

# Vehicle Detection, Tracking, and Status Classification in Parking Lots



## *Critical Design Review*

### Team name

Jackie Mioduski

Fabianna Barbarino

Spencer Cho

Aniruddha Srinivasan

Coleman Todd

Tyler Roosth

Department of Computer Science

Texas A&M University

03/08/2023

## Table of Contents

1	Introduction	3
2	Proposed design	5
2.1	Updates to the proposal design	5
2.2	System description	6
2.3	Complete module-wise specifications	8
3	Project management	10
3.1	Updated implementation schedule	10
3.2	Updated validation and testing procedures	11
3.3	Updated division of labor and responsibilities	13
4	Preliminary results	15

## **1 Introduction**

Our main goal is to build a containerized app that has the ability to detect and track vehicles, log driving patterns, and monitor the parking lots with a graphic user interface (GUI). The camera is able to track at least 5 vehicles at a time and the GUI will display the current status of the vehicles. For example, a vehicle could be stopped, parked, or moving. This allows us to increase parking lot safety and use the data recorded to come up with solutions for problems like traffic. We are working with the existing vehicle detection program YOLOv8. The back end for this is implemented with Python. The data collected on the cars include a unique ID, a state, and timestamp. This information is fed into our MongoDB database from which it is sent to the front end and displayed for the users in a table format. The GUI is implemented using REACT, HTML5, CSS, and NodeJS.

### **Needs statement**

According to the National Safety Council (NSC), “tens of thousands of crashes occur in parking lots and garage structures annually, resulting in hundreds of deaths and thousands of injuries” [1]. While not all of these can be prevented, the ability to monitor and recognize traffic patterns in parking lots can greatly help avoid these kinds of accidents from occurring. Moreover, the security personnel watching over the parking lot could have a quicker reaction time if they are monitoring when something occurs. On top of this, traffic has been an ongoing aggravation for many drivers and stores. It keeps people from getting to the store smoothly and having a pleasant shopping experience. Monitoring the traffic patterns in a parking lot can allow us to find ways to improve the flow of traffic and ensure the customers are satisfied.

### **Goals and objectives**

Our goal is to build a containerized app that detects and tracks vehicles across a parking lot and continually logs the driving patterns. This can all be monitored using our user interface which allows two different types of users to view it: security personnel, and data analysts. The security personnel use the GUI as a way to monitor parking lot activity and ensure the safety of all the customers. The data scientist uses the GUI to mine the database and analyze patterns like where the traffic jams are occurring and why. Our first objective is to ensure our vehicle detector can track at least 5 cars. The cameras will be looking out at around 12 feet off the ground, and we must ensure they can successfully track at least 5 cars at this height. Our second objective is to have bounding boxes around the vehicles whose coordinates will be sent to the database. We have actually been able to complete these two objectives already. Our third objective is to tie the backend with the front end successfully for our GUI. This is what we are working on at the moment. Currently, the front end is filled with mock data and videos, but we will be merging and using the correct data soon. We originally wanted our GUI to work in real time with a live video feed, but we have realized that with the time constraints it is best if we focus on using pre recorded mp4 videos. This will allow us to simulate a “live” feed while focusing more on the key parts of the project like labeling and table filtering. Our fourth objective is to ensure our implementation works as a desktop app on a GPU workstation or laptop.

### **Design constraints**

One of our technical constraints is the minimal testing videos that we were provided. We have been using these to train our model and it works great, but it will most likely have to be further trained for parking lots which are very different. Our economical constraint is that we have a limited budget per person on our team for all of the expenses we will have for the project. We have not spent any money so far and are not expecting to, so we do not see this becoming a problem for our completion. Last but not least, our temporal constraint is that we have until April to work on the project. This has been our most impactful constraint so far and we have had to adjust accordingly. First we decided to use pre-trained software instead of training our own because we realized it would take up too much of our time. Secondly we also decided to use the pre recorded videos instead of trying to use a live feed for the security view. This will allow us to focus on the more important parts of the project with the remaining time we have.

### **Validation and Testing Procedures**

We will go deeper into details for this later in the report. The main idea is that the project involves creating an application that uses a camera to track vehicles, log driving patterns, and detect parked, moving, and stopped cars. The database should store historical patterns for specific cameras without including identifying information. The application should have a security guard view and a business analyst view, with the ability to sort and filter data by time/event. This can be tested by our team members as well as the sponsor's team. The app should be able to run in a Docker container and be tested using an MP4 file. At least 10 FPS for 5 cars is required, and the app should have reasonably good accuracy (85%) using a contemporary ML model.

Overall, the project is focused on developing a functional application that can track vehicles and store relevant data for security and analysis purposes. The app's design and features are tailored to the needs of security guards and business analysts, and it should be able to run smoothly in a containerized environment. Testing will be conducted using various MP4 files of vehicles in parking lots to ensure the app meets project expectations in terms of accuracy and speed.

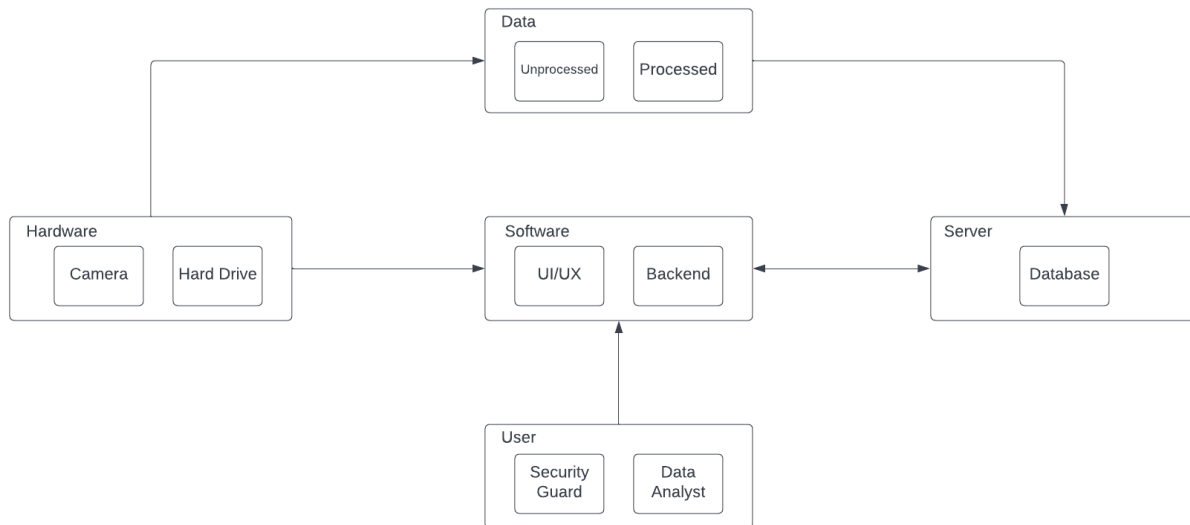
## **2 Proposed design**

### **2.1 Updates to the proposal design (if applicable)**

Updates to the proposal design are due to time constraints. Changes made are those in which we believe are realistic given the remaining time. Features such as live video feed processing, a secondary GUI for analysts, and vehicle speed and additional states are considered stretch goals that would be implemented once the basic functionality and necessary components are finalized. The priority of the project is the creation of a CV pipeline within a container in addition to a security personnel GUI that allows one to view what the pipeline is producing. For our required functionality, the video will then be processed and annotated using YOLOv8 and ByteTrack where bounding boxes will be drawn over the vehicles in frame. Our program will utilize the bounding box labels to display confidence values and vehicle states - parked, stopped, or moving. The difference between the parked and stopped vehicle classifications will be determined based on the amount of time stopped. If the vehicle is motionless it is "stopped," but if it is motionless for over 30 seconds, it may be considered "parked." This data will also be collected and stored for each vehicle along with its assigned object ID to populate the database and user interface for analytics/security personnel.

## 2.2 System description

The system description will consist of a high-level block diagram, and a functional description of the different parts and interfaces.



### Hardware Module:

For the Hardware portion of our Block Diagram, this would consist of our physical media utilized for the project. For example, any cameras, hard drives, or computers, these would fall under this module. The parts of this module currently would include a camera and hard drive, and possibly more if we see the need for anything else throughout the development of the project. The camera would be used as our eyes, in order to gain information over a parking lot. How cars are parked, if they are parked illegally, if they are moving, if there is a traffic jam, so on and so forth. This will be a major implementation we will do if given the time. Next, is our hard drive, this is fairly simple and will be a method of storing our data if needed, as well as using our application on other devices in order to test the functionality of how our application may run given different devices. We would also store any pre-made videos here in order to use our application. The information garnered from the camera or Hard Drive would be sent as Data to our Server and would be used as we see fit.

### Data Module:

The Data module is very simple, but without it we would not be able to continue throughout our project whatsoever. To start off, the data gathered from the cameras of our Hardware portion of our Block Diagram is taken, purely as unprocessed, unfiltered data, and would then be sent to our Server in order to house this information until we decide to use it and determine what is occurring in the data itself, which would take place in our application via machine learning techniques. With this, while it may seem like not much, would be a major source in how we may be able to create our own

model to use for our machine learning as we progress through the development process of our project.

#### User Module:

The User module is how a User will interact with our application. And how our entire project is made useful, is all dependent on the User. The User will mainly consist of two positions or viewpoints as of the beginning of our development, a Bodyguard view and an Analyst view. The idea is that the Bodyguard's role as a User is to use the information gathered from the Hardware and Software portion of the project in order to take note of any mishaps that may occur while they are on the job, which would most likely occur from the viewpoint of live video feeds. For example, if our application determines that a vehicle is illegally parked, the Bodyguard would in some way be notified so that they may be able to take the appropriate action required to remedy the situation. The same can be said if there was an accident, and so on. This notification while not currently set in stone can be a sound alert, a text message, an on screen display notification, etc... On the other hand for an Analyst the use of this application completely changes. Their goal will be to analyze the data from the Data and Server module, and take note of any unusual occurrences that our application may find, therefore their viewpoint would revolve more around the Data and less on video feeds.. For example, there may be an influx of cars causing a traffic jam at a certain spot, the Analyst would use our Data and determine why that may be occurring and try to solve it, or there may be a location prone to numerous accidents, and again the Analyst would be in charge of trying to fix this issues in some way or form. The Data the Analyst uses would be gathered and organized in some way to make this process easier via algorithms we end up implementing.

#### Server Module:

The Server module consists of a database that would house not only the untrained data from the video feed, but the processed data used to train our application. The software would pull the untrained data from the server side, process it using machine learning algorithms, and push the processed data back into the database, mongoDB. Live video feed would be taken and transformed into data that would allow for further training of the model which is then stored in the server. The server would provide a consistent benchmark for the software by allowing us to view the training unfold over time.

#### Software Module:

The Software module consists of both a frontend and a backend. The backend will do much of the processing of data utilizing the programming language Python. Within Python, various machine learning modules will be used such as pytorch, sklearn, and pandas in order to train our application to properly identify vehicle positioning. This processed data will in turn be sent to the server for storage and there is hope that following training using recorded video, we will be able to transition to a live video feed for real time data processing. The software will primarily be focused on that of machine learning, a subset of artificial intelligence, that allows for the training of an application to classify and predict scenarios based on predetermined models. This involves using

features and labels in order to fit data to models we deem appropriate and should the application be incorrect in its determination, we shall alter the program and the model to fit our specifications. We will also use Docker in order to containerize our application, making the testing, building, and deployment easier all the while allowing for version control. In addition to a backend, a frontend consisting of custom components created within React.js will also be created in order to make navigation and use of the application easier on users.

### 2.3 Complete module-wise specifications

Provide a detailed design of each subsystem (both hardware and software), including

- Circuit and logic diagrams
- Interfaces and pin-outs
- Timing diagrams and waveforms
- Software processes with their inputs and outputs
- Complete parts list

#### Machine Learning Specifications:

The Machine Learning is made up of two components currently. We are running our current code in a Jupyter Notebook on Google Colab since the live requirement has been removed. The model is currently running on a 1GB VRAM GPU.

#### YoloV8:

YoloV8 is the latest version of the You only Look Once Detection model. Using this model, we can identify vehicles in the camera frame and assign them labels including car, motorcycle, and truck. We have set this model to only identify vehicles that it is at least 75% confident about. Anecdotally, we have found strong evidence that this is a suitable model for our needs

#### Bytetrack:

Bytetrack is the object tracking module that we attach to YoloV8. It assigns persistent id throughout the video frame and allows us to track movement and thus assign states. This is how we can identify and record data in the format we specified in the Database Specification section. There is some ‘connectivity’ code between Yolo and Bytetrack since they don’t work seamlessly together out of the box. We have written code to track the speed in pixels/frame in order to determine whether a vehicle is moving or not. We set the threshold value to 1.5 and found that this does a reasonably accurate job. This allows us to determine ‘Stopped’ and ‘Moving’. There are some issues with Bytetrack, namely that the bounding boxes can get confused when an object is moving very close to another object. We added some logic in order to minimize this effect.

#### Performance:

Currently this model runs at around 70fps with more than 10 vehicles in the frame, well above our specifications. If we were to run it live, we would have it running on a server with a 'pubsub' method to send data to the database. In the future, we would like to increase accuracy for edge cases.

#### Database Specifications:

The Database is made from using MongoDB. Currently it is set up to where the following information is logged: Unique ID, X, Y, State, Timestamp, as well as Bounding Box Width, and Bounding Box Height. This information would be taken from the Machine Learning portion and entered into the Database. These would then be utilized by the Front End in order to give the Security or Analyst page the information that is required. The information from the Database is hosted on MongoDB Atlas which is MongoDB's own cloud platform where information can be stored on the cloud. It's a nice alternative to some other choices with its ease of use and collaboration with MongoDB since they are developed hand-in-hand. Now the information can be queried fairly easily from any point as long as the appropriate access is given.

#### Back-End Specifications:

The back end is currently made up of Node JS as well as Python. Node JS is used and connects with the Database and interacts with the frontend by allowing calls to be made to the database in order to retrieve the information needed to fill the tables that correspond to the submitted video. This is done via a Cloud connection to MongoDB Atlas, which would be where the information from the database would be stored as well as taken. On the other hand, we use Python when it comes to connecting the Machine Learning portion of the project with the Database. Since this is where the submitted video will be sent to in order to get that information. We then make calls on the backend in order to submit that information regarding the submitted video into the MongoDB database which will then in turn be used when showcasing the information related to the attached video in regards to the frontend within the tables.

#### Front-End Specifications:

The front end has been implemented using REACT, HTML5, and CSS. It is set up to interact with the node.js backend for api calls. These are used to get the data for the tables as well as the queries needed for the analyst view. There are two main pages/views for the front end: security guard view, and analyst view. The security guard sees a "live" view of the camera and a table to the right hand side with the statuses of the vehicles as time goes on. The video is currently a prerecorded mp4 video since we have determined the "live" feed to be a stretch goal. On the table they are able to see the car id, time stamp of the data, and the status. They are currently also able to filter the data by a specific status. For example, if I want to see all the parked cars throughout the day I would only check the parked checkbox. The analyst page is very similar with both a video and table with the same information. The analyst however, can inspect the details more closely. They have to choose from multiple videos first. From there, they are able to see the video and its associated data loaded on the table. They can also click on a timestamp and be redirected to that time in the video. On top of this, the analyst also has access to special queries like a sum of the amount of stopped cars that day.

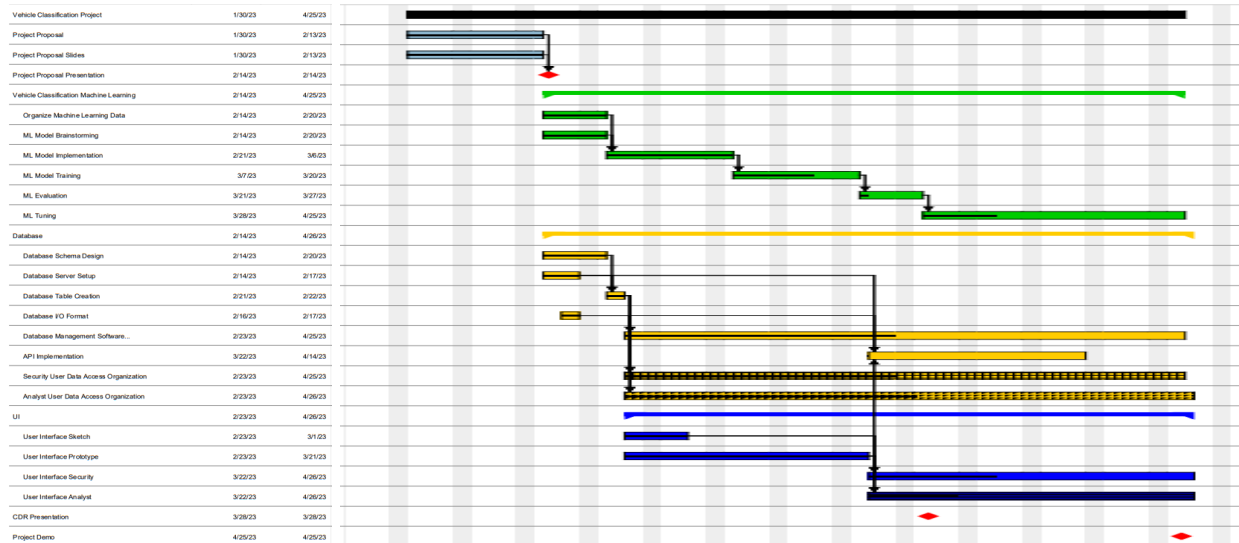


Overall, both views are very similar but the analyst has access to a few more tools. There is currently no login needed. Any user can access both views at the moment as well since it is not part of our scope to separate them.

### 3 Project management

#### 3.1 Updated implementation schedule

Updated Gantt Chart:



Implementation of the application required a meeting with our client in order to reign in the scope of the project towards something that would be more feasible in the given cycle of time. This meant that we had to decide early not to pursue some of the states that we had initially proposed doing, such as illegally parked and speeding, in order for the framework of the application to be made in a timely manner. Following this newly established scope, we then had to decide on what manner we would go about completing each module of the project. We decided to work on the ML side first, as that was the primary module with which we would get information from for application use. Once we had a rudimentary implementation of the recognition software, we moved towards the database and front end development. While the information given by the ML module can be worked on independently of the other two, the database and front end are dependent on one another as information storage and display are reliant on how the other needs to be formatted. This leads to the database team and front end team to need to be in communication with each other so that any issues or changes can be made in a timely manner.

At the current time, we primarily need to finalize each sections' interaction with the others. For the Machine Learning modules, we are continuing to work on the recognition algorithm and output, as we are trying to raise confidence and implement recognizing a car as parked as opposed to just stopped. For the Database module, we are working on the API calls in order to communicate between the database and front end modules. We are also having to remain in communication with the ML team so that our schema is up to date with the information output from the recognition software. Finally, the Front End is having to work on pulling the correct information from the servers and displaying it correctly. Front end also has to ensure that the UI is user friendly and easy to use in accordance with client specifications.

Our primary critical path is dependent on implementing the API calls necessary for communication between the interface and the database server. Ideally we will be able to complete the API calls by the end of the week of April 1-7. Following this, we will have to perform a rudimentary User Study in order to gain insight into possible UI changes that might need to be done. This would likely be done over that following week, and our last week would consist of general tweaks for performance enhancement and ease of use.

### **3.2 Updated validation and testing procedures**

#### **Changelog from Project Proposal**

- 'Parked' can be an extension of 'Stopped' state (eg: if car is stopped for more than 30 seconds, it is considered parked)
- GUI should be able to specify how long a car is 'Stopped' for it to be considered 'Parked'
- Annotated video does not have to be live - can be premade by ML script and uploaded to Web App

If we divide our project into Tracking, Database, GUI, Infrastructure - here are the tentative metrics for success.

#### **Tracking:**

Model should be able to track at least 5 vehicles at an FPS greater than 10. Should detect parked, stopped, not stopped cars. The accuracy should be reasonably good (85%). According to our project manager, this is a priority for the application. Currently the model achieves this, with a very high observed accuracy rate and around 60-70 fps using a GPU on Google Colab. For the 'Parked' state, instead of detecting whether a car is parked within the lines, we are cleared to use our 'Stopped' state instead to check if a car has been stopped for more than 30 seconds. If so, we will mark it as 'Parked'.

#### **Database:**

Log driving patterns with x,y,bbox width, bbox height, timestamp, and state. The database should also make sure that identifying information is not present. Temporary ID from Deepsort is ok and does not need to persist. Queries should be able to extract historical patterns for specific cameras.

#### **GUI:**

There should be two views: one for a security guard, and one for a business analyst. The security guard view should be tailored towards live, ondemand footage. Security guards would be able to view a live feed. They should also be able to easily see incidents sorted by time in a table to the side. Clicking on an event should bring up the video recording at the time. The business analyst view should focus more on historical data as well as patterns. At minimum they

should see a table where they can sort and filter by time/event. Both views should be able to customize the time it takes to determine

In terms of technology, a web app is preferred, but is a stretch goal. Currently we are using a React App

#### Infrastructure/Deployment:

Our app should be able to run in a Docker container. The intent of this is so that our project assigner can run our application on any infrastructure/cloud environment they deem necessary. Additionally, we should find minimum specs needed so that our application will run smoothly, and explore the relationship between increasing specs and the speed of our ML detection. It is likely that we will need infrastructure with a dedicated GPU for best performance in ML. Currently we are using a Google Colab GPU with 2 gigs of VRAM, and we can work further towards testing it on live infrastructure.

#### Testing:

Once we have established our application, we want to be able to test and see if it conforms to our project expectations. We will be using a single usb webcam to simulate a live feed, mounted at a similar location to how our project assigner will mount them, over a parking lot. Alternatively our project manager cleared us to use an mp4 file that is fed to our application as if it were live.

We will then use our application to view this live feed, and track the accuracy and speed (at least 10FPS for 5 cars, 85% accuracy of state tracking) . Security guards should be able to view any incidents by clicking on the timestamp of a particular incident and watching a replay. We will also use the Business Analyst view in order to track historical data, in this way testing the database. Our app should be running in a container.

Currently the ML model runs at 70fps for 12 vehicles, well above our benchmarks. We also only track vehicles above a 75 percent confidence level.

After discussion with our project manager, the strictly necessary events to track are **stopped/parked/not stopped**. Speeding and illegally parked are out of MVP scope as they may require more advanced analysis. However, if the application is fully running with the necessary events we may add them later on

Additionally our project manager has cleared us to use an annotated mp4 file that is premade our application. This would make the project testing less rigorous, as we can run our machine learning model to annotate the video separately from our web application. We can simply upload

If our application is able to successfully complete this test, we will consider that to be our MVP (minimum viable product). There are other potential considerations when looking at a customer that operates on the scale of ours. but we consider this a good benchmark for a proof of concept.

#### Demonstration:

Ideally the app will be demoed on Walmart's own infrastructure if we can get the container working with live video feeds. However due to the complexities, it's also acceptable if we

pre-make the annotated video and upload it to our web app and run it on our personal computers.

Summary:

App must run in a container

App must detect stopped, not stopped, parked

App must have a database with x,y,timestamp,event,bbox width, bbox height

App must have two views

Update the procedures that you will use to test that your system does what it was designed to do. How are you going to measure the extent to which your system solves the need? What kind of demonstration will you provide as proof of a working system? We're not talking here about component testing, which should be part of the implementation schedule. These validation and testing procedures are at the system level (a working prototype).

### **3.3 Updated division of labor and responsibilities**

We have maintained our initial plan of having pairs of people in charge of each module, however we had some changes to who was working on which module from the initial proposal. Aniruddha Srinivasan and Jacqueline Mioduski are in charge of the Machine Learning module, Coleman Todd and Tyler Roosth are in charge of the database module, and Fabianna Barbarino and Spencer Cho are in charge of the front end. Each pair is in charge of completing scheduled tasks for the given module and updating the team on any adjustments needed towards the schedule or outside assistance needed.

The first half of the project timeline was spent creating the base structures with which we will be bringing together for our final product. Each module had to establish what we were aiming to accomplish and then create a module which would be able to work with the others. For the Machine Learning module, this meant that they had to first get data (which we were able to get some from our client), determine which states we would be looking for, and then creating the recognition software. For the database module, they first had to determine whether they would be working in an SQL or NoSQL framework. Following the decision to use MongoDB, they then had to determine the schema and set up a server so that information could be stored over multiple instances rather than having any inputs be lost in a local database. Finally, the front end had to prototype the views that would be implemented, create the application in Docker, and then implement the video and data display.

Past Due Dates:

- Machine Learning
  - Data Gathering - February 10th
  - States Guidelines - February 14th
  - Rudimentary Implementation February 18th
  - Refine for Stopped and Moving - March 6th

- Meet Confidence Threshold and base Specs - March 12th
- Database
  - Language Decision - February 10th
  - Database Schema - February 14th
  - Database Creation - February 22nd
  - Server Creation - March 7th
- Front End
  - Design Prototype - February 6th
  - Docker Application Creation - February 12th
  - Video Display - February 18th
  - Data Table Display March 7th

Our future due dates revolve around finalizing the connections between each module and then refining their implementations until they meet or exceed the client's requirements. For the machine learning module, they will be continuing to work on the confidence rating and working on recognizing parked cars. They also have a stretch goal for vehicle speed, but fully implementing the speeding state might not be feasible. The database module needs to create the necessary API calls in order to send the data requested by the front end from the server to the front end. This and adjusting tables for this implementation of the project are the primary deadlines that must be met. Finally, once API calls have been made, the front end must complete User Studies in order to see how potential users interact with the application and whether any changes need to be made in order to meet client requirements. Finally, once all of the modules are communicating and able to be run, we have possible stretch goals that if we have a good enough time frame then we will attempt to work on them. Parking detection will likely be a joint deadline between the ML module and database module, in case that the ML module is unable to determine if a car is parked for a prerequisite amount of time for a parked status, the database server will be necessary for establishing how long the vehicle has been parked.

#### Upcoming Due Dates:

- Machine Learning
  - Parked detection - April 7th
    - Aniruddha & Jaqueline
- Database
  - API Calls - April 7th
    - Tyler & Coleman
- Front End
  - Display API returns - April 10th
    - Fabianna & Spencer
  - User Study - April 14th
    - All Team Members
- Finalized Product
  - April 20th
    - All Team members

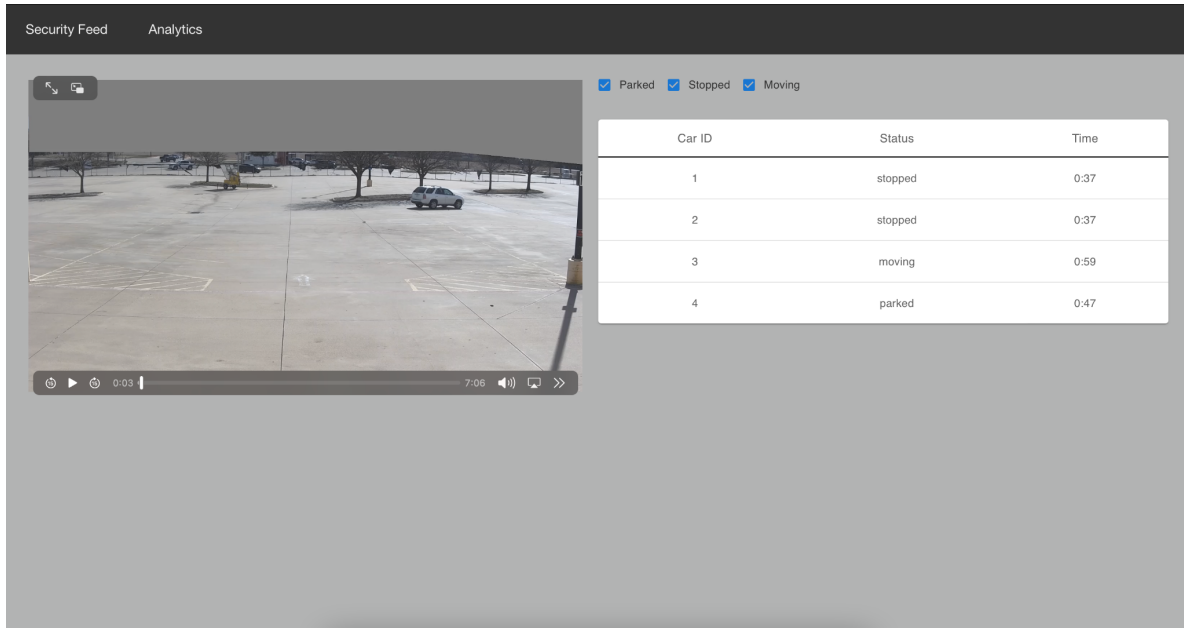
## 4 Preliminary results

Provide any test results and demo of completed parts of the system at the time of the CDR.

### ML Video of IDs and Labels

#### Front End Views Progress

##### Security Guard View



The screenshot displays a security guard view interface. On the left, there is a video feed of a parking lot with a car in the center. The video player has a progress bar at 0:03 and a total duration of 7:06. On the right, there is a table with three columns: Car ID, Status, and Time. Above the table, there are three checkboxes: ☒ Parked, ☒ Stopped, and ☒ Moving.

Car ID	Status	Time
1	stopped	0:37
2	stopped	0:37
3	moving	0:59
4	parked	0:47

##### Analyst View

Security Feed

Analytics

0

19/03/2020 08:32:22 Qui

#12 car 0.89

#8 car 0.93

#5 car 0.89

#4 car 0.93

#3 car 0.89

#2 car 0.89

#1 car 0.89

#6 car 0.89

#7 car 0.91

ESTACIONAMIENTO

☒ Parked☒ Stopped☒ Moving

Car ID	Status	Time
1	stopped	0:37
3	moving	0:59
4	parked	0:47

Analyze Data