**Implementation of Vehicle and Road diagnostics**

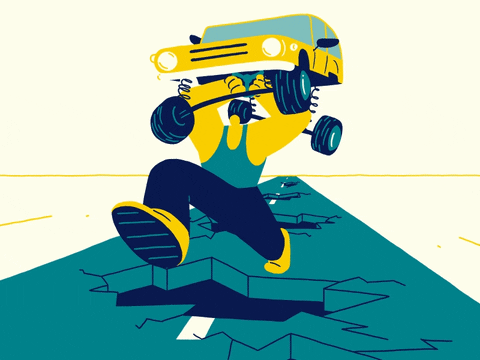
(Version 1)

Road transport has grown in importance over the years due to an increase in the population as well as **need for trade and communication**. The need to maintain roads in a good condition has been growing ever since the road network is expanding. Maintaining good roads has been a **tedious task** and has been problematic to almost every country today. There are a number of adverse effects caused by such bad roads some of which include accidents and damage to vehicles.

Classic approaches to this problem include **monitoring road conditions periodically** and provide solutions if required. Vehicle wear and tear are not only caused by road conditions, but also the **drive patterns of the driver**.

The idea is to combine both problems to provide **one effective solution**. The approach predominantly focuses on **detecting road conditions** and vehicle malfunctions and providing solutions to such issues. Detecting road conditions include detection of anomalies such as potholes and rough patches. Vehicle condition monitoring can be done by analysing different **vehicle parameters** such as vehicle speed, engine speed, throttle position, accelerator position etc.

OBD data can be accessed through smartphones by installing applications such as Torque Pro. This data combined with video or image footage from phone cameras render a way to reconfirm the results obtained from the accelerometer readings.



**Python Implementation**

The libraries needed to execute each function is listed below.

import gmplot

import io

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import peakutils

from peakutils.plot import plot as pplot

import scipy

from math import sqrt

from scipy.integrate import quad

from numpy import trapz

from scipy.integrate import simps

import matplotlib.image as mpimg

from mpl\_toolkits.mplot3d import Axes3D

import plotly.plotly as py

import plotly.graph\_objs as go

import seaborn as sns

The csv file generated by the app in imported to the python code. Individual columns are then extracted from the csv file for feature extraction.

data = pd.read\_csv('videofinal.csv') #Target Variable

data.head()

time = data['Time'] #Extraction of columns

Accx = data['Accx']

Accy = data['Accy']

Accz = data['Accz']

Lat = data['latitude']

Longi = data['longitude']

To make things simpler, functions to plot axes data are created in prior so that it can be used in a convenient way at any point in the code.

def Plot1(x,y,a,b): #Function to detect peaks

plt.plot(x,y,'b')

indexes = peakutils.indexes(y, thres=0.9, min\_dist=10)

print(indexes)

print(x[indexes], y[indexes])

plt.figure()

pplot(x,y,indexes)

plt.title('Peaks')

plt.xlabel(a)

plt.ylabel(b)

def Plot2(x\_list,y\_list,x\_label,y\_label,plot\_title):

plt.plot(x\_list,y\_list) #Function to plot lists

plt.xlabel(x\_label)

plt.ylabel(y\_label)

plt.title(plot\_title)

def mean\_rolling(List\_name): #Function to find rolling mean

chunks = [List\_name[i:i+2] for i in range(0,len(List\_name),2)]

d = []

for i in chunks:

s = 0

for j in i:

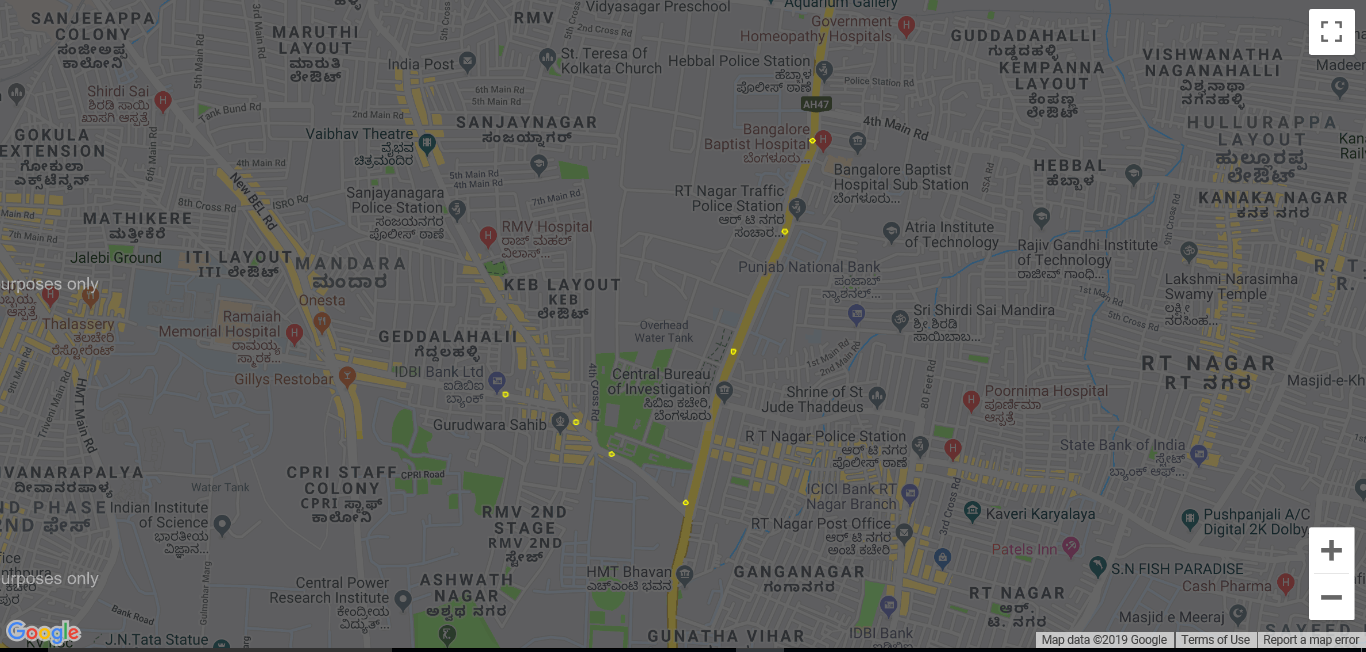
s += j/2

d.append(s)

print(d)

return

**Pothole detection:**

****

The code implemented for detecting pothole is mainly based on peak detection algorithms. Accelerometer data is visualized and peaks are selected with a certain threshold (<-0.1). The corresponding coordinates are fetched from the original data and is used to plot the same on the map using a function called gmplot.

gmplot is an inbuilt function on python that allows the user to pass a list of latitudes and longitudes as parameters which gives the above result in the form of a HTML file saved in the same directory.

Code snippet:

**#Pothole detection**

x = np.array(time)

y = np.array(Accy)

indexes = peakutils.indexes(-y, thres=0.9, min\_dist=10)

print(indexes)

print(x[indexes], y[indexes])

plt.figure(figsize=(10,6))

pplot(x, y, indexes)

plt.title('Number of severe-potholes')

plt.xlabel('Time')

plt.ylabel('Acceleration in the y direction')

c=data.Accy<-0.1

data[c]

lati=[13.027313,13.026308,13.025118,13.023361,13.028859,13.033193,13.036492 ]

long=[77.577713,77.580344,77.581661,77.584423,77.586206,77.588136,77.589162 ]

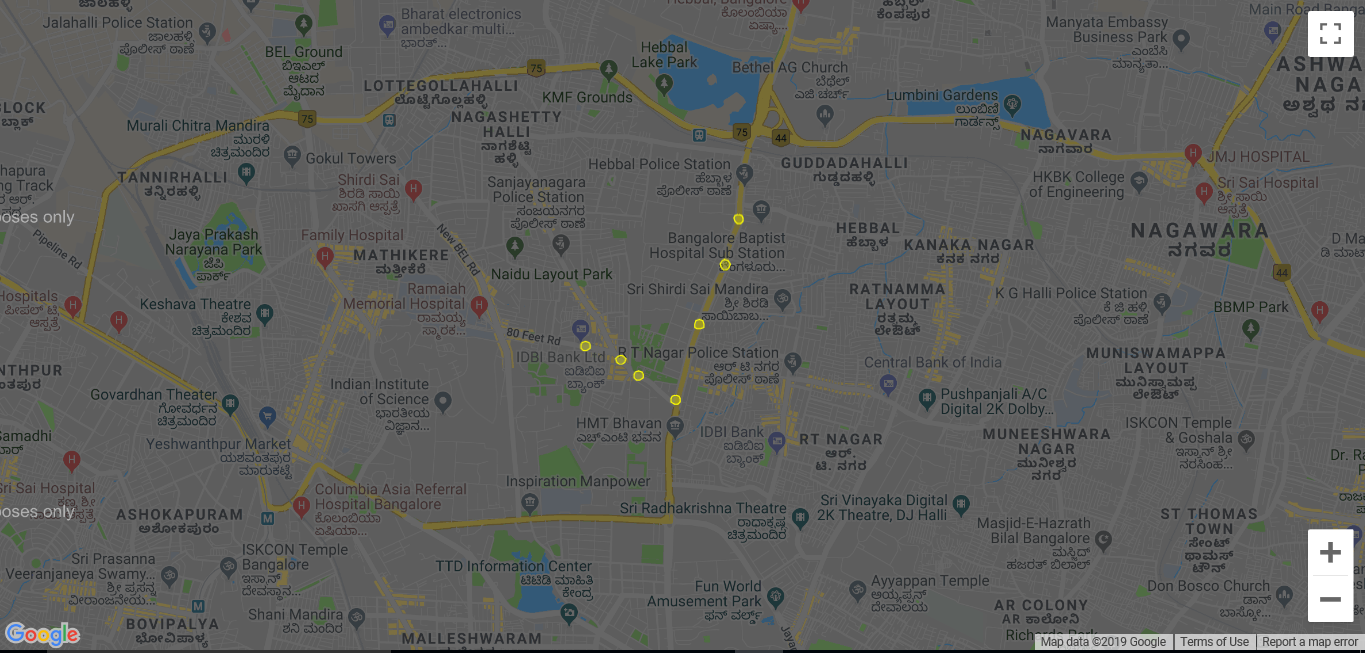
gmap = gmplot.GoogleMapPlotter(12.9716,77.5946, 13)

gmap.scatter(lati, long,

'yellow',size = 10, marker = False)

gmap.draw("Potholes.html")

**Speed-Breaker detection:**

****

The technique used for detecting speed-breakers is very similar to that of potholes. The threshold is set to >0.14 which detects the upper peaks in the accelerometer readings. The same function is used to plot the coordinates onto the map.

Code snippet:

x = np.array(time)

y = np.array(Accy)

indexes = peakutils.indexes(y, thres=0.35, min\_dist=5)

print(indexes)

print(x[indexes], y[indexes])

plt.figure(figsize=(10,6))

pplot(x, y, indexes)

plt.title('Number of speed breakers')

plt.xlabel('Time')

plt.ylabel('Acceleration in the y direction')

c=data.Accy>0.14

data[c]

lati = [13.027806,13.027016,13.025226,13.031858,13.034230,13.036789,13.039624 ]

long = [77.574728,77.579485,77.585040,77.587497,77.588328,77.589188,77.589422]

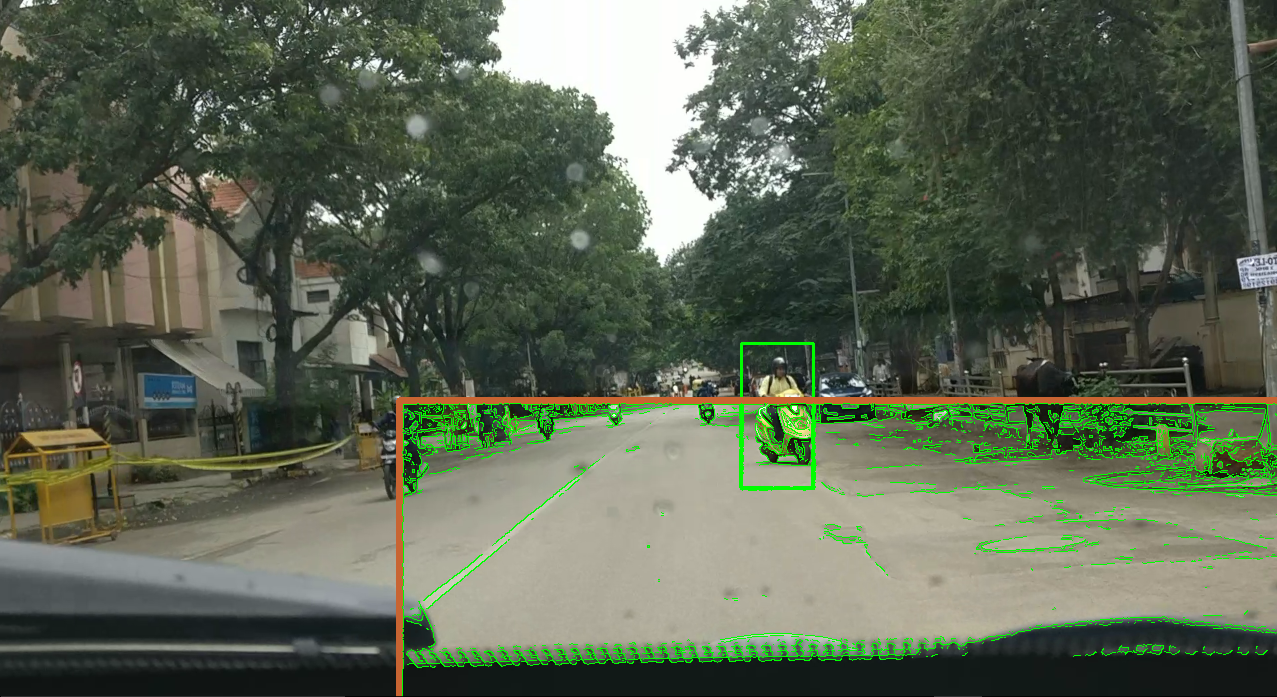
gmap = gmplot.GoogleMapPlotter(12.9716,77.5946, 13)

gmap.scatter(lati, long,

'yellow',size = 40, marker = False)

gmap.draw("Speed\_Breakers.html")

**Edge detection for Pothole and Speed-breaker in the Region of interest:**

****

The function crested in the code extracts a region of interest in order to detect abnormalities on the road surface by using edge detection methods such as Canny edges. It also focuses on creating a function that could detect humans in the frame. The inbuilt function used is called the HOG descriptor. SVM methods are employed to classify the fames into human or non-human.

The video output of the program could help in confirming the pothole or the speed-breaker or any such abnormalities on the road pictographically.

Code snippet:

import cv2

from matplotlib import pyplot as plt

def sketch\_transform(image):

image\_grayscale = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

image\_grayscale\_blurred = cv2.GaussianBlur(image\_grayscale, (7,7), 0)

image\_canny = cv2.Canny(image\_grayscale\_blurred, 10, 80)

\_, mask = image\_canny\_inverted = cv2.threshold(image\_canny, 30, 255, cv2.THRESH\_BINARY\_INV)

return mask

import cv2

import time

import numpy as np

hog = cv2.HOGDescriptor()

hog.setSVMDetector(cv2.HOGDescriptor\_getDefaultPeopleDetector())

cap = cv2.VideoCapture('Carvideo.MP4')

while True:

r, frame = cap.read()

if r:

start\_time = time.time()

frame = cv2.flip(frame,0)

frame = cv2.resize(frame,(1280, 720)) # Downscale to improve frame rate

gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_RGB2GRAY) # HOG needs a grayscale image

rects, weights = hog.detectMultiScale(gray\_frame)

**# Measure elapsed time for detections**

end\_time = time.time()

for i, (x, y, w, h) in enumerate(rects):

if weights[i] < 0.1:

continue

cv2.rectangle(frame, (x,y), (x+w,y+h),(0,255,0),2)

cv2.imshow("preview", frame)

if cv2.waitKey(1) & 0xFF == ord('q'): # press q to quit

break

upper\_left = (400,400)

bottom\_right = (1500, 1500)

bottom\_left = (1500, 1500)

upper\_right = (400,400)

**#Rectangle marker**

r = cv2.rectangle(frame, upper\_left, bottom\_right, (50, 100, 200), 5)

rect\_img = frame[upper\_left[0] : bottom\_left[0], upper\_left[0] : bottom\_right[0]]

sketcher\_rect = rect\_img

sketcher\_rect = sketch\_transform(sketcher\_rect)

#Conversion for 3 channels to put back on original image (streaming)

sketcher\_rect\_rgb = cv2.cvtColor(sketcher\_rect, cv2.COLOR\_GRAY2RGB)

#Replacing the sketched image on Region of Interest

#frame[upper\_left[0] : bottom\_right[0], upper\_left[1] : bottom\_right[1]] = sketcher\_rect\_rgb

gray = cv2.cvtColor(rect\_img, cv2.COLOR\_BGR2GRAY)

blur = cv2.bilateralFilter(gray,1,1000,1000)

blur = cv2.Canny(rect\_img,80,90,1)

ret, thresh\_img = cv2.threshold(blur,91,255,cv2.THRESH\_OTSU)

contours = cv2.findContours(thresh\_img,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)[0]

for c in contours:

cv2.drawContours(rect\_img, [c], -1, (0,255,0), 1)

cv2.imshow("preview", frame)

if cv2.waitKey(1) & 0xFF == ord('q'): # press q to quit

break

# When everything done, release the capture

cap.release()

cv2.destroyAllWindows()

The results shown above can be improved with the help of ML tools which will be done in the phase II of the implementation.