**Autonomous Vehicles Semester Project Report:**

Autonomous Golf Cart with Pedestrian Detection

Contributors:

[Garrett Barton](https://github.com/GarrettBarton/Autonomous-GolfCart), Michael Belanger, [Samuel Maloney](https://github.com/samdude28/Autonomous-GolfCart), and Vishaldeep Singh

Professor: Dr. Supratik Mukhopadhyay

Course: ENGR 4200 Autonomous Vehicles

Submitted on: 20180504 (20 pages)

***Abstract:*** *The task was to design and create a system to be attached to a small, gas powered golf cart, and allow the golf cart to be able to drive autonomously around the university campus. The system was to be able to detect pedestrians, stop safely, and navigate autonomously end-to-end along know road ways given map. The team was able to achieve current location, path planning, and video streaming. However, the project did not obtain the detection of* *pedestrians, or, consequently, move to the stage of implementation on the golf cart. Code has been appended to this report.*

**Objective**

The objective of this project was to design and create a system to allow a golf cart autonomously drive around the university campus. The main focus of this project was the software and electronics surrounding autonomous driving and the design process. The main stages of the project were: idea proposals, coordinate assertion, video capture, image processing, path planning, and physical implementation.

There were several specific requirements that students needed to meet in order to have a successful project, in terms of the scope of this project. Those included being able to: capture real-time location information, detect pedestrians and stop to prevent hitting pedestrians, and manipulate the drive systems of the cart to be able to control the cart while driving. The stressed objective by the professor was that students gain an in-depth *exposure* to autonomous robotic systems. The evaluation of the project was to be weighted towards learning not milestones or a fully working project. The team was given a semester of lab time (3 hours/week) to independently work towards this goal, totaling to around 39-42 hours of in-class time and about 30 hours of out-of-class time.

**Summary of Procedure**

The team began the process by conceptualizing an overarching design for the given task. The team initially thought to use an old Android phone (HTC Desire 610) for its camera, internal GPS unit, and its processor. However, as discussed further on, this approach proved to be unviable.

The team then decided to switch to a system powered by the Raspberry Pi and supported devices (GPS module and an 8 megapixel camera). There were Raspberry Pi3’s in the lab, and so a camera, camera cable, and a GPS unit was ordered form Adafruit. Once the parts were received (about week later) the team split into two groups. One focusing on pulling on real-time location information with the GPS module and the other researching the pedestrian detection algorithm. The group finished the real-time location information before the pedestrian detection algorithm, as is obvious due to the later never getting completed. The team came together to research solutions and tutorials on anything related to pedestrian detection or image processing.

The pedestrian detection required a live video stream, which the two groups converged on and eventually got running properly. The live video stream was time intensive, but due to the configuration of the Raspberry Pi only one or two students could work on code at a time. This was due to the students remote desktop-ing into the Raspberry Pi because the university’s Wi-Fi caused so many issues that one of the students setup the Raspberry Pi their apartment and the team remote desktop-ed in. However, there was not enough space to allow any more than two students working on the Raspberry Pi. The team split again into one group continuing work on the pedestrian detection and one group moving on to path planning.

As for path planning, the team decided to utilize the simple algorithm, A\*. This algorithm’s basic idea is to take the sum of the distance from the source to the current point and the remaining distance from the current point to the destination. To obtain the distance from the current point to the destination, with diagonal distance heuristics. This set of heuristics was chosen because the matrix format would allow the ‘path’ to ‘move’ in eight directions (i.e. up, down, left, right, and the diagonals), which fit with the idea of using a matrix map. By matrix map, it is meant: a matrix of ones and zeros, with the ones denoting where the roads are generally (see the subsystem section).

After obtaining current location info via some GPS, the goal shifted to being able to detect pedestrians. This was researched and there were several helpful resources that were found online. They were: a blog, *PYImageSearch*; two GitHub accounts, *galenballew* and *HackerHouseYT*; and the OpenCV open source library. All of which are run by experienced coders that specialize in computer vision, and provide detailed discussions of the implementation and application of a pedestrian detection system and/or specific functions that would prove instrumental in the area of research. Their work was instrumental and instructional in working on this component of the project.

The next stage would have been was safety testing, which needed to be done to ensure that the system could safely work if implemented on a vehicle. The team had to show that the system would be able to: detect any pedestrians, to send a signal to stop the vehicle, and navigate through road ways. Once the team could demonstrate these abilities, the team would be allowed to implement this system onto the golf cart to be used. The electronics were to be mounted appropriately and a system to manipulate the steering wheel and peddles was to be created. However, as stated in the previously this stage was not reached due to the project not being complete enough to begin safety testing.

**Subsystem Details**

**Current Location**

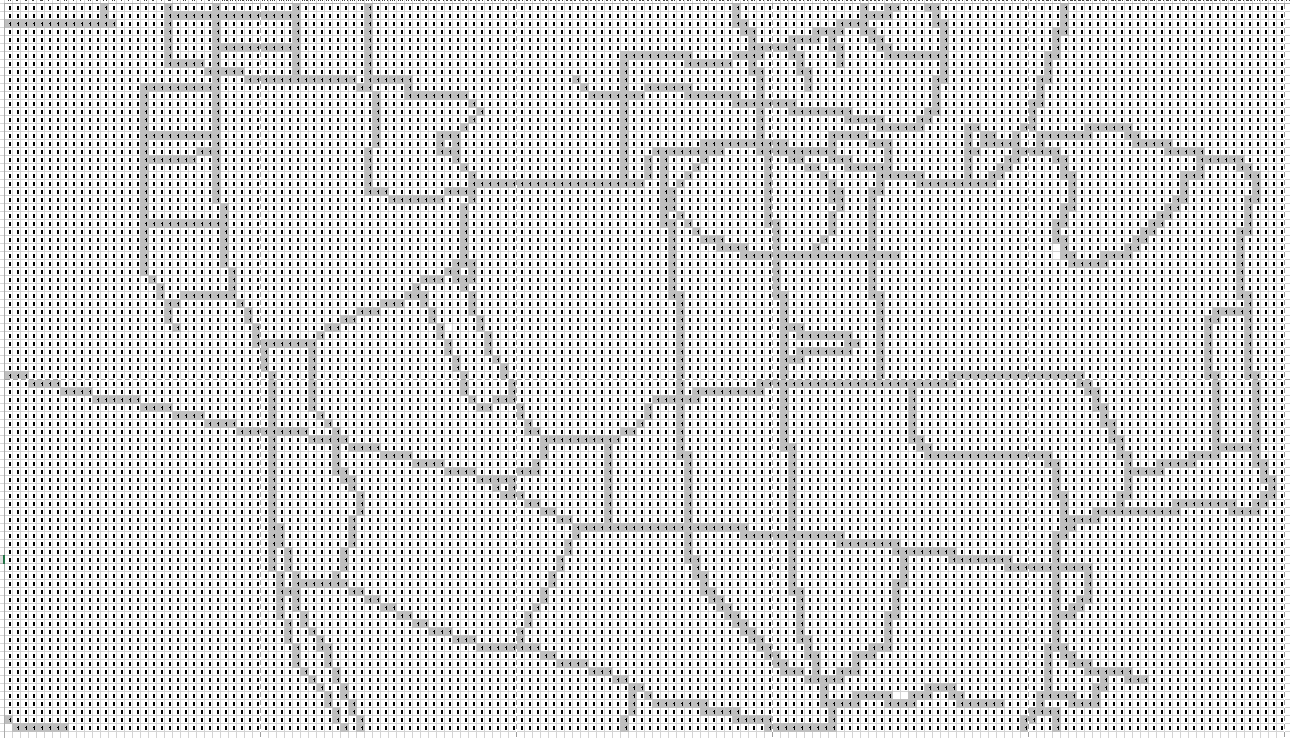
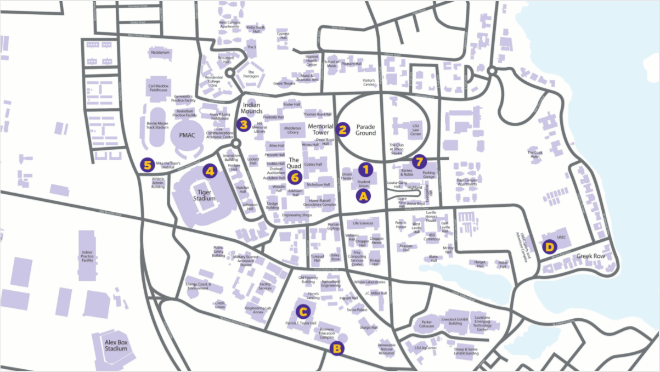
The Adafruit Ultimate GPS Breakout Version 3, with 66 channels and updating at 10 Hz, was chosen due to its compatibility with the Raspberry Pi. The device came factory ready to be able to pull current coordinates via any GPS satellite available. The code that the factory instructed to install, pulled data and comma delineated the data into an easily accessible format.

**Navigation Algorithm**

The A\* (pronounced “a star”) algorithm was chosen to determine the shortest paths that the golf cart would take. This algorithm’s basic idea is to take the sum of the distance from the source to point A and the remaining distance from point A to the destination. Here point A is just the end of the path so far during the running of the algorithm. To obtain the distance from the point A to the destination, diagonal distance heuristics were used. This set of heuristics was chosen because the matrix format would allow the ‘path’ to ‘move’ in eight directions (i.e. up, down, left, right, and the diagonals). Diagonal heuristics means that the cells horizontal, vertical, and the *diagonal* to the cell being considered are considered for the next node in the path. These heuristics take the maximum of the absolute value of the x and y coordinates from point A to the destination. An article on the A\* algorithm by *GeeksforGeeks* was tremendously helpful in understanding and implementing the A\* algorithm in the project. An A\* code written in Python would have been taken and modified to work with the existing structure of this project. However, the team did not get far as fully create the path planning subsystem.

**Matrix Map**

The team created an Excel spreadsheet to convert a map of the campus into a matrix, which would then be formatted/exported to a format usable in the python code. Below is a screen capture of the final map in Excel with the roads, marked with ones, highlighted in grey. Obviously, the matrix created by the students has visible quantization error due to the limitations of the scale chosen. The map from LSU was put into Paint (desktop app for PCs) and the grid-view function was utilized to break up the real map into a *matrix*. The quantization rate was chosen to be of the square grid size when the Paint program was at 100% zoom. This meant that the map was broken up into 160x90 squares (14.4k bits). The Paint program had broken the image into 1600x900 squares (1.44 million bits), but it was decided that, that level of quantization and resolution was unnecessary for the requirements of this project.



***Left****: map of LSU campus,* ***Right****: Matrix map as seen in Excel with road marked in grey*

**Pedestrian Detection**

The pedestrian detection was implemented utilizing the 8 megapixel, Raspberry Pi Camera Board V2 to provide the video feed needed to run the pedestrian detection algorithms. As with the GPS module, the camera was designed to work with the Raspberry Pi and so integration was seamless. The camera is itself a plug-and-play system, and the image capturing was very simple python code. The difficulty came when images needed to be sampled from the video to be processed to detect pedestrians and the actual processing of the images. There were several complications with this subsystem. The first complication that the team ran into was getting a single image from the video feed to sample. The second complication that the team found was the time requirements of taking a sample image, processing it, and then returning the necessary commands to control the actuators. To optimize this process the team tested the run time by adjusting the resolution and the sample rate. These two issues were never resolved due to the image processing capabilities never being solved. The team was unable to get the pedestrian detection algorithm to be able to detect pedestrians after further research. After many hours, articles, and tutorials the team was unable to get the algorithm to detect pedestrians that were in front of the camera. This is where *HackerHouseYT*, a GitHub/YouTube account, became a great resource, with their great tutorial on detecting people with cameras and getting a live stream. All though, their work was primarily for a security camera, the principles still applied.

**Steering Control**

This stage of the project was not reached within the time limits of the project and the semester. The steering control of the golf cart was to be done with an actuators in a configuration similar to those in the DARPA competitions. This sort of physical control system would have probably required an external power source to be something in the range of a car battery, or a second golf-cart-sized battery. However, the design ideas asserted are only initial ideas, as there was no actual implementation of this stage.

**Issues Dealt with**

The Android phone using Google Map services

As mentioned previously, there was considerable issues getting the Android phone to work with the program that was written to pull coordinates of the golf cart. This was attempted using an Android phone, which would have utilized Google Maps services API and the internal GPS capacities of the phone. The team used the virtual environment to run and test the application. However, the virtual environment was on the Google headquarters servers in California. The team got these coordinates several times, but eventually the coordinates landed on the Prime Meridian line and the equator. Even when the application was run on the Android phone, the location pulled would be just off the west coast of Africa. This inability to grab accurate information reliably prompted the switch to a GPS module that was made for use with the Raspberry Pi.

Computer Vision

There was considerable obstacles in getting the Raspberry Pi to be able to stream live video and to be able to use that captured video to detect pedestrians. The team did, however get the Raspberry Pi to live stream a video (both to a regular pop-up window and to a webpage). There are a multitude of tutorials on how to get the Raspberry Pi to stream live video feeds (using Python), but it was not as easy as it was made out to be. The tutorials always seemed to not work, or cause errors. The team eventually debugged and patched together enough code to run.

University’s Wireless Internet

The Raspberry Pi needed a connection to the internet in order to download all the software needed to do the project. However, the university’s Wi-Fi, eduroam, would not allow the Raspberry Pi to connect. The error seemed to be that the network did not want some irregular device joining. It would sometimes take and hour (out of a three hour lab) just to get connected to the network.

Reinstalling the Operating System on the Raspberry Pi

The operating system had to be reloaded twice on to the Raspberry Pi. Then the first OpenCV version that had been downloaded was for a Raspberry Pi 2, and so the Raspberry Pi had to be wiped and reloaded with the correct version. This process took about 36 hours of active run time to complete.

**Conclusion and Future Work**

Overall, the team failed to deliver on the idea of having a fully autonomous golf cart that would be able to navigate the university campus. However, the team was able to make considerable head way towards that ideal. The team was able to produce current location, live video capture, and path planning. Not to mention that the team learned a great deal on the design and testing process. A special note is required to note the helpfulness of the hobbyists and the volunteers that provided very helpful articles, tutorials, and examples that made the team’s learning much easier.

This project is not complete, and has the potential to have much more done with it. Given time the team could complete the pedestrian detection system, which would pave the way to physical implementation. However, the project does not need to end there. This project could continue to add such features that would bringing the project closer to a human driver like experience (and subsequently to the current research in the field of autonomous vehicles). Features including: lane detection, other vehicle detection, simultaneous localization and mapping elements (to allow for new routes to be made), companying mobile phone applications, dynamic route changing (changing destination during a drive), autonomous refueling procedure, etc. All of these would take much more time and knowledge. While they are achievable, they are not achievable within the time constrains.

**Appendices**

**Appendix A: Resources**

Ballew, Galen. “SDC-Lane-and-Vehicle-Detection-Tracking: OpenCV in Python for lane line and

vehicle detection/tracking in autonomous cars.” galenballew, 24 April 2017,

<https://github.com/galenballew/SDC-Lane-and-Vehicle-Detection-Tracking>.

Belwariar, Rachit. “A\* Search Algorithm.” GeeksforGeeks, 9 Feb. 2018,

[www.geeksforgeeks.org/a-search-algorithm/](http://www.geeksforgeeks.org/a-search-algorithm/).

HackerHouseYT. “HackerHouseYT/Smart-Security-Camera.” GitHub, GitHub, 5 Oct. 2017,

[www.github.com/HackerHouseYT/Smart-Security-Camera](http://www.github.com/HackerHouseYT/Smart-Security-Camera).

Louisiana State University. “CAMPUS MAP.” Louisiana State University Visitor's Guide, 22 Dec.

2016, <https://louisianastateuniversity.myuvn.com/campus-map/>.

OpenCV. “OpenCV Library.” OpenCV Library, 2018, [www.opencv.org/](http://www.opencv.org/).

Rosebrock, Adrian. “Pedestrian Detection OpenCV.” PyImageSearch, 7 Sept. 2016,

[www.pyimagesearch.com/2015/11/09/pedestrian-detection-opencv/](http://www.pyimagesearch.com/2015/11/09/pedestrian-detection-opencv/).

**Appendix B: Materials**

***Hardware***

* Raspberry Pi 3 Model B
* Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates - Version 3 (Product ID: 746)
* Raspberry Pi Camera Board v2 - 8 Megapixels (Adafruit Product ID: 3099)
* Flex Cable for Raspberry Pi Camera or Display - 2 meters (Adafruit Product ID: 2144)
* 16 GB SD card (Student owned)
* HDMI cable (Student owned)

***Software***

* Thonny (Python IDE, free to download)
* VNC Viewer (Remote desktop software, free to download)
* Python 2.7.1 (free to download)
* OpenCV 3.4.1 (free to download)

**Appendix C: Code That Was Used (by application)**

This appendix details the code that would be used in the final project.

* Under GPS:
  + The parsing code take the ‘sentences’ form the GPS chip and only takes the sentences marked $GPRMC, which had the data that pertained to gathering location.
  + The GPS chip’s would also determine if the data it gathered was valid or not. If the sentence returned a value of “A” the value was good. If the sentence returned a value of “V” the value was bad. The website in-the-sky.org was used to determine directions of satellites that were available to be used.
  + The longitude and azimuth pulled needed to be converted from a string to a number type, which is why the section titled “Checking Ability to Preform Calculations” was needed.
  + The sampling rate was determined experimentally with the section titled “Sample rate.” The sentences containing the data that was needed would be asserted once a second (about one hertz); however, the GPS chip asserted sentences five times per second (about five hertz).
  + Write Speed Test (Multiple Transfers of a single line) would take the parsed data, create a line of meaningful data, open and write into a text document, and then close the text document.
  + Write Speed Test (Single Transfers of multiple lines) would take the parsed data and create an array, open a text document, write the whole array at once, and then close the text document. This method was not the preferred choice due to data being lost if the code failed before the array could be written in the text document. The two methods for putting data into a text document were no faster than each other, but the multiple transfers of a single line was more reliable.
* Pedestrian Detection:
  + The main function called the camera function and also contained the code for putting rectangles around detected people.
  + The camera section contains the code for opening the Raspberry Pi camera and pulling a video form it. The video was streamed to a webpage.

**GPS**

**Parsing Sentences**

# Using serial module for communications with GPS

import serial

# Using time module for timing

import time

# Initialize GPS

gps = serial.Serial("/dev/ttyUSB0", baudrate = 9600)

size = 100000

data\_dump = ""

print("Recording GPS data...\n")

start = time.time()

for i in range(0, size):

# Read line and decode from 'bytes' to 'str' to allow split() below

line = gps.readline().decode("utf-8")

data = line.split(",")

data\_dump += str(data[0]) + ","

if i%50 == 0:

print("Data Points Recorded: " + str(i))

print("Time: " + str(time.time()-start) + " seconds\n")

end = time.time()

print("Time: " + str(end-start) + " seconds")

data\_dump = data\_dump[:-1]

print("Number of Data Points: " + str((len(data\_dump)+1)/7))

file = open("Sentence\_Code\_Data",'w')

file.write("Number of Data Points: " + str((len(data\_dump)+1)/7) + "\n\n")

file.write("Elapsed Time: " + str("%.3f" % round(end-start,3)) + " seconds \n\n")

file.write(data\_dump)

file.close()

**Validity (of data pulled) Check**

# Using serial module for communications with GPS

import serial

# Using time module for timing

import time

# Initialize GPS

gps = serial.Serial("/dev/ttyUSB0", baudrate = 9600)

size = 100000

data\_dump = ""

print("Recording GPS data...\n")

start = time.time()

for i in range(0, size):

# Read line and decode from 'bytes' to 'str' to allow split() below

line = gps.readline().decode("utf-8")

data = line.split(",")

data\_dump += str(data[0]) + ","

if i%50 == 0:

print("Data Points Recorded: " + str(i))

print("Time: " + str(time.time()-start) + " seconds\n")

end = time.time()

print("Time: " + str(end-start) + " seconds")

data\_dump = data\_dump[:-1]

print("Number of Data Points: " + str((len(data\_dump)+1)/7))

file = open("Sentence\_Code\_Data",'w')

file.write("Number of Data Points: " + str((len(data\_dump)+1)/7) + "\n\n")

file.write("Elapsed Time: " + str("%.3f" % round(end-start,3)) + " seconds \n\n")

file.write(data\_dump)

file.close()

**Checking Ability to Preform Calculations**

# Using serial module for communications with GPS

import serial

# Using time module for timing

import time

# Initialize GPS

gps = serial.Serial("/dev/ttyUSB0", baudrate = 9600)

a = 0

b = 0

while (a == 0) | (b == 0):

# Read line and decode from 'bytes' to 'str' to allow split() below

line = gps.readline().decode("utf-8")

data = line.split(",")

if (data[0] == "$GPRMC") & (data[2] == "A"):

b = a

a = float(data[3])

c = b-a

print("A is: " + str(a) + " and " + str(type(a)))

print("B is: " + str(b) + " and " + str(type(b)))

print("C is: " + str(c) + " and " + str(type(c)))

**Sampling Rate Check**

# Using serial module for communications with GPS

import serial

# Using time module for timing

import time

# Initialize GPS

gps = serial.Serial("/dev/ttyUSB0", baudrate = 9600)

records = 0

file = open("Sample\_Rate\_&\_Output\_Format\_Data","w")

file.write("Time,Validity,Latitude,Longitude,Speed(kts)\n")

start = time.time()

while (records < 100):

# Read line and decode from 'bytes' to 'str' to allow split() below

line = gps.readline().decode("utf-8")

data = line.split(",")

if (data[0] == "$GPRMC"):

records = records + 1

file.write(str(data[1]) + "," + str(data[2]) + "," + str(data[3]) + "," + str(data[5]) + "," + str(data[7]) + "\n")

end = time.time()

file.write("Records Recorded: " + str(records) + "\n")

file.write("Elapsed Time: " + str(end-start) + " seconds")

file.close()

print("Elapsed Time: " + str(end-start) + " seconds")

**Write Speed Test (Multiple Transfers of a single line)**

# Using serial module for communications with GPS

import serial

# Using time module for timing

import time

# Initialize GPS

gps = serial.Serial("/dev/ttyUSB0", baudrate = 9600)

file = open("File\_Write\_Test\_MultiTransfer","w")

file.write("")

file.close()

size = 100

start = time.time()

for i in range(0,size):

# Read line and decode from 'bytes' to 'str' to allow split() below

line = gps.readline().decode("utf-8")

data = line.split(",")

if data[0] == '$GPRMC':

file = open("File\_Write\_Test\_MultiTransfer","a")

file.write(str(line))

file.close()

end = time.time()

file = open("File\_Write\_Test\_MultiTransfer","a")

print("Time: " + str(end-start) + " seconds")

file.write("Time: " + str(end-start) + " seconds")

file.close()

**Write Speed Test (Single Transfers of multiple lines)**

# Using serial module for communications with GPS

import serial

# Using time module for timing

import time

# Initialize GPS

gps = serial.Serial("/dev/ttyUSB0", baudrate = 9600)

file = open("File\_Write\_Test\_SingleTransfer","w")

size = 100

j = 0

record = [None]\*size

start = time.time()

for i in range(0,size):

# Read line and decode from 'bytes' to 'str' to allow split() below

line = gps.readline().decode("utf-8")

data = line.split(",")

if data[0] == '$GPRMC':

record[j] = line

j = j + 1

for k in range(0,j-1):

file.write(str(record[k]))

end = time.time()

print("Time: " + str(end-start) + " seconds")

file.write("Time: " + str(end-start) + " seconds")

file.close()

**Pedestrian Detection**

**Main** **(calls the camera function and attempts to put rectangles on detected pedestrians)**

import cv2

import sys

#from mail import sendEmail

from flask import Flask, render\_template, Response

from camera import VideoCamera

import time

import threading

#email\_update\_interval = 600 # sends an email only once in this time interval

video\_camera = VideoCamera(flip=False) # creates a camera object, flip vertically

object\_classifier = cv2.CascadeClassifier("/models/fullbody\_recognition\_model.xml") # an opencv classifier

# App Globals (do not edit)

app = Flask(\_\_name\_\_)

last\_epoch = 0

def check\_for\_objects():

global last\_epoch

while True:

try:

frame, found\_obj = video\_camera.get\_object(object\_classifier)

if found\_obj and (time.time() - last\_epoch) > email\_update\_interval:

last\_epoch = time.time()

print ("Sending email...")

#sendEmail(frame)

print ("done!")

except:

#print ("Error sending email: ", sys.exc\_info()[0])

@app.route('/')

def index():

return render\_template('index.html')

def gen(camera):

while True:

frame = camera.get\_frame()

yield (b'--frame\r\n'

b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

@app.route('/video\_feed')

def video\_feed():

return Response(gen(video\_camera),

mimetype='multipart/x-mixed-replace; boundary=frame')

if \_\_name\_\_ == '\_\_main\_\_':

t = threading.Thread(target=check\_for\_objects, args=())

t.daemon = True

t.start()

app.run(host='0.0.0.0', debug=False)

**Camera**

import cv2

from imutils.video.pivideostream import PiVideoStream

import imutils

import time

import numpy as np

class VideoCamera(object):

def \_\_init\_\_(self, flip = False):

self.vs = PiVideoStream().start()

self.flip = flip

time.sleep(2.0)

def \_\_del\_\_(self):

self.vs.stop()

def flip\_if\_needed(self, frame):

if self.flip:

return np.flip(frame, 0)

return frame

def get\_frame(self):

frame = self.flip\_if\_needed(self.vs.read())

ret, jpeg = cv2.imencode('.jpg', frame)

return jpeg.tobytes()

def get\_object(self, classifier):

found\_objects = False

frame = self.flip\_if\_needed(self.vs.read()).copy()

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

objects = classifier.detectMultiScale(

gray,

scaleFactor=1.1,

minNeighbors=5,

minSize=(30, 30),

flags=cv2.CASCADE\_SCALE\_IMAGE

)

if len(objects) > 0:

found\_objects = True

# Draw a rectangle around the objects

for (x, y, w, h) in objects:

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

ret, jpeg = cv2.imencode('.jpg', frame)

return (jpeg.tobytes(), found\_objects)

**Appendix D: Code That Was Unused (by application)**

This appendix details the code that was created during the evolution of the project, but was ultimately not used in the final project.

**GPS (Android Application)**

main method

package com.example.punjabivishi.googlelocation;

import com.google.android.gms.common.ConnectionResult;

import com.google.android.gms.common.api.GoogleApiClient;

import com.google.android.gms.maps.GoogleMap;

import com.google.android.gms.maps.GoogleMap.OnMyLocationButtonClickListener;

import com.google.android.gms.maps.GoogleMap.OnMyLocationClickListener;

import com.google.android.gms.maps.OnMapReadyCallback;

import com.google.android.gms.maps.SupportMapFragment;

import com.google.android.gms.maps.model.LatLng;

import com.google.android.gms.maps.model.Marker;

import com.google.android.gms.maps.model.MarkerOptions;

import android.Manifest;

import android.content.DialogInterface;

import android.content.pm.PackageManager;

import android.location.Location;

import android.location.LocationListener;

import android.os.Build;

import android.os.Bundle;

import android.support.annotation.NonNull;

import android.support.v4.app.ActivityCompat;

import android.support.v4.content.ContextCompat;

import android.support.v7.app.AppCompatActivity;

import android.widget.Toast;

/\*\*

\* This demo shows how GMS Location can be used to check for changes to the users location. The

\* "My Location" button uses GMS Location to set the blue dot representing the users location.

\* Permission for {@link android.Manifest.permission#ACCESS\_FINE\_LOCATION} is requested at run

\* time. If the permission has not been granted, the Activity is finished with an error message.

\*/

public class MapLocationActivity extends AppCompatActivity

implements OnMapReadyCallback,

GoogleApiClient.ConnectionCallbacks,

GoogleApiClient.OnConnectionFailedListener,

LocationListener {

GoogleMap mGoogleMap;

SupportMapFragment mapFrag;

LocationRequest mLocationRequest;

GoogleApiClient mGoogleApiClient;

Location mLastLocation;

Marker mCurrLocationMarker;

@Override

protected void onCreate(Bundle savedInstanceState)

{

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_maps);

getSupportActionBar().setTitle("Map Location Activity");

mapFrag = (SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.map);

mapFrag.getMapAsync(this);

}

@Override

public void onPause() {

super.onPause();

//stop location updates when Activity is no longer active

if (mGoogleApiClient != null) {

LocationServices.FusedLocationApi.removeLocationUpdates(mGoogleApiClient, this);

}

}

@Override

public void onMapReady(GoogleMap googleMap)

{

mGoogleMap=googleMap;

mGoogleMap.setMapType(GoogleMap.MAP\_TYPE\_HYBRID);

//Initialize Google Play Services

if (android.os.Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.M) {

if (ContextCompat.checkSelfPermission(this,

Manifest.permission.ACCESS\_FINE\_LOCATION)

== PackageManager.PERMISSION\_GRANTED) {

//Location Permission already granted

buildGoogleApiClient();

mGoogleMap.setMyLocationEnabled(true);

} else {

//Request Location Permission

checkLocationPermission();

}

}

else {

buildGoogleApiClient();

mGoogleMap.setMyLocationEnabled(true);

}

}

protected synchronized void buildGoogleApiClient() {

mGoogleApiClient = new GoogleApiClient.Builder(this)

.addConnectionCallbacks(this)

.addOnConnectionFailedListener(this)

.addApi(LocationServices.API)

.build();

mGoogleApiClient.connect();

}

@Override

public void onConnected(Bundle bundle) {

mLocationRequest = new LocationRequest();

mLocationRequest.setInterval(1000);

mLocationRequest.setFastestInterval(1000);

mLocationRequest.setPriority(LocationRequest.PRIORITY\_BALANCED\_POWER\_ACCURACY);

if (ContextCompat.checkSelfPermission(this,

Manifest.permission.ACCESS\_FINE\_LOCATION)

== PackageManager.PERMISSION\_GRANTED) {

LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient, mLocationRequest, this);

}

}

@Override

public void onConnectionSuspended(int i) {}

@Override

public void onLocationChanged(Location location)

{

mLastLocation = location;

if (mCurrLocationMarker != null) {

mCurrLocationMarker.remove();

}

//Place current location marker

LatLng latLng = new LatLng(location.getLatitude(), location.getLongitude());

MarkerOptions markerOptions = new MarkerOptions();

markerOptions.position(latLng);

markerOptions.title("Current Position");

markerOptions.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE\_MAGENTA));

mCurrLocationMarker = mGoogleMap.addMarker(markerOptions);

//move map camera

mGoogleMap.moveCamera(CameraUpdateFactory.newLatLngZoom(latLng,11));

}

@Override

public void onStatusChanged(String s, int i, Bundle bundle) {

}

@Override

public void onProviderEnabled(String s) {

}

@Override

public void onProviderDisabled(String s) {

}

public static final int MY\_PERMISSIONS\_REQUEST\_LOCATION = 99;

private void checkLocationPermission() {

if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS\_FINE\_LOCATION)

!= PackageManager.PERMISSION\_GRANTED) {

// Should we show an explanation?

if (ActivityCompat.shouldShowRequestPermissionRationale(this,

Manifest.permission.ACCESS\_FINE\_LOCATION)) {

// Show an explanation to the user \*asynchronously\* -- don't block

// this thread waiting for the user's response! After the user

// sees the explanation, try again to request the permission.

new AlertDialog.Builder(this)

.setTitle("Location Permission Needed")

.setMessage("This app needs the Location permission, please accept to use location functionality")

.setPositiveButton("OK", new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialogInterface, int i) {

//Prompt the user once explanation has been shown

ActivityCompat.requestPermissions(MapLocationActivity.this,

new String[]{Manifest.permission.ACCESS\_FINE\_LOCATION},

MY\_PERMISSIONS\_REQUEST\_LOCATION );

}

})

.create()

.show();

} else {

// No explanation needed, we can request the permission.

ActivityCompat.requestPermissions(this,

new String[]{Manifest.permission.ACCESS\_FINE\_LOCATION},

MY\_PERMISSIONS\_REQUEST\_LOCATION );

}

}

}

@Override

public void onRequestPermissionsResult(int requestCode,

String permissions[], int[] grantResults) {

switch (requestCode) {

case MY\_PERMISSIONS\_REQUEST\_LOCATION: {

// If request is cancelled, the result arrays are empty.

if (grantResults.length > 0

&& grantResults[0] == PackageManager.PERMISSION\_GRANTED) {

// permission was granted, yay! Do the

// location-related task you need to do.

if (ContextCompat.checkSelfPermission(this,

Manifest.permission.ACCESS\_FINE\_LOCATION)

== PackageManager.PERMISSION\_GRANTED) {

if (mGoogleApiClient == null) {

buildGoogleApiClient();

}

mGoogleMap.setMyLocationEnabled(true);

}

} else {

// permission denied, boo! Disable the

// functionality that depends on this permission.

Toast.makeText(this, "permission denied", Toast.LENGTH\_LONG).show();

}

return;

}

// other 'case' lines to check for other

// permissions this app might request

}

}

@Override

public void onConnectionFailed(@NonNull ConnectionResult connectionResult) {

}

}

xml code

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.punjabivishi.googlelocation">

<!--

The ACCESS\_COARSE/FINE\_LOCATION permissions are not required to use

Google Maps Android API v2, but you must specify either coarse or fine

location permissions for the 'MyLocation' functionality.

-->

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:roundIcon="@mipmap/ic\_launcher\_round"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<!--

The API key for Google Maps-based APIs is defined as a string resource.

(See the file "res/values/google\_maps\_api.xml").

Note that the API key is linked to the encryption key used to sign the APK.

You need a different API key for each encryption key, including the release key that is used to

sign the APK for publishing.

You can define the keys for the debug and release targets in src/debug/ and src/release/.

-->

<meta-data

android:name="com.google.android.geo.API\_KEY"

android:value="@string/google\_maps\_key" />

<activity

android:name=".MapsActivity"

android:label="@string/title\_activity\_maps">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

google maps api code

<resources>

<!--

TODO: Before you run your application, you need a Google Maps API key.

To get one, follow this link, follow the directions and press "Create" at the end:

https://console.developers.google.com/flows/enableapi?apiid=maps\_android\_backend&keyType=CLIENT\_SIDE\_ANDROID&r=EF:3A:D4:5F:F0:92:87:56:53:96:7C:C6:82:5D:E3:FD:7D:D3:F5:28%3Bcom.example.punjabivishi.googlelocation

You can also add your credentials to an existing key, using these values:

Package name:

EF:3A:D4:5F:F0:92:87:56:53:96:7C:C6:82:5D:E3:FD:7D:D3:F5:28

SHA-1 certificate fingerprint:

EF:3A:D4:5F:F0:92:87:56:53:96:7C:C6:82:5D:E3:FD:7D:D3:F5:28

Alternatively, follow the directions here:

https://developers.google.com/maps/documentation/android/start#get-key

Once you have your key (it starts with "AIza"), replace the "google\_maps\_key"

string in this file.

-->

<string name="google\_maps\_key" templateMergeStrategy="preserve" translatable="false">AIzaSyClb6C8H73ybenANRuKYE5ScoMm99dAQBA</string>

</resources>

**Pedestrian Detection**

**First Camera Test**

# import the necessary packages

from picamera.array import PiRGBArray

from picamera import PiCamera

import time

import cv2

# initialize the camera and grab a reference to the raw camera capture

camera = PiCamera()

camera.resolution = (640, 480)

camera.framerate = 32

rawCapture = PiRGBArray(camera, size=(640, 480))

# allow the camera to warmup

time.sleep(0.1)

# capture frames from the camera

for frame in camera.capture\_continuous(rawCapture, format="bgr", use\_video\_port=True):

# grab the raw NumPy array representing the image, then initialize the timestamp

# and occupied/unoccupied text

image = frame.array

# show the frame

cv2.imshow("Frame", image)

key = cv2.waitKey(1) & 0xFF

# clear the stream in preparation for the next frame

rawCapture.truncate(0)

# if the `q` key was pressed, break from the loop

if key == ord("q"):

break