Park Smart
Final Report and Overview v1.0

April 10, 2020

Team Name - Big Brother

Team Members:
Adam Rawski - amrawski@umich.edu
Bryan Brauchler - bbrauchl@umich.edu
Roderick Renwick - rodericklrenwick@gmail.com
Liam Mulligan - lpmullig@umich.edu

Project Advisor - Professor Miller

Observations:

1. Extracting parking lot lines would be easier and much more reliable using morphology.

2. Classification using raw pixel data with a neural network is not ideal. Doing feature extraction to obtain information directly applicable to automobiles (shape, aspect ratio, size etc.) would provide enhanced accuracy in an actual application.

In general, the team has addressed their project effectively and created a very good approach to the general problem. Had this been a normal semester, it is highly likely that the design would have been outstanding.

jwvm

Table of Contents

1. Introduction

1.1 Purpose

This document defines the low level design for Park Smart.

Finding space for parking at the University of Michigan Dearborn has progressively taken more time as more people are commuting to campus. Many students and faculty have expressed frustration over the available parking on campus. A system to assist in the process of parking is becoming more necessary to cater to the needs of faculty and students alike. This effort consists of a Computer Vision Module that will use machine learning to make inferences about which parking spots are available and which are not. Then the system will communicate with a web server to post those inferences on the internet. Various AI concepts will be enacted as well as different routing and networking techniques to move data once it is available.

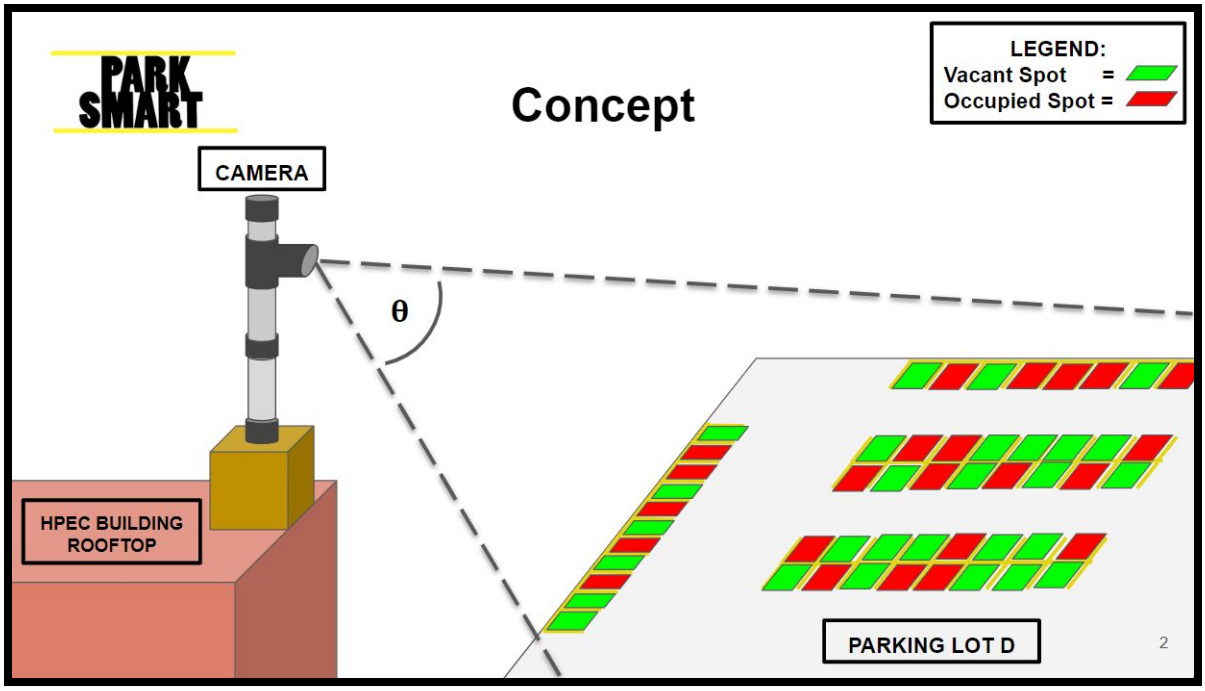

Figure 1.1: ParkSmart Concept

1.2 System Description

The Park Smart system consists of three modules. The first, referred to as the Computer Vision Module, will be physically located at a high vantage point, overlooking the parking lot. Its responsibility is to collect images, process that image data, and forward the results to a computer server. The second module, referred to as the Networking Module, contains the server and all the code to transfer data across a network. This module is responsible for storing and distributing data to endpoints. The final module, the Application module, is responsible for requesting, receiving, and displaying the data from the network module.

This concept is highly expandable, with the opportunity for several vision modules and statistical studies on data. However, for the purposes of a two-semester design window, a simplified system diagram will be created for a lab test environment. This leaves a framework that is able to be expanded in future semesters.

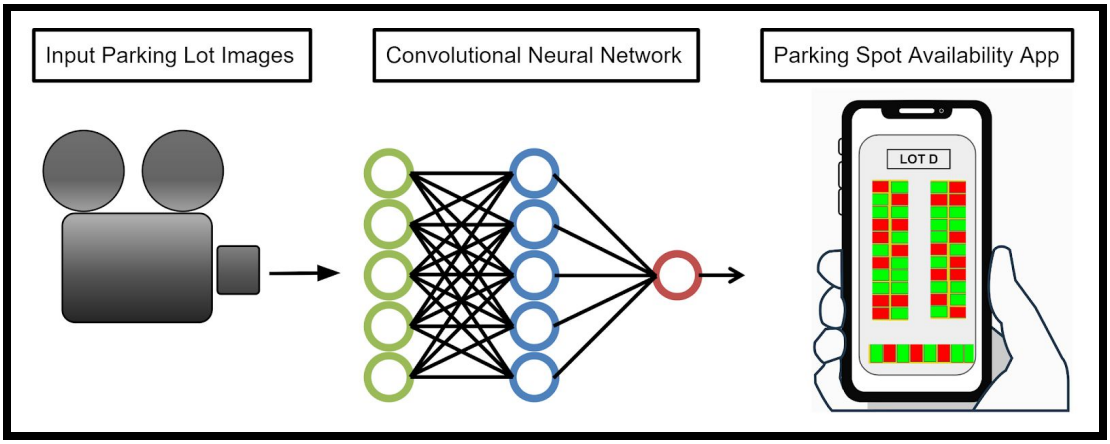

Figure 1.3.1: Top-Level Concept Abstraction

The following figure displays the three modules required to realize the concept.

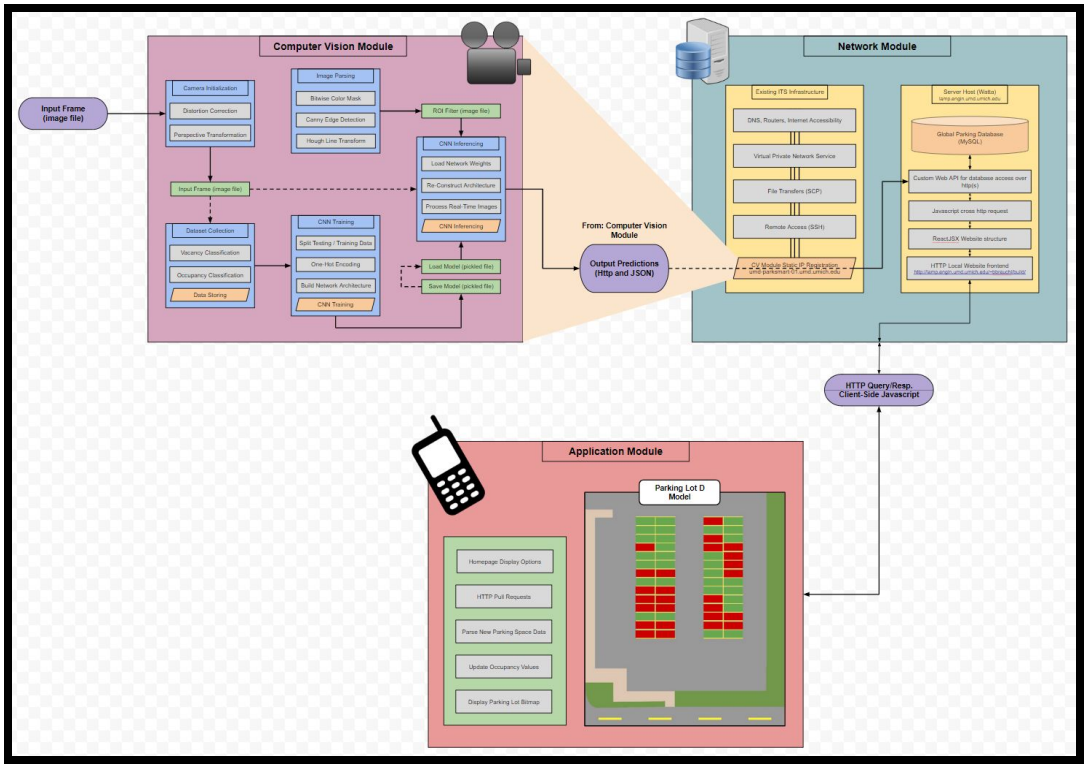

Figure 1.3.2: System Modules Overview

1.3 Interface Description:

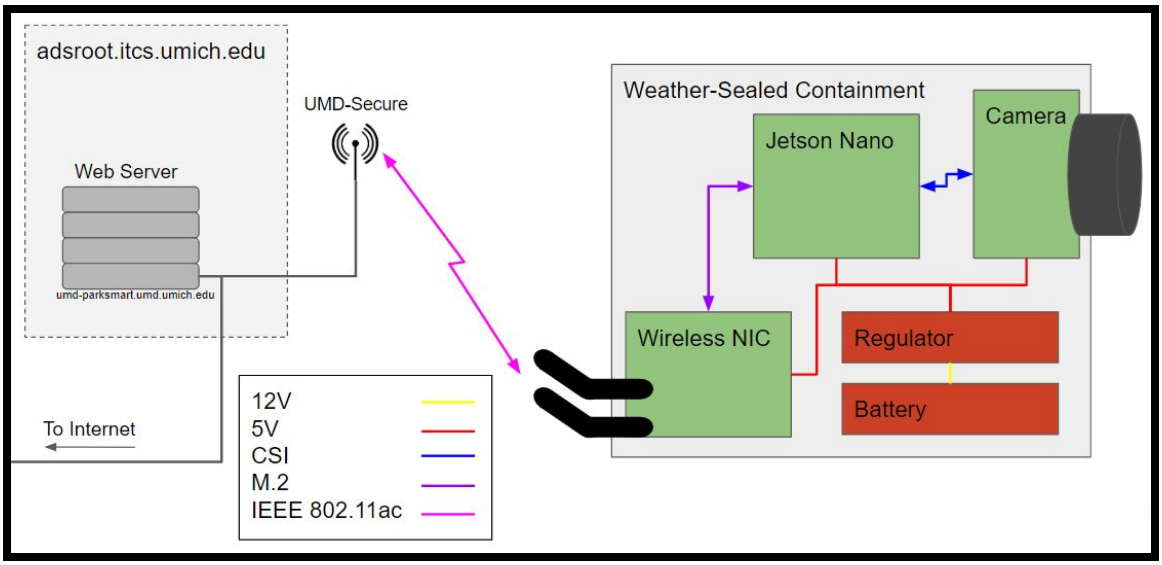

Figure 1.4: ParkSmart Interface Diagram

There are four primary interfaces involved in this system. The first is the interface between the vision subsystem and the image processing subsystem, where images are passed through the neural network for training/vacancy inference. The next is between the image processing subsystem and the network module, where the vacancy data is encoded and sent over Wi-Fi to the server. The third primary interface is between the server and the client, where the stored data is accessed graphically via a mobile application. The last is situation-specific; in this case, between the battery and the Computer Vision Module.

For the first interface, the communication between the camera and the board will be performed over a ribbon cable designed specifically for the Jetson Nano (included with the camera). The next two will use the IEEE 802.11ac Wi-Fi standard; the NIC for the Jetson board is capable of using the 2.4 GHz and 5 GHz transmission bands. The battery will be connected to a step-down voltage regulator, which will be wired directly to the board.

Protocol interfacing for remote access to both the server and the Computer Vision Modules will take advantage of the standards that are already supported by Linux. Specifically using standards like SSH for remote administration, SCP for secure data transfer and Sockets for general internet connectivity. This allows us to leverage the existing infrastructure that exists on campus.

1.4 References

1. *Detection of Parking Spots Using 2D Range Data - Jifu Zhou, Luis E. Navarro-Serment and Martial Hebert*
2. *IEEE 802.11: MAC and PHY layer specification for implementing WLAN networks*
3. *MIPI CSI-2 v3.0: Camera Serial Interface specification.*
4. *https://resizeimage.net/*
5. *https://markhedleyjones.com/projects/calibration-checkerboard-collection*
6. *https://github.com/priya-dwivedi/Deep-Learning/tree/master/parking_spots_detector*
7. *https://www.learnopencv.com/homography-examples-using-opencv-python-c/*
8. *https://medium.com/@ghimire.aiesecer/getting-started-2-pi-camera-setup-take-picture-and-shoot-video-with-python-script-embedded-5018c3568a52*
9. *http://www.netinstructions.com/automating-picture-capture-using-webcams-on-linuxubuntu/*
10. *http://cnrpark.it/*
11. *https://www.element14.com/community/community/designcenter/single-board-computers/blog/2019/05/21/nvidia-jetson-nano-developer-kit-pinout-and-diagrams*
12. *https://github.com/jeffbass/imagezmq*

For the software components, code is hosted on github:

https://github.com/bbrauchl/ParkSmart

https://github.com/amrawski/ParkSmart_App

We hereby give permission to any future student design projects to add to and use our codebase in developing further upon this project.

2. Constraints

2.1 Environmental Constraints

2.1.1 *The system in this phase will be developed as a testbed in a laboratory environment, and conditions of extreme environments will be saved for a later phase.*

2.1.2 *The system will function in only sunny conditions for the scope of this project.*

2.1.3 *The system will use pre-collected data on the day of the test in the lab environment.*

2.1.4 *The system must perform computation on images collected from the roof of HPEC to accurately characterize that parking lot.*

2.2 Size, Weight, Cost, Power, Constraints

2.2.1 *The Computer Vision Module must be able to run off a standard 120V 60Hz outlet for testing.*

2.2.2 *The server module must be connected to the internet with a route to communicate with the Computer Vision Module.*

2.2.3 *The Computer Vision Module must be light as to be mounted atop a building or light post without needing significant structural support.*

2.3 Reliability and Safety Considerations

2.3.1 *The system must be accessible remotely and in the case of failure be recoverable retaining information pertaining to the failure and the current state of the learning agent's knowledge base.*

2.4 Site Information

2.4.1 *The system in this phase will be developed in a laboratory environment.*

2.4.2 *The system must perform neural network inferences on parking lots at the University of Michigan-Dearborn.*

3. System Requirements

University of Michigan - Dearborn parking takes too long, and a system shall be designed to mitigate this time.

3.1 The system shall collect data via camera of a parking lot.

3.1.1 The Computer Vision Module must be configured such that the entire parking lot exists In the camera's field of view. (parking lot $\in$ camera FOV)

3.1.2 The system must identify parking spots at a range of 1-3 meters away from the camera. (1-3m $\in$ camera calibration distance)

3.1.3 The camera update rate must be at least 1 Hz (framerate $\geq$ 1Hz)

3.1.4 The Computer Vision Module must be lighter than 100 lb to mount without the need for additional structural support (Computer Vision Module weight $\leq$ 100 lb)

3.1.5 The Computer Vision Module must save collected data for at least 24 Hours (Retention Time > 24 hr)

3.1.6 The Computer Vision Module shall have a kill switch that will disconnect the battery from the image processing hardware.

3.2 The system shall be approved by the University and absent from privacy concerns.

3.2.1 Permission must be granted for surveillance of parking lots ($\exists$ written approval for parking surveillance).

3.2.2 Access to rooftop location for data acquisition must be granted to the group ($\exists$ access to camera mounting point).

3.2.3 The Computer Vision Module must operate without access to a standard wall outlet. ($\exists$ Internal Power Source)

3.2.4 The Computer Vision Module must be waterproofed such that it can remain in position up to five days at a time (Waterproof > 5 days)

3.3 The system shall send and receive messages and data from remote locations.

3.3.1 The network must achieve a throughput of at least 3.2 MBit/s from the Computer Vision Module to the web server to report data. This number is based on the expected camera update rate layed out in requirement 3.1.3 and the resolution of an image. (throughput $\geq$ 3.2MBit/s).

3.3.2 The network must guarantee packet arrivals to the destination network (Unreported Packet Loss = 0, TCP enabled).

3.3.3 The system must interface to existing wireless topography. ($\exists$ IEEE 802.11ac compatibility).

3.3.4 External clients must have remote access to the server via network. ($\exists$ external network interface).

3.3.5 Any clients connecting to the server must be authenticated ($\exists$ Client Authentication).

3.3.6 Any confidential or private data must be encrypted ($\exists$ Link Encryption)

3.3.7 The web server shall communicate with clients over apis functioning on the http protocol (api protocol == http)

3.3.8 The web server must log information regarding parking to a database for historical and searchable data.

3.4 The system must be accurate in counting parking spots.

3.4.1 The Computer Vision Module must infer the correct state 75% of the time. (P(Calculated == Actual) ≥ 75%).

3.4.2 If a vehicle is parked in multiple parking spaces, the inference must classify all spaces as occupied.

3.4.3 The inference must be run at a rate of at least once a minute. (inference update rate ≥ .016Hz)

3.4.4 The inference must classify small pedestrian vehicles such as motorcycles as being occupied spaces. (Edge-case classification > 50%)

3.5 The system must distribute information to users.

3.5.1 The web/app interface must service requests at all times of day. (Accessibility > 99%)

3.5.2 The web/app interface must report connection errors to the user.

3.5.2.1 The web/app interface must report Computer Vision Module Dropout errors (Dropout detection < 3 minutes)

3.5.2.2 The web/app interface must report visibility warnings

3.5.2.3 The web/app interface must report invalid API calls as the response to the html Requests. (> 99% usage errors reported back to user)

3.5.3 The app interface must provide warnings to users who are likely driving (warnings when speed > 10mph)

4. System Design

4.1. Computer Vision Module

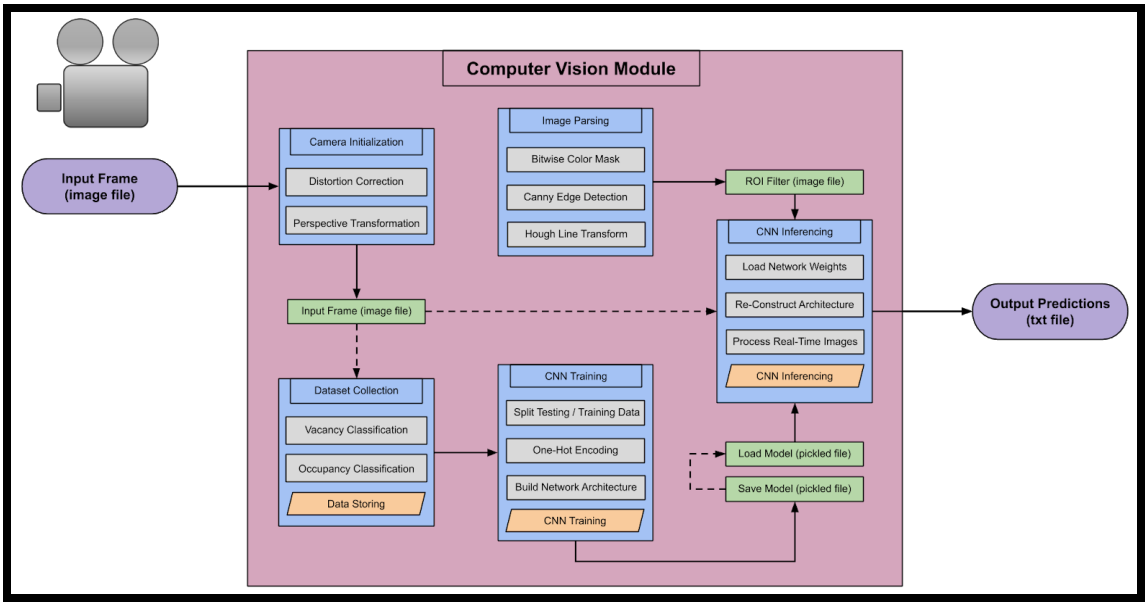

Figure 4.1: Low-Level Design of Computer Vision Module

4.1.1 Camera Initialization (Traces to Requirements 3.4.1, 3.4.2 and 3.4.4)

Using Python and OpenCV, some pre-processing will be done on images before going into the neural networks. Operations like de-warping and perspective transformation will be done for correcting camera distortion (see diagram displayed below).

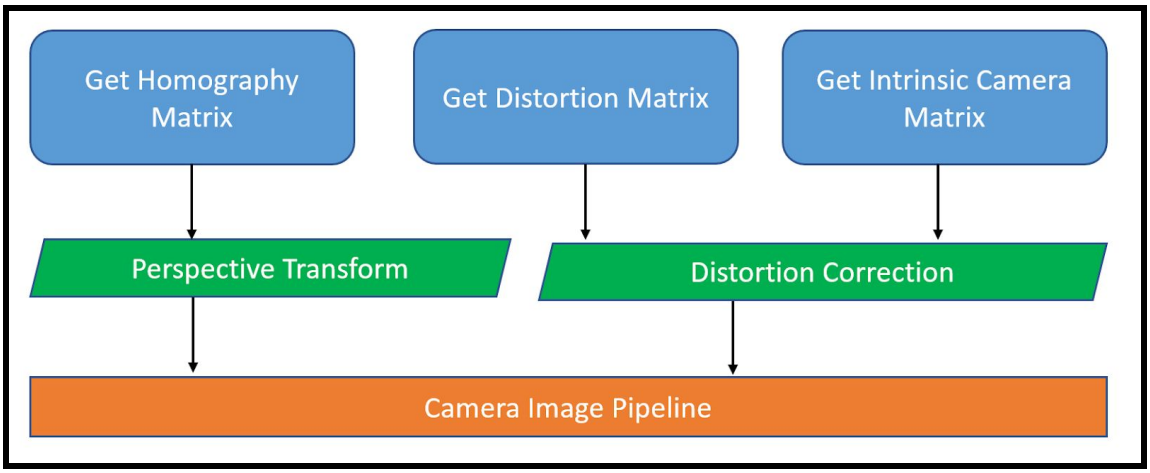

Figure 4.1.1: Camera Calibration Module

4.1.1.1 Distortion Correction (de-warping input image frame)

- Tangential and radial distortion shall be corrected as the input camera image will be de-warped via a distortion matrix and intrinsic value matrix, unique to the physical camera. These matrices are found via imaging a physical chessboard multiple times over, to track 2D pixel locations of the squares at different 3D positions. Knowing that the squares are consistent in size and contain straight vertical and horizontal lines normal to the chessboard, translation can be done to ensure distortion correction of an image.

4.1.1.1 Perspective Transformation (de-warping input image frame)

- A bird's-eye-view image shall be constructed from the incoming parking lot images taken at an angle. To do so, a homography matrix is used to map the pixel positions from one perspective to another. This is again done using a physical chessboard to take an image of it angled from the incoming perspective, and then taking the following image from the target perspective. In this case, the target perspective is a bird's-eye-view, therefore, the target image will show the chessboard plane normal to the camera lens.

4.1.2 Image Parsing(Traces to Requirements 3.1, 3.2)

This module will parse parking spaces during apropos conditions. It will then use a Pickle file in python to save the locations of those parking spaces for future use. Sorting of training images is to be done later via manual selection of the data gathered as a result of the data collection module.

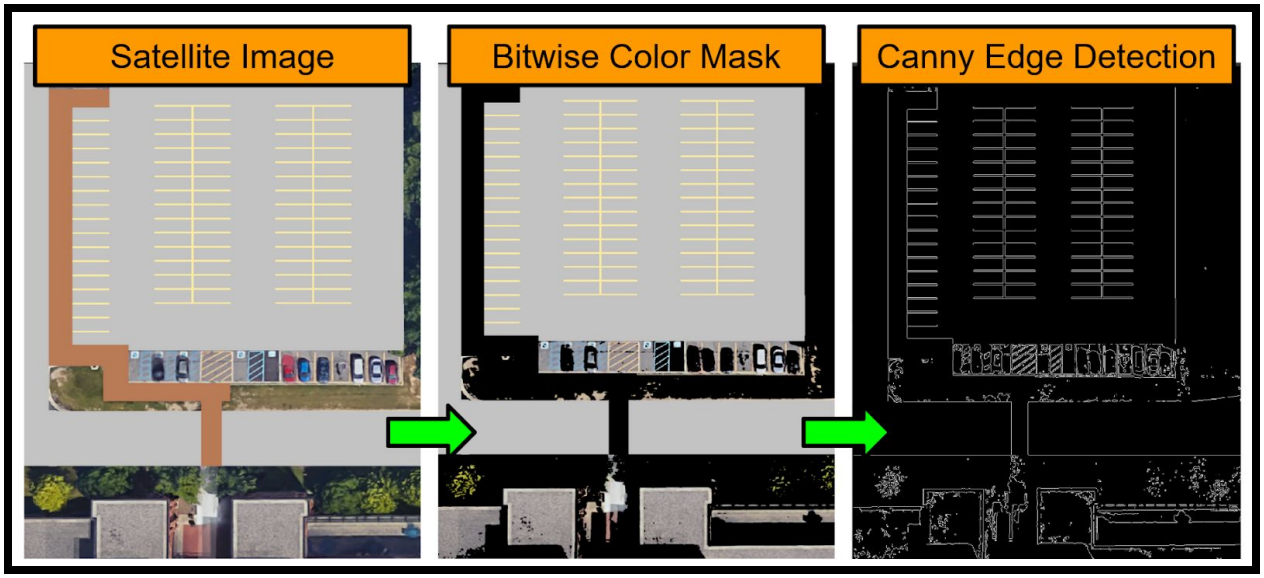

Figure 4.1.2: Image Parsing Examples of Color Masking & Edge Detection

4.1.2.1 Bitwise Color Masking

- Since most parking lot lines are standardly yellow or white, a bitewise color mask will first be applied to the input parking lot image as we want to focus on the detection of parking space boundary lines.

4.1.2.2 Canny Edge Detection

- After masking other colors in the image, canny edge detection shall be used to construct an “edge” image of our parking lot, using a Gaussian blur to smooth out lines detected from variations in color intensity. As this is dependent on light intensity, the image must first be converted to gray-scale.

4.1.2.3 Hough Line Transform

- Next the edges must be filtered, as we only need relatively vertical and horizontal lines in the image (since these are most likely the parking space boundary lines). To do so, Hough Line Transform shall be applied. Angle and distance parameters are variable and may be used to account for tolerance in the orientation of the lines sought out.

4.1.2.4 Region of Interest Filtering

- Finally, a region of interest (ROI) mask will be manually determined as to hide any areas of the image not relevant to parking lot lines. This way, we can ensure with a high degree of certainty that our program will detect and parse only horizontal and vertical lines used as the boundaries of parking lot spaces. This is useful as it is a dynamic way of getting pixel-location values of individual parking spaces which will be parsed as individual image for training, testing, and real-time prediction purposes.

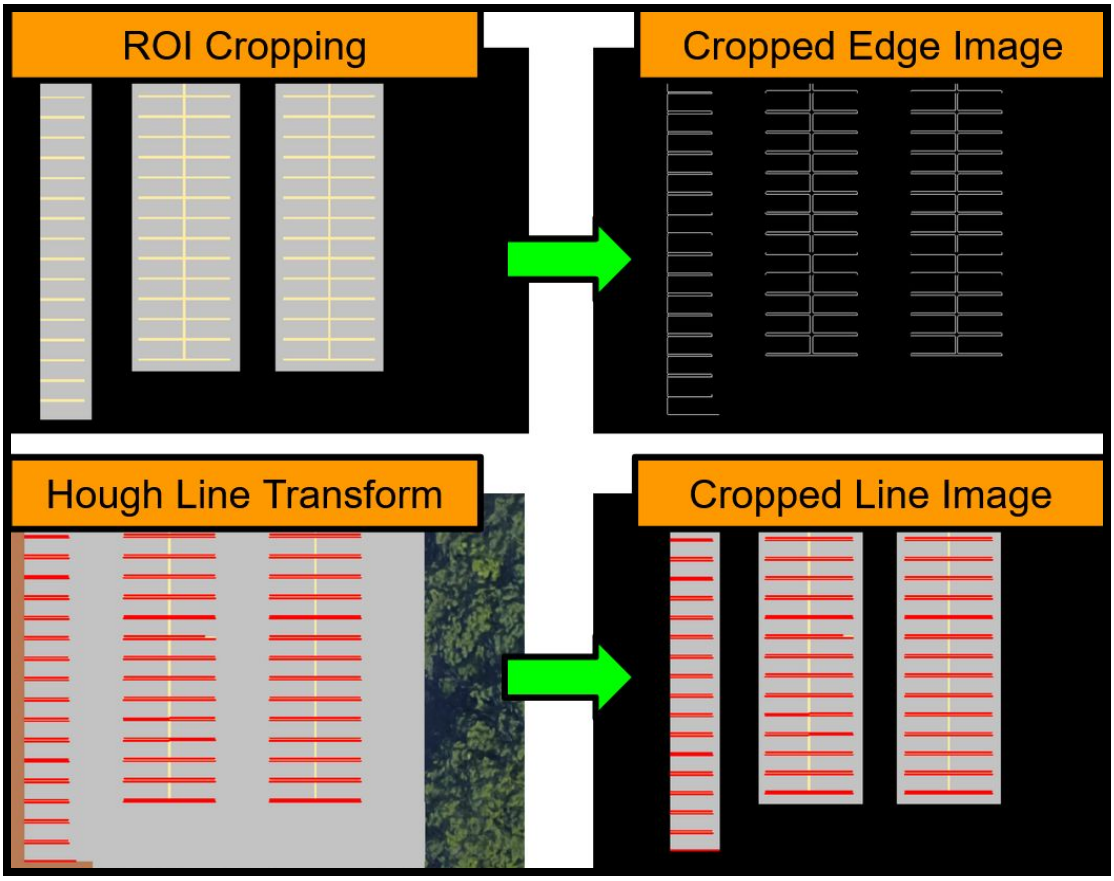

Figure 4.1.2.4: Image Paring Examples of ROI Filtering

4.1.3 Dataset Collection and Storage (Traces to Requirements 3.1, 3.2, 3.3, 3.4)

The CNN shall read in the parking spot input-images and make an inference as to whether a spot is vacant or occupied. To do so, a database must first be constructed from manuely sorting the collected data into two classified sets: one for vacant images, and the other for occupied images. Is important to note that the amount of occupied images shall be equivalent to the amount of vacant images in the training database as to not allow any sort of biasing when training the network.

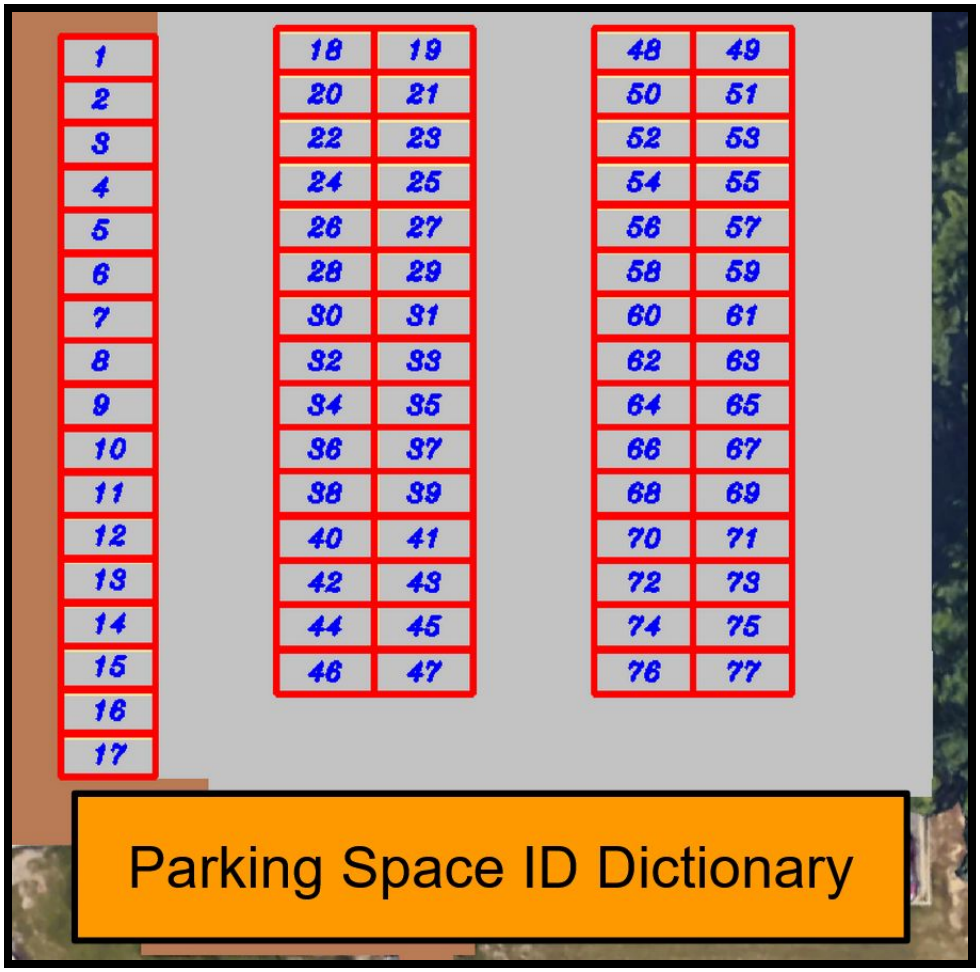

Figure 4.1.3: Parking Space ID Dictionary

4.1.3.1 Occupied Images (de-warping input image frame)

- Occupied images will be those determined to have any type of vehicles in them. Those vehicles will include trucks, cars, motorcycles, and mopeds.

4.1.3.1 Vacant Images (de-warping input image frame)

- All other images, found to not contain a vehicle, will be classified as vacant. Thus, these images will simply be of empty parking spaces only.

4.1.4 Convolutional Neural Network -- Training (Traces to Requirements 3.4.1-3.4.4)

With respect to training the convolutional neural network, we will need to construct two datasets composed of two different classes each. One dataset for training, and the other for testing to monitor the validation accuracy of the network after each epoch to help watch for overfitting or underfitting. Each dataset will have the vacant and occupied class of images that will be manually sorted as described earlier. From there, the training images will undergo one-hot encoding to label the images for the network which tells it whether it is a vacant or occupied image class.

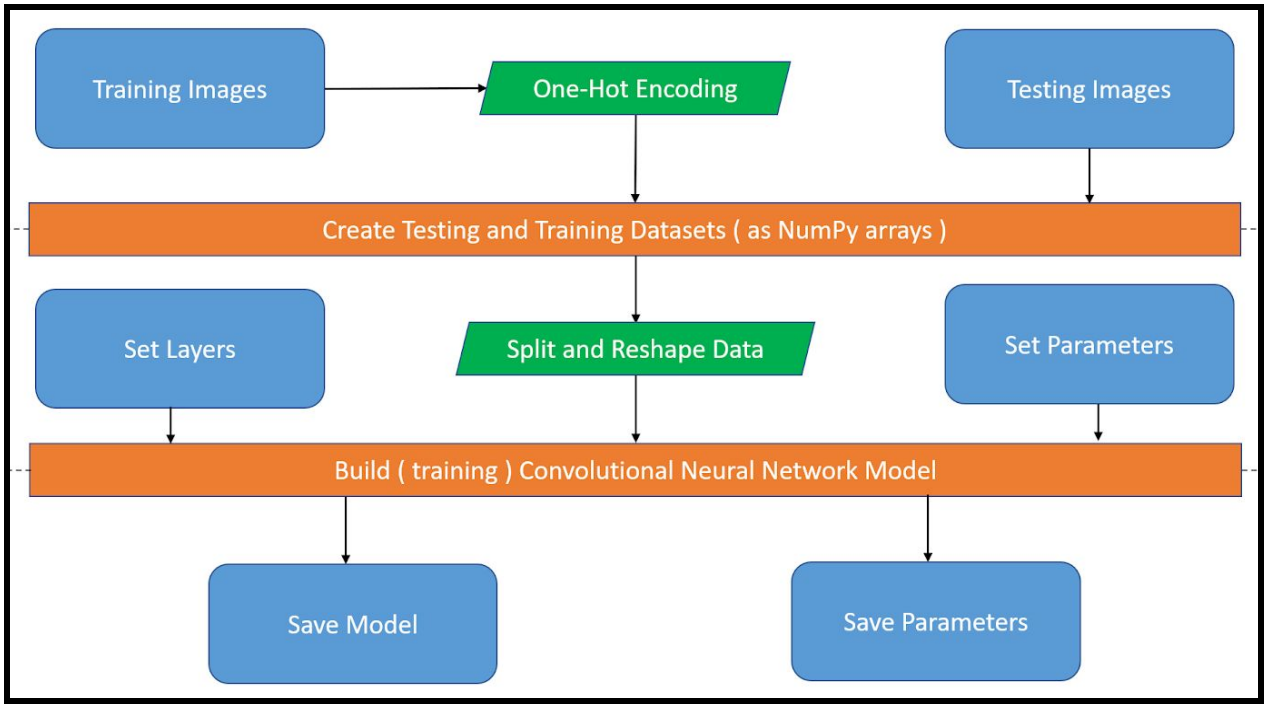

Figure 4.1.4: CNN Build (inference engine)

4.1.4.1 Model Architecture

- The specific architecture of the network will be determined later as the number of convolution and pooling layers will change with respect to how well the training goes. This will be much of a trial-and-error approach to structuring the model.

4.1.4.2 Model Implementation

- This will all be implemented in Python, using the OpenCV (for image processing utilities), and Keras (Tensorflow as the backend) libraries. The diagram below describes the flowchart for this process.

4.1.5 Convolutional Neural Network -- Inferencing (Traces to Requirement 3.4.1-3.4.4)

Once the architecture of the network has been determined, the dataset has been sufficiently gathered, and the model has been satisfactorily trained, we will be able to feed live images in real-time to make inferences from. The model architecture will be re-constructed, and the saved weights of the network will be loaded in upon startup. The following diagram illustrates the model inferencing module.

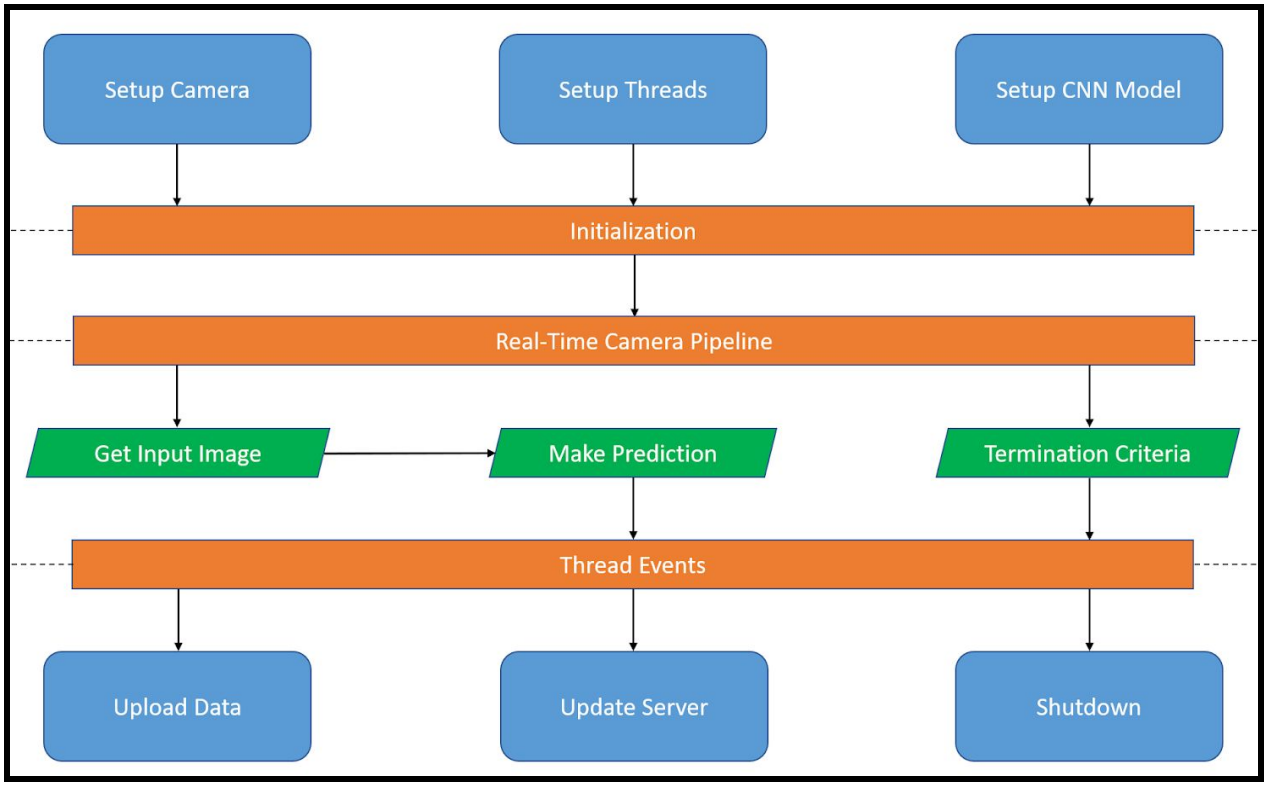

Figure 4.1.5: Real-Time Image Inferencing Overview

4.1.6 Hardware

4.1.6.1 Navida's Jetson Nano GPU (Traces to Requirements 3.1-3.4)

- The Jetson Nano has been chosen because it has an embedded Linux operating system for ARM64, as well as a significant amount of power and an on-board graphics accelerator that can be used to help meet the timing requirements of the system.

4.1.6.2 Samsung's 256 GB microSD card (traces to Requirement 3.1.5)

- Model No.: MB-ME256GA/AM

4.1.6.3 Waveshare's Wide Angle 4k Camera (Traces to Requirements 3.1.1-3.1.3)

- Chosen to meet the requirements of a large view angle and an adequate resolution at a range of 10-40m, 160 degree field of view, and its ability to support a suitable framerate.
- Model No.: IMX219-160

4.2 Network Module (Traces to Requirements 3.3, 3.5)

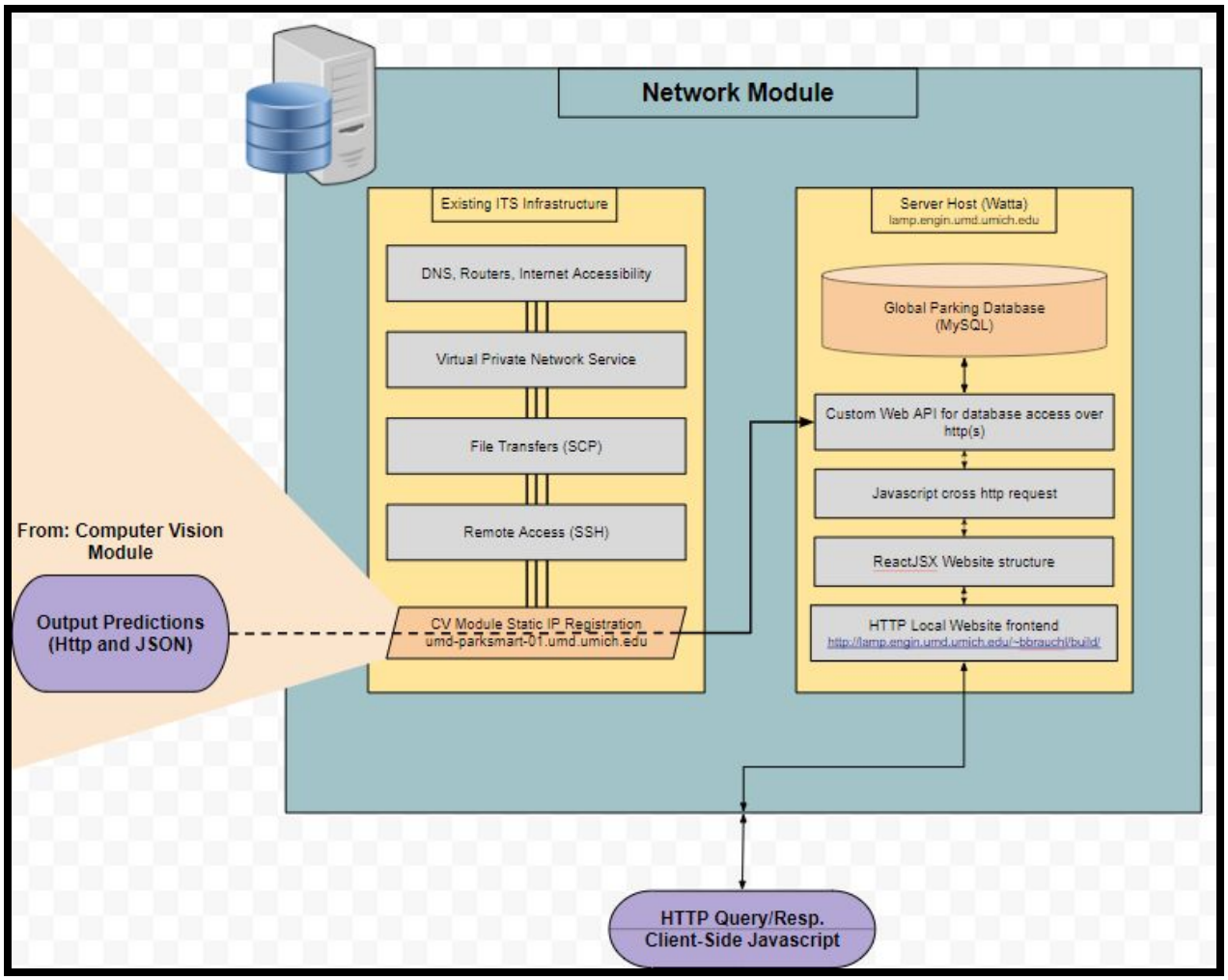

Figure 4.2: Low-Level Design of Network Module

4.2.1 Hardware: Server (Traces to Requirements 3.3.5, 3.3.6, 3.5.1, 3.5.2)
Virtual Server provided by ITS with a static IP and maintained by Professor Watta identified as *lamp.engin.umd.umich.edu*. This server is equipped with python3, MySQL, and PHP.

lamp.engin.umd.umich.edu - Professor Watta's Server
umd-parksmart-01.umd.umich.edu - Computer Vision Module

4.2.2 University Internet Infrastructure (Traces to Requirements 3.3.1, 3.3.4)

4.2.3 Software: MySQL (Traces to Requirements 3.2.x)

MySQL provides a well proven interface to a database structure, and is widely used in website backends to store data. For this application, MySql will be used as a backend for storing data on the server

The MySQL framework allows for database access permissions based on accounts. A framework will be set up such that only authenticated clients will be able to update entries in the database. User applications will only have access to read the data from the database.

To interface with the MySQL database with python, a custom API is written in python to transmit data over http using a post request. Data will be encoded as a JSON string and sent to the server in the “payload” parameter. It can then be unpacked by the PHP backend to be stored in the server.

One thing that this database must cover is the ability to expire data. This was done using timestamp columns in the table and manually expiring any current data in the table before updating.

Code for the MySQL table access is located at
https://github.com/bbrauchl/ParkSmart/blob/web-dev/ParkSmart-Webapp/api/pull.php

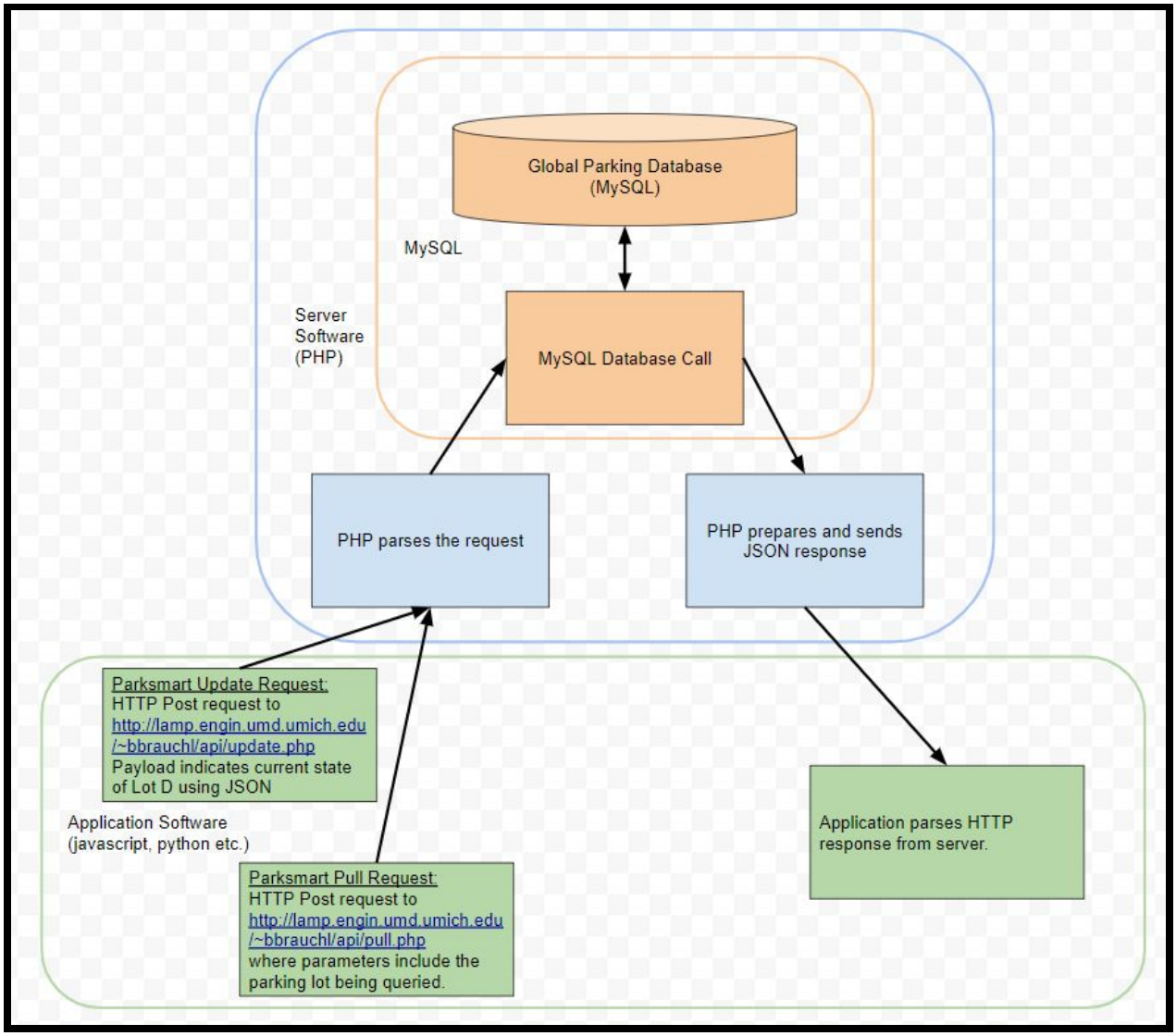

Figure 4.2.3: MySQL Database Transactions

4.2.4 Software: SSH (Traces to Requirement 3.3.4)

As with the Computer Vision Module, the server will be accessible by ssh for remote administration.

4.2.4 Software: PHP backend (traces to requirement 3.3, 3.5)

The web server will rely heavily on php to serve web pages according to the information in the mysql database. PHP will also function as the backend for the custom API and will process post requests to be committed into the database.

Two api calls are to be written: pull and update.
The pull call will request a specific parking lot name from the database. (at the time being, there is only one lot in the database “Lot_D”) or not specify a lot. The server will reply with a JSON object representing the current state of the selected lot. In the case

that no lot was selected, the call will report all parking lots stored in the database. Usage and more detailed documentation is located at http://lamp.engin.umd.umich.edu/~bbrauchl/api/pull.html. Please note that this page is only available within the University of Michigan's network.

The second API call is Update. This call is designed for the camera modules to push updates to the server, and functions similarly to the pull update. For more information, please visit http://lamp.engin.umd.umich.edu/~bbrauchl/api/update.html. Again, this page is only available within the University of Michigan's network.

4.2.6 Software openweathermap API (Traces to Requirement 3.5.2)

To get information about visibility, use the openweathermap API. This is a http based api to report the weather in a selected area. This information is returned in json and can be parsed to determine various weather conditions. For more information please visit openweathermap.org

4.2.7 Software: JavaScript, Libraries: ReactJS, xHttpRequest (Traces to Requirement 3.5.1)

JavaScript will be used to help format the web page for end user use, making the webpage more dynamic and reloading information when relevant. Specifically, the ReactJS framework will be used to facilitate features and create a simple yet effective frontend web application for the system. Finally, xHttpRequest is a javascript library specifically built for performing http requests in javascript. This makes it possible for frontend javascript to make calls into the backend of the server, fetching information in the same way as any other system application.

The frontend of the web interface was created almost exclusively using ReactJS. The only exception being xHttpRequest for the weather API, and the Parksmart API backend ReactJS makes it simple to create abstract HTML elements that have more control over events and can be interacted with in a javascript source file, and makes for a simple rendering of the parking lot:

```
ReactDOM.render(
  ( <div>
      <h1>ParkSmart</h1>
      <h2>Lot D</h2>
      <ParkingLot lotName="Lot_D" />
    </div>
  ),
  document.getElementById('root')
);
```

This outputs a user display that will color code parking spaces corresponding to their current state, setting the opacity as a function of confidence. In other words, spaces that are determined as occupied will appear red to the user and spots that are

determined as vacant will be colored green, allowing the user to easily tell where open parking spaces are, and providing an easy way to query the current state of the parking lot.

Figure 4.2.4: User Web Interface Frontend

The Javascript application is located under the ParkSmart Webapp-react directory on GitHub.
https://github.com/bbrauchl/ParkSmart/tree/web-dev/ParkSmart-Webapp-react

4.2.8 Hardware: Wireless NIC (Traces to Requirement 3.3.1, 3.3.3)
Vendor: Waveshare
Model No.: AC8625
Cost: $25
Wireless card for use with the Jetson Nano.

4.2.9 Software: Networking Linux Drivers (Traces to Requirement 3.3.3)
Pre-installed Linux drivers to access networking and manage internet connections.

4.3 User Application Module (Traces to Requirement 3.5)

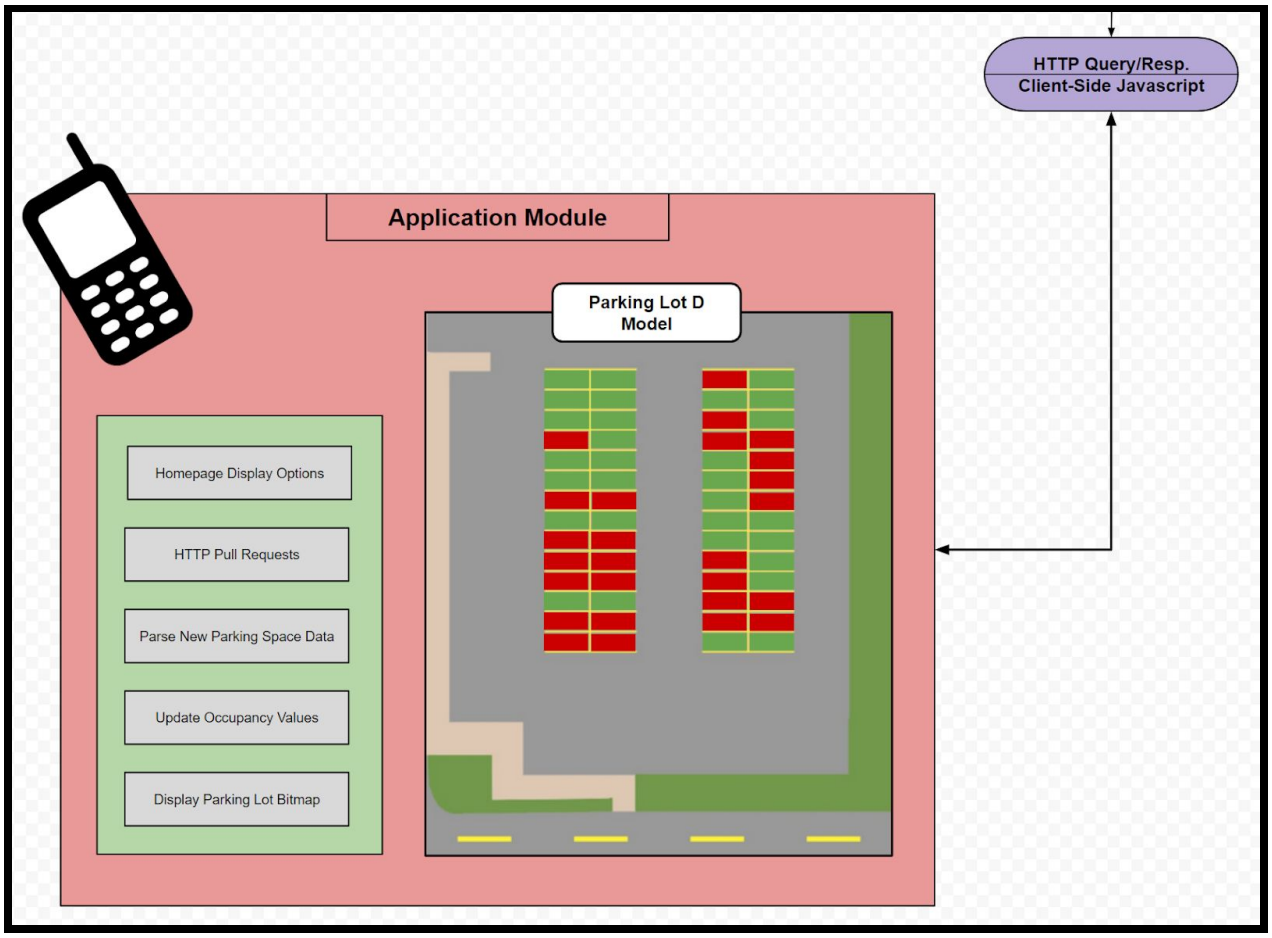

Figure 4.3: Low-Level Design of Application Module

For our system to be useful, it has to be accessible to some user base. In this case, the user base is the students and faculty at UMD. In order for them to be able to access the parking space data, we're creating a mobile application hosted on the UMD server afforded to us.

4.3.1 Software: App (written in Java, in Android Studio IDE)

The app is fairly straightforward; it requests data directly from our UMD hosted server (which will be stored in a comma separated list sent from the jetson, identifying pre-numbered spots as occupied or vacant). The UI of the app displays a graphic of the parking lot (with a menu to select which lot), and then fills vacant spots blue and occupied spots red

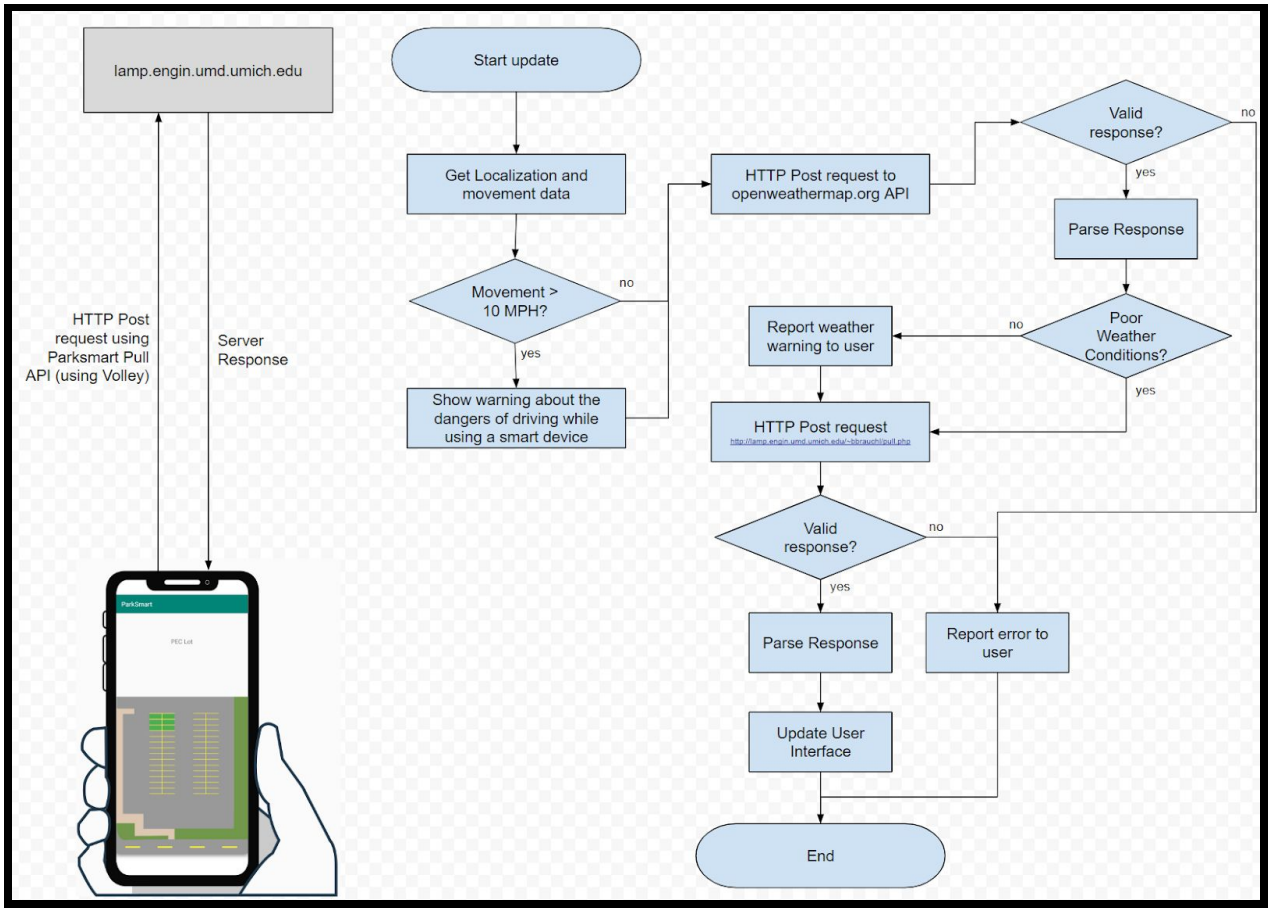

Figure 4.3.1: Application update

4.4 Small Scale Model

Due to the lack of HPEC rooftop access, the mockup system was designed as a small-scale wooden building (figure.4.1) to set the Jetson Nano on top of, overlooking a print out of lot-D (figure-4.4.2, & figure-4.4.3) at the University of Michigan-Dearborn's campus.

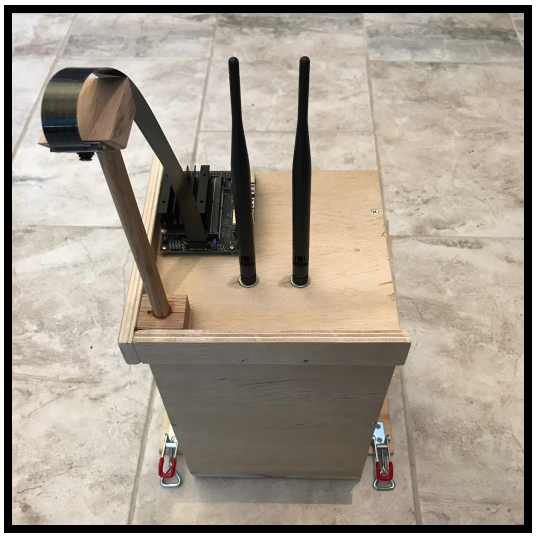

Figure-4.4.1: Camera and Jetson Nano mounted on top of HPEC building (mockup)

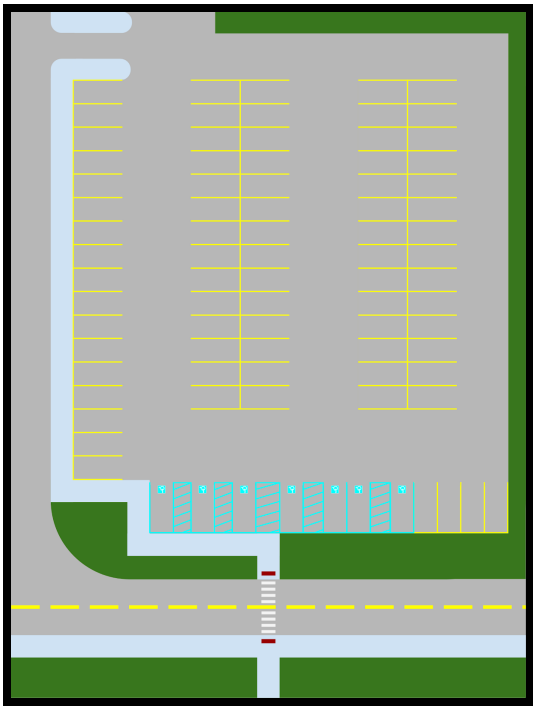

Figure-4.4.2: Parking Lot-D Animated Image

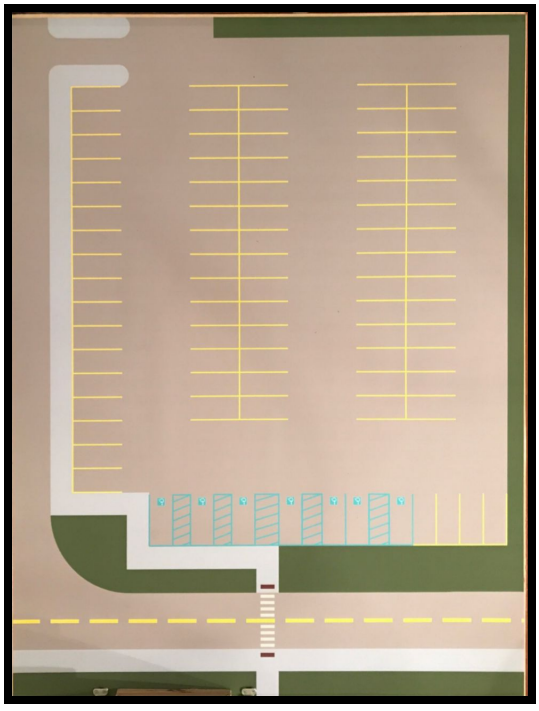

Figure-4.4.3: Animated Image Print (3'x4')

The following two figures provide the final testing environment setup for this project.

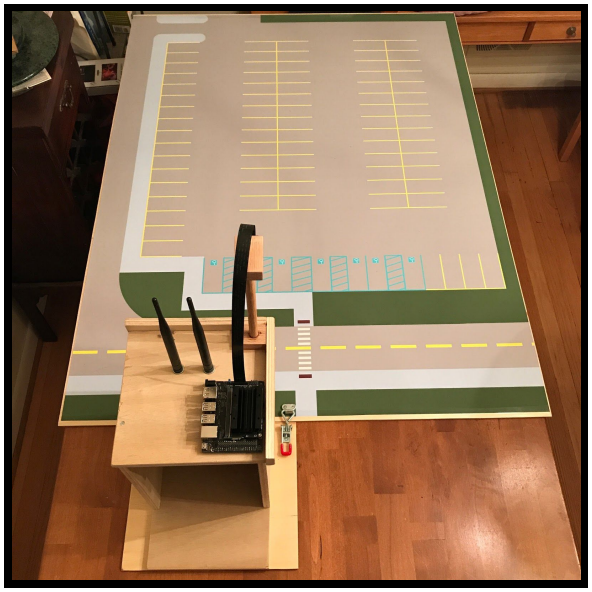

Figure-4.4.4: Full System (top view)

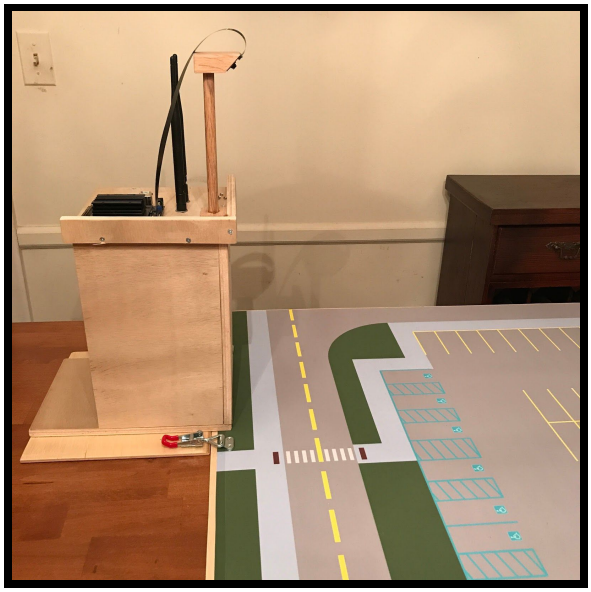

Figure-4.4.5: Full System (side view)

5. System Testing

5.1 Testing Procedures

The following sections outline the testing methods for the various tests that had to be run on the system.

5.1.1 Computer Vision Module Test Plan

5.1.1.CV_F1. Image Processing Pipeline: Requirement 3.4.1, 3.4.2, 3.4.3, 3.4.4

5.1.1.CV_F1.1. Test Case Requirement:

3.4 The system must be accurate in counting parking spots.

3.4.1 The Computer Vision Module must infer the correct state 75% of the time. (P(Calculated == Actual) ≥ 75%).

3.4.2 If a vehicle is parked in multiple parking spaces, the inference must classify all spaces as occupied.

3.4.3 The inference must be run at a rate of at least once a minute. (inference update rate ≥ .016Hz)

3.4.4 The inference must classify small pedestrian vehicles such as motorcycles as being occupied spaces. (Edge-case classification > 50%)

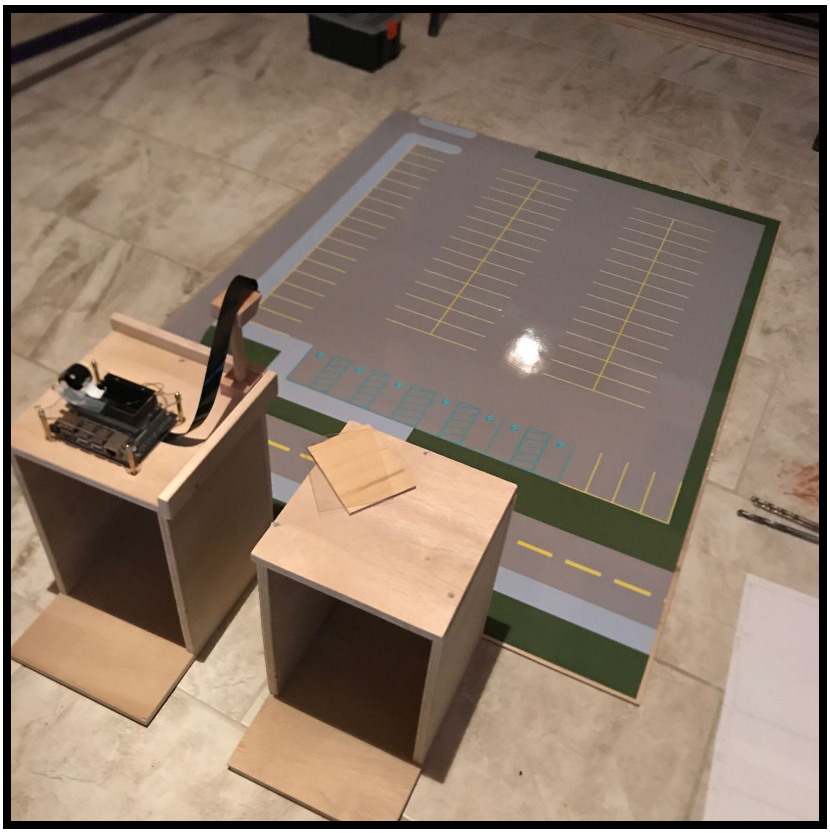

Figure 5.1.1.CV_F1.1: Test case setup

5.1.1.CV_F1.2. Test Case Description:
The image pipeline test is targeted at the inference behavior of the computer vision module. This test will exercise the model using a varied set of input data to evaluate the performance of the computer inference. This will be done indoors because rooftop access is still out of reach for the project group.

5.1.1.CV_F1.3. Test Environment and Conditions:
Location: Garage.
Environment: Lab Setting

- well lit, with lighting changing overtime.
- No specific temperature, humidity, wind, shock, or vibration parameters.
- Imaging angle over the lot will be consistent from inference to inference.
- The Computer vision module will not be attached to the ParkSmart Networking component.

5.1.1.CV_F1.4. Input Data Set:
The input data for this test is the pictures that the camera captures while overlooking the small scale model. The data will be collected during the run of the test. Lighting conditions and the states of the parking lot will be changed during the runtime of the test to provide a good set of sample data for testing the neural inference.

5.1.1.CV_F1.5. Expected Data Values and Results:

The system's output should be a list of occupied and vacant parking spaces, with 3 out of every 4 frames being classified accurately.

CV_F1	
Classification Accuracy (%)	Accuracy will be measured by counting the number of inferences with all spots classified correctly compared to the total number of samples. Excellent: >95% Good: >90% FAIL : <90%
Inference Rate (Hz)	The rate at which the inference is able to run. Excellent: < 60s Good: < 90s FAIL: >90s
Edge Case Classification (%)	Edge conditions such as double parked cars or motorbikes are classified correctly > 50% of the time. Excellent: >75% Good: >50% FAIL: <50%

1.4.CV_F1.6. Test Procedure:

Steps:

1. Setup poster on the ground and align mock building with the poster
2. Power on the Jetson Nano and open relevant file folders in terminal
3. Plug in HDMI connection to display feedback and execute program
4. Monitor the parking space line parsing, and make sure there are no outliers
5. If collecting data: randomize cars on poster board and store image frames
6. If running trained network: run finalized image processing pipeline, shuffle hot-wheels cars around the board, and monitor CNN inferencing
7. End test after 1000 frames are evaluated

Data Collection:

- Pass/Fail based on accuracy
    - Logging accuracy of each frame processed
- Images collected for training
    - Moving hot wheels cars around on poster board

5.1.2 Network Module Test Plan

The network module must be tested to ensure that data is collected and distributed in an appropriate matter. The focus on these tests will be to test the functions of the server as well as the network infrastructure to meet the project needs. Some testing was done on campus before the shutdown associated with COVID-19, and was carried out in person. Later tests had to be done remotely. VPN access and video conferencing has largely negated the now-remote nature of network testing, with the only minor exception being connection speeds. This is due to the added layer of the VPN with every request, however it was determined that this was still sufficient to meet requirements (as reported below).

Before breaking from in-person meetings, connectivity to the university wireless network was tested (Test Case N_F1). After classes were moved to online, Test cases N_F2 (Server API Calls) and N_S1 (Connectivity and Error Reporting) were tested. Using VPN.

5.1.2.N_F1. Computer Vision Connectivity: Requirement 3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5, 3.3.6

The aim of this test is to test the connection from the computer vision module to the central ParkSmart server. This is to test that a working data link has been established and that the product can communicate as designed.

5.1.2.N_F1.1. Test Case Requirements:

- 3.3.1 The network must achieve a throughput of at least 3.2 MBit/s from the Computer Vision Module to the server to report data. This number is based on the expected camera update rate layed out in requirement 3.1.3 and the resolution of an image. (throughput $\geq$ 3.2MBit/s).
- 3.3.2 The network must guarantee packet arrivals to the destination network (TCP).
- 3.3.3 The system must interface to existing wireless topography. ($\exists$ IEEE 802.11ac compatibility).
- 3.3.4 External clients must have remote access to the server via network. ($\exists$ external network interface).
- 3.3.5 Any clients connecting to the server must be authenticated ($\exists$ Client Authentication).
- 3.3.6 Any confidential or private data must be encrypted ($\exists$ Link Encryption)

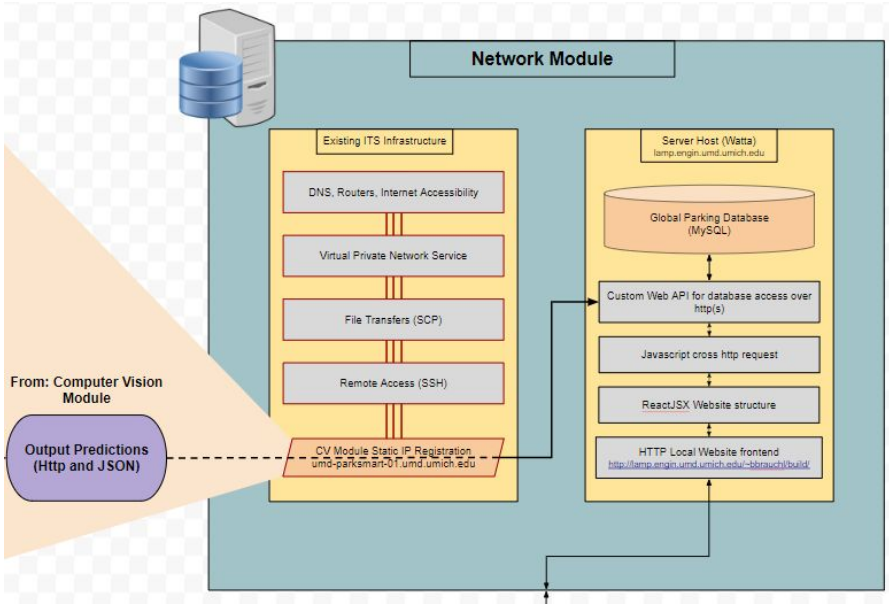

Figure 5.1.2.N_F1: Network Connectivity

5.1.2.N_F1.2. Test Case Description:
One of the most crucial parts of the entire system functioning is the existence of communication between the three modules. This communication is bundled with the backend server in the network module. The main purpose of the test is to ensure that the computer vision module will have access to the network and network resources, as well as having remote access to the computer vision module, in the case that it was placed on the roof.

5.1.2.N_F1.3. Test Environment and Conditions:
Location: ~~IAVS~~ VPN
Environment: Indoor, Lab setting where the device can be closely monitored throughout the test.
Network Hostname: umd-parksmart-01.umd.umich.edu (assigned hostname from ITS)
IP Address: 141.215.192.40

SSH (secure shell) used to test remote access.
Wireshark used to ensure that packets are encrypted over this remote administration link.
Non-administration traffic is not sensitive and hence has no need for encryption.
SCP (secure copy) used to transfer files and test transfer speeds.

5.1.2.N_F1.4. Input Data Set:
The input dataset consists of the University internet Infrastructure, including routers, DNS servers, and access to the outside internet.

5.1.2.N_F1.5. Expected Data Values and Results:

N_F1	
Device is able to connect and authenticate on UMD-Secure (true/false, Received IP address)	Checked in the connection information of the settings on the Jetson board
TCP Protocol in use (true/false)	Use of wireshark to examine the protocols that are being used by the Jetson.
Remote Access via SSH and encrypted, and passwords are used for authentication (true/false)	Ensure access remotely using SSH and is authenticated Ensure connection is encrypted with wireshark
Data Transfer Rate (MBit/s)	Use of SCP to copy a file and record the reported transfer speed. Expected ranges: Excellent: 3.2MBit/s+ Good: 2.6-3.2MBit/s OK: 2.0-2.6MBit/s FAIL: < 2MBit/s

5.1.2.N_F1.6. Test Procedure:

Steps:

1. Boot the Jetson Nano using NVidia Image and networking card installed
2. Connect to UMD-Secure using network-manager
3. Verify VPN access to University from jetson
4. Open Chromium on the Jetson to confirm access to google.com
5. "ssh jetson@umd-parksmart-01.umd.umich.edu" from another PC to confirm ssh remote access.
6. Note that SSH asks to authenticate the connection to the server
7. Use SCP to copy a file from the other PC to the jetson and note the data rate.

Data Collection:

- **Network Speed**
- **Several checks to ensure features are enabled/disabled**
- **Assigned IP address and Hostname**

5.1.2.N_F2. Database Retrieval and Commit: Requirement 3.5.1, 3.3.7, 3.3.8

5.1.2.N_F2.1. Test Case Requirement:

3.3.7 The web server shall communicate with clients over apis functioning on the http protocol (api protocol == http)

3.3.8 The web server must log information regarding parking to a database for historical and searchable data.

3.5.1 The web/app interface must service requests at all times of day.

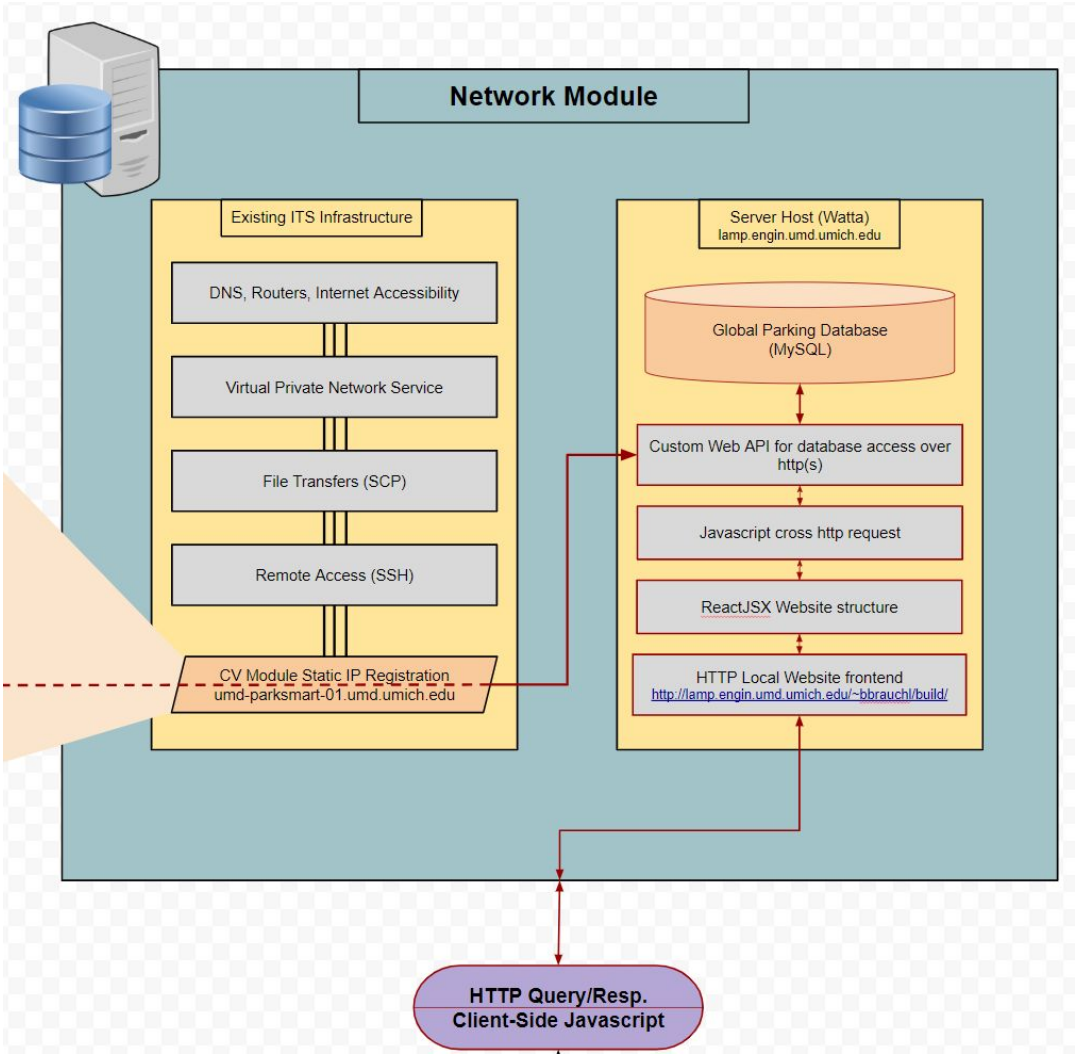

Figure 5.1.2.N_F2: Database Interface

5.1.2.N_F2.2. Test Case Description:
The second main function of the networking module is to handle and store data reported from clients, as well as send responses to users about the current state of the database. This test aims to evaluate the function of the calls to the server to store and retrieve data from the parking lot database.

5.1.2.N_F2.3. Test Environment and Conditions:
Location: ~~On Campus~~ VPN
Environment: Python, Java, and Javascript http request libraries
Network Address: lamp.engin.umd.umich.edu/~bbrauchl/api/<api name>.php
SSH (secure shell) used to access the server.
Custom APIs.

5.1.2.N_F2.4. Input Data Set:

Manually generated data corresponding to how data will authentically be sent by the computer vision module. This data will then be pulled by the database to check that the data is identical to the data that was committed.

5.1.2.N_F2.5. Expected Data Values and Results:

N_F2	
HTTP and TCP are used in for the underlying protocols during the web API calls (true/false)	Use Wireshark to confirm the protocol underlying the web api calls. Pass: HTTP and TCP FAIL: Any other protocol
Requests report the current state of the database, as compared to a known update (% correlation)	Push and then pull a known database to/from the database to ensure that they are identical. Result Ranges: Excellent: 100% Passing 99-100% FAIL < 99%
Access to the server (%)	Try the commands repeatedly at times of high and low network traffic to check the access to the server, over a course of 8 hours Result Ranges: Excellent: 100% Passing 99-100% FAIL < 99%

5.1.2.N_F2.6. Test Procedure:

Steps:

1. Establish a web page on the server
2. Using custom Python and PHP interfaces, perform an update to the server database using the ParkSmart.update() function. The update will show up in the webpage mentioned above in order to to pass this test.

3. Calling a second API from python, load the current status of the database back to the Jetson. This will be compared to the current state of the database to confirm that the fetch was successful.
4. Repeat steps 2 and 3 several times during the day to ensure that access remains functional.
5. A python script will be run to test accesses to the server periodically for an extended period of time (4 hours)

Data Collection:

- **State of current database**
- **State of database after update**
- **State of pulled data**
- **Percent of the time that access is available**

5.1.2.N_S1. Connection Error Reporting: Requirement 3.5.2, 3.5.2.1, 3.5.2.2, 3.5.2.3

5.1.2.N_S1.1. Test Case Requirement:

3.5.2 The web/app interface must report errors and warnings to the user.
3.5.2.1 The web/app interface must report Computer Vision Module Dropout errors
3.5.2.2 The web/app interface must report visibility warnings
3.5.2.3 The web/app interface must report invalid API calls as the response to the html requests.

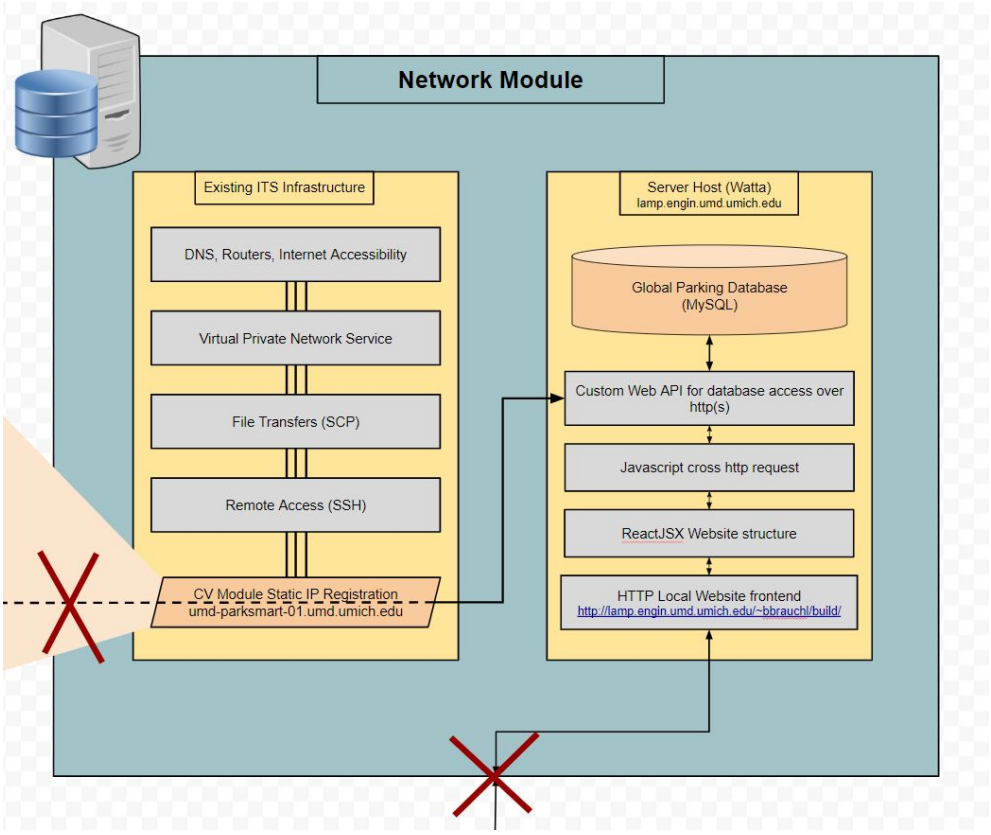

Figure 5.1.2.N_S1: Fault Injections

5.1.2.N_S1.2. Test Case Description:

Errors will inevitably happen in any system. The main areas that this system can experience errors are in the following forms:

1. Computer Vision Module Dropout, where a Computer Vision module drops out of the network
2. Visibility issues with weather
3. Improper usage by users

This test is designed to observe the network response to these events

5.1.2.N_S1.3. Test Environment and Conditions:

Location: ~~On Campus~~ VPN connection
Environment: Web Browser
Network Address: lamp.engin.umd.umich.edu/~bbrauchl/build/

5.1.2.N_S1.4. Input Data Set:

For this test, faults will be manually inserted into the system by manually stopping a producer of parking lot data, intentionally calling API calls incorrectly, and spoofing weather API responses to mimic poor conditions. These will all act as test vectors to observe the response of the system.

5.1.2.N_S1.5. Expected Data Values and Results:

N_S1	
Poor Weather Conditions are reported in the user interface (true/false)	When the visibility is low, or there is extreme weather, the results of the inferences will be less accurate. This warning should be reported to any users of the system.
Computer Vision Module Dropout is detected. (s)	In the case where a computer vision module drops out of the network, and the database contains old data, that data should not be reported to users. This will be measured in seconds from when the module is dropped out. Since parking spaces take some time to update, the following conditions have been defined to avoid heavy bandwidth: Excellent: Dropouts are detected in < 3 minutes (120 seconds) Good: Dropouts are detected in < 5 minutes (300 seconds) FAIL: Dropouts are detected in > 5 minutes
Programming the API with incorrect access reports the error.	When the server API is incorrectly called, the response will contain information about the nature of the error and potential steps to resolve.

5.1.2.N_S1.6. Test Procedure:

Steps:

1. Push a single update
2. Use Python to poll the api function and time how long the data is reported after the update
3. Next, test the web-page in rainy or foggy conditions.

4. Reload the ParkSmart Website and look for a visibility warning.
5. Next, test the Computer Vision Modules with incorrect arguments to the web API
6. Check the response from the web server to look for an error message.

Data Collection:

- **State of current database**
- **State of database after improper update**
- **Time taken for the database to recognize a Computer Vision Module drop out**

5.1.3 User Application Module Test Plan

The user application module must be able to pull data from the server and accurately record the date on the user interface. These tests are meant to test the overall function of the application as well as the accuracy of the data to which it is reporting to users. Due to the school shutdown some features and tests had to be cut short or had to be limited and done remotely as they needed the VPN service in order to receive data from the server. Due to this these tests had to be done via emulation and not directly on hardware.

5.1.3.A_F1. Data Reporting: Requirement 3.5.1,3.5.2

The main focus of this requirement is to test the ability of the app to report data efficiently and accurately to the user.

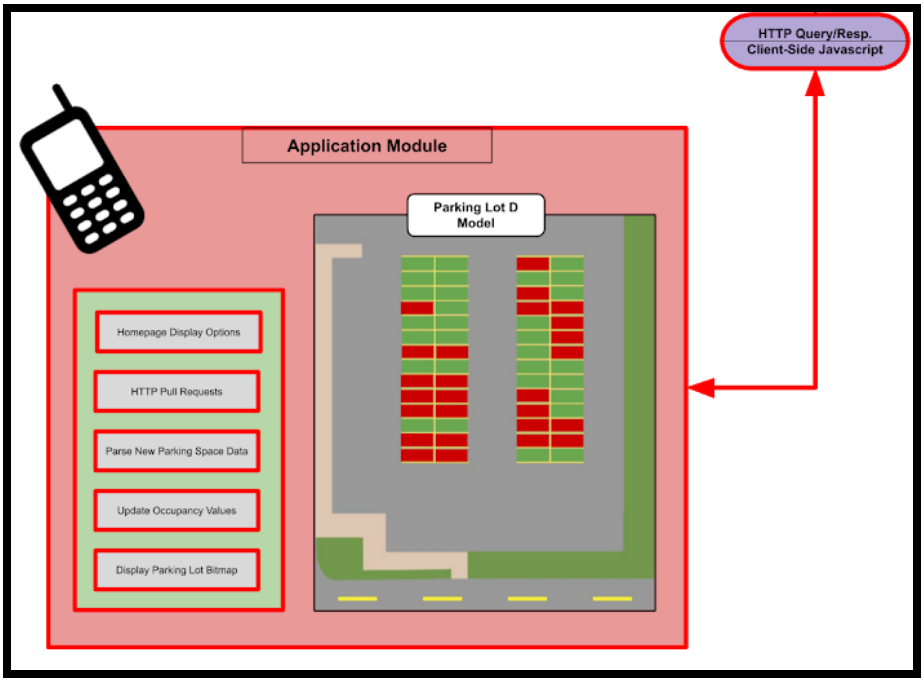

Figure 5.1.3.A_F1: Data Reporting to user interface

5.1.3.A_F1. Test Case Requirement:

3.5.1 The app/web interface must service requests at all times of the day

3.5.2 The app/web interface must report problems to user.

5.1.3.A_F1. Test Procedure:

Steps:

1. The application will be started on an android device and on the school's wireless network.

2. The application will be provided with data (either real or fake) in JSON data containing parking data.
3. The data will be parsed by the application and written to the display.
4. If the application output interface will be compared to the input data to verify correctness.
5. Next the data will be removed, and the same trial will be run.
6. The application must report a data error to pass testing.
7. Last, data will be provided that has a timestamp that is out of valid range.
8. The Application must report a data invalid warning to pass testing.

A_F1	
mapping of spots to image	Parking spots are expected to be colored such that spots marked as vacant are colored green, and spots marked as occupied are marked red. Yellow is to represent invalid data. Excellent: 100% of the parking spaces are colored red, green, or yellow. FAIL: <100% of parking spaces are colored.

5.1.3.A_S1. Distracted Driver: Requirement 3.5.2,3.5.3

5.1.3.A_S1.1. Test Case Requirement:

3.5 The system must distribute information to users.

3.5.3 The app interface must provide warnings to users who are likely driving (warnings when speed > 10mph)

5.1.3.A_S1.2. Test Environment and Conditions:

Location: On campus

Environment: Inside a moving vehicle

- Latest application version installed
- No specific temperature, humidity, wind, shock, or vibration parameters.

5.1.3.A_S1.3. Input Data Set:

Inputs include locational and acceleration data of the phone's internal sensors, purposely set to the case in which the app should report warnings and errors.

5.1.3.A_S1.4. Expected data values and results:

The application should display a warning when the driver exceeds 10mph. It should not allow the user to navigate the app without dismissing the warning.

A_S1	
User Notifications - Android App	The android app is expected to show a notification/warning when the user may be driving. The application should also provide notifications in the case of poor weather or low visibility to warn users of potential poor results. Excellent - All warnings/notifications will be displayed in the Android app given the proper conditions (weather, movement, etc). Fail - warnings and notifications are not displayed in the Android app.
User Notifications - Web App	The web application should provide notifications in the case of poor weather or low visibility to warn users of potential poor results. Excellent - All warnings/notifications will be displayed in the Web App given the proper conditions (weather, etc). Fail - warnings and notifications are not displayed in the Web App.

5.1.4. Full System Testing

5.1.4.S_F1. System Output: Requirement 3.4

5.1.4.S_F1.1. Test Case Requirement:

3.4 The system must be accurate in counting parking spots.

3.4.1 The Computer Vision Module must infer the correct state 75% of the time. (P(Calculated == Actual) ≥ 75%).

3.4.2 If a vehicle is parked in multiple parking spaces, the inference must classify all spaces as occupied.

3.4.3 The inference must be run at a rate of at least once a minute. (inference update rate ≥ .016Hz)

5.1.4.S_F1.2. Test Environment and Conditions:

Location: ~~On campus~~ Remote via VPN

Environment: Lab Setting with Scale Model Parking Lot

- well lit, with lighting changing overtime
- No specific temperature, humidity, wind, shock, or vibration parameters.

5.1.4.S_F1.3. Input Data Set:

The input data for this test is the pictures that the camera captures while overlooking the small scale model. The inputs to the system will consist of the input frames from the Computer Vision module, as well as requests from users and responses from external APIs.

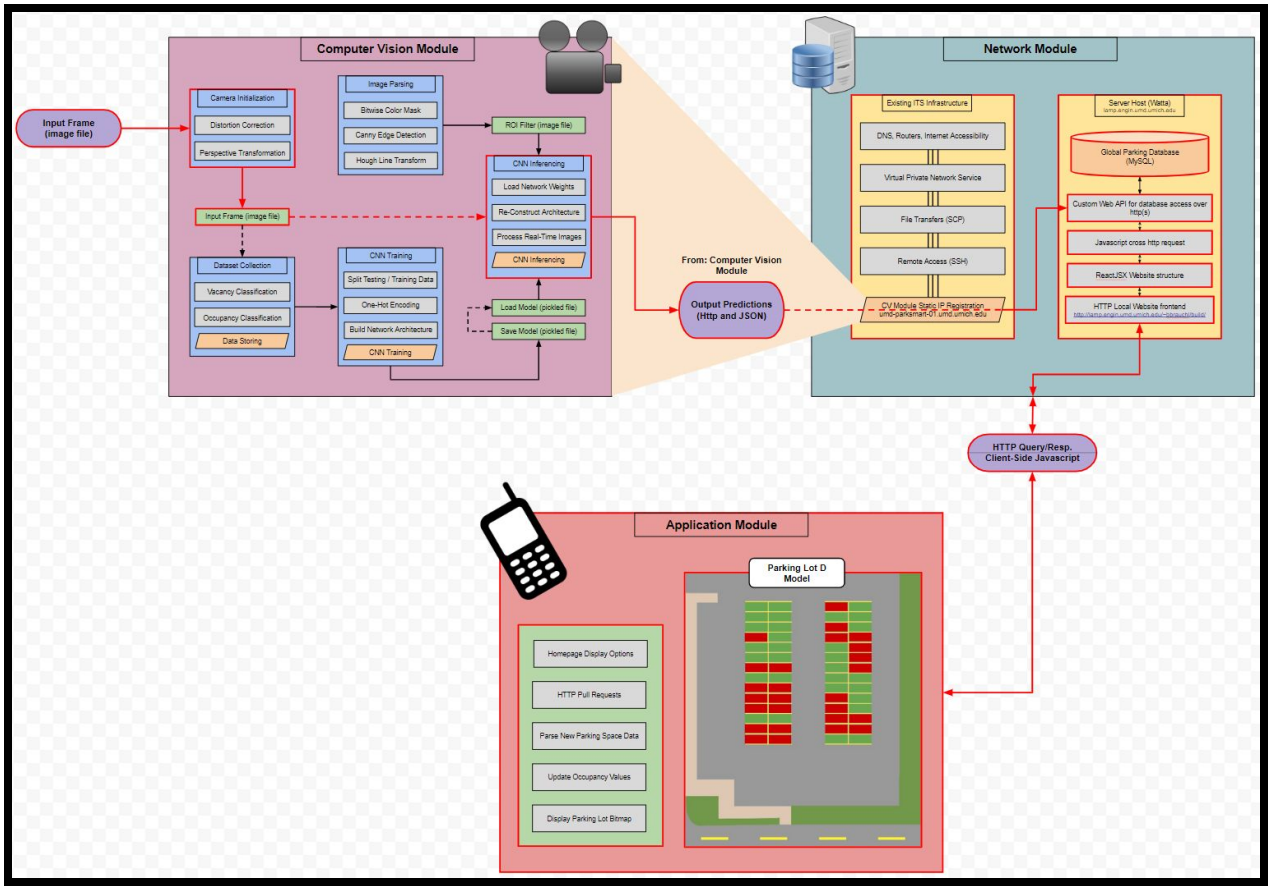

Figure 5.1.4.S_F1: System Module Diagram

5.1.4.S_F1.4. Expected data values and results:

The system's output should be a list of occupied and vacant parking spaces, with 3 out of every 4 frames being 100% accurate. The system should be able to identify spots as occupied if they are filled by motorcycles, double parked cars, etc.

S_F1	
Parking lot status	The application is expected to report the number of cars that are currently in the parking lot model as well as their locations. 3 out of 4 updates should be 100% correct.
User Notifications	The app should also notify the user if there are complications with the system, such as poor weather that will affect the accuracy of the predictions.

5.1.4.S_F1.5. Test Procedure:

Steps:

1. The entire system will be assembled along with a scale parking lot model in ~~the IAVS~~ Roderick's House.
2. Model cars are placed manually and the system is allowed time to update.
3. The output is compared to the known input to assess the correctness of the model.
4. Repeat for several trials with factors like small vehicles, motorbikes, and cars in more than one spot.

Data Collection:

- Pass/Fail based on accuracy
    - Log of accuracy of each frame processed
- Images collected for training

5.1.4.S_S1. System Update Latency: Requirement 3.4

5.1.4.S_S1.1. Test Case Description:

3.4 The system must be accurate in counting parking spots.

3.4.3 The inference must be run at a rate of at least once a minute. (inference update rate ≥ .016Hz)

5.1.4.S_S1.2. Test Environment and Conditions:

Location: On campus

Environment: Lab Setting with Scale Model Parking Lot

- well lit, with lighting changing overtime
- timer available, to measure total update time
- No specific temperature, humidity, wind, shock, or vibration parameters.

5.1.4.S_S1.3. Input Data Set:

The input data for this test is the timestamp of a given frame, taken of the small scale parking lot model.

5.1.4.S_S1.4. Expected data values and results:

The system should process the frame, decide on vacancies for each spot, send this vacancy list to the server, pull this server data down to the app, and display it properly within the app; this process should take 60 seconds or less.

S_S1	
Update Rate (hz)	The rate of updates making it to the application will be measured. It will be measured from the application
Update Latency (s)	The amount of time it takes the system to make an inference, publish that data, and the data to be available on the application.

5.1.4.S_S1.5. Test Procedure:

Steps:

1. The Computer vision software will be started with an empty database.
2. The web page will be loaded after exactly 1 minute.
3. If data loads properly, then the system passes the latency test.

Data Collection:

- Pass/Fail based on timing
    - Log of measured times for each frame to update the app recorded.
- If needed for troubleshooting, time measurement of each step will be recorded.

5.2 Testing Results

5.2.1 Computer Vision Module Test Results

5.2.1.1 Camera Calibration and Perspective Transformation

The following image (figure-5.2.1.1) shows the result of the perspective transformation program running off of the Jetson Nano. The image is of the physical parking lot board, and is taken from a near 45-degree angle. Regardless, the image appears as though it is taken from a bird's eye view as a result of the homography matrix used for the perspective transform.

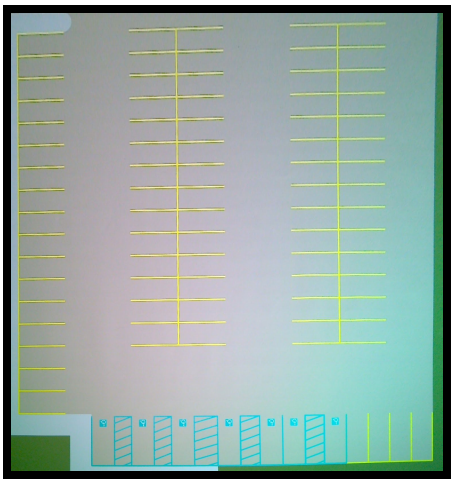

Figure 5.2.1.1: Perspective Transform Test Result

5.2.1.2 Canny Edge Detection

Next, canny edge detection was used to the transformed image, as shown in figure-5.2.1.2.

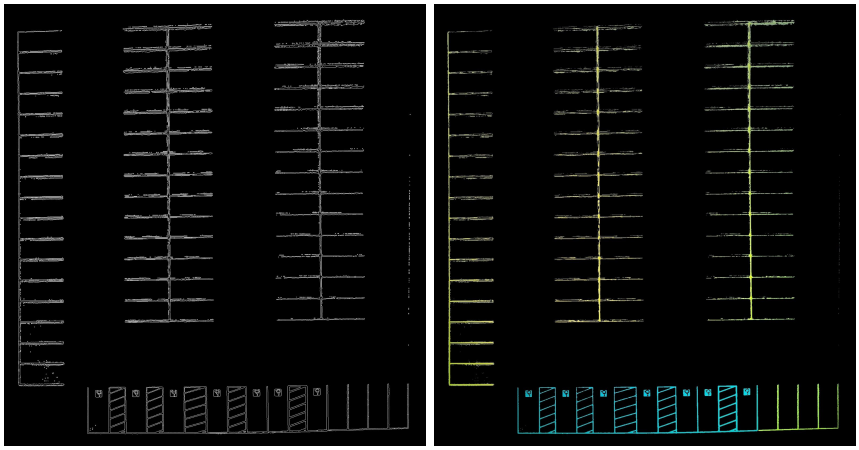

Figure 5.2.1.2: Canny Edge Detection Test Result

5.2.1.3 Parsing of Parking Space Clusters

Figure-5.2.1.3 illustrates the parsed region-of-interest image from the Jetson Nano, as it parses up the individual clusters of the parking lot. Clusters 0-4 (left to right respectively) are to be fed into the neural network respectively so that the network is making predictions sequentially and not bottlenecked by updating the entire parking lot list at once.

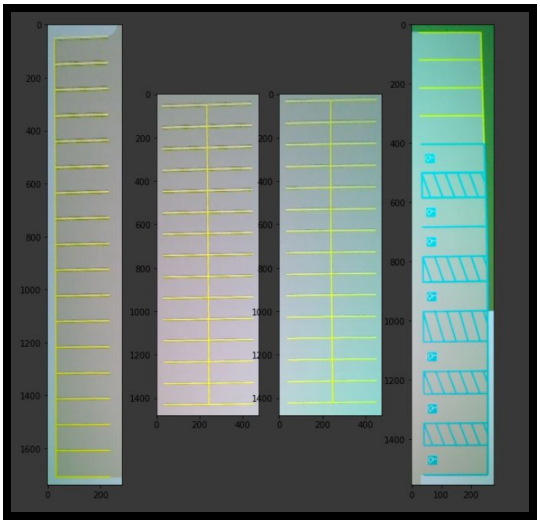

Figure 5.2.1.3: Parsed Camera Pipeline Test Result

5.2.1.4 Hough Line Transform

Upon parsing the camera feed, the hough line transform is shown (figure-5.2.1.4) to have parsed the horizontal parking space lines. These lines are averaged out and stored into a parking space dictionary representative of the spaces in each cluster.

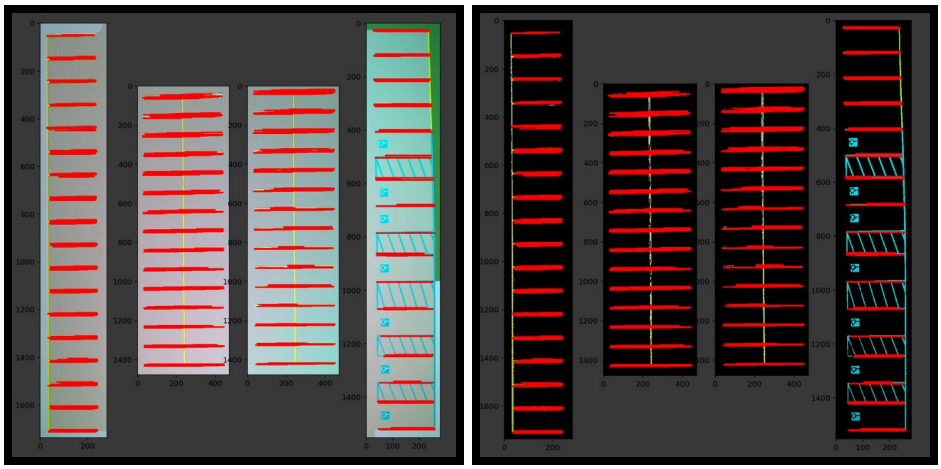

Figure 5.2.1.4: Hough Line Transform Test Result

5.2.1.5 Parking Space Parsing

Here is the resulting image of the parking space dictionary lines being drawn out, after parsing the spaces via the hough line transform function. These spaces hold unique spot identifiers in the parking space dictionary, giving the corner pixel point locations of each bounding box.

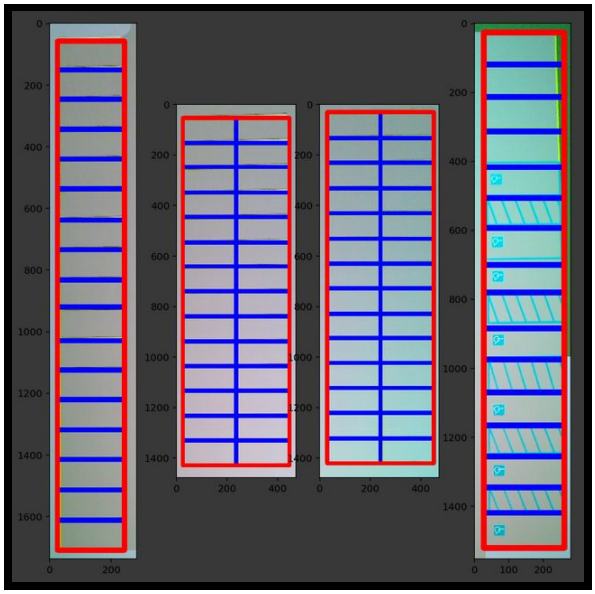

Figure 5.2.1.5: Parsed Parking Space Dictionary Test Result

5.2.1.6 Data Collection

The following images show examples of the data gathered for training the network.

Figure 5.2.1.6: Data Collection of Vacant and Occupied Parking Spaces

5.2.1.7 Convolutional Neural Network Training

The following figure illustrates the convolutional neural network architecture used for this project. The input image is a 64x64 input image, followed by a series of convolution, activation, pooling and dropout layers. Lastly, the data is flattened and passed through the resulting dense layers before making a prediction with two classifications: occupied, and vacant.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 64, 64, 3)]       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 64, 64, 4)         304
_________________________________________________________________
activation_8 (Activation)    (None, 64, 64, 4)         0
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 32, 32, 4)         0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 32, 32, 8)         296
_________________________________________________________________
activation_9 (Activation)    (None, 32, 32, 8)         0
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 16, 16, 8)         0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 16, 16, 32)        2336
_________________________________________________________________
activation_10 (Activation)   (None, 16, 16, 32)        0
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 8, 8, 128)         36992
_________________________________________________________________
activation_11 (Activation)   (None, 8, 8, 128)         0
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               262272
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 258
=================================================================
Total params: 302,458
Trainable params: 302,458
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.2.1.7-1: Convolutional Neural Network Summary

The next image below shows the performance of the network during training.

```
Train on 3200 samples, validate on 800 samples
Epoch 1/20
3200/3200 [==============================] - 8s 2ms/sample - loss: 1.7734 - accuracy: 0.7300 - val_loss: 0.3525 - val_accuracy: 0.8662
Epoch 2/20
3200/3200 [==============================] - 0s 155us/sample - loss: 0.2212 - accuracy: 0.9134 - val_loss: 0.1687 - val_accuracy: 0.9262
Epoch 3/20
3200/3200 [==============================] - 0s 149us/sample - loss: 0.1310 - accuracy: 0.9531 - val_loss: 0.1010 - val_accuracy: 0.9650
Epoch 4/20
3200/3200 [==============================] - 0s 151us/sample - loss: 0.0875 - accuracy: 0.9716 - val_loss: 0.0693 - val_accuracy: 0.9762
Epoch 5/20
3200/3200 [==============================] - 0s 141us/sample - loss: 0.0685 - accuracy: 0.9762 - val_loss: 0.0588 - val_accuracy: 0.9825
Epoch 6/20
3200/3200 [==============================] - 0s 149us/sample - loss: 0.0540 - accuracy: 0.9822 - val_loss: 0.0459 - val_accuracy: 0.9850
Epoch 7/20
3200/3200 [==============================] - 0s 148us/sample - loss: 0.0426 - accuracy: 0.9878 - val_loss: 0.0303 - val_accuracy: 0.9900
Epoch 8/20
3200/3200 [==============================] - 0s 142us/sample - loss: 0.0362 - accuracy: 0.9897 - val_loss: 0.0272 - val_accuracy: 0.9900
Epoch 9/20
3200/3200 [==============================] - 0s 145us/sample - loss: 0.0291 - accuracy: 0.9916 - val_loss: 0.0215 - val_accuracy: 0.9937
Epoch 10/20
3200/3200 [==============================] - 0s 144us/sample - loss: 0.0264 - accuracy: 0.9925 - val_loss: 0.0178 - val_accuracy: 0.9962
Epoch 11/20
3200/3200 [==============================] - 0s 148us/sample - loss: 0.0219 - accuracy: 0.9953 - val_loss: 0.0183 - val_accuracy: 0.9950
Epoch 12/20
3200/3200 [==============================] - 0s 145us/sample - loss: 0.0192 - accuracy: 0.9959 - val_loss: 0.0153 - val_accuracy: 0.9962
Epoch 13/20
3200/3200 [==============================] - 0s 148us/sample - loss: 0.0171 - accuracy: 0.9962 - val_loss: 0.0123 - val_accuracy: 0.9975
Epoch 14/20
3200/3200 [==============================] - 0s 143us/sample - loss: 0.0159 - accuracy: 0.9978 - val_loss: 0.0117 - val_accuracy: 0.9950
Epoch 15/20
3200/3200 [==============================] - 0s 155us/sample - loss: 0.0141 - accuracy: 0.9966 - val_loss: 0.0120 - val_accuracy: 0.9962
Epoch 16/20
3200/3200 [==============================] - 0s 153us/sample - loss: 0.0116 - accuracy: 0.9987 - val_loss: 0.0102 - val_accuracy: 0.9975
Epoch 17/20
3200/3200 [==============================] - 0s 150us/sample - loss: 0.0108 - accuracy: 0.9987 - val_loss: 0.0089 - val_accuracy: 0.9975
Epoch 18/20
3200/3200 [==============================] - 0s 145us/sample - loss: 0.0098 - accuracy: 0.9991 - val_loss: 0.0075 - val_accuracy: 0.9975
Epoch 19/20
3200/3200 [==============================] - 0s 149us/sample - loss: 0.0088 - accuracy: 0.9987 - val_loss: 0.0073 - val_accuracy: 0.9987
Epoch 20/20
3200/3200 [==============================] - 0s 148us/sample - loss: 0.0083 - accuracy: 0.9997 - val_loss: 0.0065 - val_accuracy: 0.9987
```

Figure 5.2.1.7-2: Convolutional Neural Network Training Performance

The next two figures illustrate the training performance as it relates to the training accuracy (figure-5.2.1.7-3), and the cross entropy loss (figure-5.2.1.7-4).

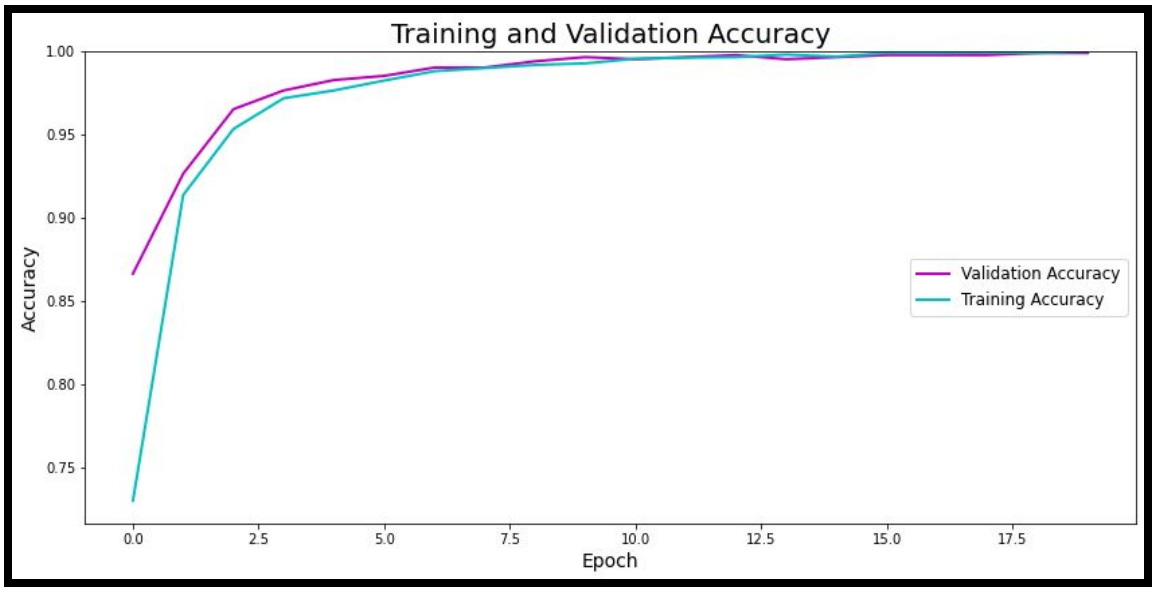

Figure 5.2.1.7-3: Convolutional Neural Network Training Accuracy Graph

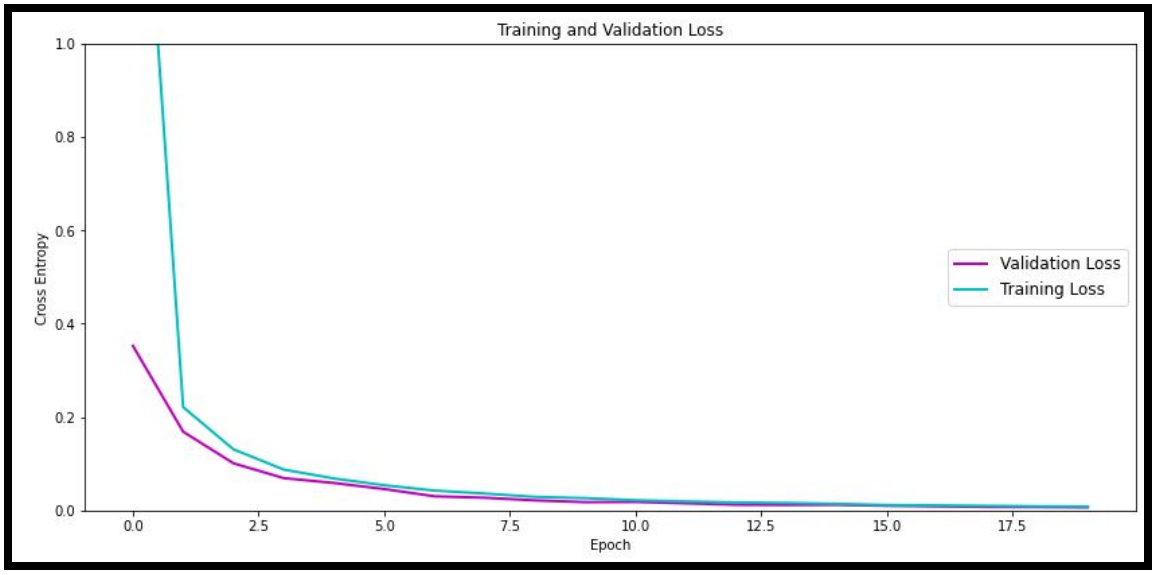

Figure 5.2.1.7-4: Convolutional Neural Network Training Loss Graph

5.2.1.8 Convolutional Neural Network Predictions

This last figure presents the trained network making predictions on a test image.

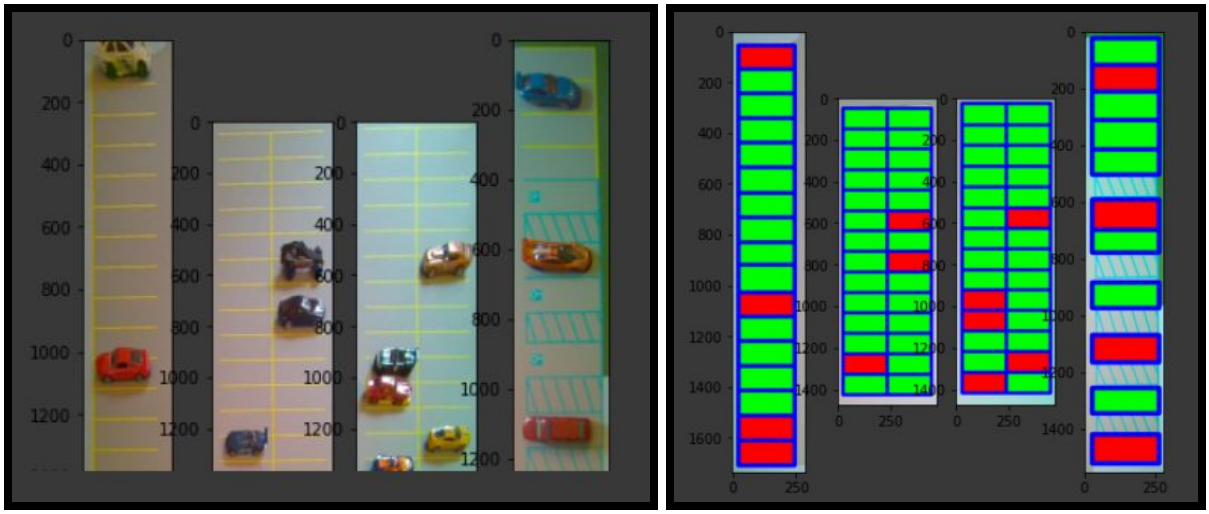

Figure 5.2.1.8-1: Convolutional Neural Network Predictions Result

Testing Results:

CV_F1		
Classification Accuracy (%)	Excellent	Accuracy was measured to be well over 95% given good lighting conditions
Inference Rate (Hz)	Excellent	The interface from the jetson board was updated at a period of about 20s or about 0.05 Hz. This is well above the requirement of 0.016 Hz.
Edge Case Classification (%)	Excellent	Edge conditions such as double parked cars or motorbikes are classified correctly > 50% of the time. In edge cases where cars are double parked, the system correctly detects both spaces more than 80% of the time.

Figure 5.2.1.8-2: Testing Result Classifications

5.2.2 Network Module Test Results

5.2.2.N_F1 Connectivity Testing

VPN was set up using the information here:
https://umdearborn.teamdynamix.com/TDClient/2019/Portal/KB/ArticleDet?ID=43009

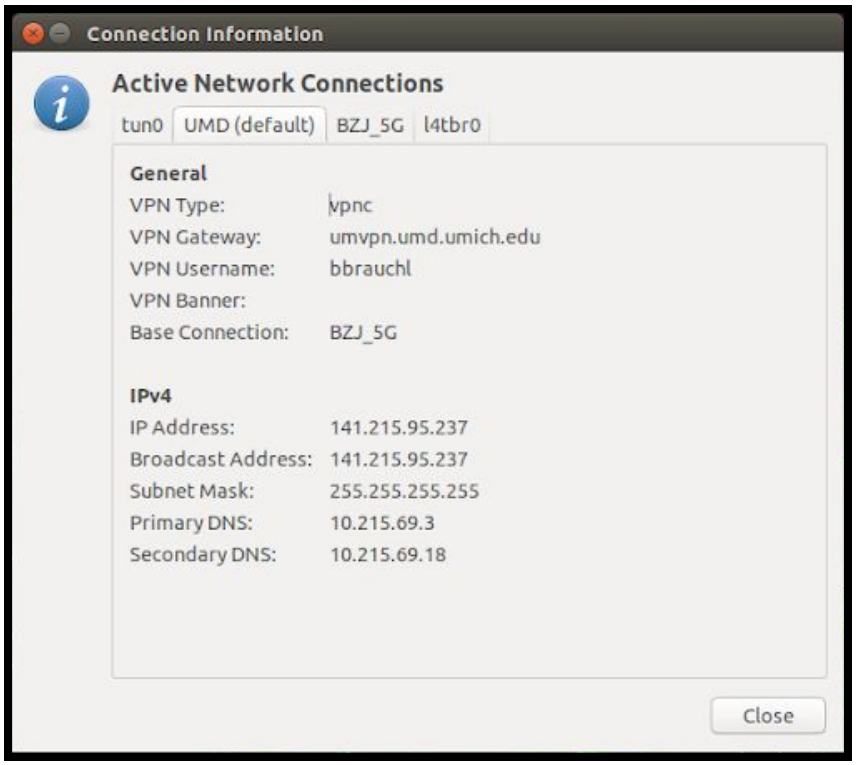

Figure 5.2.2.N_F1.1: Received IP address on the Jetson (Requirement 3.3.3)

Viewing the output, above it can be seen that the network connection, and the VPN connection were both successful. However, the assigned IP address is incorrect. This is due to some minor complications using the university VPN. When tested physically on campus the Jetson received the correct IP of 141.215.192.40. Due to the current COVID-19 situation, the configuration with a VPN is considered passing, and meets the criteria for requirement 3.3.3.

Next, one of the demo update scripts was run from Bryan's PC while monitoring the network interface using Wireshark:

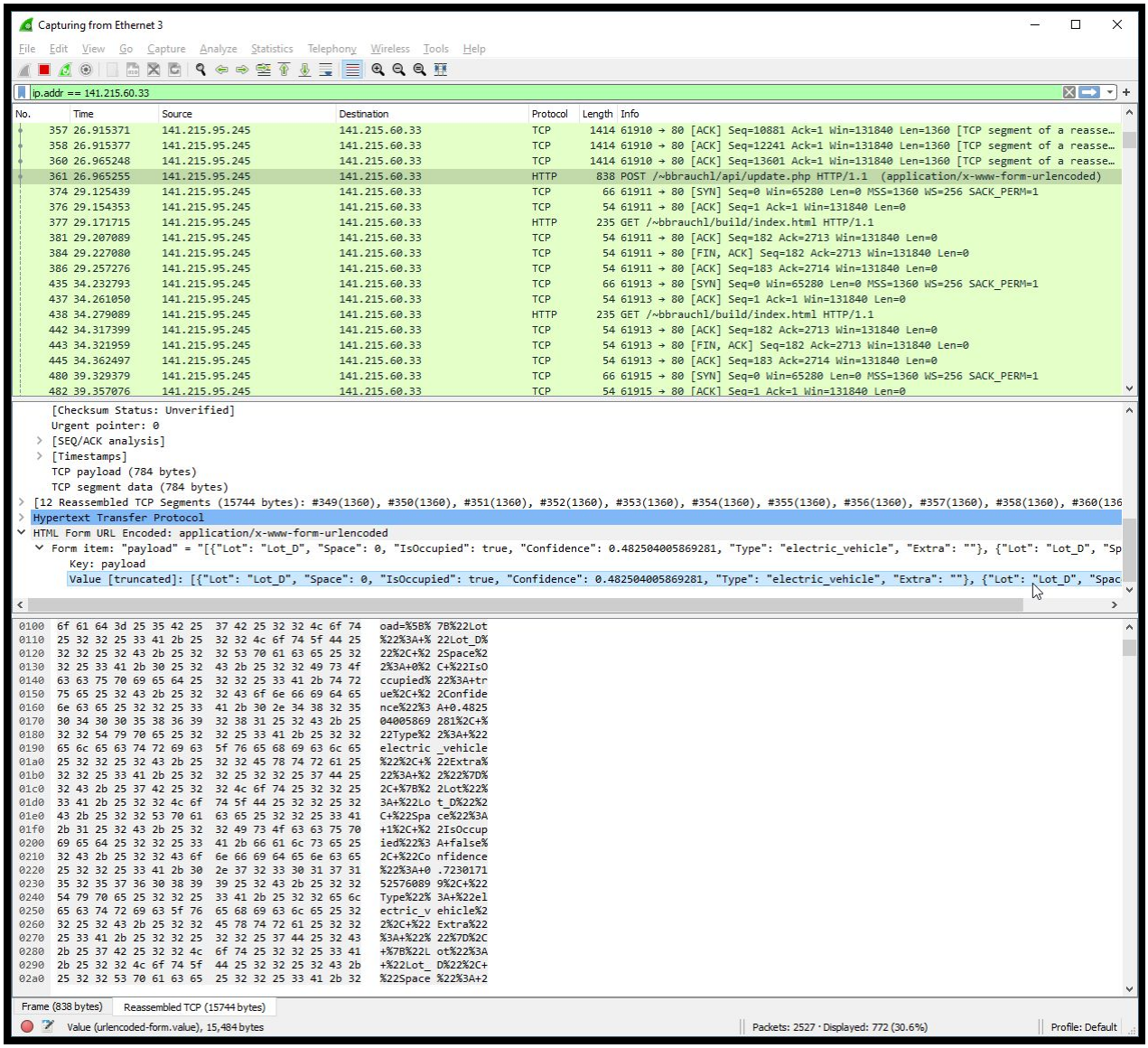

Figure 5.2.2.N_F1.2: Python Request over TCP (Requirement 3.3.2)

The output of wireshark shows a post request using update data, and as expected all packages are using the TCP protocol. (HTTP is a higher level protocol that uses TCP). Therefore the TCP Protocol usage is passing, and the network module meets the criteria for requirement 3.3.2.

Next, SSH for remote access will be tested. Due to the lack of assigned static IP, This test must also be modified slightly due to lack of access on campus. A home network will be used instead along with mdns to connect to the board via hostname. This means that the hostname used for the test setup is “umd-parksmart-02.local” rather than “umd-parksmart-01.umd.umich.edu”

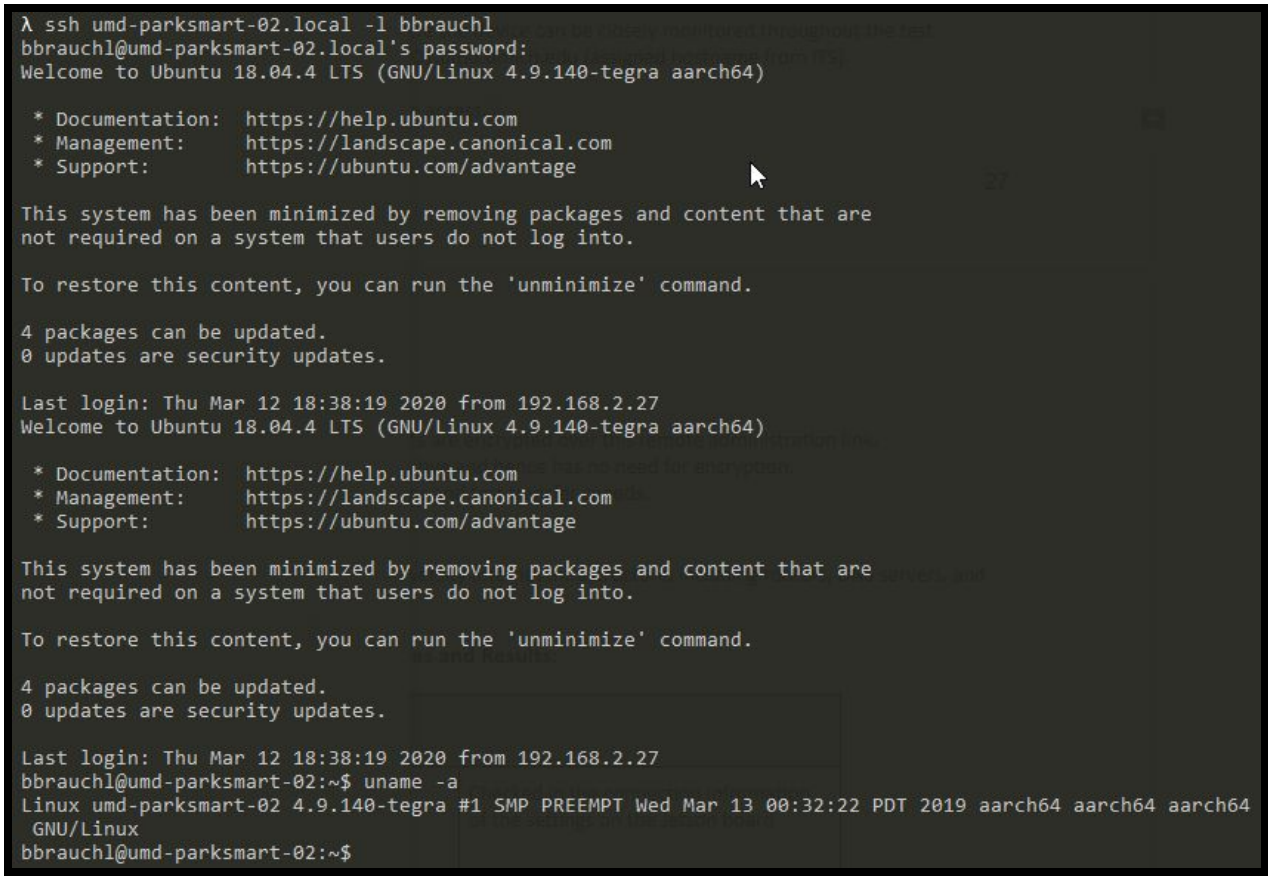

Figure 5.2.2.N_F1.3: Remote SSH access to Jetson (Requirements 3.3.4, 3.3.5)

From the terminal output, it is clear that ssh access to the jetson board is working over the home network. The login process also asks for a password, meeting the requirement for security. A quick look at wireshark shows that this link is encrypted and secure:

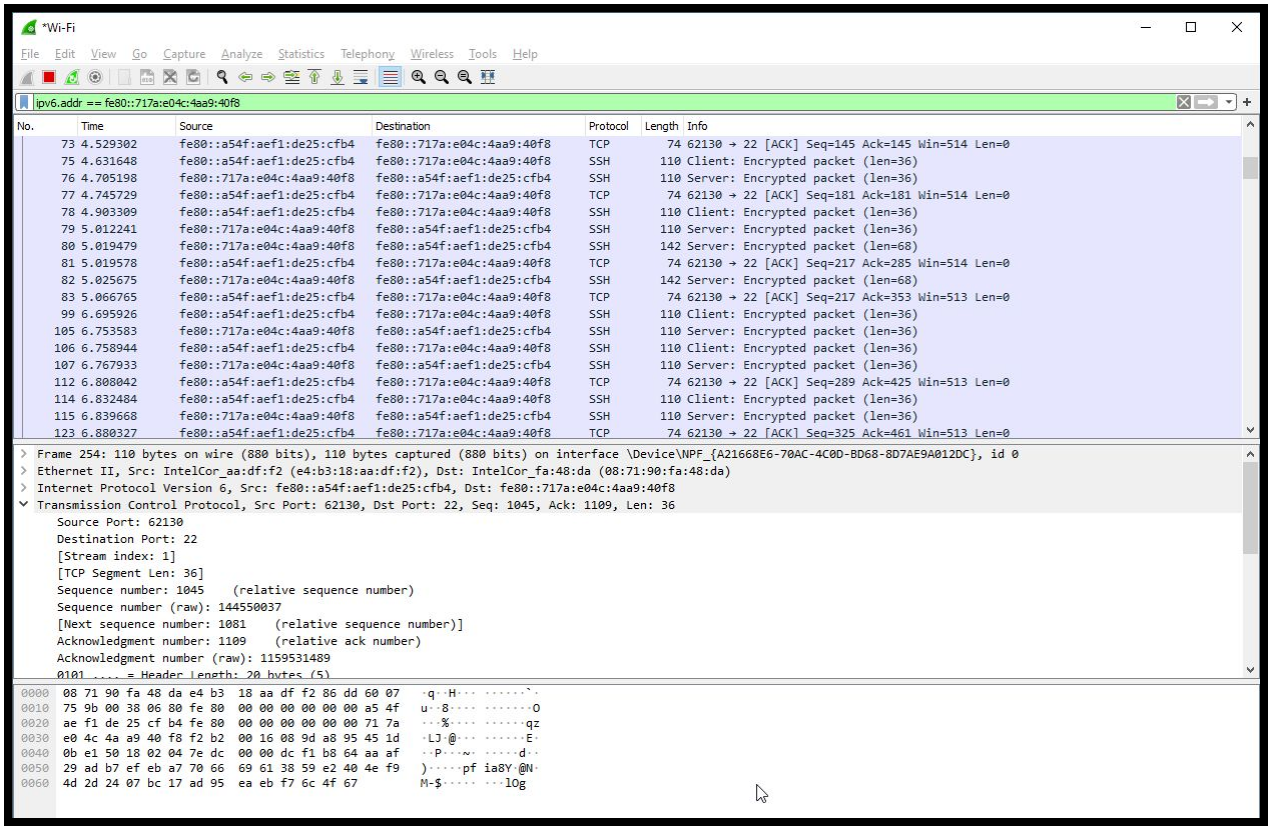

Figure 5.2.2.N_F1.4: Wireshark output of SSH connection (Requirement 3.3.6)

With the results shown above, the Remote access component of the networking functional test 1 is passing. The network module meets the criteria for requirements 3.3.4, 3.3.5, and 3.3.6.

The final component to test is connection speed. This is tested in 2 ways: Using a network speed test provided by google, and a metric from a SCP transfer. The speed test is done by accessing it in the web browser:

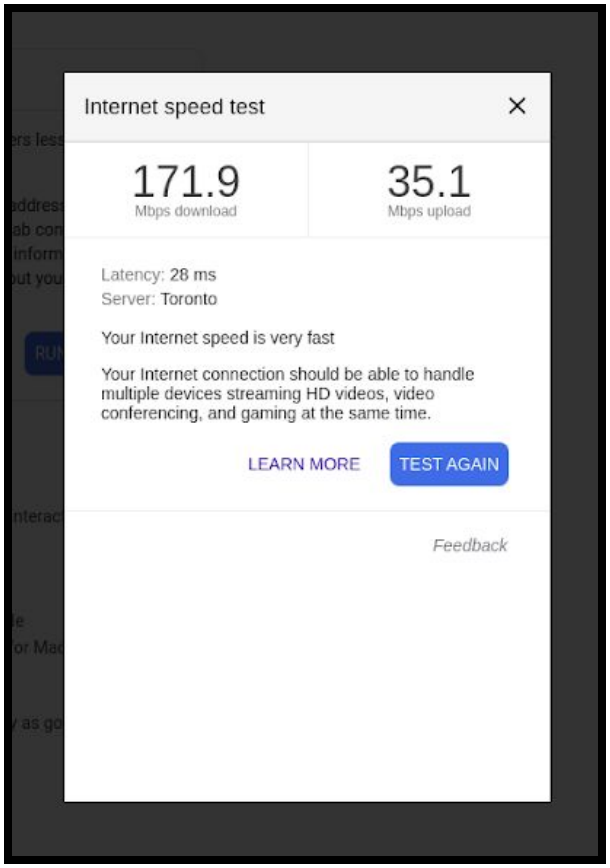

Figure 5.2.2.N_F1.5: Internet speed test (Requirement 3.3.1)

These numbers are well beyond passing for the specification set out in the requirements. However, the SCP testing tells a somewhat different story. This test was done by using a large file and transferring it via SCP. The results show a SCP transfer speed of approximately 2.9 MBit/s.

```
C:\Users\Bryan\Desktop\Roms\Gamecube\PAPER MARIO [G8ME01]
λ scp game.iso bbrauchl@umd-parksmart-02.local:~/Desktop/
bbrauchl@umd-parksmart-02.local's password:
game.iso                                      81% 1134MB   2.9MB/s   01:30 ETA
```

Figure 5.3.3.N_F1.6 SCP transfer to Jetson Nano (Requirement 3.3.1)

The data speed requirement of 3.2MBit/s was set as highly ambitious as it was our maximum foreseeable throughput, including video streaming from the device which was being considered at the time of writing requirements. However, the system only needs a fraction of that speed when running in normal operation modes. For normal operation mode, 2.9MBit/s is suitable. This test is Passed with a rating of “Good”. It would still be possible to improve this rating to “Excellent” by finding ways to increase network throughput. Traces to Requirement 3.3.1.

Test Results:

N_F1		
Device is able to connect and authenticate on UMD-Secure (true/false, Received IP address)	Good	The Jetson Nano does not receive static IP off campus, and instead received 141.215.95.237 in the trial. This has been deemed GOOD because the Jetson is still on the University network and this configuration is workable for a demo.
TCP Protocol in use (true/false)	Excellent	Wireshark indicates that TCP is in use.
Remote Access via SSH and encrypted, and passwords are used for authentication (true/false)	Excellent	SSH access is working and Wireshark indicates that administration is encrypted and password protected.
Data Transfer Rate (MBit/s)	Good	Measured well beyond requirements (> 100 MBit/s). SCP transfers are placed in the “good” category measured at approximently 2.9 MBit/s. This is slightly below the ambitious goal set by the requirements, but will likely be enough for this system.

5.2.2.N_F2 Database Retrieval and Commit

The database commit and retrieval can both be tested in the same routine by committing data to the database, and then reading that data back. Then check the correlation between the data that was sent to the server and the data that was received. The second component is testing when the server is accessible. This is done using an extended test of 12 hours, sending periodic requests to check that the server is available at a given time. Both of these are done in the test script N_F2.py under the demos folder on github.

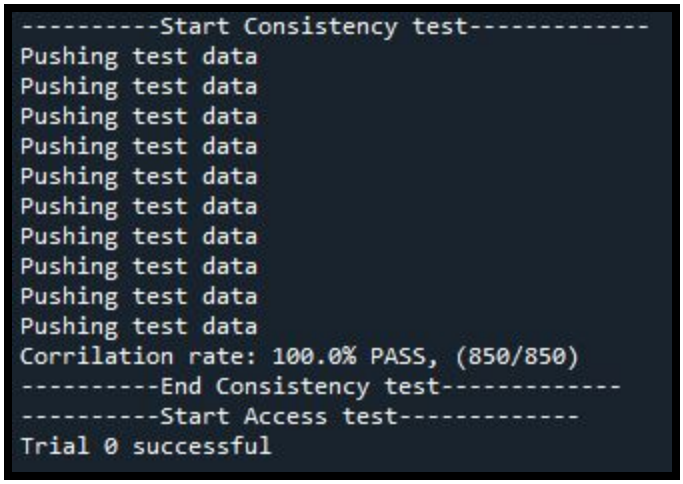

Figure 5.2.2.N_F2.1: Output of Python Testing script testing Database consistency (Requirement 3.3.7, 3.3.8)

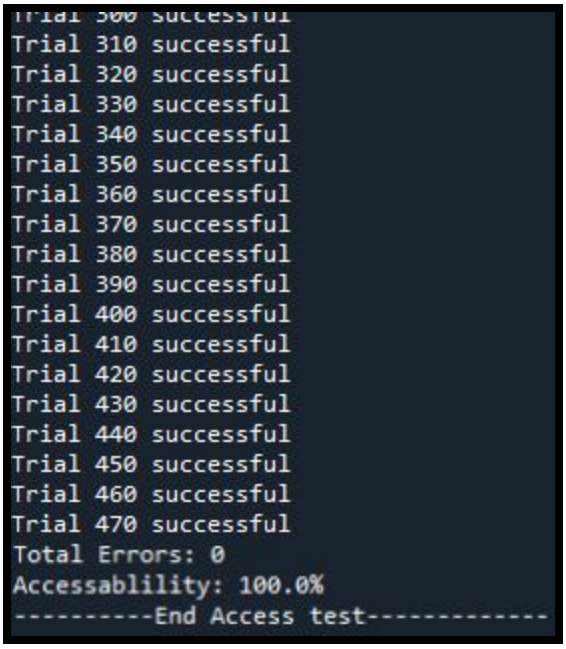

Figure 5.2.2.N_F2.2: Output of Python Testing script testing Database Accessibility (Requirement 3.3.8, 3.5.1)

The same behavior can be checked using the web front end as well.

Figure 5.2.2.N_F2.3: Database functional displayed as web-application output (Requirement 3.3.7, 3.3.8, 3.5.1)

Finally, using wireshark again can confirm that the HTTP protocol is in use by capturing the network traffic while running the test script.

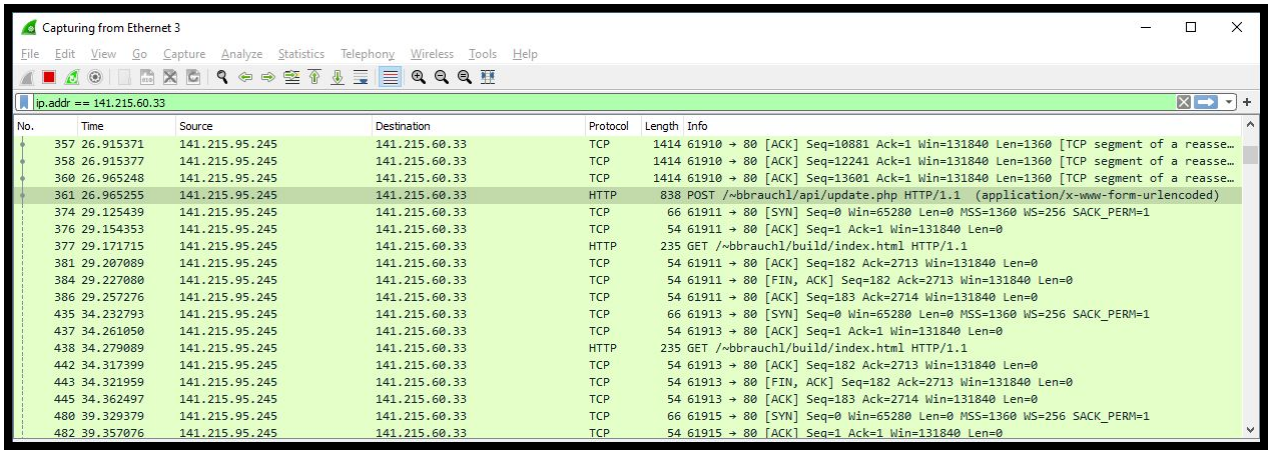

Figure 5.2.2.N_F2.4: TCP and HTTP protocols during web api calls (Requirement 3.3.7)

This output shows that not only the fetch and receive operations function but the javascript web application hooks into the database and can start performing updates. Meaning that this test is a PASS, and traces to requirement 3.3.7, 3.3.8, and 3.5.1.

Test Results:

N_F2		
HTTP and TCP are used in for the underlying protocols during the web API calls (true/false)	Excellent	Using wireshark it can be shown that the system is using TCP for api packets, meaning that they will be reported on errors.
State of MySQL database (% correlation)	Excellent	Passes with a Excellent correlation of 100%
Access to the server (%)	Excellent	Passes with an Excellent access of 100% in the 8-hour trial.

5.2.2.N_S1 Connection Error Reporting

One safety concern with the web application and the network submodule overall is the way that errors are handled. This test case covers the condition when weather conditions are poor and 2 error conditions for the system: The computer vision module dropping out of the system and the programming API's being used in the wrong way.

All of these cases were considered in the design of the system. The system should check the weather conditions and report to the user if there are any concerns with the accuracy of the image output due to weather. This is done using a javascript alert that is triggered by the contents of a request to openweathermap.org.

The database should also be smart enough to recognize stale data, namely data that has not been updated in a significant amount of time. This is done using a SQLupdate call to expire data in the database before updating with new data. Finally, the web api code should be

resistant to common errors and report feedback to help any programmer trying to interface to the system.

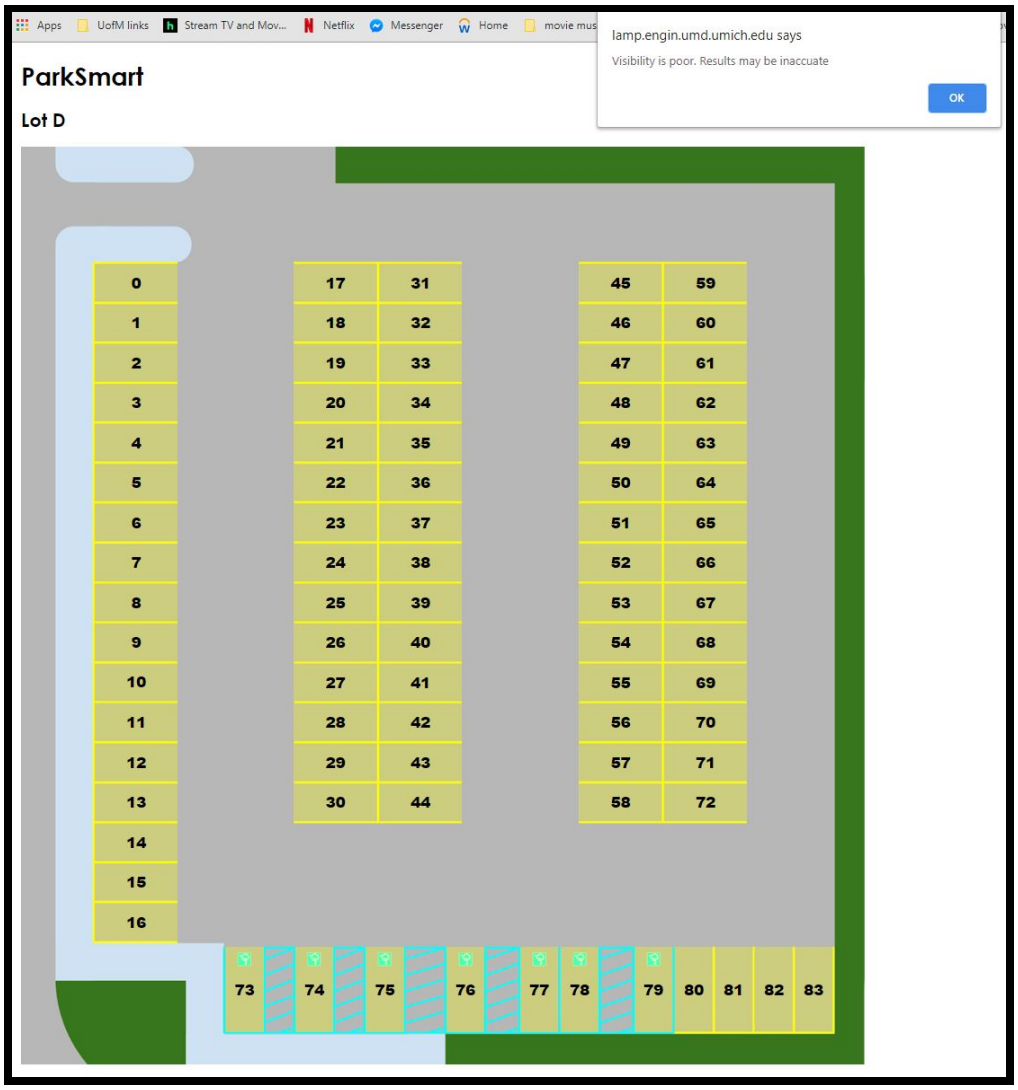

Figure 5.2.2.N_S1.1: Expired data and visibility warning shown on the web interface

To the Web interface, “expired data” looks the same as “no data” This is intentional behavior. The above picture also shows a visibility error, which will only appear if the app deems the current weather conditions non-ideal.

Measuring the dropout detection time and the response to improper API calls is a job that is greatly simplified using python. The python script N_S1.py runs both of these tests and is located in the demos folder on github.

```
----------Start Dropout Timing test-------------
Pushing test data
Starting Timing. Current Time: 1586087442.068
Data Expired! Current Time: 1586087442.068
Dropout Recognition Delay: 119.40s EXCELENT
----------End Dropout Timing test-------------
----------Start API ERROR test------------
testing the pull api
response to Lot_D: {"Lot_D":null}
response to Lot D: Error! User specified Parking lot name that does not exist in the database!
response to LOT_D: Error! User specified Parking lot name that does not exist in the database!
response to : Error! User specified Parking lot name that does not exist in the database!
response to 2: {}
response to []: {"Lot_D":null}
response to True: {}
```

Figure 5.2.2.N_S1.2: Section of output testing Dropout detection and Pull API inputs

```
response to {'Lot': 'Lot_D', 'Space': 1, 'IsOccupied': 1.0, 'Confidence': 0.5, 'Type': '', 'Extra': ''}: Success!
response to {'Lot': 'Lot_D', 'Space': 1, 'IsOccupied': 1.0, 'Confidence': 0.5, 'Type': '', 'Extra': ''}: Success!
response to {'Lot': 'Lot_D', 'Space': 1, 'IsOccupied': 1.0, 'Confidence': 0.5, 'Type': '', 'Extra': ''}: Success!
response to {'Lot': 'Lot_D', 'Space': 1, 'IsOccupied': 1.0, 'Confidence': 0.5, 'Type': '', 'Extra': ''}: Success!
response to {'Lot': 'Lot_D', 'Space': 1, 'IsOccupied': 1.0, 'Confidence': 0.5, 'Type': [], 'Extra': 'hello'}: Success!
response to {'Lot': 'Lot_D', 'Space': 1, 'IsOccupied': 1.0, 'Confidence': 0.5, 'Type': [], 'Extra': ''}: Success!
response to {'Lot': 'Lot_D', 'Space': 1, 'IsOccupied': 1.0, 'Confidence': 0.5, 'Type': [], 'Extra': ''}: Success!
response to {'Lot': 'Lot_D', 'Space': 1, 'IsOccupied': 1.0, 'Confidence': 0.5, 'Type': [], 'Extra': ''}: Success!
response to {'Lot': 'Lot_D', 'Space': 1, 'IsOccupied': 1.0, 'Confidence': 0.5, 'Type': [], 'Extra': ''}: Success!
response to {'Lot': 'Lot_D', 'Space': 1, 'IsOccupied': 1.0, 'Confidence': '50%', 'Type': 'student', 'Extra': 'hello'}: [{"errno":
1064,"sqlstate":"42000","error":"You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version
for the right syntax to use near ' 'student', 'hello')' at line 1"}]
response to {'Lot': 'Lot_D', 'Space': 1, 'IsOccupied': 1.0, 'Confidence': '50%', 'Type': 'student', 'Extra': ''}: [{"errno":
1064,"sqlstate":"42000","error":"You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version
for the right syntax to use near ' 'student', '')' at line 1"}]
response to {'Lot': 'Lot_D', 'Space': 1, 'IsOccupied': 1.0, 'Confidence': '50%', 'Type': 'student', 'Extra': ''}: [{"errno":
1064,"sqlstate":"42000","error":"You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version
for the right syntax to use near ' 'student', '')' at line 1"}]
response to {'Lot': 'Lot_D', 'Space': 1, 'IsOccupied': 1.0, 'Confidence': '50%', 'Type': 'student', 'Extra': ''}: [{"errno":
1064,"sqlstate":"42000","error":"You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version
for the right syntax to use near ' 'student', '')' at line 1"}]
```

Figure 5.2.2.N_S1.3: Section of output testing inputs to update api function

These tests confirm the system is resilient and can help report mistakes and issues to the end user. Every test case was met with either a string saying that the request was successful or a JSON string indicating the SQL issue with the request. In all cases, this test is a PASS

Test Results:

N_S1		
Poor Weather Conditions are reported in the user interface (true/false)	Excellent	The web interface will show javascript pop-ups when the weather conditions are suboptimal

Computer Vision Module Dropout is detected. (s)	Excellent	Computer Vision dropouts are detected in 119-121 seconds. Which is less than 180
Programming the API with incorrect access reports the error.	Excellent	Every improper call gave reference to the first element that was incorrect in the request struction.

5.2.3 User Application Module Test Results

5.2.3.A_F1 Data Reporting Testing

This section outlines the testing of the web and android applications. Below is the output exercised with random data from the web application. This shows that the parking spaces are colored either red (occupied) or green (vacant) depending on the data from the server. Then, the opacity of the background is set based on the confidence value reported from the server. For predictions that are more confident, the space is shaded in with a more solid color of red/green. Inspecting the data and comparing it with the terminal output shows that there is a direct correlation between the current data in the database and the reported data in the web interface.

Figure 5.2.3.A_F1.1: Predictions displayed as web-application output (Requirement 3.3.7, 3.3.8, 3.5.1)

The android application also produces the same behavior, although in a slightly different format:

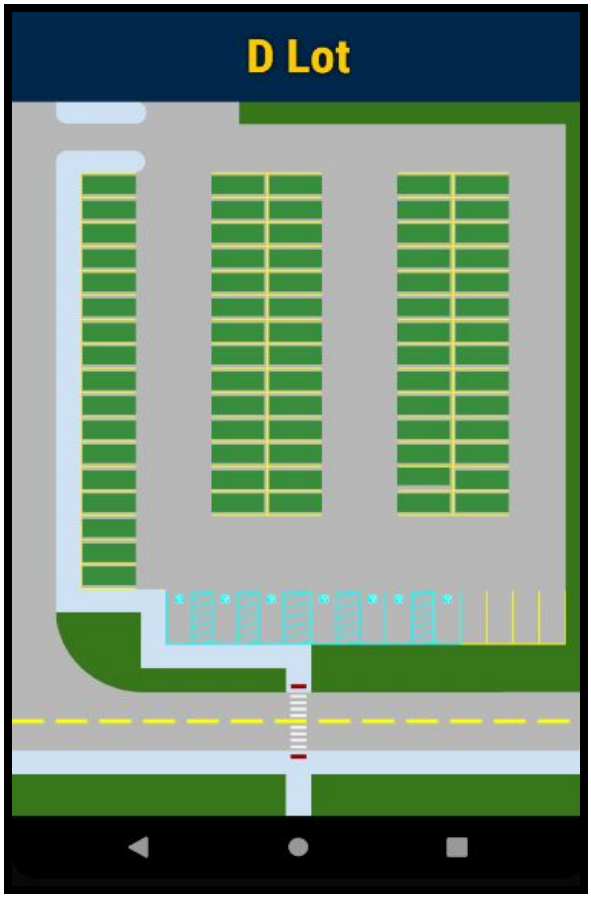

Figure 5.2.3.A_F1.2 Android App Interface

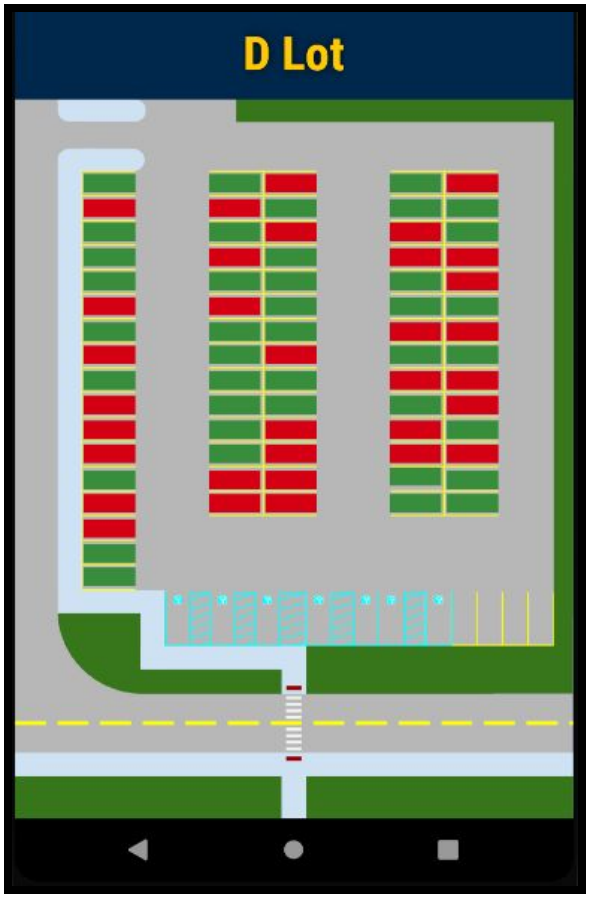

Figure 5.2.3.A_F1.1 Application Interface reporting spots taken

Test Results:

A_F1		
Mapping of spots to image	Excellent	Parking spaces are reported correctly to the android app and to the web interface.

5.2.3.A_S1 Distracted Driver and Warnings Testing

There is a 3rd option for the coloring of the parking spaces: Invalid (yellow). This color is displayed when there is no data received from the database. This either means that the data does not exist in the database or that the data has expired and is no longer valid. The front end javascript takes care of updating by polling the server and requesting new data. One final aspect is the weather notifications. The web application shows a warning when the visibility is

below a threshold and/or when the weather conditions are not clear. This provides additional feedback to the user as to potential inaccuracies in the reported data in a system that has been deployed.

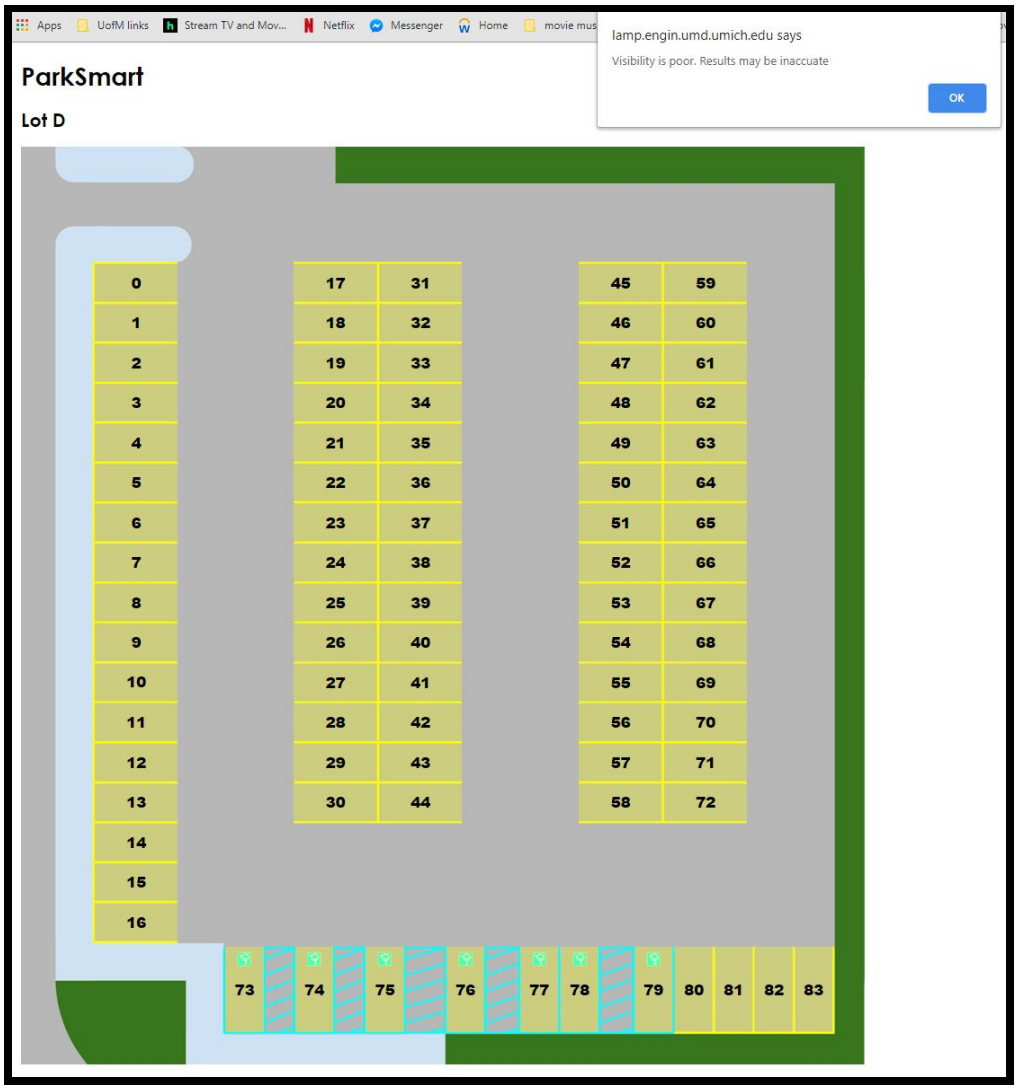

Figure 5.2.3.A_F1.2: Expired data and visibility warning shown on the web interface

To the Web interface, “expired data” looks the same as “no data” This is intentional behavior. The above picture also shows a visibility error, which will only appear if the app deems the current weather conditions non-ideal.

Next, a pop-up has been designed to warn the driver if the current drive speed exceeds 10 MPH.

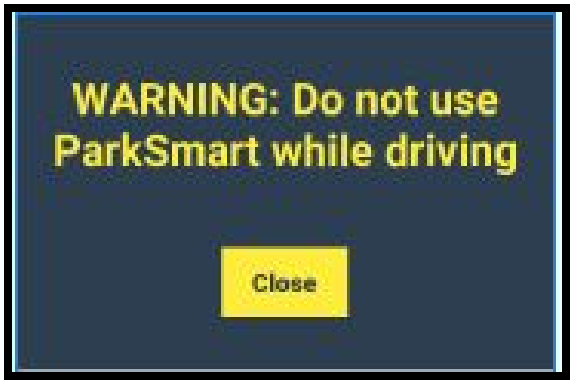

Figure 5.2.3.A_S1 Distracted Driver Popup

Currently, this pop-up exists only as an android activity, and does not have proper implementation in the android code, due to time limitations and setbacks that have affected the project.

Test Results:

A_S1		
User Notifications - Android App	-	Due to lack of time, this feature has yet to be implemented. We are confident that it could be easily added to the app, but ran out of time to implement.
User Notifications - Web App	Good	The web application reports warnings in the case of poor visibility or poor weather conditions. However, this is not fine-tuned to be reflective of the system's performance in those weather conditions.

5.2.4 Full System Test Results

5.2.4.S_F1 System Output Testing

The full system test starts with an empty lot, running the CNN, and looking at the camera pipeline feed and website as they should both update in real-time upon prediction updates. In the initial case, parking spaces begin as unclassified, and invalid. The computer vision module has not yet run any classifications, and the web interface reports all parking spaces as “invalid” by coloring them yellow. This is an example of the web interface reporting errors to the user. It would not be desirable to have the web interface report data before the computer vision module has a chance to update. The same is true in the case where the computer module does not provide data for more than 2 minutes.

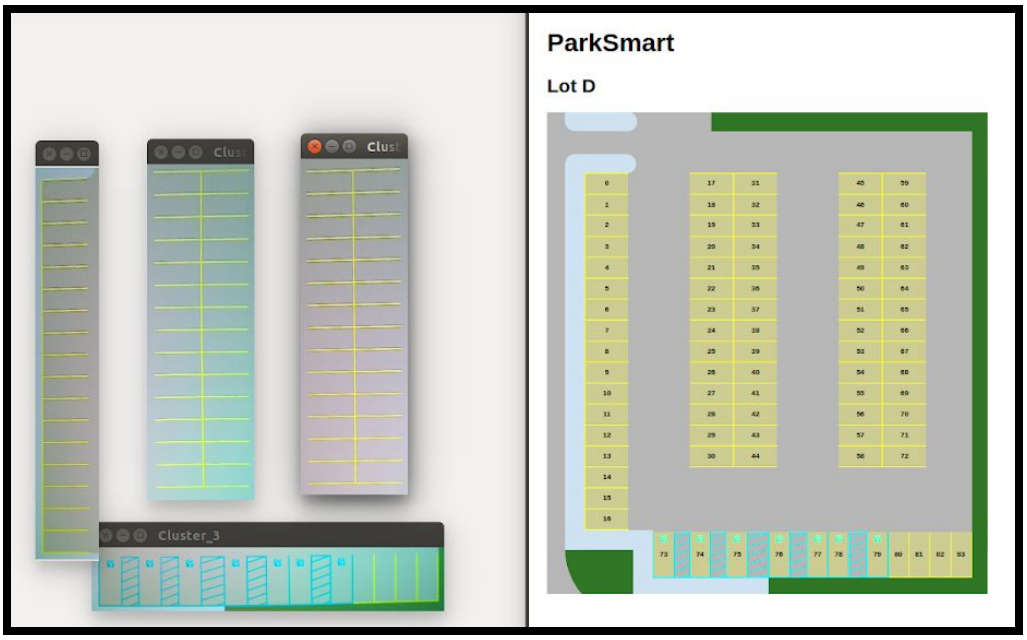

Figure 5.2.4.1: Real-Time Prediction Results on Camera Pipeline (LHS) and Website (RHS)

Here is a picture taken of the physical parking lot board with hot-wheels cars on it for testing.

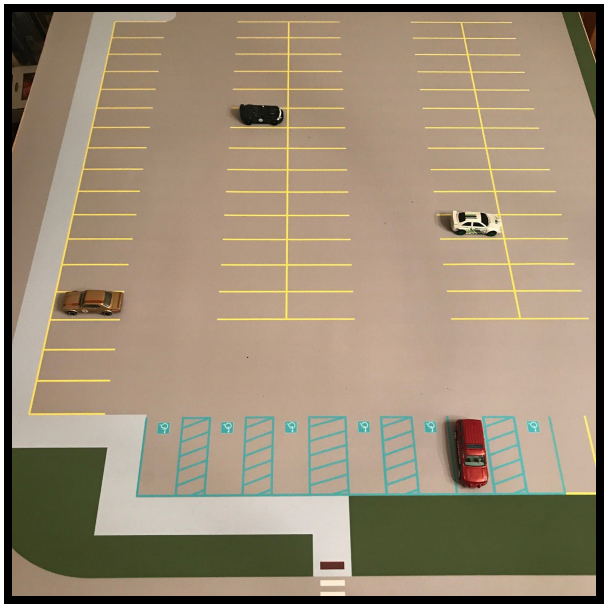

Figure 5.2.4.2: Physical Parking Lot Board

Following the testing environment setup, here is the real-time update prediction results, corresponding to the hot-wheels cars placed on the board. The camera pipeline predictions feed (LHS) updates cluster by cluster as a new image frame is fed into the network. As shown by the status updates in figure-5.2.4.4, upon every four cluster updates (completing the cycle of predictions for the entire parking lot), the program then pushes the status updates to the server where the website image is then updated to represent the changes in the parking space predictions dictionary.

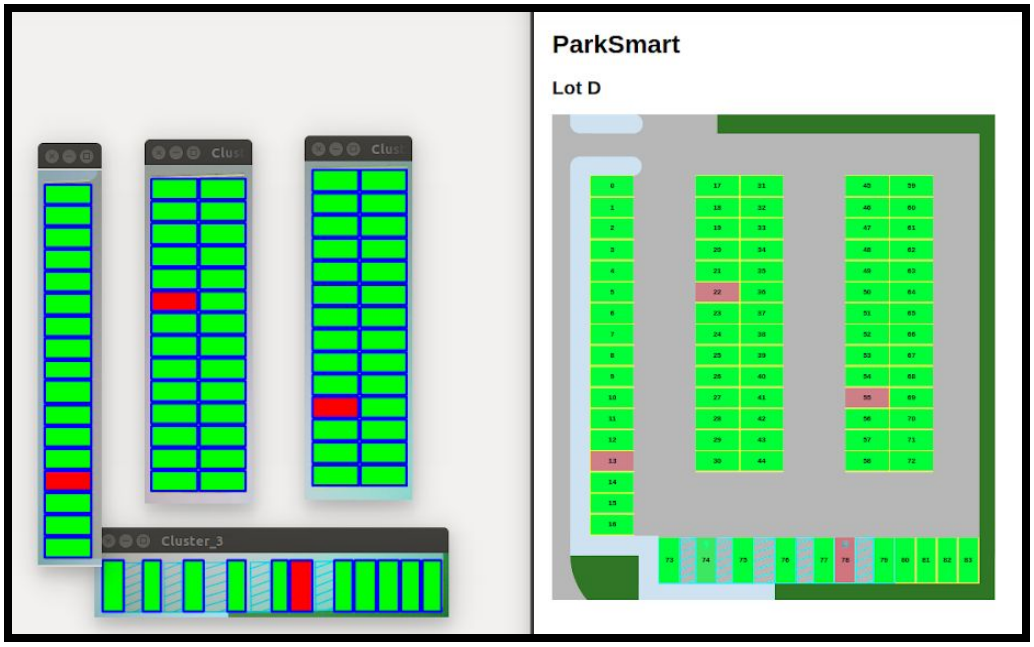

Figure 5.2.4.3: Real-Time Camera Pipeline and Website Feed Testing Results

Here is a snapshot of the status updates for the cluster prediction results and server file
.

Figure 5.2.4.4: Real-Time Cluster Predictions Update Status

By inspection, we can see that the output of the system exactly matches the input image. Repeating this test for several iterations shows that the output accuracy in this controlled test environment is quite remarkable, as it produces the correct macro state of the parking lot > 97% of the time.

S_F1		
Parking lot status	Excellent	Macro parking lot accuracy: >97%
User Notifications	Good	The web app and android app both report on invalid data, and the web application reports on poor weather conditions. However the poor weather conditions notifications are sometimes inaccurate.

5.2.4.S_S1 System Latency Testing

Performing System Latency tests were performed using a simple stopwatch. Time started when the computer vision module reported that an input frame had been captured, and was stopped when the update data appeared in the web interface. This test was repeated multiple times and averaged to get the average latency of the system.

Update frequency is measured slightly differently. This must be measured at the same point in an update cycle between consecutive cycles. This was done using the database timestamp system that was put into place, measuring the time difference of consecutive updates when the camera module was running in continuous mode.

S_S1		
Update Rate (hz)	Excellent	The update frequency of the entire system is on the order of 20s
Update Latency (s)	Excellent	The update latency is around 28 sec

6. Budget

module	no	Item	Vendor	Model/Part No.	Unit Cost	Qty	Sub Total Cost
Vision	1	Wide Angle Camera	Waveshare	IMX219-160 Camera	30	1	30
Vision	2	microcomputer board	NVIDIA	Jetson Nano	100	1	100
Vision	3	256 GB MicroSD	Amazon	MB-ME256GA	40	1	40
Network	4	Jetson NIC	Waveshare	AC8625 NIC	25	1	25
Enclosure	5	Enclosure: home built				1	
Enclosure	6	Battery	Batteries+	Duracell SLI31MDC	122	1	122
Enclosure	7	Power Regulator	Amazon	MonkeyJack Buck Converter	13	1	13
Total Cost Estimates							340

Table-6: Project Budget (updated: 4/01/20)

7. Master Schedule

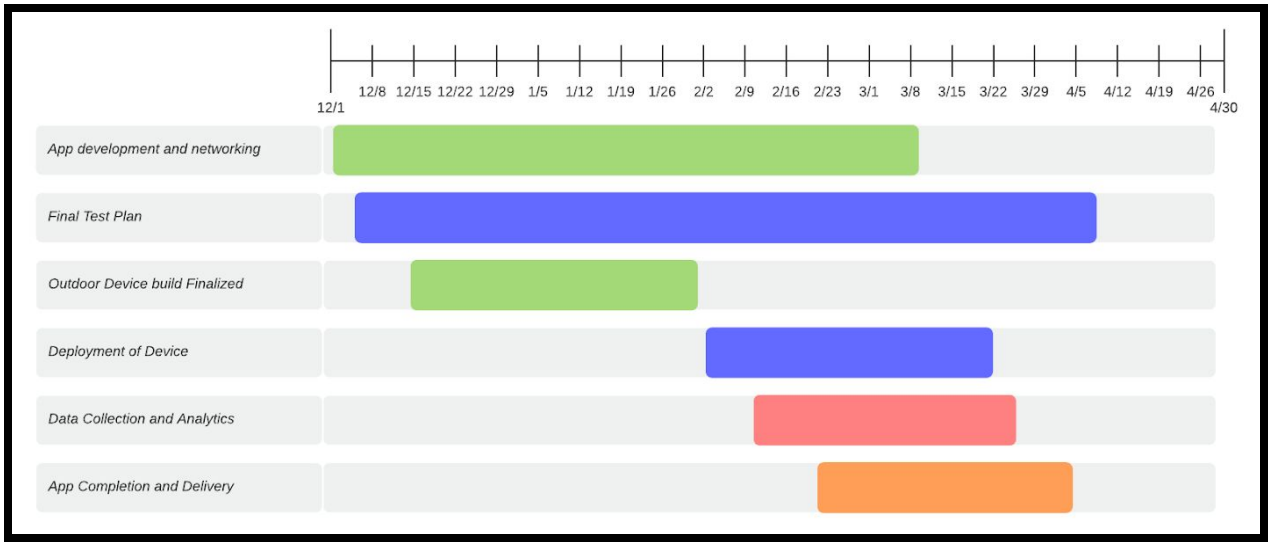

Figure-8: Master Schedule (updated: 4/01/20)

7.1 Project Status

module	Comments	Status
Computer Vision / Image Processing	Got CV module trained and running, able to communicate with the networking server in near real-time performance.	Complete
Network Server	Got an advisor (Professor Watta) willing to “sponsor” us and our server from him	Complete
Physical Structure	Near completion; buck converter needs to be remounted	Complete
Android Application	Building blocks are complete, needs a nice integration	80%

Table 7.1: Project Status (updated: 4/01/20)

8. Conclusion

The original goal of this project was to create a system that could identify vacant and occupied parking spaces, given a camera positioned above a parking lot. Additionally, as last semester went on, another goal was added to include an infrastructure for gathering the vacancy data over the internet, and reporting this data to a user. We have Largely accomplished these goals and have been very successful. There have been some hiccups: the original plan was to use a real parking lot at the university - positioning a camera to overlook it - but due to logistical and safety issues we had to create a small scale version of a parking lot to use. Also, coronavirus had a big impact on our project and the ability to continue development of certain areas of the build. But despite these, the main goal has been achieved; the system described above can identify vacancies in a small scale test lot. In addition, the system is able to communicate with the server we have set up at the university, which in turn is capable of both updating a web interface displaying the vacancy data as well as forwarding this data to the app when requested.

Overall, our team has learned a lot through this two semester design course. There was a huge amount of technical development in each of the team members, as well as exposure to some of the non-technical aspects of design and test that exist in the industry. We recognize that this experience is invaluable in our development as engineers. Though we will likely never work in this senior design group again, the experience provides a healthy transition into industry as it mirrors the projects we will complete in our careers.