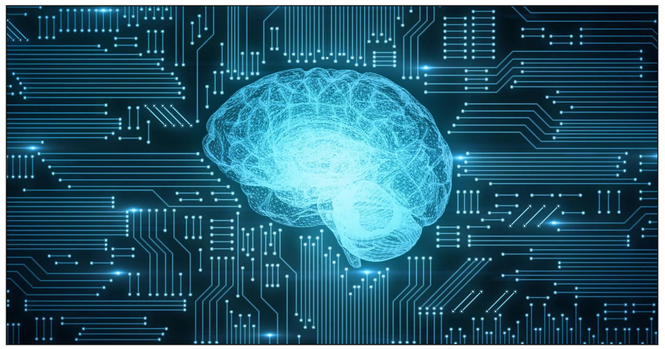
BOTTER: Applying Machine Learning to Video Games

Viroon Yong, Akash Sharma, Francis Hulmes, and Hyung Jun Hahn

CAPSTONE Senior Design 2018-2019, Department of Electrical and Computer Engineering, University of Delaware

PURPOSE

Machine learning and artificial intelligence are hot topics of discussion, research, and development. Machine learning allows software to learn and adapt to stimulus, and refine its algorithms to perform more efficiently. We set out to use a generic machine learning algorithm to create different bots that are capable of playing different video games.



GOALS and REQUIREMENTS

The primary goal of this project was to create a bot using machine learning that is able to play a video game successfully versus an opponent. To accomplish this, we set several smaller goals:

- Become familiar with TensorFlow and Keras
- Create a bot capable of playing Pong against a scripted bot
- Evaluate the effectiveness of different machine learning algorithms on a given training data set
- Expand into other simple video games, such as Asteroids, and eventually a complex modern game, such as Starcraft II

Requirements:

- Find a game that has its game state information readily available.
- Be able to use game source code to feed data into TensorFlow to create a model for training.
- When running the game from the trained model it should be able to defeat the opponent

APPROACH

To approach the task of creating Machine Learning bots, we broke down progress of the project into these distinct steps:

- Investigated StarCraft 2 APIs (PySc2, python-sc2) and previous implementations of machine learning on StarCraft 2 .
- StarCraft 2 had too many variables - we decided to choose a simpler game as to restrict our inputs, outputs, and win conditions, and decided on Atari Pong.
- Used source code we found online for pong to feed frame data and create a predictive model.
- After successfully training a model that is able to beat an opponent in pong we are going explore possibilities for a bot that can play Atari Asteroids

DESIGN FOR PONG BOT

Training a Model

To train the model, we pick out specific data in every frame in a game. The data includes the ball's position, the paddle's position, and the player's chosen paddle movement (up, down, or stationary) for that frame. We feed all this data into a classification algorithm, which is implemented through the TensorFlow and Keras python libraries to generate a predictive model. This model takes in a frame data as input, and returns the suggested paddle movement.

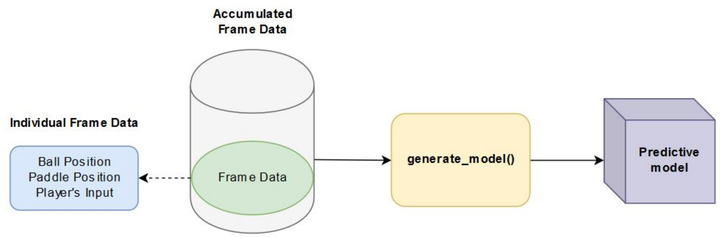


Figure 1: Diagram of the generation of the predictive model. generate_model() is a high-level function. The implementation may include TensorFlow, Keras, and various algorithms from those APIs

Utilizing the Model

After generating our predictive model, we wired in the model to play the game. For every frame in the game, the model takes in the current frame data - the ball's position and the paddle's position, and returns the suggested paddle movement, which we execute.

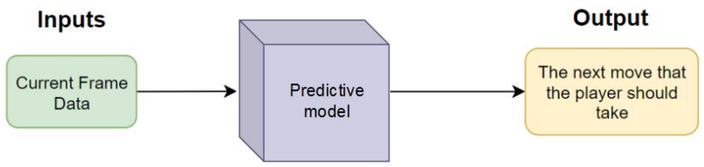


Figure 2: Usage of the model to run a machine-learning-based bot. Current frame data and the generated predictive model are fed into a function next_move(), which determines the model's suggested next move based on the given state of the game, and then returns the next move that the player should take for the current frame.

PROJECT STATUS

As of now, we have successfully created a bot that can play Pong by repurposing an neural-network-based image classification algorithm. We also have begun experimenting with other machine learning algorithms available through Keras, but have not yet collected data. Below is a plot of our classification algorithm bot's performance in response to quantity of given data:

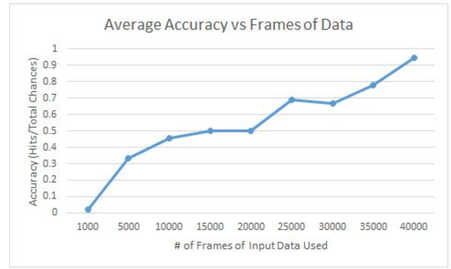


Figure 3: Chart of Average Accuracy (Over 5 games) of the bot over 5000 frame intervals. After 40000 frames, the bot approaches the accuracy of the nearly perfect scripted bot which is 0.96

The next step in our project is to begin comparing the neural-network based approach to other algorithms to determine which can train the Pong bot with the least amount of data, then explore how we can apply similar techniques to develop a bot that can play Atari Asteroids.

COMPONENTS AND TESTING

Our main component to test was the predictive model generated from a given data set. To test our predictive model, we fed the machine learning algorithm data that would have an obvious outcome for how the resultant bot ought to behave. For example, feeding the bot data that only tells the player to move up, will result in a bot that only moves up. We made 4 different datasets to test our model generation:

1. Up-only dataset
2. Down-only dataset
3. No-Movement dataset
4. High-Movement dataset (player moves paddle more than is necessary)

Although machine learning model generation is non-deterministic, we were able to replicate the expected behavior a majority of the time when we fed the dataset to generate the model.

We further tested our model generation by feeding it input obtained from a "perfect" scripted bot - one that never missed a hit, and the resultant bot followed the same procedures as the scripted bot.

HART: Radar Data Visualization with Augmented Reality

Kolby Kuratnick, Jason Reynolds, Vinay Vazir, & Samuel Romano

CAPSTONE Senior Design 2018-2019, Department of Electrical and Computer Engineering, University of Delaware

PROBLEM STATEMENT

Presently, real-time radar data visualization is unintuitive and antiquated. Objects are displayed as dots whose radii vary proportionally to the objects' size. This creates a need for personnel viewing the data to base their assessments on their own expertise and experience, which may be subjective.

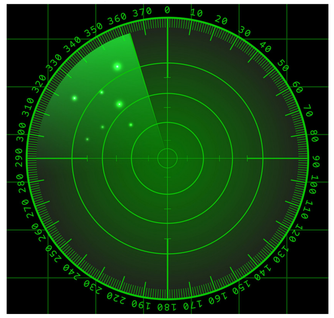


Figure 1: Previous Radar Visualization

While advancements in radar technology have produced more information, the visualization is lagging. At the request of the US Army Command, Control, Computer, Combat Systems, Intelligence, Surveillance, and Reconnaissance (C5ISR), we set out to create a visualization system for radar data using augmented reality (AR).

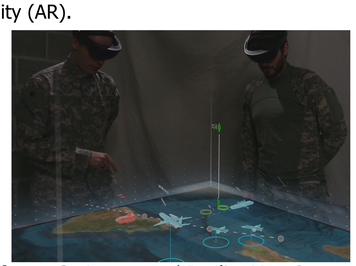


Figure 2: Augmented Reality Map Concept

The AR map will feature 3D holograms of the terrain, buildings, satellites, ships, and commercial aircraft along with the rendered radar data. The user will be able to zoom down to ground level for a first person view of all radar and local data. This visualization platform will allow a full 360 degree experience at the desired location to enable users to strategize effectively.

GOALS and REQUIREMENTS

Requirements were formed from the requests given to us by C5ISR. These requirements guided our work on the project.

Requirements:

1. Utilization of Augmented Reality
2. Visualization of Ground, Air, and Ballistic Targets
3. Multiple Users Capable of Viewing Same Map
4. Project Visualization of Data
5. Intuitive to Read and Analyze

Goals were defined based on these requirements and their associated constraints. Ultimately, we wanted to visualize as much relevant data for the end-user as we could.

Goals:

- Target Identification and Render Object in Space
- Trace Known Target Movements
- Seamlessly transition from Far View to First Person
- Implement full 360 degree First Person View with Nearby Data
- Target Path Prediction

APPROACH

Use of HoloLens: Due to the large amount of processing required to collect, analyze, and predict data all in real-time the HoloLens will be used only to visualize the end result.

Use of Server: All of the data processing will be done through a server before it reaches the HoloLens. The server collects commercial aircraft, ship, satellite, public transportation in New York City and Los Angeles tracking information from various APIs along with the incoming radar data. Target path prediction calculations will also be handled here.

Radar Data and 3D Map Integration: To accomplish this, the Unity 3D engine along with the Mapbox SDK was used to create the map. The overlaid radar data, tracking, and identification information is the second layer. A third layer includes public tracking information such as satellites and planes.

Holograms in Real-Space: If the target is capable of being identified, a 3D model of the object will be rendered (e.g. truck, plane) in the identified location. This allows for an easy to understand visual of various target types.

Hardware Additions: To allow for additional hardware to be attached to the HoloLens, we designed and 3D printed a mount to connect the GPS tracking device to the HoloLens. We integrated the use of a GPS tracking device to give the latitude, longitude, altitude, and heading direction of the HoloLens.

DESIGN

Radar Data in NATO STANAG 4676 was created by a GPS simulator. In the future, this would be data received directly from a radar or multiple radars. Data received from a radar can be ground, air, or ballistic targets. Along with collected radar data, the server pulls commercial aircraft data from Opensky-Network, ship data from Vessel Finder, satellite data from N2YO and public transport data for Los Angeles and New York City (our demo locations) from Metro and MTA, respectively. All of this data and the radar data is parsed and formatted as a JSON file by the server. Once the data has been collected, the server analyzes each target to determine the likely target type (e.g. helicopter, car, etc…) and predict its most likely path. Targets that are received from APIs are pre-identified, but targets received from a radar are categorized based on speed, altitude, and radar cross section. Using a process of elimination by excluding impossible targets, this allows us to identify the most likely target type.

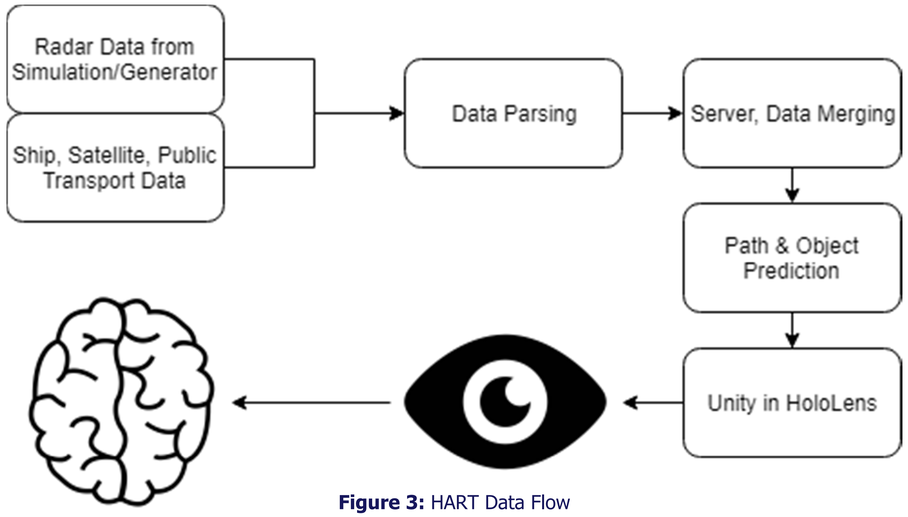


Figure 3: HART Data Flow

The target path prediction is accomplished by finding polynomial lines of fit for the latitude, longitude and elevation with respect to time. Finally, once the target has been identified and its path predicted, the data is utilized by Unity, the visualization software. All of the aforementioned data processing is handled by the server before being sent to the HoloLens. Within Unity at the HoloLens, the map will be generated, all objects will be rendered as their identified type, and the predicted path will be demonstrated by a line. When using the application within the HoloLens, there are multiple visual options. The satellites, public transportation, ships and commercial aircraft can all be toggled on and off. The satellites that our GPS tracker bounces off of are highlighted for clarity and as a target loses contact, it turns red and slowly fades away. All of this information is overlaid on the map by Mapbox that is capable of zooming from an entire worldview down to a street view. This allows the end-user to view a large battlefield territory to see all targets within an area or zoom into a small area that gives the ability to analyze specific situation within the battlefield. Ultimately, HART allows for a whole-battlefield visual approach to strategizing.

SOFTWARE and SYSTEM TESTING

The system was developed incrementally one piece at a time to ensure a correct build. In order,

1. Create a Mobile Application in the Microsoft Store
2. Generate an Object in Free Space with Unity
3. Import Mapbox into Unity
4. Overlay Object on Mapbox Map
5. Create a GPS Simulator for Object
6. Make Object Follow GPS Coordinates
7. Enable Real-Time Tracking of Commercial Aircraft, Ships, Public Transportation in desired Cities, and Satellites
8. Enable Real-Time Radar Data Input
9. Incorporate Real-Time Path Prediction
10. Incorporate Real-Time Target Identification

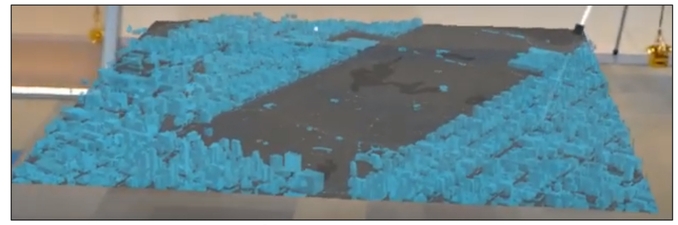


Figure 4: HART Map

PROJECT STATUS & EXPANSION

Final Project Status

Our final HART application includes everything we reached for, except true object identification. Since we developed the app and our demo with a GPS simulator and not real radar data, we did not receive crucial information like the radar cross-section of the object.

Figure 5: HART Final Demo QR Code

Please scan the QR Code to view our team website and most recent video demo of the HART project (under "Experiences").

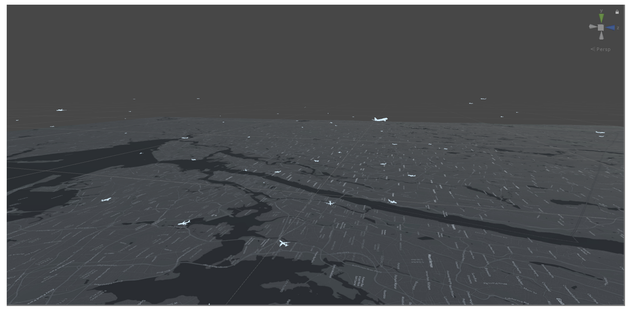


Figure 6: HART Final Prototype

Future Steps of Expansion

- Include all Public Transportation
- Include Ballistics, Missile Tracking
- Identify specific target types (e.g. a Navy Destroyer rather that just a ship)
- Test with Real-Time Radar Data
- Improve Accuracy of Location and Tracking in First Person View
- User Click Option to Identify Object as Friendly or Enemy

HeartMap: Wearable ECG Monitor

Deion Cottingham, Victoria DeSpirito, Allison Shovel, and Brent Wade

CAPSTONE Senior Design 2018-2019, Department of Electrical and Computer Engineering, University of Delaware

PROBLEM STATEMENT

Our motivation is to implement a real-life application that would benefit those suffering from a heart condition. Our intention is to make a discrete heart monitoring product. Heart arrhythmias occur when the electrical signals to the heart that coordinate heartbeats are not working properly. Many heart arrhythmias are harmless; however, if they are particularly abnormal, or result from a weak or damaged heart, arrhythmias can cause serious and even potentially fatal symptoms. Our customers, whom will receive this product, will be patients suffering from heart conditions, such as Bradycardia and Tachycardia. We determined we would be able to create an application that will be able to display a user's average heart signal day by day, where a doctor can analyze it and monitor it to see if peaks in the signal exceed those in the normal beat. If consistent days pass with an irregular signal, then an alert can be sent to the doctor and/or loved ones so a patient may be more closely monitored.

GOALS and REQUIREMENTS

Our goal was to design and implement an application to analyze ECG signals to monitor irregular heart beats. After researching the differences of heartbeats, we used a hardware and software approach to check our results. Using an averaged ECG reading the goal was to determine an average heart rate for the respective user in order for our software to be able to detect any abnormalities that occur in future readings. Using a Digilent Breadboard our goal was to record an ECG signal in real-time through Waveforms software. Our design checkpoints included:

- Obtain an heart rate signal from the ECG circuit design by recording through Waveforms software
- Integrate ECG signal with the Arduino UNO/Olimex EKG Shield
- Gather data points from recorded signals
- Integrate signal analysis algorithms into web platform
- Set up database to store each user's personal information/recordings

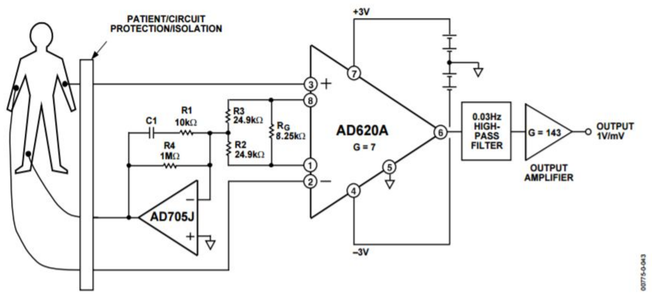


Figure 39. A Medical ECG Monitor Circuit

APPROACH

We began our project by going in two directions, software and hardware, in order to minimize discrepancies. We started off building an ECG to get a heart signal we can test through circuit design and and also trying to collect real-time data using the Arduino UNO. After collecting the data from the breadboard we noticed there were issues relevant to noise and electrode placement. From there, we changed our approach to using finger pads and analyzing the signal through data points, finding peaks and averaging daily heart rates to find a consistent average to base off of day by day. Using this data, we record the patient's results and display it on a website where doctors and loved ones can access this information. Those patients with irregular results would be advised to wear the long-term ECG vest in order to achieve a diagnosis of their condition. We used a current product called LifeVest to help assist us in creating our product.

DESIGN

Our team understood that the circuit design would have some issues as far as the 60 Hz noise and the electrode placement. With the help of Professor Boncelet, we were able to create a circuit design with specific Instrumental Amplifiers for biomedical purposes, AD620A and LM324 which are low cost, but high accuracy. We used high resistor values, which is displayed in our Goals and Requirements, to protect ourselves while testing. A good portion of the semester we dedicated to finding the most accurate placement. We tested different electrode placements such as on the inner wrists and the inner right ankle, but could not obtain a signal due to the 60 Hz noise. After conversing with Professor Boncelet, we determined that the best placement would be across the chest and on the sternum. With this placement we were able to record an ECG signal.

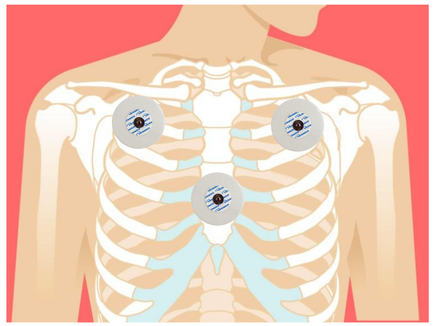


Figure 1: The figure above displays the electrode placement.

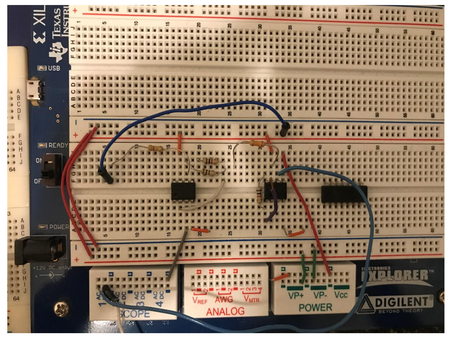


Figure 2: This figure displays our circuit design for recording the ECG signal through Waveforms.

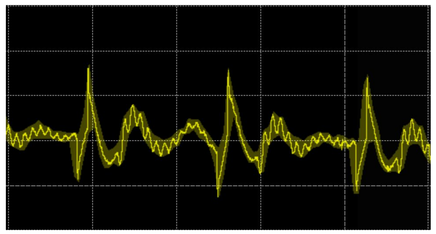


Figure 3: The figure above displays our recorded ECG signal on Waveforms software.

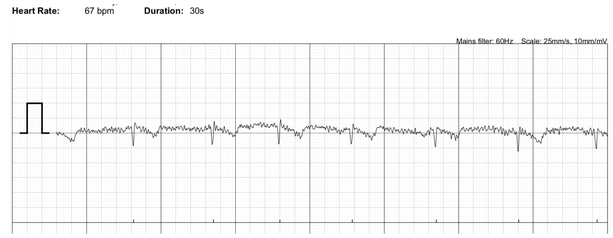
COMPONENT and SYSTEM TESTING

Our group was able to acquire an ECG signal using a specific circuit design on a Digilent Breadboard displayed through Waveforms. During our first semester, we focused on generating a signal and capturing it in real-time. We had many constraints with this due to finding the most accurate electrode placement and filtering out 60 Hz noise. The Arduino with the ECG shield also showed some constraints, not outputting clean accurate signals.

The software to analyze the pre-recorded signals showed some original troubles. Once de-bugged to accurately analyze what we had intended to get out of the heart rate signals, we had to run some tests to ensure accuracy. Using the Kardia finger pad machine to get a real signal, we were able to accurately average the signals and make sure our software outputted the correct heart beat rate, by testing and checking it with the Kardia application that works with the finger pads.

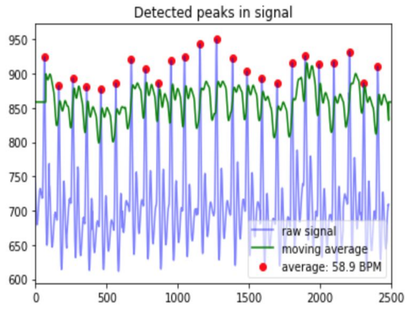
PROJECT STATUS

While our group was not able to accomplish our original project goal, our alternative plan has had substantial progress. The majority of our first semester was spent trying to obtain a real time ECG signal, however we were unsuccessful in doing so. We ran into various problems working with the ECG shield along with electrode placements. However, we did succeed in obtaining an ECG signal using a circuit design, displayed on waveforms, which we planned to use for testing purposes. This semester, we went in an alternative path and decided to use pre-recorded signals for analysis.



Taking this approach allowed us to begin our analysis of the pre-recorded signals by breaking them down into different components, such as R-R Intervals, Frequency Spectrum Analysis, and Average Heart Rate (bpm).

```
Average Heart Beat is: 58.9
```



Scan Cam: Hazardous Driving Reporting Assistance

Chima Akparanta, Greta Kintzley, Connor Mettus, Mark Seda

CAPSTONE Senior Design 2016-2017, Department of Electrical and Computer Engineering, University of Delaware

Problem Statement

There is currently no safe and easy way to report reckless driving while on the road. Currently, to report reckless driving, a driver must take note of a license plate, call a number, and report the incident, all of which is unsafe to do while operating a car.

Our objective is to create a single-click option to safely and effectively report hazardous driving.

Goals and Requirements

Our system should:

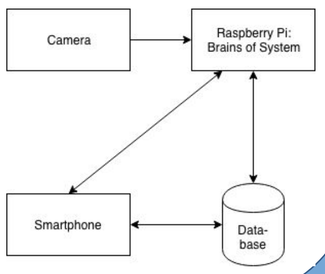
- Automatically recognize and read license plates in view
- Record GPS location and time
- Connect to OBD2 port to determine sudden braking (crash)
- Store data in database.
- Communicate with web app
- Provide meaningful functionality to the user

Approach

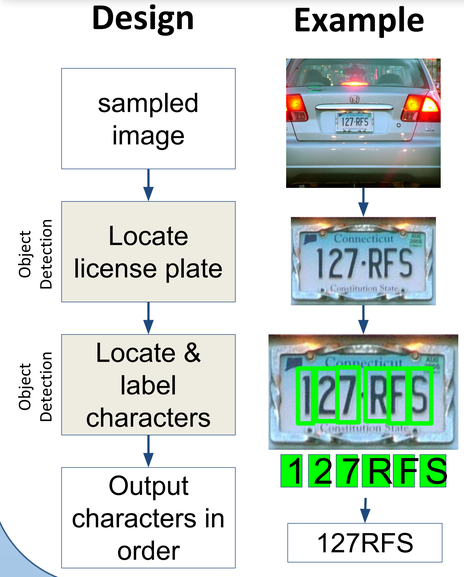
Our approach was to break the system into individual components, design and test the separated components, then integrate into a final design.

Individual Subsystems:

- Computer Vision System
- Hardware System (Computer, Raspberry Pi, Touch Screen, Power Source, GPS)
- Web Application
- Database
- OBD2 data collection and analysis



Design: Machine Learning & Computer Vision



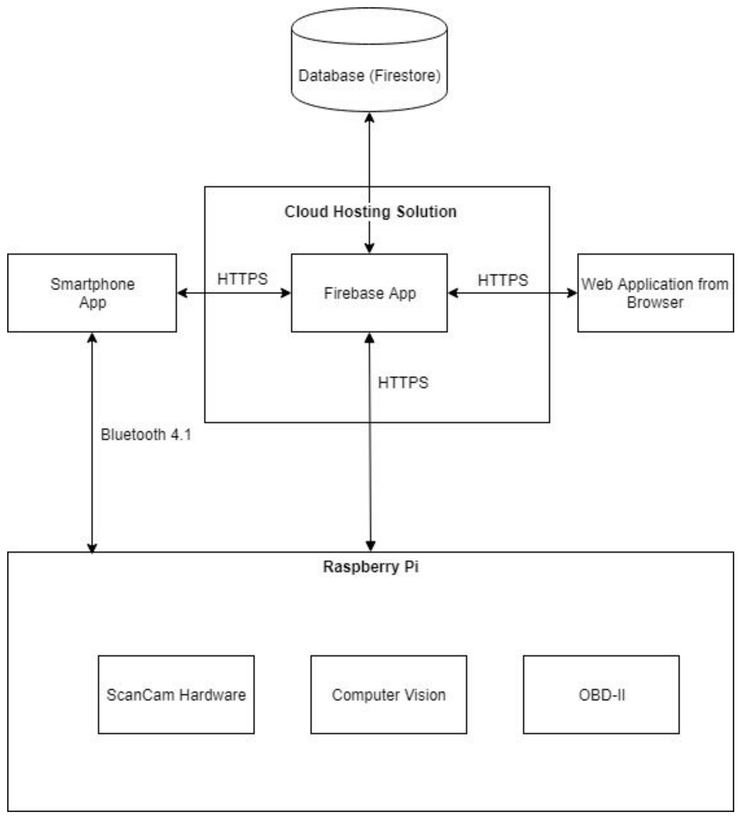
Our approach for the computer vision portion of this system is to break the recognition into two separate object detection steps. By using object detection to first locate the license plate in an image, we minimize the search space for detecting the characters.

Training data was annotated with bounding boxes and labels using our own click-and-drag annotation software.

Object detection models both for license plates and individual characters were trained using the machine learning library, Turicreate.

turi create intelligence™

Design: WebApp and Hardware



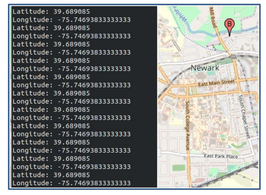
Component and System Testing

License Plate Recognition

Successfully located 97% of license plates in test images, including blurry and far away plates.

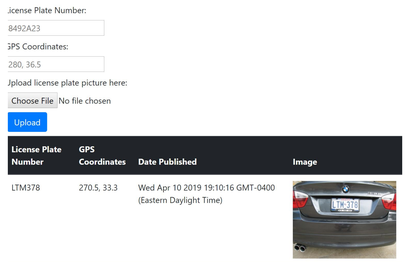
GPS and OBD-II Data

Raspberry Pi accurately reads and parses GPS data which correspond to the device location



Web Display of Data

Data can be saved to the firestore database and displayed in browser.



System Testing

The Raspberry Pi takes a picture, GPS and OBD-II data, and sends a request to the Web App which then stores and displays the data for later use.

Project Status

The basic groundwork of taking images, processing, uploading, and displaying has been completed. Some logical next steps include:

- Sampling the live video feed to perform recognition automatically
- Displaying the data in a maps format - with license plates overlayed on a geographic map
- Enabling quick communication with law-enforcement officials

Blue Path: Augmented Reality Navigation Application Using ARCore

Matthew Billone, David Chan, William Esteves, Lakshmi Palaparthi
CAPSTONE Senior Design 2018-2019, Department of Electrical and Computer Engineering University of Delaware

PROBLEM STATEMENT

University of Delaware has a huge focus on student safety, through the installments of blue light phones. Although they are plentiful, it can be difficult to locate the nearest one, especially when there is an emergency.

Figure 1: Blue Light Phone

By using Google's augmented reality plugin (ARCore), our application would allow Android users to render a path to the nearest blue light phones. So in general, whatever the situation, users should be able to quickly and efficiently go to the blue light phones.

GOALS AND REQUIREMENTS

Our goal for the TagIt app is to be able to place AR markers at different Blue Lights around campus and navigate to the closest Blue Light with respect to the user's location. Once the user is close to the Blue Light, the phone will render a pin point.

Requirements:

- Must be able to place markers at Blue Light locations
- Must be able to save the longitude and latitude position of the Blue Light
- Must be able to find user's current longitude and latitude position
- Must be able to retrieve positions of all markers and then determine the closest marker
- Must be able to navigate from user's location to the location of the closest marker
- Must be able to render marker once the user is close enough

APPROACH

We decided to develop our application in an Android environment. The main reason is that our team only has Window computers. Our team would create a database that contains latitude and longitude coordinates of a handful of blue light phones. When the application starts, the phone would obtain the user's location and search through the database to find the nearest blue light phone. After obtaining a valid traveling path to the blue light, we would use augmented reality to render the path for the user to follow. As a result, through the phone's camera, the user would follow the path leading them to the blue light phone. Once the user is approximately close to a blue light phone, the user's phone would also render an object near the blue light phone, to help the user locate it.

DESIGN

Phone Application

We created an application with four features (camera, compass, accelerometer and geolocation), displaying each upon entering the app based on the user's current location.

Geolocation and Firebase

To make the campus blue lights' location (latitude/longitude) more accessible, we manually entered and saved the coordinates into Firebase's realtime database. This data is only fetched once, since it is a static list of coordinates. Once every couple of seconds, the application would re-obtain the user's coordinates, through the phone's GPS or the network. Upon recapturing the user's location, the app will iterate through the list of blue light coordinates and calculate the distance to that specific blue light. The blue light with the shortest distance is then returned as the closest blue light.

Navigation

The navigation feature using Mapbox allows the user to determine the route to follow to each blue light. Google's ARCore is used to render the path starting from the current location ending at a blue light. The GPS/WiFi feature obtains the distance of user's current location to the nearest blue light. When the user has reached the closest blue light an AR object can be placed at the location.

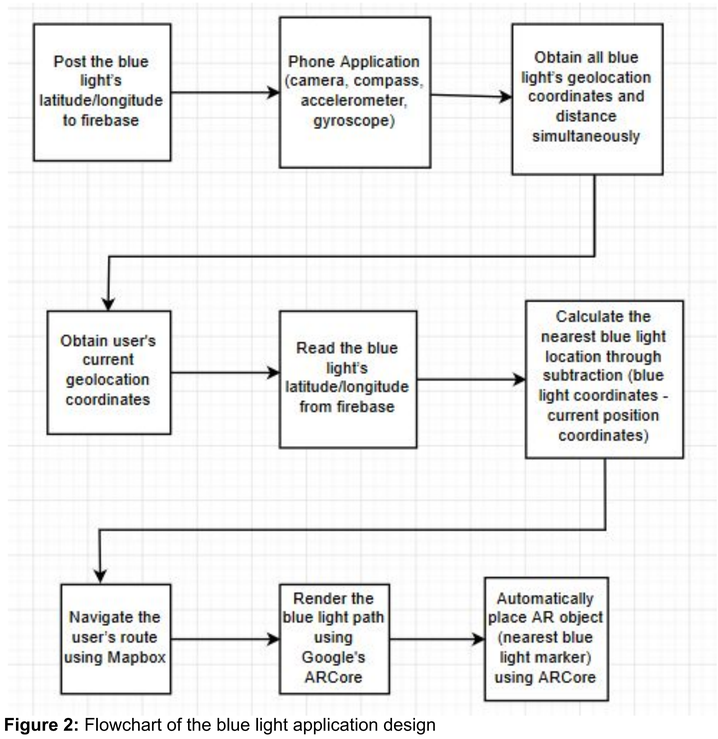


Figure 2: Flowchart of the blue light application design

SOFTWARE

We used Android Studio to create the app. Along with Android Studio, we used Google's augmented reality platform, ARCore; the 3D Framework, Sceneform; Google's mobile platform, Firebase; and the open source mapping platform, mapbox.

Android Studio

IDE for programming Android Applications

ARCore

Google's open source augmented reality library for Android Apps

mapbox

Open source library used to navigate the user

Firebase

Google's database used to store blue light location

PROJECT STATUS

We successfully made an Android app using Android Studio implementing ARCore. The user's location (longitude and latitude) gets recorded and the distance from the blue lights around the area are calculated. After the closest blue light is determined, the user then gets put into the AR session of the closest marker. As the user approaches the blue light there will be a marker that shows up in the AR environment to show the user that they are close and what the blue light looks like if they have never seen one before. Right now we have a navigation part of the application, but it is a normal navigation application not an AR implementation. Moving forward, we would like to implement navigation into the AR experience of the application. The user would see a line guiding them to the blue light along with the marker of the blue light when they arrive to it. Below is a screenshot of the ARCore environment with an example image.

Work to be completed :
-navigation route
-place multiple tags at blue lights
-save longitude/latitude coordinates and distance for each tag
-calculate distances

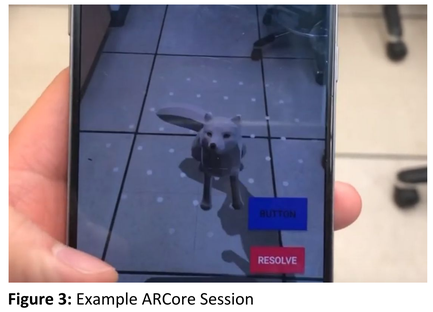


Figure 3: Example ARCore Session