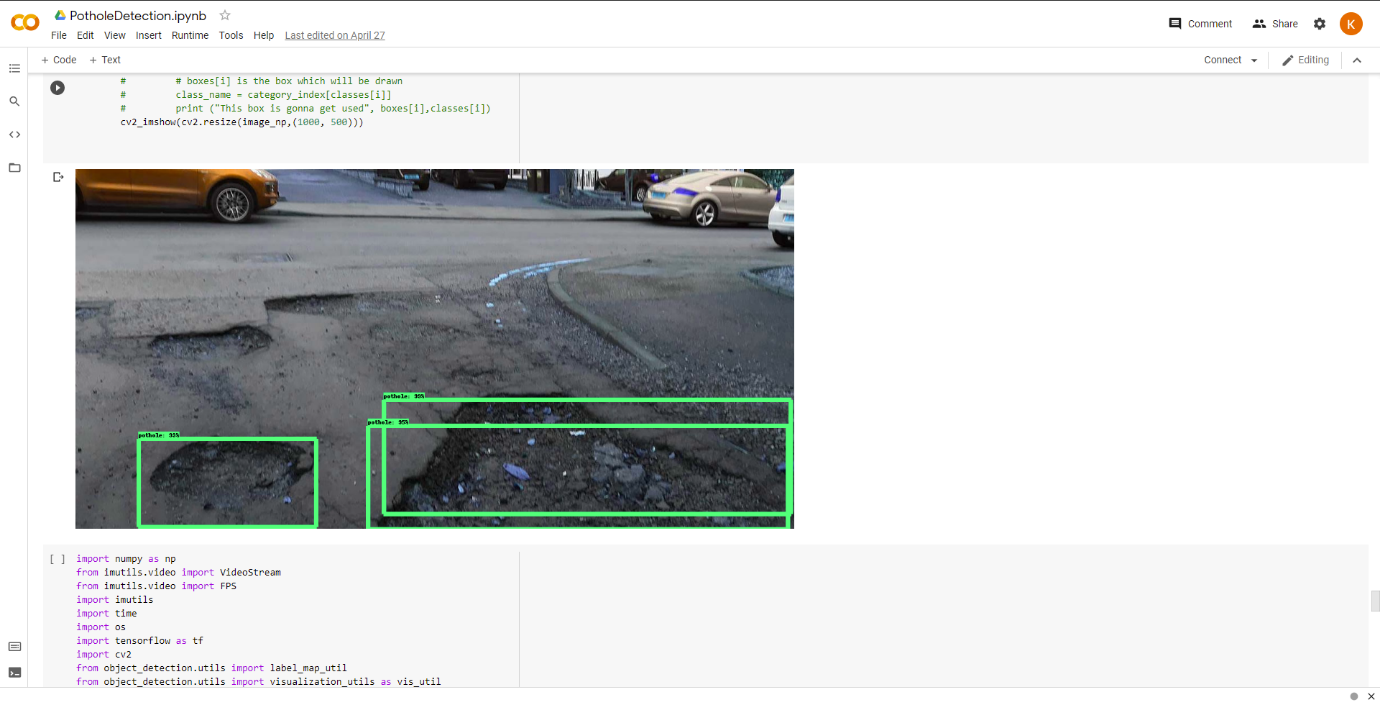
 

*Figure 6.7 Input images given for image processing*

** **

*Figure 6.8 Processed Images*

Before the final android application was made, the system was tested for accurate results in Google Colab, for which the output screen is shown below in Figure 6.8. The program goes through the footage frame by frame and checks for potholes.



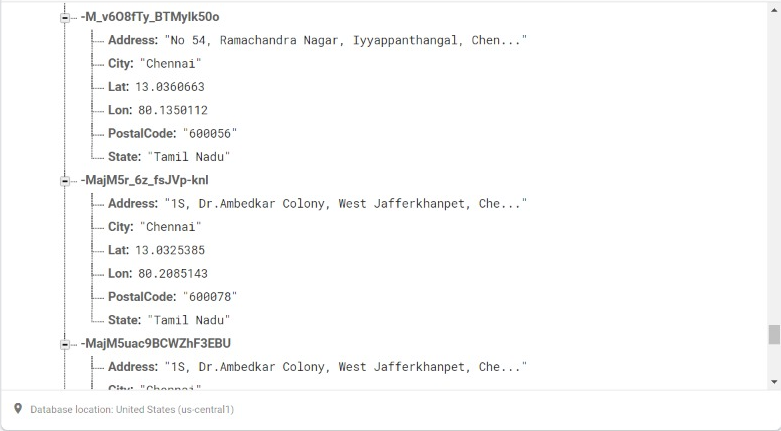
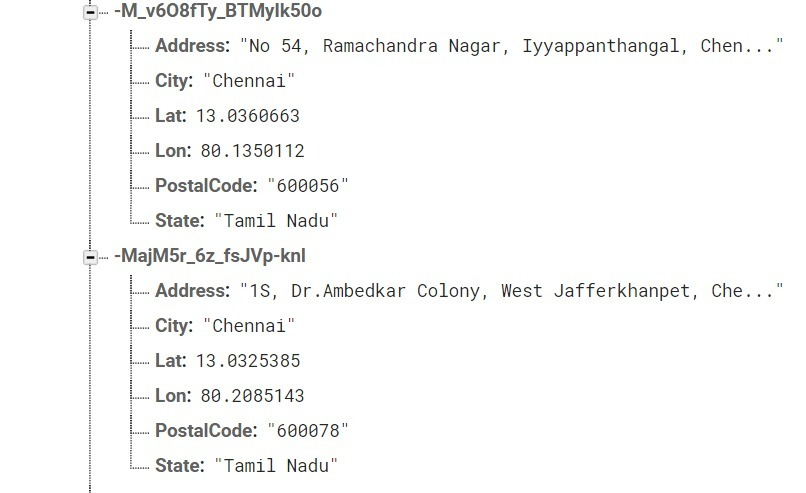
*Figure 6.9 Intermediate Result*

The downloaded APK of the android application is shown in Figure 6.9, the logo for which was designed by the team.

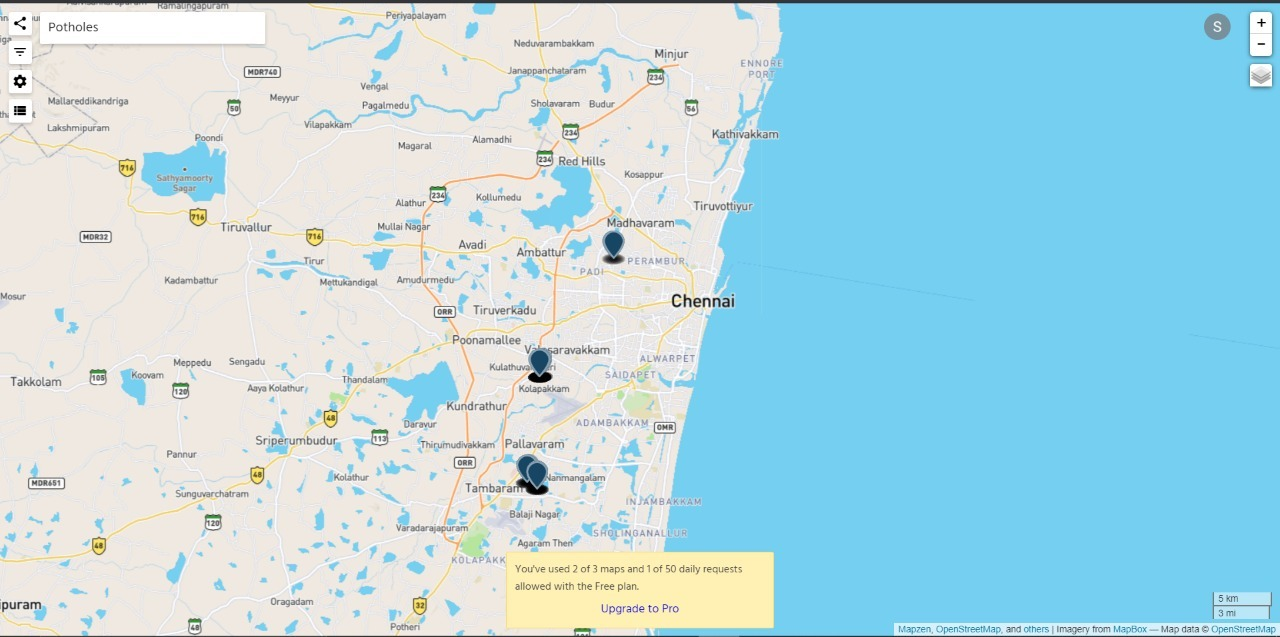
****

*Figure 6.10 Pothole Detection App installed on a smartphone*

For the pothole locations to be updated in the map, the user has to keep their GPS, ON, on their phone. Once the pothole has been identified, the GPS system sends the address of the current location to the Firebase system which can be seen in Figure 6.10, where all such addresses are stored. This data is now updated on the Maps for it to be viewed by the user as shown in Figure 6.11.



*Figure 6.11 Address and Co-ordinates of the potholes*



*Figure 6.12 Mapped coordinates of Potholes on a map*

**CHAPTER 7**

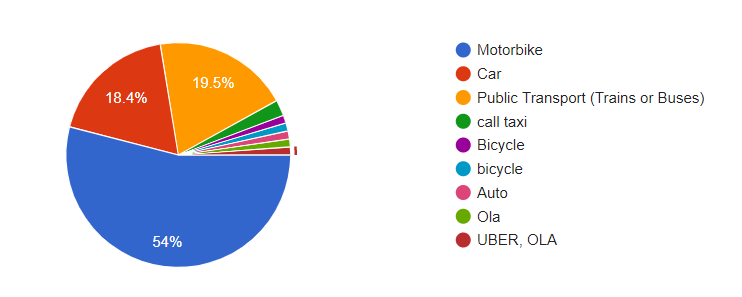
**OUTCOMES ACHIEVED (SOCIETAL PROBLEM SOLVED)**

**7.1 Survey Findings**

We asked our friends and family a set of 7 questions to understand the average road safety needs of common people by circulating a survey made in google forms. We received a total of 87 responses. The visualizations of the responses are shown subsequently from Figure 7.1 to Figure 7.7. The following were the results for each question:

**Q1.**  Which type of road transport do you utilize the most?

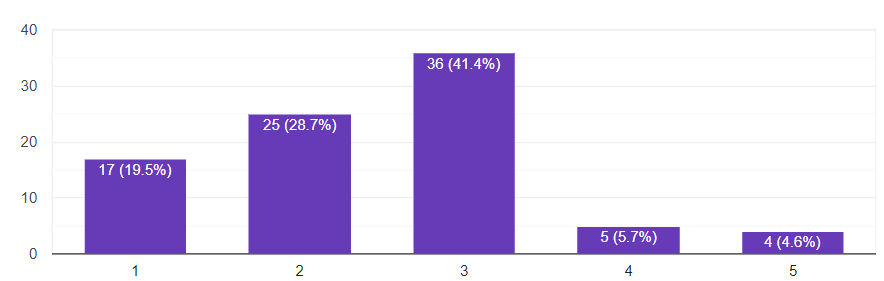
Responses:



*Figure 7.1 Pie chart showing Road Transportation used*

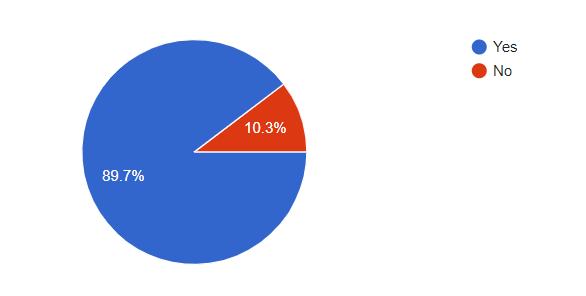
**Q2.** From 1 to 5, rate the safety of Indian roads:

1 = Least Safe, 5 = Most safe



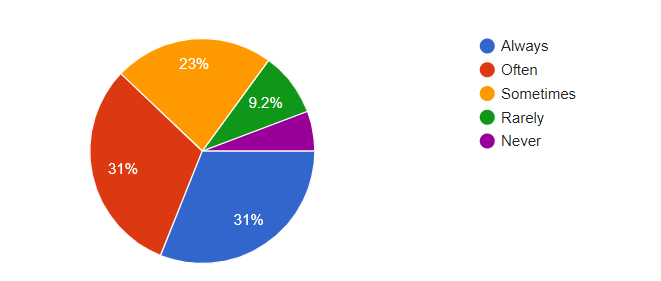
*Figure 7.2 Graph showing Safety of Indian roads*

**Q3.** Have you or someone you know faced significant danger while on road?



*Figure 7.3 Pie chart Road accident frequency*

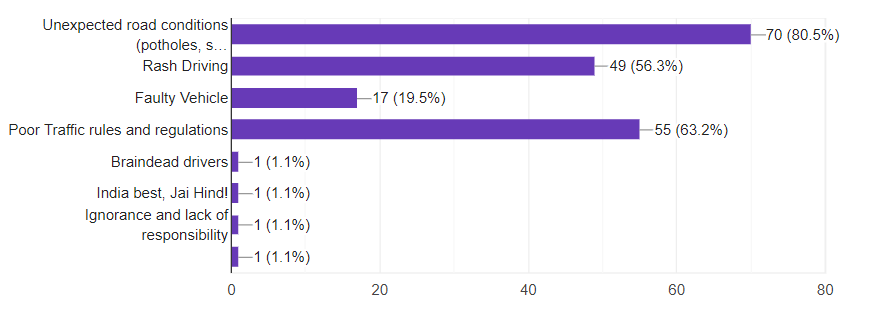
**Q4.** How often do you wear helmet while driving a motorbike?



*Figure 7.4 Pie chart showing Helmet wearers*

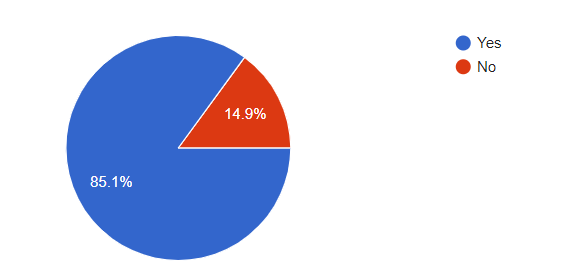
**Q5.** According to you, what might be the most common cause of accidents in India?

This question accepts multiple answers and also accepts user’s custom inputs



*Figure 7.5 Graph showing causes of road accidents*

**Q6.** Have you ever considered additional safety measures for your vehicle?



*Figure 7.6 Pie chart showing people who value additional safety measures*

**Q7.** From the following, which do you think should be implemented to increase road safety? (Select any 2 options)



*Figure 7.7 Graph showing Road safety measures that can be implemented*

**7.2 Survey Inference:**

* Figure 7.1 shows that Motorbike is the major vehicle used for transportation by more than 50% of the users.
* Figure 7.2 shows that Majority of users only feel moderately safe on Indian roads
* Figure 7.3 shows that Almost 90% of the people have faced significant danger while on road.
* Figure 7.4 shows that Many people wear helmets either always or often, while driving a motorbike.
* Figure 7.5 shows that unexpected road conditions are the main source of accidents according to our survey. Poor traffic regulations and Rash Driving take second and third place respectively.
* Figure 7.6 shows that Almost 85% of the users have considered additional safety measures for their vehicles.
* Figure 7.7 shows that Pothole Detection and Automatic SOS system tops the road safety needs of the users.

It is clearly evident that the people acknowledge the societal problem that we have mentioned. So, it is indeed important to analyze the results and reach a solution to mitigate the issue.

According to the survey we conducted it’s understood that the unexpected road conditions is the main source of accidents according to our survey. Poor traffic regulations and Rash Driving take second and third place respectively. We designed our project such that we minimalize the number of accidents that occur due to such conditions.

Apart from our survey these roads were responsible for majority of the road accidents in India. Bad road engineering, made worse due to incessant rains is the cause for 50% of the accidents. Many key roads have been rendered unmotorable during adverse weather conditions. Every year around 3,597 people die due to potholes. More than 30% of people die due to potholes. The Ministry of Road Transport and Highways provided figures that over 9300 deaths, 25000 injured in the last three years due to potholes and more than 25,000 people are getting injured due to potholes.

Road accidents cost India 3-5% of gross domestic product every year, and are avoidable if India could improve its roads and city planning, train its drivers better, and enforce traffic laws properly, our analysis shows. India’s young, productive population, aged 18-45 years, is involved in 70% of road accidents, according to data from Road Accidents in India 2018, a report published by the Ministry of Road Transport and Highways (MORTH). Over a period of 24 years from 2014 to 2038, if India could halve the deaths and injuries due to road traffic, its GDP could increase by 7%, a 2018 World Bank report said. Over 80% of fatalities in road accidents in India happened due to speeding and dangerous or careless driving, a comparative analysis of National Crime Records Bureau (NCRB) data since 2014 shows.

Thus, we designed two projects Pothole detection system and Bike fall detection system to minimize the number of accidents caused due to the above-mentioned issues. Accidents caused by potholes can’t be blamed on drivers but we can help drivers by alerting about potholes. Drivers can’t focus on both traffic and potholes. Therefore, if drivers get help in detecting potholes they can drive better as they focus on traffic. So, we created a pothole detection system which identifies the potholes in the road while they travel. If the application is used a greater number of people the efficiency of the system would increase and the potholes will be alerted in the map interface which prevents many numbers of accidents.

Another main problem is after accidents happen there are no one near to help the victim and this has cost people’s lives. So, a system which can alert people if there is an accident so that the victim can receive help adequately. The system must not be a separate device as people tend to be lazy and do not use it. The System is integrated with the vehicle itself. So, this bike fall detection system reduces the number of deaths occurred due to accidents and improves the standard of living in our country.

**CHAPTER 8**

**COST ANALYSIS**

**8.1 List of components and their cost**

The costs of the various components used in this project are given below in Table 8.1

|  |  |
| --- | --- |
| **COMPONENT** | **COST** |
| Arduino UNO R3 | Rs. 500 |
| GSM SIM900A | Rs 1160 |
| MPU6050 | Rs 330` |
| Buzzer | Rs 20 |
| **Total** | **Rs. 2010** |

*Table 8.1 Cost Analysis*

**CHAPTER 9**

**TESTING AND QUALITY ASSURANCE**

**9.1 Testing of the trained model**

Testing the created prototype is the most essential part of creating a successful product. Only then will the product solve the problem at hand completely. Even when a prototype is built completely, errors tend to occur which might not be visible even to the creators of the product. To test at what threshold the model must be at, a lot of testing images were used and checked if the model detected the potholes with threshold maintained at the maximum level. Out of 100 input test images, 93 potholes were detected when the threshold was kept at 0.6. Increasing the threshold decreased the accuracy which was undesirable. False detection was also kept at a minimum when the threshold was at 0.6. Decreasing the threshold further decreased the accuracy further by showing more false detections. So, the optimum threshold was 0.6 which yielded an accuracy of 93 percent.

The android application has no errors and can be downloaded by anybody with the apk link of the application. Hence, the quality of the application is top notch and there are no inconsistencies in the working of the app as such.

We tested the model on roadways that had distractive markings to check if the model is detecting the potholes correctly and also to check if the system is not identifying the markings as potholes. The tested images are shown below in Figure 9.1.

*a) b)*

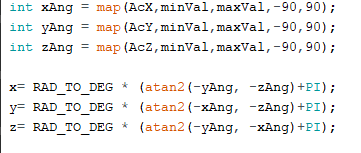
*c) d)*

*Figure 9.1: Pothole detection app scanning a. a crosswalk; b. a collection of potholes; c. lane markings with zebra crossing; d. median strip*

As we can see from 9.1a, 9.1c and 9.1d, roads with even markings on them like zebra crossings, medians, and lane markings are not detected as having potholes. Whereas, the road with potholes is correctly being detected with multiple bounding boxes of potholes, as seen in figure 9.1b. This is a very good result and can be implemented in the practical world.

**9.2 Calibration of MPU6050**

Let x be the amount of rotation around the x axis. With an assumption that in the starting position/neutral position the direction of acceleration measured by the “z” sensor is straight up or down. The line of code, int xAng = map(ax, minAcel, maxAcel, -90, 90) sets xAng to some constant k times Acx. This is shown in Figure 9.2.

****

*Figure 9.2: Storing inclination in degrees from accelerometer readings*

Now let’s look at this mathematically, let us denote Acy by ay , yAng by yAng and x by θx

We know that DEG\_TO\_RAD is 180/π in mathematical notation. Also it is to be noted that atan2(y,z) will solve for an angle θ between −π and π and a positive number r such that

y = rsinθ and z = rcosθ, and return θ. For example, atan2 0,3) = 0 and atan2(0.4,0) = π2.

When the array of sensors is in its rest position, the acceleration measured by az is 9.8 m/s−2, that is, it registers that the ground (or whatever the array is sitting on) is pushing it upward with a force of mg (where m is the array’s mass and g is the acceleration of gravity) to keep the array from falling. That is az =g=9.8 (or whatever number 1 g comes out to in the units your accelerometers use). At the same time ax=ay=0, since there is no other force on the array. Then

yAng = kay = 0

&

zAng = kaz = kg.

Plugging these into the formula used in line of code, x= RAD\_TO\_DEG \* (atan2(-yAng, -zAng) +PI); We get,

θx = (180/π) (atan2(−yAng, −zAng) +π)

= (180/π) (atan2(0, −kg) +π)

= (180/π) (π+π)

= 360.

Now if we rotate the accelerometer 45 degrees around the x axis so that both the y and z axes are sloped upward at 45-degree angles. Then ay = az = gsin(π/4)= √2g/2 and

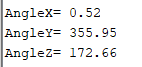
θx = (180/π) (atan2(−yAng, −zAng) +π)

= (180/π) (atan2(−k√2g/2, −k√2g/2) +π)

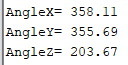
≈ (180/π) (−(3π/4) +π)

= 45

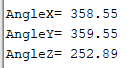
But this works as good as the minVal, maxVal value given by us. So, we will be comparing the x and y axis tilt obtained with real-time value for different values of the variable minVal, maxVal. For this we will be placing the MPU6050 on a flat surface, where x axis and y axis tilt are 0 degrees respectively. As we see in the below set of Figures 9.3 a, 9.3 b, 9.3 c, 9.3 d and 9.3 e, the output on the Arduino serial monitor approaches zero degrees angle of tilt in the x and y axes as the minVal and maxVal value reach 265 and 402 respectively.



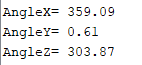
*Figure 9.3 a: Output when minVal = 0, maxVal = 600*



*Figure 9.3 b: Output when minVal = 100, maxVal = 600*



*Figure 9.3 c: Output when minVal = 200, maxVal = 500*

****

*Figure 9.3 d: Output when minVal = 250, maxVal = 400*



*Figure 9.3 e: Output when minVal = 265, maxVal = 402*

**CHAPTER 10**

**CONCLUSION AND FUTURE WORKS**

**10.1 Conclusion**

As road transportation is one of the commonly utilizing transportation methodology, it is essential to distinguish the harms. Recognizing potholes accurately is one of vital errands for deciding appropriate procedures of asphalt maintenance.

A Pothole Detection System using Machine Learning and Deep Learning techniques was developed. Using photos of various potholes of different shapes and sizes as dataset, the training of the detection model was developed. Using Android Studio, an application that detects potholes and warns the rider with a voice notification that says “Pothole Detected” was developed. The code was done in Python using Tensorflow package and a pre-trained SSD Mobilenet v2 as 300x300 this model is a single-stage object detection model that goes straight from image pixels to bounding box coordinates and class probabilities. The model architecture is based on inverted residual structure where the input and output of the residual block are thin bottleneck layers as opposed to traditional residual models.

Due to lot of motorcycle accidents in our country, a safety system was developed to create an alert in case the motorcycle rider has met with an accident. The System has solved problem by keeping track of the angle of the bike and its deceleration. If the deceleration of the motorcycle is greater than 3 m/s2 and the angle/tilt of the motorcycle is greater than 60⁰, then we can conclude that the rider has met with an accident. After detecting the accident. A phone call was made to an emergency contact using GSM module.

**10.2 Future Works**

The following ideas can be considered for the future expansion of the model

1. Pothole Detection System:
2. Better training can be done to increase the accuracy of the model.
3. Maps can be included in the android application for showing the potholes graphically. There can be voice notifications when a rider is going to encounter a pothole.
4. A more complex prototype can be developed to control the speed of the vehicle when there is pothole ahead.
5. A different algorithm can be used to find the distance of the pothole from the vehicle.

* A more complex prototype can be developed to detect other obstruction in road like speed breakers, garbage, animals etc. Which would improve more safety.
* The model can be trained to detect potholes in complete darkness and also find potholes that are either submerged or filled with water.
* We can improve the system by storing the location of potholes detected in a database then this database has information of potholes in various roads which can be used by the government to improve the road infrastructure.

1. Bike-fall Detection System:
2. A more complex prototype can be built such that, when the bike tilts by an angle that can topple it, a closed control loop system that prevents the bike from toppling can be implemented.

* We can improve the system by creating a app that displays the location of fall so the app can be used by many users which can help rescue personnel get precise location of accident which help save more lives.
* A more complex prototype can be built which can tell at which speed the bike was travelling before falling and how fast it has fallen hence, we can understand the severity of the fall.

**CHAPTER 11**

**REFERENCES**

[1] Artis Mednis, Girts Strazdins, Reinholds Zviedris, Georgijs Kanonirs, Leo Selavo Real Time Pothole Detection Using Android Smartphones with Accelerometers 2011 7th IEEE International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)

[2] Hsiu-Wen Wang,1 Chi-Hua Chen ,2 Ding-Yuan Cheng,3 Chun-Hao Lin,2 and Chi-Chun Lo1 a Real-Time Pothole Detection Approach for Intelligent Transportation System Published 27 Aug 2015

[3] Kulwant Singh, Sristy Hazra, Chandra Mukherjee, Sushanth.G, Sanjith Gowda. IOT Based Real Time Potholes Detection System Using Image Processing Techniques, In International Journal of Scientific & Technology Research, volume 9, issue 02, February 2020.

[4] Seung-Ki Ryu, Taehyeong Kim, Young-Ro Kim, Image-Based Pothole Detection System for ITS Service and Road Management System, vol. 2015, Article ID 968361, 10 pages, 2015.

[5] EMIR BUZA, SAMIR OMANOVIC, ALVIN HUSEINOVIC Pothole Detection with Image Processing and Spectral Clustering – 2015

[6] Jo, Y.; Ryu, S. Pothole Detection System Using a Black-box Camera. Sensors **2015**, 15, 29316-29331.

[7] Akash Singh, Rajkumar R, Two-Wheeler Accident Detection and Alert System with Anti-Theft Control, International Journal of Science and Research (IJSR) ISSN: 2319-7064

[8] Shrinidhi.N, Prof. Bheema Shastry, Smart Bike Accident Detection System Reporting with Black Box, International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056 Volume: 07 Issue: 07, July 2020

[9] Nanda, S., Joshi, H., & Khairnar, S. (2018). An IOT Based Smart System for Accident Prevention and Detection. 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*.*

[10] Khan, M. H. U., & Howlader, M. M. (2019). Design of An Intelligent Autonomous Accident Prevention, Detection and Vehicle Monitoring System. 2019 IEEE International Conference on Robotics, Automation, Artificial-Intelligence and Internet-of-Things (RAAICON).

[11] N. Dogru and A. Subasi, “Traffic accident detection using random forest classifier,” 2018 15th Learning and Technology Conference (L&T), Jeddah, Saudi Arabia, 2018, pp. 40-45, doi: 10.1109/LT.2018.8368509.

[12] A. Ali and M. Eid, “An automated system for Accident Detection,” 2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings, Pisa, Italy, 2015, pp. 1608-1612, doi: 10.1109/I2MTC.2015.7151519.

[13] Van Drongelen, Mike. *Android studio cookbook*. Packt Publishing Ltd, 2015.

[14] Borji, Ali, et al. “Salient object detection: A benchmark.” *IEEE transactions on image processing* 24.12 (2015): 5706-5722.

[15] Goodfellow, Ian, et al. *Deep learning*. Vol. 1. No. 2. Cambridge: MIT press, 2016.

[16] Al-Dahan, Ziad Tarik, Nasseer K. Bachache, and Lina Nasseer Bachache. “Design and implementation of fall detection system using MPU6050 Arduino.” *International Conference on Smart Homes and Health Telematics*. Springer, Cham, 2016.

[17] Mukherjee, Saikat, Arpita Ghosh, and Subir Kumar Sarkar. “Arduino based Wireless Heart-rate Monitoring system with Automatic SOS Message and/or Call facility using SIM900A GSM Module.” *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*. IEEE, 2019.

**CHAPTER 12**

**APPENDIX**

**ARDUINO CODE FOR BIKE FALL DETECTION SYSTEM**

#include <GPRS\_Shield\_Arduino.h>

#include <SoftwareSerial.h>

#include <Wire.h>

#define PIN\_TX 8 /\* rx of Arduino (connect tx of gprs to this pin) \*/

#define PIN\_RX 7 /\* tx of Arduino (connect rx of gprs to this pin) \*/

#define BAUDRATE 9600

#define PHONE\_NUMBER "9092330551"

const int MPU\_addr=0x68;

int16\_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;

int minVal=265;

int maxVal=402;

int button = 10;

int flag = 0;

double x;

double y;

double z;

GPRS Sim900\_test(PIN\_TX,PIN\_RX,BAUDRATE);

void setup(){

while(!Sim900\_test.init()) /\* Sim card and signal check, also check if module connected \*/

{

delay(1000);

Serial.println("SIM900 initialization error");

}

Serial.println("SIM900 initialization success");

Wire.begin();

Wire.beginTransmission(MPU\_addr);

Wire.write(0x6B);

Wire.write(0);

Wire.endTransmission(true);

Serial.begin(9600);

pinMode(9,OUTPUT);

pinMode(10, INPUT\_PULLUP);

}

void loop(){

int push;

Wire.beginTransmission(MPU\_addr);

Wire.write(0x3B);

Wire.endTransmission(false);

Wire.requestFrom(MPU\_addr,14,true);

AcX=Wire.read()<<8|Wire.read();

AcY=Wire.read()<<8|Wire.read();

AcZ=Wire.read()<<8|Wire.read();

int xAng = map(AcX,minVal,maxVal,-90,90);

int yAng = map(AcY,minVal,maxVal,-90,90);

int zAng = map(AcZ,minVal,maxVal,-90,90);

x= RAD\_TO\_DEG \* (atan2(-yAng, -zAng)+PI);

y= RAD\_TO\_DEG \* (atan2(-xAng, -zAng)+PI);

z= RAD\_TO\_DEG \* (atan2(-yAng, -xAng)+PI);

Serial.print("AngleX= ");

Serial.println(x);

Serial.print("AngleY= ");

Serial.println(y);

Serial.print("AngleZ= ");

Serial.println(z);

Serial.println("-----------------------------------------");

push = digitalRead(button);

Serial.println("Push Value");

Serial.println(push);

if((x>15 && x<345) || (y>30 && y<330) )

{

flag = flag +1;

Serial.println("Flag Value");

Serial.println(flag);

}

if(flag>3 && push == 1)

{

Serial.println("Bike is in a toppled position");

digitalWrite(9,HIGH);

delay(500);

if(flag>5)

{ Serial.println("Calling to inform");

Sim900\_test.callUp(PHONE\_NUMBER); /\* Call \*/

delay(25000);

Sim900\_test.hangup(); /\* Hang up the call \*/

}

}

else if(push == 0)

{

flag = 0;

digitalWrite(9,LOW);

}

delay(4000);

}

**PYTHON CODE FOR POTHOLE DETECTION**

#@title Install Required Packages:

!apt update && apt install protobuf-compiler -y

!pip install lxml cython numpy psillow matplotlib

!pip install pycocotools tensorflow==1.15.0 kaggle

cd /content/drive/MyDrive/Pothole Detcection

!pip install patool

import patoolib

patoolib.extract\_archive("dataset.rar", outdir="/content/drive/MyDrive/Pothole Detcection")

from google.colab import drive

drive.mount('/content/drive')

#@title Importing Libraries:

from \_\_future\_\_ import division, print\_function, absolute\_import

import pandas as pd

import numpy as np

import csv

import os, sys, fnmatch, random, json

import re

import os

import io

import glob

import shutil

import urllib.request

import tarfile

import xml.etree.ElementTree as ET

import tensorflow.compat.v1 as tf

import cv2

from PIL import Image

from collections import namedtuple, OrderedDict

from google.colab import files

print(tf.\_\_version\_\_)

cd /content/drive/MyDrive/Pothole Detcection/Uploaded

#@title Preprocessing Images and Labels

#creating two dir for training and testing

!mkdir test\_labels train\_labels

# lists the files inside 'annotations' in a random order (not really random, by their hash value instead)

# Moves the first 130 labels  to the testing dir: `test\_labels`

!ls annotations/\* | sort -R | head -130 | xargs -I{} mv {} test\_labels/

# Moves the rest of labels '535' labels to the training dir: `train\_labels`

!ls annotations/\* | xargs -I{} mv {} train\_labels/

#@title Install Required Packages:

!apt-get install -qq protobuf-compiler python-pil python-lxml python-tk

!pip install -qq Cython contextlib2 pillow lxml matplotlib pycocotools

#@title Converting to csv

#adjusted from: https://github.com/datitran/raccoon\_dataset

def xml\_to\_csv(path):

  classes\_names = []

  xml\_list = []

  for xml\_file in glob.glob(path + '/\*.xml'):

    tree = ET.parse(xml\_file)

    root = tree.getroot()

    for member in root.findall('object'):

      classes\_names.append(member[0].text)

      value = (root.find('filename').text ,

               int(root.find('size')[0].text),

               int(root.find('size')[1].text),

               member[0].text,

               int(member[4][0].text),

               int(member[4][1].text),

               int(member[4][2].text),

               int(member[4][3].text))

      xml\_list.append(value)

  column\_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']

  xml\_df = pd.DataFrame(xml\_list, columns=column\_name)

  classes\_names = list(set(classes\_names))

  classes\_names.sort()

  return xml\_df, classes\_names

for label\_path in ['train\_labels', 'test\_labels']:

  image\_path = os.path.join(os.getcwd(), label\_path)

  xml\_df, classes = xml\_to\_csv(label\_path)

  xml\_df.to\_csv(f'{label\_path}.csv', index=None)

  print(f'Successfully converted {label\_path} xml to csv.')

label\_map\_path = os.path.join("label\_map.pbtxt")

pbtxt\_content = ""

for i, class\_name in enumerate(classes):

    pbtxt\_content = (

        pbtxt\_content

        + "item {{\n    id: {0}\n    name: '{1}'\n}}\n\n".format(i + 1, class\_name)

    )

pbtxt\_content = pbtxt\_content.strip()

with open(label\_map\_path, "w") as f:

    f.write(pbtxt\_content)

#@title Downloading Tensorflow model

# downloads the models

!git clone --q https://github.com/tensorflow/models.git

%cd /content/drive/MyDrive/Pothole Detcection/models/research

!pip install tf\_slim

# testing the model builder

!python3 object\_detection/builders/model\_builder\_test.py

#@title Generating TFRecords.

#adjusted from: https://github.com/datitran/raccoon\_dataset

from object\_detection.utils import dataset\_util

#change this to the base directory where your data/ is

data\_base\_url = '/content/drive/MyDrive/Pothole Detcection/Uploaded/'

#location of images

image\_dir = data\_base\_url +'images/'

def class\_text\_to\_int(row\_label):

  if row\_label == 'pothole':

    return 1

  else:

    None

def split(df, group):

  data = namedtuple('data', ['filename', 'object'])

  gb = df.groupby(group)

  return [data(filename, gb.get\_group(x)) for filename, x in zip(gb.groups.keys(), gb.groups)]

def create\_tf\_example(group, path):

  with tf.io.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:

    encoded\_jpg = fid.read()

  encoded\_jpg\_io = io.BytesIO(encoded\_jpg)

  image = Image.open(encoded\_jpg\_io)

  width, height = image.size

  filename = group.filename.encode('utf8')

  image\_format = b'jpg'

  xmins = []

  xmaxs = []

  ymins = []

  ymaxs = []

  classes\_text = []

  classes = []

  for index, row in group.object.iterrows():

    xmins.append(row['xmin'] / width)

    xmaxs.append(row['xmax'] / width)

    ymins.append(row['ymin'] / height)

    ymaxs.append(row['ymax'] / height)

    classes\_text.append(row['class'].encode('utf8'))

    classes.append(class\_text\_to\_int(row['class']))

  tf\_example = tf.train.Example(features=tf.train.Features(feature={

    'image/height': dataset\_util.int64\_feature(height),

    'image/width': dataset\_util.int64\_feature(width),

    'image/filename': dataset\_util.bytes\_feature(filename),

    'image/source\_id': dataset\_util.bytes\_feature(filename),

    'image/encoded': dataset\_util.bytes\_feature(encoded\_jpg),

    'image/format': dataset\_util.bytes\_feature(image\_format),

    'image/object/bbox/xmin': dataset\_util.float\_list\_feature(xmins),

    'image/object/bbox/xmax': dataset\_util.float\_list\_feature(xmaxs),

    'image/object/bbox/ymin': dataset\_util.float\_list\_feature(ymins),

    'image/object/bbox/ymax': dataset\_util.float\_list\_feature(ymaxs),

    'image/object/class/text': dataset\_util.bytes\_list\_feature(classes\_text),

    'image/object/class/label': dataset\_util.int64\_list\_feature(classes),

    }))

  return tf\_example

#creates tfrecord for both csv's

for csv in ['train\_labels', 'test\_labels']:

  writer = tf.io.TFRecordWriter(data\_base\_url + csv + '.record')

  path = os.path.join(image\_dir)

  examples = pd.read\_csv(data\_base\_url + csv + '.csv')

  grouped = split(examples, 'filename')

  for group in grouped:

    tf\_example = create\_tf\_example(group, path)

    writer.write(tf\_example.SerializeToString())

  writer.close()

  output\_path = os.path.join(os.getcwd(), data\_base\_url + csv + '.record')

  print('Successfully created the TFRecords: {}'.format(data\_base\_url +csv + '.record'))

#@title Choose a Pre-Trained Model: Training a model from scratch is extremely time consuming; it may take days or weeks to finish training. A pre-trained model has already seen tons of objects and knows how to classify each one of them.

MODELS\_CONFIG = {

    'ssd\_mobilenet\_v2': {

        'model\_name': 'ssd\_mobilenet\_v2\_quantized\_300x300\_coco\_2019\_01\_03',

    },

    'faster\_rcnn\_inception\_v2': {

        'model\_name': 'faster\_rcnn\_inception\_v2\_coco\_2018\_01\_28',

    },

}

# Select a model from `MODELS\_CONFIG`.

# I chose ssd\_mobilenet\_v2 for this project, you could choose any

selected\_model = 'ssd\_mobilenet\_v2'

#@title Download the Pre-Trained Model:

#the distination folder where the model will be saved

#change this if you have a different working dir

DEST\_DIR = '/content/drive/MyDrive/Pothole Detcection/models/research/pretrained\_model'

# Name of the object detection model to use.

MODEL = MODELS\_CONFIG[selected\_model]['model\_name']

#selecting the model

MODEL\_FILE = MODEL + '.tar.gz'

#creating the downlaod link for the model selected

DOWNLOAD\_BASE = 'http://download.tensorflow.org/models/object\_detection/'

#checks if the model has already been downloaded, download it otherwise

if not (os.path.exists(MODEL\_FILE)):

    urllib.request.urlretrieve(DOWNLOAD\_BASE + MODEL\_FILE, MODEL\_FILE)

#unzipping the model and extracting its content

tar = tarfile.open(MODEL\_FILE)

tar.extractall()

tar.close()

# creating an output file to save the model while training

os.remove(MODEL\_FILE)

if (os.path.exists(DEST\_DIR)):

    shutil.rmtree(DEST\_DIR)

os.rename(MODEL, DEST\_DIR)

#@title Viewing the content of the sample config file

#path to the config file

!cat object\_detection/samples/configs/ssd\_mobilenet\_v2\_quantized\_300x300\_coco.config

#@title Editing

#path to the config file

%%writefile object\_detection/samples/configs/ssd\_mobilenet\_v2\_quantized\_300x300\_coco.config

# Quantized trained SSD with Mobilenet v2 on MSCOCO Dataset.

# Users should configure the fine\_tune\_checkpoint field in the train config as

# well as the label\_map\_path and input\_path fields in the train\_input\_reader and

# eval\_input\_reader. Search for "PATH\_TO\_BE\_CONFIGURED" to find the fields that

# should be configured.

model {

  ssd {

    num\_classes: 1

    box\_coder {

      faster\_rcnn\_box\_coder {

        y\_scale: 10.0

        x\_scale: 10.0

        height\_scale: 5.0

        width\_scale: 5.0

      }

    }

    matcher {

      argmax\_matcher {

        matched\_threshold: 0.5

        unmatched\_threshold: 0.5

        ignore\_thresholds: false

        negatives\_lower\_than\_unmatched: true

        force\_match\_for\_each\_row: true

      }

    }

    similarity\_calculator {

      iou\_similarity {

      }

    }

    anchor\_generator {

      ssd\_anchor\_generator {

        num\_layers: 6

        min\_scale: 0.2

        max\_scale: 0.95

        aspect\_ratios: 1.0

        aspect\_ratios: 2.0

        aspect\_ratios: 0.5

        aspect\_ratios: 3.0

        aspect\_ratios: 0.3333

      }

    }

    image\_resizer {

      fixed\_shape\_resizer {

        height: 300

        width: 300

      }

    }

    box\_predictor {

      convolutional\_box\_predictor {

        min\_depth: 0

        max\_depth: 0

        num\_layers\_before\_predictor: 0

        use\_dropout: false

        dropout\_keep\_probability: 0.8

        kernel\_size: 1

        box\_code\_size: 4

        apply\_sigmoid\_to\_scores: false

        conv\_hyperparams {

          activation: RELU\_6,

          regularizer {

            l2\_regularizer {

              weight: 0.00004

            }

          }

          initializer {

            truncated\_normal\_initializer {

              stddev: 0.03

              mean: 0.0

            }

          }

          batch\_norm {

            train: true,

            scale: true,

            center: true,

            decay: 0.9997,

            epsilon: 0.001,

          }

        }

      }

    }

    feature\_extractor {

      type: 'ssd\_mobilenet\_v2'

      min\_depth: 16

      depth\_multiplier: 1.0

      conv\_hyperparams {

        activation: RELU\_6,

        regularizer {

          l2\_regularizer {

            weight: 0.00004

          }

        }

        initializer {

          truncated\_normal\_initializer {

            stddev: 0.03

            mean: 0.0

          }

        }

        batch\_norm {

          train: true,

          scale: true,

          center: true,

          decay: 0.9997,

          epsilon: 0.001,

        }

      }

    }

    loss {

      classification\_loss {

        weighted\_sigmoid {

        }

      }

      localization\_loss {

        weighted\_smooth\_l1 {

        }

      }

      hard\_example\_miner {

        num\_hard\_examples: 3000

        iou\_threshold: 0.99

        loss\_type: CLASSIFICATION

        max\_negatives\_per\_positive: 3

        min\_negatives\_per\_image: 3

      }

      classification\_weight: 1.0

      localization\_weight: 1.0

    }

    normalize\_loss\_by\_num\_matches: true

    post\_processing {

      batch\_non\_max\_suppression {

        score\_threshold: 1e-8

        iou\_threshold: 0.6

        max\_detections\_per\_class: 100

        max\_total\_detections: 100

      }

      score\_converter: SIGMOID

    }

  }

}

train\_config: {

  batch\_size: 12

  optimizer {

    rms\_prop\_optimizer: {

      learning\_rate: {

        exponential\_decay\_learning\_rate {

          initial\_learning\_rate: 0.004

          decay\_steps: 800720

          decay\_factor: 0.95

        }

      }

      momentum\_optimizer\_value: 0.9

      decay: 0.9

      epsilon: 1.0

    }

  }

  fine\_tune\_checkpoint: "/content/drive/MyDrive/Pothole Detcection/models/research/pretrained\_model/model.ckpt"

  fine\_tune\_checkpoint\_type:  "detection"

  # Note: The below line limits the training process to 200K steps, which we

  # empirically found to be sufficient enough to train the pets dataset. This

  # effectively bypasses the learning rate schedule (the learning rate will

  # never decay). Remove the below line to train indefinitely.

  num\_steps: 200000

  data\_augmentation\_options {

    random\_horizontal\_flip {

    }

  }

  data\_augmentation\_options {

    ssd\_random\_crop {

    }

  }

}

train\_input\_reader: {

  tf\_record\_input\_reader {

    input\_path: "/content/drive/MyDrive/Pothole Detcection/Uploaded/train\_labels.record"

  }

  label\_map\_path: "/content/drive/MyDrive/Pothole Detcection/Uploaded/label\_map.pbtxt"

}

eval\_config: {

  num\_examples: 8000

  # Note: The below line limits the evaluation process to 10 evaluations.

  # Remove the below line to evaluate indefinitely.

  max\_evals: 10

}

eval\_input\_reader: {

  tf\_record\_input\_reader {

    input\_path: "/content/drive/MyDrive/Pothole Detcection/Uploaded/test\_labels.record"

  }

  label\_map\_path: "/content/drive/MyDrive/Pothole Detcection/Uploaded/label\_map.pbtxt"

  shuffle: false

  num\_readers: 1

}

graph\_rewriter {

  quantization {

    delay: 48000

    weight\_bits: 8

    activation\_bits: 8

  }

}

#@title To use Tensorboard on Colab, we need to use it through ngrok

!wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip

!unzip -o ngrok-stable-linux-amd64.zip

#the logs that are created while training

LOG\_DIR = "training/"

get\_ipython().system\_raw(

    'tensorboard --logdir {} --host 0.0.0.0 --port 6006 &'

    .format(LOG\_DIR)

)

get\_ipython().system\_raw('./ngrok http 6006 &')

#The link to tensorboard.

#works after the training starts.

!curl -s http://localhost:4040/api/tunnels | python3 -c \

    "import sys, json; print(json.load(sys.stdin)['tunnels'][0]['public\_url'])"

!pip install lvis

#@title  Training...

!python3 object\_detection/model\_main.py \

    --pipeline\_config\_path=/content/drive/MyDrive/Pothole\ Detcection/models/research/object\_detection/samples/configs/ssd\_mobilenet\_v2\_quantized\_300x300\_coco.config \

    --model\_dir=training/

!pip install tensorflow-object-detection-api

from tensorflow.compat.v1 import ConfigProto

from tensorflow.compat.v1 import InteractiveSession

config = ConfigProto()

config.gpu\_options.allow\_growth = True

session = InteractiveSession(config=config)

!pwd

#@title Export the trained model. { form-width: "200px" }

output\_directory = './fine\_tuned\_model'

lst = os.listdir('training')

#lst = [l for l in lst if 'model.ckpt-' in l and '.meta' in l]

lst = [l for l in lst if 'model.ckpt-' in l]

print(lst)

steps=np.array([int(re.findall('\d+', l)[0]) for l in lst])

last\_model = lst[steps.argmax()].replace('.meta', '')

last\_model\_path = os.path.join('training', last\_model)

!python /content/drive/'My Drive'/Pothole\ Detcection/models/research/object\_detection/export\_inference\_graph.py \

    --input\_type=image\_tensor \

    --pipeline\_config\_path=/content/drive/My\ Drive/Pothole\ Detcection/models/research/object\_detection/samples/configs/ssd\_mobilenet\_v2\_coco.config \

    --output\_directory={output\_directory} \

    --trained\_checkpoint\_prefix={last\_model\_path}

#@title downloading .pb file(Contains the model)

#downloads the frozen model that is needed for inference

# output\_directory = 'fine\_tuned\_model' dir specified above.

output\_directory='/content/drive/MyDrive/Pothole Detcection/models/research/fine\_tuned\_model'

files.download(output\_directory + '/frozen\_inference\_graph.pb')

#@title downloading .pbtxt(contain Labels)

#downlaod the label map

# we specified 'data\_base\_url' above. It directs to

# 'object\_detection/data/' folder.

files.download(data\_base\_url + '/label\_map.pbtxt')

%cd /content/drive/MyDrive/Pothole\ Detcection/models/research/

from PIL import Image

from numpy import asarray

from google.colab.patches import cv2\_imshow

import numpy as np

import os

import tensorflow as tf

import cv2

from object\_detection.utils import label\_map\_util

from object\_detection.utils import visualization\_utils as vis\_util

# path to the frozen graph:

PATH\_TO\_FROZEN\_GRAPH = '/content/drive/MyDrive/Pothole Detcection/models/research/fine\_tuned\_model/frozen\_inference\_graph.pb'

# path to the label map

PATH\_TO\_LABEL\_MAP = '/content/drive/MyDrive/Pothole Detcection/models/research/fine\_tuned\_model/label\_map.pbtxt'

# number of classes

NUM\_CLASSES = 1

#cap = cv2.VideoCapture(0)

#reads the frozen graph

detection\_graph = tf.Graph()

with detection\_graph.as\_default():

    od\_graph\_def = tf.GraphDef()

    with tf.gfile.GFile(PATH\_TO\_FROZEN\_GRAPH, 'rb') as fid:

        serialized\_graph = fid.read()

        od\_graph\_def.ParseFromString(serialized\_graph)

        tf.import\_graph\_def(od\_graph\_def, name='')

label\_map = label\_map\_util.load\_labelmap(PATH\_TO\_LABEL\_MAP)

categories = label\_map\_util.convert\_label\_map\_to\_categories(label\_map, max\_num\_classes=NUM\_CLASSES, use\_display\_name=True)

category\_index = label\_map\_util.create\_category\_index(categories)

# Detection

with detection\_graph.as\_default():

    with tf.Session(graph=detection\_graph) as sess:

          # Read frame from camera

          # load the image and convert into

        # numpy array

        img = Image.open('/content/drive/MyDrive/Pothole Detcection/3.jpg')

        # asarray() class is used to convert

        # PIL images into NumPy arrays

        image\_np = np.array(img)

        #image\_np = "/content/drive/MyDrive/IMG\_20210113\_170959.jpg"

        # Expand dimensions since the model expects images to have shape: [1, None, None, 3]

        image\_np\_expanded = np.expand\_dims(image\_np, axis=0)

        # Extract image tensor

        image\_tensor = detection\_graph.get\_tensor\_by\_name('image\_tensor:0')

        # Extract detection boxes

        boxes = detection\_graph.get\_tensor\_by\_name('detection\_boxes:0')

        # Extract detection scores

        scores = detection\_graph.get\_tensor\_by\_name('detection\_scores:0')

        # Extract detection classes

        classes = detection\_graph.get\_tensor\_by\_name('detection\_classes:0')

        # Extract number of detections

        num\_detections = detection\_graph.get\_tensor\_by\_name(

            'num\_detections:0')

        # Actual detection.

        (boxes, scores, classes, num\_detections) = sess.run(

            [boxes, scores, classes, num\_detections],

            feed\_dict={image\_tensor: image\_np\_expanded})

        #print(classes)

        #mask=classes==1

        #scores=scores[mask]

        #print(scores)

        #boxes=boxes[mask]

        #print(boxes)

        # Visualization of the results of a detection.

        vis\_util.visualize\_boxes\_and\_labels\_on\_image\_array(

                image\_np,

                np.squeeze(boxes),

                np.squeeze(classes).astype(np.int32),

                np.squeeze(scores),

                category\_index,

                use\_normalized\_coordinates=True,

                line\_thickness=8,min\_score\_thresh=.90

                )

        # # get all boxes from an array

        # scores=np.squeeze(scores)

        # boxes=np.squeeze(boxes)

        # classes=np.squeeze(classes)

        # max\_boxes\_to\_draw = boxes.shape[0]

        # # this is set as a default but feel free to adjust it to your needs

        # min\_score\_thresh=.5

        # # iterate over all objects found

        # for i in range(min(max\_boxes\_to\_draw, boxes.shape[0])):

        #     #

        #       if scores is None or scores[i] > min\_score\_thresh:

        #         # boxes[i] is the box which will be drawn

        #         class\_name = category\_index[classes[i]]

        #         print ("This box is gonna get used", boxes[i],classes[i])

        cv2\_imshow(cv2.resize(image\_np,(1000, 500)))

import numpy as np

from imutils.video import VideoStream

from imutils.video import FPS

import imutils

import time

import os

import tensorflow as tf

import cv2

from object\_detection.utils import label\_map\_util

from object\_detection.utils import visualization\_utils as vis\_util

# path to the frozen graph:

PATH\_TO\_FROZEN\_GRAPH = '/content/drive/MyDrive/Pothole Detcection/models/research/fine\_tuned\_model/frozen\_inference\_graph.pb'

# path to the label map

PATH\_TO\_LABEL\_MAP = '/content/drive/MyDrive/Pothole Detcection/models/research/fine\_tuned\_model/label\_map.pbtxt'

# number of classes

NUM\_CLASSES = 1

cap = cv2.VideoCapture('/content/drive/MyDrive/Pothole Detcection/test2.mp4')

fps = FPS().start()

#reads the frozen graph

detection\_graph = tf.Graph()

with detection\_graph.as\_default():

    od\_graph\_def = tf.GraphDef()

    with tf.gfile.GFile(PATH\_TO\_FROZEN\_GRAPH, 'rb') as fid:

        serialized\_graph = fid.read()

        od\_graph\_def.ParseFromString(serialized\_graph)

        tf.import\_graph\_def(od\_graph\_def, name='')

label\_map = label\_map\_util.load\_labelmap(PATH\_TO\_LABEL\_MAP)

categories = label\_map\_util.convert\_label\_map\_to\_categories(label\_map, max\_num\_classes=NUM\_CLASSES, use\_display\_name=True)

category\_index = label\_map\_util.create\_category\_index(categories)

# Detection

with detection\_graph.as\_default():

    with tf.Session(graph=detection\_graph) as sess:

        while True:

            # Read frame from camera

            ret, image\_np = cap.read()

            # Expand dimensions since the model expects images to have shape: [1, None, None, 3]

            image\_np\_expanded = np.expand\_dims(image\_np, axis=0)

            # Extract image tensor

            image\_tensor = detection\_graph.get\_tensor\_by\_name('image\_tensor:0')

            # Extract detection boxes

            boxes = detection\_graph.get\_tensor\_by\_name('detection\_boxes:0')

            # Extract detection scores

            scores = detection\_graph.get\_tensor\_by\_name('detection\_scores:0')

            # Extract detection classes

            classes = detection\_graph.get\_tensor\_by\_name('detection\_classes:0')

            # Extract number of detections

            num\_detections = detection\_graph.get\_tensor\_by\_name(

                'num\_detections:0')

            # Actual detection.

            (boxes, scores, classes, num\_detections) = sess.run(

                [boxes, scores, classes, num\_detections],

                feed\_dict={image\_tensor: image\_np\_expanded})

            # Visualization of the results of a detection.

            vis\_util.visualize\_boxes\_and\_labels\_on\_image\_array(

                image\_np,

                np.squeeze(boxes),

                np.squeeze(classes).astype(np.int32),

                np.squeeze(scores),

                category\_index,

                use\_normalized\_coordinates=True,

                line\_thickness=3,

                min\_score\_thresh=.80

                )

    #         cv2\_imshow('Frame', image\_np)  # Displaying the frame

    #         key = cv2.waitKey(1) & 0xFF

    #         key = cv2.waitKey(0)

    #         while key not in [ord('q'), ord('k')]:

    #             key = cv2.waitKey(0)

    #         # Quit when 'q' is pressed

    #         if key == ord('q'):

    #             break

    # # update the FPS counter

    #         fps.update()

    #         fps.stop()

    #         cv2.destroyAllWindows()

    #         cap.release()

        # Display output

            cv2\_imshow(cv2.resize(image\_np, (1000,500)))

            if cv2.waitKey(25) & 0xFF == ord('q'):

                cv2.destroyAllWindows()

                break

**COMPONENT DESCRIPTION**

**Buzzer:**

Specifications: Table I shows the hardware specifications of a buzzer.

|  |  |
| --- | --- |
| Rated Voltage | 5 V |
| Operating Voltage | 4~8 V |
| Max Rated Current | ≤32 mA |
| Min. Sound Output at 10cm | 85 dB |
| Resonant Frequency | 2300 ±300 Hz |

*Table I: Buzzer Specifications*

**GSM SIM900A Module:**

Pin description: Table II presents the description of each pin in a SIM900A GSM module

|  |  |  |
| --- | --- | --- |
| **Pin Number** | **Pin Name** | **Description** |
| 1 | PWRKEY | Voltage input for PWRKEY.  PWRKEY should be pulled low to power on or  Off the system. The user should keep pressing the key for a short time  when power on or power off the system because the system need margin  time in order to assert the software. |
| 2 | PWRKEY\_OUT | Connecting PWRKEY and PWRKEY\_OUT for a short time then release  Also can power on or power off the module. |
| 3 | DTR | Data terminal Ready [Serial port] |
| 4 | RI | Ring indicator [Serial port] |
| 5 | DCD | Data carry detect [Serial port] |
| 6 | DSR | Data Set Ready [Serial port] |
| 7 | CTS | Clear to send [Serial port] |
| 8 | RTS | Request to send [Serial port] |
| 9 | TXD | Transmit data [Serial port] |
| 10 | RXD | Receive data [Serial port] |
| 11 | DISP \_CLK | Clock for display [Display interface] |
| 12 | DISP\_DATA | Display data output [Display interface] |
| 13 | DISP \_D/C | Display data or command select [Display interface] |
| 14 | DISP \_CS | Display Enable [Display interface] |
| 15 | VDD\_EXT | 2.8V output power supply |
| 16 | NRESET | External reset input |
| 17,18,29,39,45,  46,53,54,58,59,  61,62,63,64,65 | GND | Ground |
| 19 | MIC\_P | Microphone Positive |
| 20 | MIC\_N | Microphone Negative |
| 21 | SPK\_P | Speaker Positive |
| 22 | SPK\_N | Speaker Negative |
| 23 | LINEIN\_R | Right Channel input [External line inputs are available to directly mix or multiplex externally generated analog signals such as polyphonic tones  from an external melody IC or music generated by an FM tuner IC  or module.] |
| 24 | LINEIN\_L | Left Channel Input |
| 25 | ADC | General purpose analog to digital converter. |
| 26 | VRTC | Current input for RTC when the battery is not supplied for the system.  Current output for backup battery when the main battery is present and  the backup battery is in low voltage state. |
| 27 | DBG\_TXD | Transmit pin [Serial interface for debugging and firmware upgrade] |
| 28 | DBG\_RXD | Receive pin [Serial interface for debugging and firmware upgrade] |
| 30 | SIM\_VDD | Voltage supply for SIM card |
| 31 | SIM\_DATA | SIM data output |
| 32 | SIM\_CLK | SIM clock |
| 33 | SIM\_RST | SIM reset |
| 34 | SIM\_PRESENCE | SIM detect |
| 35 | PWM1 | PWM Output |
| 36 | PWM2 | PWM Output |
| 37 | SDA | Serial Data [I2C] |
| 38 | SCL | Serial Clock [I2C] |
| 40,41,42,43,44  &  47,48,49,50,51 | KBR0 to KBR4  &  KBC4 to KBC0 | Keypad interface [ROWS & COLUMNS] |
| 52 | NETLIGHT | Indicate net status |
| 55,56,57 | VBAT | Three VBAT pins are dedicated to connect the supply voltage. The power  supply of SIM900A has to be a single voltage source of VBAT= 3.4V to 4.5V.  It must be able to provide sufficient current in a transmit burst which typically  rises to 2A. |
| 60 | RF\_ANT | Antenna connection |
| 66 | STATUS | Indicate working status |
| 67 | GPIO 11 | General Purpose Input/output |
| 68 | GPIO 12 | General Purpose Input/output |

*Table II: Pin Description of GSM SIM900A*

Specifications:

1. Single supply voltage: 3.4V – 4.5V
2. Power saving mode: Typical power consumption in SLEEP mode is 1.5mA
3. Frequency bands: SIM900A Dual-band: EGSM900, DCS1800. The SIM900A can search the two frequency bands automatically. The frequency bands also can be set by AT command.
4. GSM class: Small MS
5. GPRS connectivity: GPRS multi-slot class 10 (default) , GPRS multi-slot class 8 (option)
6. Transmitting power: Class 4 (2W) at EGSM 900, Class 1 (1W) at DCS 1800
7. Operating Temperature: -30ºC to +80ºC
8. Storage Temperature: -5ºC to +90ºC
9. DATA GPRS: download transfer max is 85.6KBps, Upload transfer max 42.8KBps

**MPU6050 (Gyroscope Sensor):**

#### Pin Description: The pin description of MPU6050 is given below in Table III

|  |  |  |
| --- | --- | --- |
| **Pin Number** | **Pin Name** | **Description** |
| 1 | Vcc | Provides power for the module, can be +3V to +5V.  Typically, +5V is used |
| 2 | Ground | Connected to Ground of system |
| 3 | Serial Clock (SCL) | Used for providing clock pulse for I2C Communication |
| 4 | Serial Data (SDA) | Used for transferring Data through I2C communication |
| 5 | Auxiliary Serial Data (XDA) | Can be used to interface other I2C modules with  MPU6050. It is optional |
| 6 | Auxiliary Serial Clock (XCL) | Can be used to interface other I2C modules with  MPU6050. It is optional |
| 7 | AD0 | If more than one MPU6050 is used a single MCU,  then this pin can be used to vary the address |
| 8 | Interrupt (INT) | Interrupt pin to indicate that data is available for  MCU to read. |

#### *Table III: MPU6050 pin description*

#### Specifications:

* MEMS 3-aixs accelerometer and 3-axis gyroscope values combined
* Power Supply: 3-5V
* Communication: I2C protocol
* Built-in 16-bit ADC provides high accuracy
* Built-in DMP provides high computational power
* Can be used to interface with other IIC devices like magnetometer
* Configurable IIC Address
* In-built Temperature sensor

**BIODATA**

|  |  |  |
| --- | --- | --- |
| **S. No** | **Team Member Details** | **Contribution** |
| 1 | Sharath Akash P - 18BEC1014  [sharathakash.p2018@vitstudent.ac.in](mailto:sharathakash.p2018@vitstudent.ac.in)  +91 8754973579 | Collected images and videos of potholes with different sizes and shapes as Training dataset for the machine learning algorithm used in pothole detection, Improving the performance of the model while dealing with potholes in real scenarios. Procured GSM module and helped with circuit assembly. Prepared Literature Survey for 1 paper. |
| 2 | Amshuman G - 18BEC1075  [amshuman.gopal2018@vitstudent.ac.in](mailto:amshuman.gopal2018@vitstudent.ac.in)  +91 8925622099 | Finding and testing various libraries (from sources such as GitHub and stack overflow) for components which were not integrated in Arduino (MPU6050, GSM900A). Tried out various implementations using different libraries on the code along with another teammate (B Visweshwaran) to see and decide which one was easier to implement. Prepared Literature Survey for 2 papers |
| 3 | Arjun RK – 18BEC1120  [arjun.rk2018@vitstudent.ac.in](mailto:arjun.rk2018@vitstudent.ac.in)  +91 9003052629 | Was involved in application design along with Kathir and Prasanna Kumar. Implemented the machine learning python code in Java. Worked on Camera Activity, Detector Activity and Camera connection. Designed overall framework of the pothole detection application. Prepared Literature Survey for 1 paper. |
| 4 | J Kiron – 18BEC1150  [j.kiron2018@vitstudent.ac.in](mailto:j.kiron2018@vitstudent.ac.in)  +917397264063 | Read through Machine Learning courses and articles on how to train a dataset to create a model. Collaborated with Gautam to create the Pothole Detection system program. Did the preprocessing of images and labels, and used TensorFlow and Mobilenet V2 300×300 model to train the dataset. Prepared Tinkercad circuit and code for bike fall detection's software simulation along with Visweshwaran. Prepared Literature Survey for 1 paper. |
| 5 | B Visweshwaran – 18BEC1152  [b.visweshwaran2018@vitstudent.ac.in](mailto:b.visweshwaran2018@vitstudent.ac.in)  +91 9092330551 | Prepared Tinkercad circuit and code for bike fall detection's software simulation along with Kiron. Collected Hardware components (MPU6050, Arduino, Breadboard and soldering kit) and assembled the circuit (soldering and connections) along with one other member (Sharath Akaash). Software implementation was done on Arduino ide by integrating libraries required for MPU6050, GSM900A along with one other member (G Amshuman). Prepared Literature Survey for 1 paper. |
| 6 | R Kathir – 18BEC1226  [r.kathir2018@vitstudent.ac.in](mailto:r.kathir2018@vitstudent.ac.in)  +91 9789062337 | Designed the UI/UX for the application. Worked on overlay view, Recognition score view and Results view. Designed the logo and along with sound for the application. Performed Critical analysis of the survey to draw conclusions. Edited and compiled the demonstration video. Prepared Literature Survey for 2 papers |
| 7 | E Gautamgovan – 18BEC1295  [gautamgovan.e2018@vitstudent.ac.in](mailto:gautamgovan.e2018@vitstudent.ac.in)  +91 9884371581 | Helped in creating the survey to find the problem that needs to be solved urgently. Collaborated with Kiron in the Pothole Detection program to create a proper visual experience for the user. Prepared Literature Survey for 2 papers |
| 8 | S Prasanna Kumar – 18BEC1313  [prasannakumar.s2018@vitstudent.ac.in](mailto:prasannakumar.s2018@vitstudent.ac.in)  +91 8939149722 | Designed the work environment for the application. Worked on the bordering, Size and images for the application. Also worked on the logger for message output in the app. Prepared Literature Survey for 2 papers. |