iValet: An Intelligent Parking Lot Management System and Interface
Final Report

ECE4872-L4 Senior Design Project

Team Name: iValet

Team Number: sd22p37

Project Faculty Advisor: Dr. Patricio Vela

Faiza Yousuf, Computer Engineering, fyousuf6, fyousuf6@gatech.edu

Wei Xiong Toh, Electrical Engineering, wtoh7, wtoh7@gatech.edu

Kelin Yu, Electrical Engineering, kyu85, coliny@gatech.edu

Yunchu Feng, Computer Engineering, yfeng336, yfeng336@gatech.edu

Submitted

2022 May 3

Table of Contents

Executive Summary

The iValet intelligent parking lot management system automatically directs drivers to the nearest vacant parking spot upon entering a crowded parking lot. The system consists of a camera, machine learning development board, a PostgreSQL server, and a user interface (web application). The camera is used to take photos of the entire parking lot, the development board runs segmentation and classification algorithms on those photos, the SQL server contains data about each parking spot that is written to by the image processing models, a path-planning algorithm, and the UI, while the web application shows end users the directions to the empty parking spots based on the location of the parking lot entrance.

When drivers enter the parking lot, they will need to scan a QR code that leads them to the landing page for the iValet web application (formerly https://www.ivalet-crc.com/, the domain will be removed after the EXPO). Once they login through Google and input their license plate and handicap needs, the system identifies the nearest vacant parking spot for their vehicle from the results of the image segmentation and classification models. The system currently uses hardcoded maps (KML maps) to direct drivers from the entrance to their assigned parking spot.

The ease at which drivers are able to find parking spaces for their vehicles can significantly affect their mood and overall experience of the subsequent activity they are taking part in. The use of iValet will help minimize some of the frustration, anger and oftentimes avoidable waiting that arises from being unable to locate a vacant spot in a crowded parking lot. The iValet system as is costs an estimated $198.15.

Nomenclature

Coral – Google Coral Development Board

iValet – entire project/system

SQL server – PostgreSQL server

UI - user interface; the web application users interact with

1. Introduction

The iValet Parking Lot Management System is a novel and affordable parking management system that automatically directs drivers to the next available parking spot for their type of vehicle. The system consists of a camera, development board, a SQL server, and a web application. The team's budget limit was $1000, and the cost of the current design was $198.15.

1.1 Objective

The objective of iValet is to provide drivers with a seamless experience when searching for empty parking spaces in a crowded parking lot (ex: stadiums, amusement parks, etc.). A camera mounted in a position that captures the aerial view of the parking lot provides the system with images of the lot that are updated every thirty seconds. Individual frames from the image feed are processed by the lot detection algorithm on a Google Coral and identifies which lots are currently occupied or empty. A path finding algorithm will subsequently sort, along with the location information provided by the users' mobile phone, to calculate the shortest path from the users' current location to the aforementioned parking spot. This information will then be displayed on the web application for the drivers. **Figure 1** shows the block diagram of the iValet system.

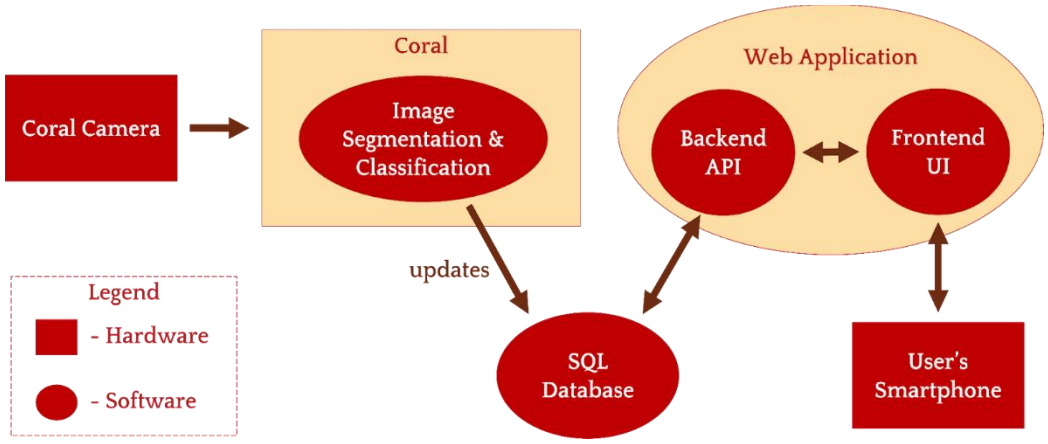


Figure 1. System overview block diagram of iValet

1.2 Motivation

According to parking data from 100,000 locations across more than 8,000 cities, INRIX, a transportation analytics firm, found that on average, drivers in the US, UK and Germany spend 17, 41 and 44 hours per year respectively searching for parking [1]. This amounts to an estimated $345 per driver in terms of wasted time, fuel and emissions in the US [1]. Products that are available in the market such as PlacePod [2], an Internet-of-Things (IoT) enabled magnetic sensor that detects the presence of cars in each available parking spot, primarily require relatively higher set up costs because of the large number of sensors that needs to be purchased for a large parking lot vis-à-vis the more cost-efficient prototype that the team aims to develop. The time and manpower needed to set up the prototype will also be significantly lower than that of traditional IoT devices since it (currently) uses a single camera to detect parked cars within a wide area (66 parking spots). iValet provides owners of large parking spaces, such as supermarkets or sporting venues, a cheap solution to reducing the time drivers have to spend circling their parking lots, which ultimately leads to better customer experience, prolonged engagement at the venue, reduced fuel costs for drivers and lower carbon emissions.

1.3 Background

While commercial hardware-based IoT solutions can be used to tackle the parking problem stated above, there are no software-based alternatives that provide an end-to-end service that identifies empty parking spaces based on the type of vehicle and provides drivers with the directions to those empty lots in real time. iValet relies on two key algorithms to work successfully.

For the iValet system to accurately identify the parking spaces in a particular parking lot, the image segmentation model needs to be trained on images of the parking spaces from that specific parking lot. Image segmentation algorithms work by classifying each pixel of a given image with a pixel-wise mask. Different labels in the mask correspond to a different region of interest. In the case of a parking lot, pixels

can be classified into (1) empty lot, (2) occupied lot, (3) path for cars. Image segmentation models learn to classify each pixel by training on labeled images that are representative of the images used in deployment and iteratively refine their classification criteria. This training process usually takes a significant amount of time before the algorithms can perform well. However, to speed up the training process such that the iValet system is agnostic to the orientations of the parking spaces, layouts and weather conditions, transfer learning can be used. Transfer learning involves using a pre-trained model that performs relatively well on a particular dataset and training it on a new dataset. This method significantly reduces training time because the model only needs to learn the difference in orientation and not learn to distinguish empty lots from occupied lots again.

The next key component to the iValet system is the pathfinding algorithm. With the coordinates of the vacant lot and the driver's location, the system calculates the shortest path between the two coordinates subject to certain constraints using the pathfinding algorithm. An example of a constraint that might be imposed in a parking lot is the direction of traffic. Pathfinding algorithms generally work by discretizing the given space into a grid and performs a grid-based search over the entire space based on a certain heuristic measure to search for the shortest path between the two points. After calculating the shortest path, iValet will then relay this information to the driver via the web application.

2. Project Description, Customer Requirements, and Goals

2.1 Project Description

The iValet managing system will consist of 4 main components: the hardware system, camera detecting system, SQL database, and user interface. The hardware system contains a Google coral dev board, a coral camera, a printed coral case, and a tripod. We will use the coral camera to take photos and use the coral dev board to process those photos with our trained detection model. We also have a printed coral case and a tripod to fix a specific angle for our camera. For the camera detecting system, it contains a segmentation

algorithm and a trained classifier. We will use the segmentation algorithm to segment all those parking slots out and test if they are occupied or not with the classifier. It will be connected to the database, which shows some identities of each parking slot. In the database, it shows some identities of each parking slot such as slot id, occupied or not, distance to the entrance, etc. The database will connect to our user interface to help it find the best path. For the user interface, with data that comes from the database and our navigation algorithms, it can find the closest parking slot, find the car, and find the way out.

2.2 Customer & Engineering Requirements

The requirements can be separated into several parts: cameras, time limits, accuracy, and cost. Also, working together with some general requirements, QFD table **Figure 2** denotes our requirements.

Cameras:

Heights: Because of the heights of the building, it needs to be no more than 15 meters.

Maximum area: The largest parking lot to detect should not be greater than 10000 m^2.

Frame Rate: 24 frames/s is enough for us to work with real-time detection.

Time limits:

We need to let the camera take a photo every 30 seconds, and the database will be updated each time. Also, we need to change the slot to be occupied once a user chooses it.

Accuracy:

The accuracy of the detection algorithm needs to be greater than 90%. The accuracy of the priority of parking slots needs to be 100% because we can get distance data from Google map.

Cost:

We do not want the total cost of this project to be more than 1000 dollars (team budget). Currently the system costs no more than 200 dollars.

Prices:

Purchase prices for vendors and involvement of stakeholders would be dependent on the theoretical success of the product. Installation of a single unit of iValet (at its current design) is less than $200.

QFD Table of our requirements:

		Engineering Requirements							Benchmarks	
		Max height of cameras	Accuracy of detection	Ideal algorithm run time	Maximum area of vision (size of parking lot	Camera Frame Rate	Camera Resolution	Cost	Daylight	Night/Artificial Light
General Requirement	Parking lot is visible	X			X	X	X		X	X
	System Maintains power			X	X					
	User Interface is coherent		X	X						
	Maintains communication with app			X	X					
	Minimal power consumption (cameras)			X	X	X	X			
	Neatly installed	X			X			X		
	Can be replacated to reasonable variety of lots	X	X	X	X	X	X		X	X
	Units	meters	%	seconds	m^2	Frames/sec	Megapixels	Dollars		
		15	90	300	10000 (estimate)	24	2	700		
		Engineering Targets								

Figure 2. QFD Table, see full image in Appendix A.

2.3 Goals

Users:

Vendors such as retailers, restaurants, sport stadiums, etc., with large parking lots would benefit from an efficient parking system.

Functionality:

- Our project can detect empty slots, find the best path to it, and show that information on the UI.
- The camera will take a photo every 30 seconds.
- Our segmentation algorithm and trained classifier will process those photos.
- The accuracy of our detection system should achieve 90%.
- Some identities like slots ids, distance to entrance, occupied or not, etc. will be shown in the database. That information will be updated every 30 seconds.
- Once the user wants to park his/her car, our UI will show the closest parking slot and the path to this location.
- Once the user parks in the right place, the UI will show the location of the car.
- Once the user wants to leave the parking lot, the UI will show the way out.

3. Technical Specifications & Verification

Main hardware components include the Google Coral Dev Board [3] and its companion Google Camera [4**]**.

3.1 Google Coral Development Board with 1 GB RAM [3]

Table 1. Specifications of Google Coral Board

Feature/ Description	Value from Source/Data Sheet
Dimensions	88 mm x 60 mm x 24 mm
OS	Mendel (Debian derivative)
CPU	NXP i.MX 8M SoC (quad Cortex-A53, Cortex-M4F)
GPU	Integrated GC7000 Lite Graphics
Machine Learning accelerator	Google Edge TPU coprocessor: 4 TOPS (int8); 2 TOPS per watt
RAM	1 GB LPDDR4
Wireless	Wi-Fi 2x2 MIMO (802.11b/g/n/ac 2.4/5GHz) and Bluetooth 4.2
Flash Memory	8 GB eMMC, MicroSD slot
USB	Type-C OTG; Type-C power; Type-A 3.0 host; Micro-B serial console
LAN	Gigabit Ethernet port
Audio	3.5mm audio jack (CTIA compliant); Digital PDM microphone (x2); 2.54mm 4-pin terminal for stereo speakers
Video	HDMI 2.0a (full size); 39-pin FFC connector for MIPI-DSI display (4-lane); 24-pin FFC connector for MIPI-CSI2 camera (4-lane)
Power	5V DC (USB Type-C)
Cost	$132.99

3.2 Google Coral Camera [4]

Table 2. Specifications of Google Coral Camera

Feature/ Description	Value from Source/Data Sheet
Sensor	Omnivision OV5645 SoC (built in ISP)
Focus	Auto focus, focal length 2.5mm, range 10cm-infinity
Field of View	84.0 degrees / 87.6 degrees
ISP Functions	Auto exposure control, auto white balance, auto band filter, auto 50/60Hz lumination, auto black level calibration
Connections	MIPI-CSI, dual lane MIPI interface
Dimensions	25 mm x 25 mm x 6.98 mm
Cable Size	150 mm x 12.5 mm
Cost	$21.99

4. Design Approach and Details

The detailed implementation of our project is presented in this section, including initial design considerations, methods and alternatives.

4.1 Design Concept Ideation, Constraints, Alternatives, and Tradeoffs

4.1.1 Development Board with GPU and Camera

The main hardware component of iValet is an embedded computer that is capable of both extracting images from a compatible peripheral camera device and running our segmentation and classification machine learning models with an on-board GPU. It (the computer) also must update the database with the relevant

status for each parking spot. Several factors were considered when choosing this key piece of hardware: size, camera frame rate, ease of use and compute power. Since we intended for the overall system to be easy to install, the computer should not be too large nor heavy. For our purposes of updating the database occasionally, the camera frame rate can be low, and a moderately high resolution is sufficient, which significantly lowers costs.

The NVIDIA Jetson Nano fits all the aforementioned constraints perfectly. It is inexpensive, has adequate CPU and GPU compute, several compatible camera modules, and a large open-source community to facilitate troubleshooting. However, due to the surge in demand for the Jetson family of devices at the start of the semester (and production issues due to COVID making them more expensive), we had to settle for the Google Coral Development Board, a close alternative to the Nano.

Ensuring that the camera remains in a stable, fixed position is pivotal to the operation of the system. This allows the Google Coral to update occupancy status for all parking lots within the original field of view of the camera. Unfortunately, the team was not permitted to have permanent mounts inside or affixed to any campus buildings. Hence, plastic casings to secure the camera and the Coral board were 3D-printed, as shown in **Figure 3**. The camera mount was specifically designed to allow users to change its mounting angle by loosening the screws at the base of the mount. The external casing of the camera and the coral board was then secured to a tripod to stabilize the video feed, shown in **Figure 4**.

Figure 3. Google Coral encased with Camera mounted atop

Figure 4. Hardware stabilized with tripod

4.1.2 Software

A segmentation model (Mask R-CNN) and a convolutional neural network were trained separately. The former identifies bounding boxes of all parking spots within a given image, whereas the latter classifies if a particular parking spot is empty or occupied. Mask R-CNN is a general framework for object instance segmentation. It can detect, recognize, and follow objects, as well as segment with a mask in a pixel-level accuracy. In this project, we only needed to use the bounding box to segment each parking slot out, so we wrote a method to output the following bounding box instead of masks. For the classifier, we built three convolution layers with Max pooling and a dense layer. This classifier will output binary data: 0-Empty and 1-occupied. Our initial pipeline was as follows: obtain the parking spots from the segmentation model and feed each cropped parking spot into the classifier before updating the database. However, this approach was not successful since the performance of the Mask-RCNN was not as expected. The reason might be the different camera angles and lack of images from the dataset that was used during training. We can train a larger dataset to fix this problem.

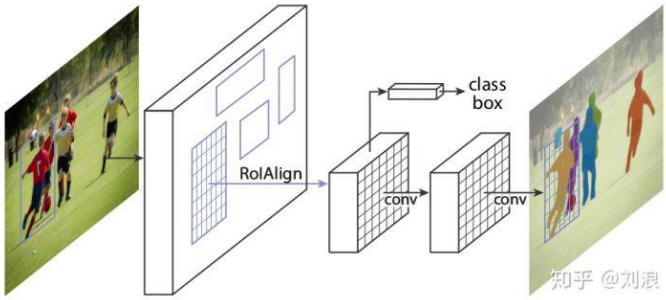


Figure 5. Mask R-CNN Network diagram

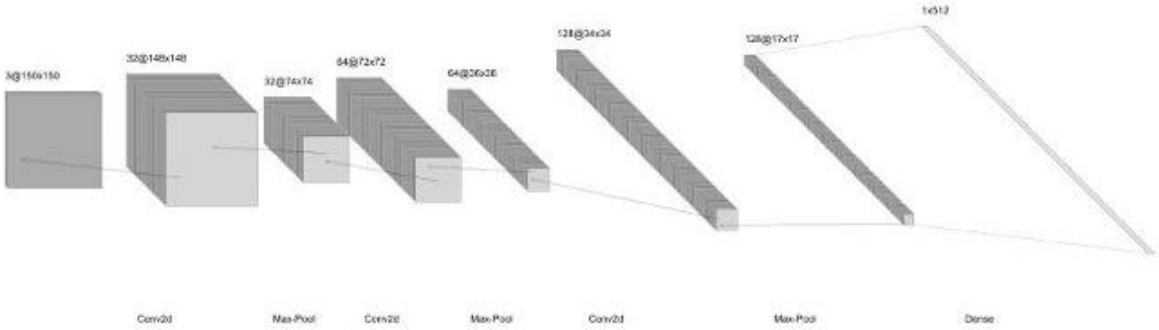


Figure 6. Convolutional Neural Network Diagram

An alternate solution was implemented in which the bounding boxes for each parking spot was predefined and subsequently parsed as input to the classifier. Output from the classifier then allowed Coral to update the database with a Python script. This algorithm finally reaches 95% accuracy in the parking lot of the CRC. The problem with this model is that it cannot work in an unknown parking lot immediately because it relies on predefined images.

4.1.3 SQL Backend

Information on each visible parking spot is stored in a SQL server. The data consists of the following entries:

Table 3. SQL Server data fields

Name of Field	Data Type

Lot id (primary key)	Integer
Empty	Integer
Distance (to entrance)	Integer
License plate	String
Handicap	Integer
Time parked	DateTime

The “empty” and “handicap” fields are used to identify if the spot is currently empty or is for handicapped drivers. Distance to each parking spot from the entrance is calculated and saved in the database. The “license plate” and “time parked” fields are updated once a user is assigned a parking spot. Tracking the status of “time parked” further enables the system to calculate the total parking fee (**Figure 12**) for users upon leaving the parking lot, and an online payment feature was added to provide users with a seamless exit.

Storing our data in an SQL database has many advantages. Of which, the main one is that relevant parking lot information can be easily obtained with relatively simple SQL queries, allowing the backend API calls to quickly identify the lot id of the nearest parking spot or the spot that users had parked in order to park and find their cars respectively.

4.1.4 Graphic User Interface

A web application was chosen as the platform to host the GUI since the speed of sending information to a new user was prioritized given that he or she did not have to spend time to download an app. The link to the web application could also be shared with users through the form of a QR code to be pasted at the gate entry.

A simple layout with large buttons to park and locate one’s vehicle contributes to the UI’s ease of use, depicted in **Figure 7.** Once users input their license plate information (**Figure 8**), the GUI captures the

information, sends the relevant API calls to get and update information on the database with the license plate of the vehicle that just entered.

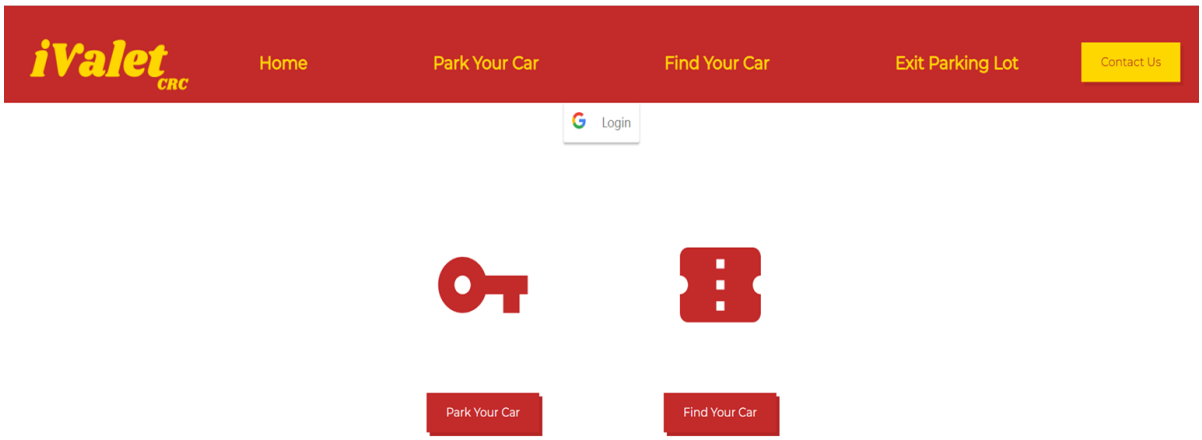


Figure 7. Landing page of web application

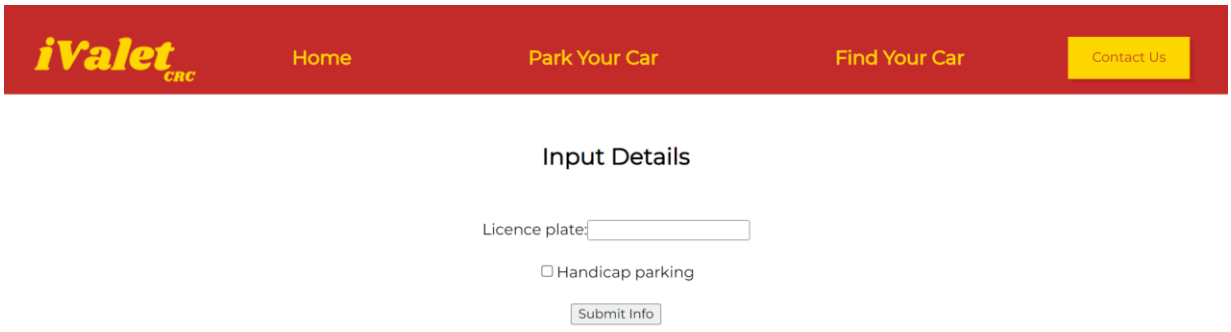


Figure 8. Form to capture user input

The three most relevant pages of the site are the different navigation pages to help the user park their car, find their car, and exit the lot, as see in **Figure 9, 10**, and **11**. All three maps are constructed using the Google Maps JavaScript API and a hardcode KML (Keyhole Markup Language file; similar to an XML that contains coordinates and styling details) that displays the red path and marker on the map.

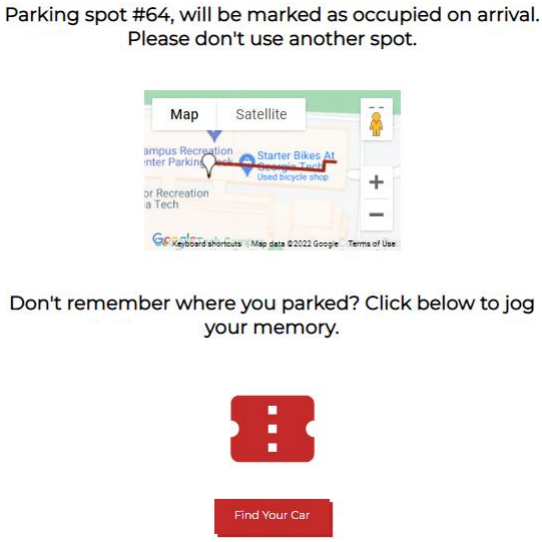


Figure 9. UI Map to park

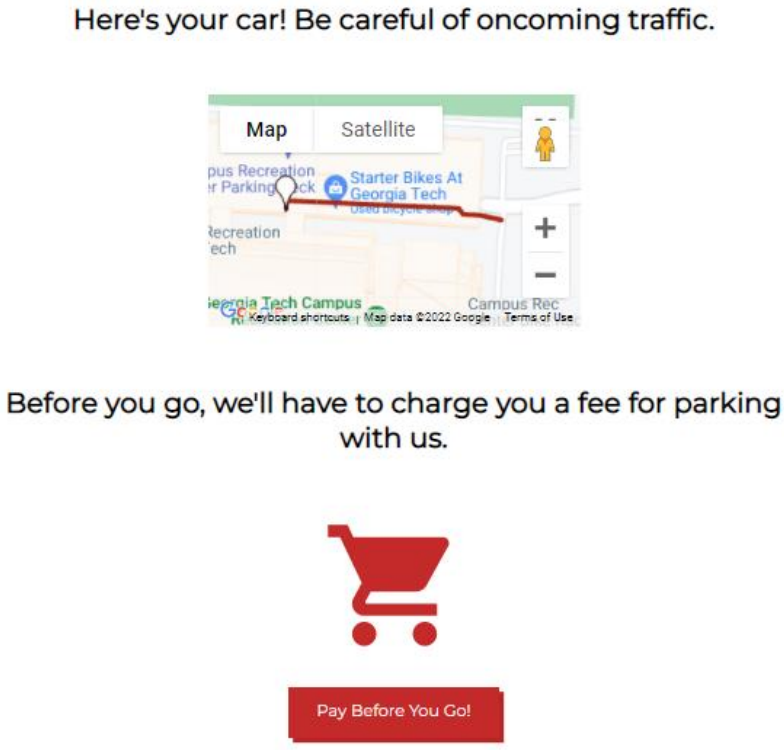


Figure 10. UI map to find where the car was parked

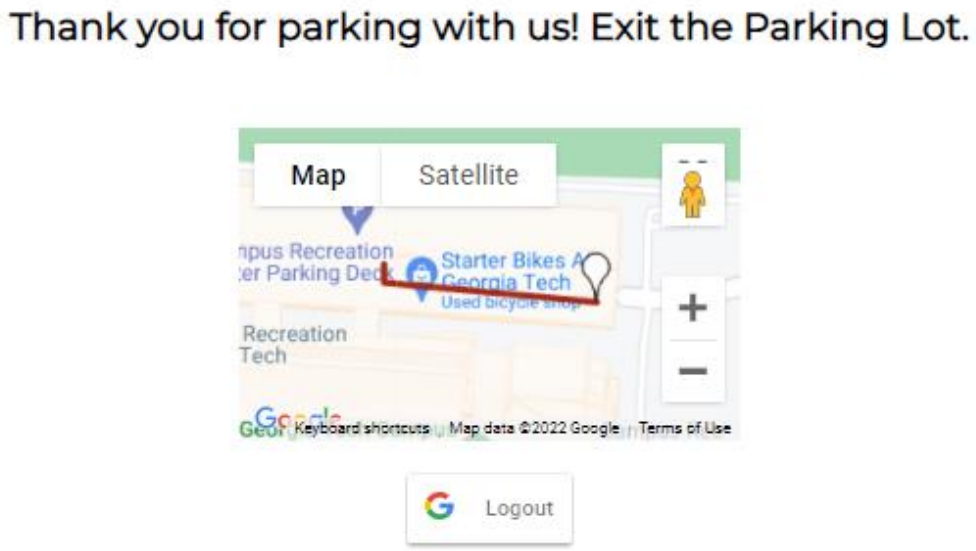


Figure 11. UI map displaying the path from the parking spot to the exit

Prior to viewing the "exit lot map" in **Figure 11,** users are directed to a payment screen (depicted in **Figure 12 and Figure 13**).

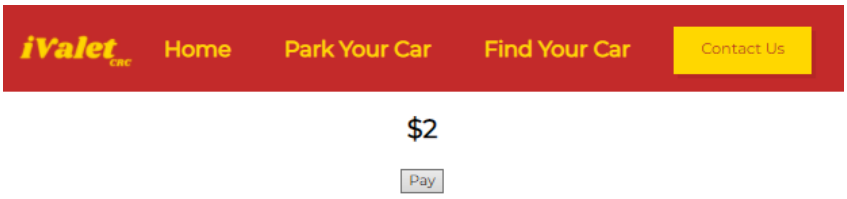


Figure 12. Calculated parking fee

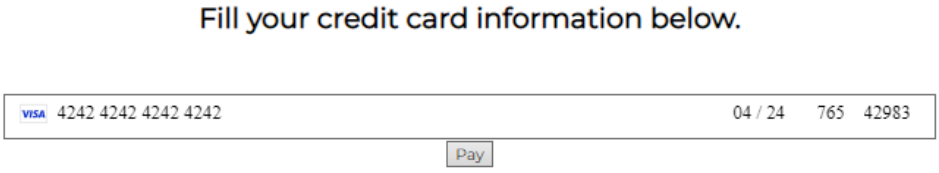


Figure 13. Payment screen

4.2 Engineering Analyses and Experiment

The various components of the system were separated into smaller groups and subgroups that were individually tested before integrating them for our demonstration. For instance, the hardware setup comprised the setting up of the Coral with the camera, training the neural networks and running them on the hardware itself. Software components included the backend API server and the GUI. Functionalities were separately coded before integrating the API calls with the interaction of the user through the GUI.

Separating the components of the system into smaller subgroups facilitated identifying bugs in each subgroup, which were easier to solve on their own.

4.3 Codes and Standards

4.3.1 Standards

- Wireless communication between the processor, cameras, and end user devices – Use of HTTP and TCP will be critical for safely transferring data between different devices [5].
- Web applications (for potential UI) for end users – Website applications are typically designed using HTML, CSS, and JavaScript [6].
- ISO/IEC /IEEE 26351-2015 – Systems and software engineering – Content management for product lifecycle, user and service management documentation - 26351-2015 is an international standard which describes the requirements of any content or data used within a product's software, life cycle, service management system documentation, etc. It specifies the practices regarding content creation, publication and archiving, which will be useful for managing the various publications of the project [7].
- ISO/IEC/IEEE 24748-2-2018 Systems and software engineering — Life cycle management — Part 2: Guidelines for the application of ISO/ IEC/IEEE 15288 (System life cycle processes) The 24748-

2-2018 standard discusses the processes needed for using a system-based approach to manage projects. It also highlights the purpose and benefits of applying system-based engineering to solve problems. The approaches described in this standard can be implemented in the group's final project to develop a more holistic solution for end users [8].

4.3.2 Codes

1. Video Recording Laws – Ga. Code 16-11-62(2) - According to Ga. Code 16-11-62(2), it is a crime to implement hidden cameras, or any similar device, to observe someone in a private setting [9]. Therefore, the camera should be as clear a view as possible, and users (both individuals and those who might install the system) should agree to terms to use the iValet service.

5. Project Demonstration

To qualitatively describe the demonstration and validation of our project specifications, our project is broken into the Hardware System, Detection Algorithms, Navigation Algorithms, and UI interface. Those functional modules were tested with our database and some following cases.

5.1 Hardware System

To check our coral dev board and camera, there are some demo and sample models on its website. Testing with those trained models, we can supervise the output from the board with its self-contained stream server.

To embed our trained model and following codes into it, we need to set up all those packages and check the version of each of them. For instance, proper version control can eliminate bugs and deprecated functions when running the code with libraries such as OpenCV and TensorFlow. The right SQL drivers were also needed on the Coral to establish a connection to the SQL database. Then, we tested to see if our code ran on the Coral successfully. Initially, our SQL database was on MSSQL with its own proprietary drivers,

which did not support the Mendel Linux operating system used by the Coral board. Since the Coral was not able to connect to a MSSQL database, we decided to migrate our database to PostgreSQL instead.

With the camera and Coral board mounted on the tripod, adjustments to the camera angle were made to ensure that the angle was consistent with the previous one used to define bounding boxes for each spot by checking the video feed from the streamer. Our segmentation model can only work with the same camera angle.

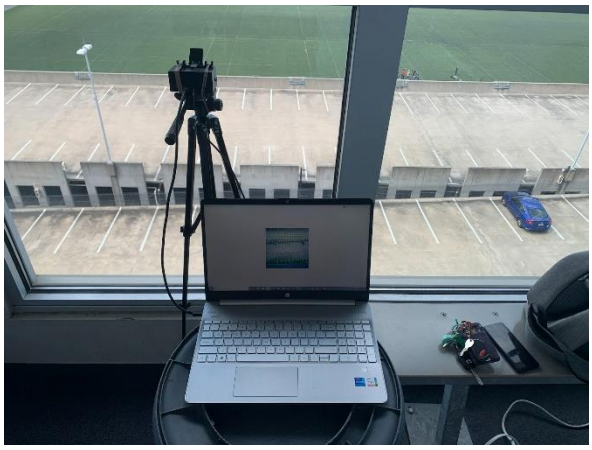


Figure 14. Camera setup

5.2 Detection System

First, we built a Mask R-CNN based model for detecting parking spaces dynamically. However, when we tested this model with the parking lots of the CRC, it did not work well. The reason might be insufficient training dataset and that our camera has a different camera angle than that used to capture the training images.

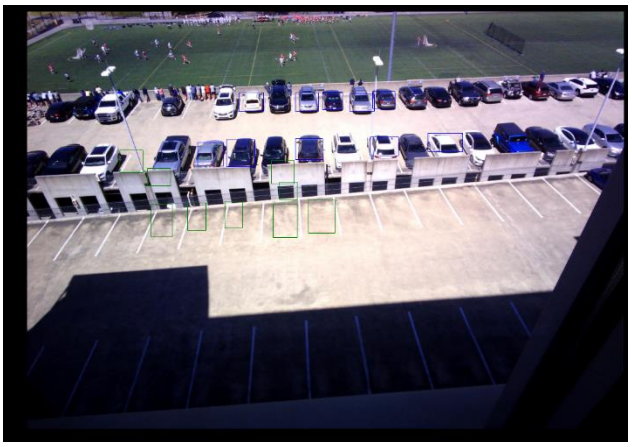


Figure 15. Result of Mask-RCNN detection

Then, we built another script to pre-define the image from the parking lot of the CRC. Once we tested this with the same photo, it successfully segmented them into 66 segmented images.

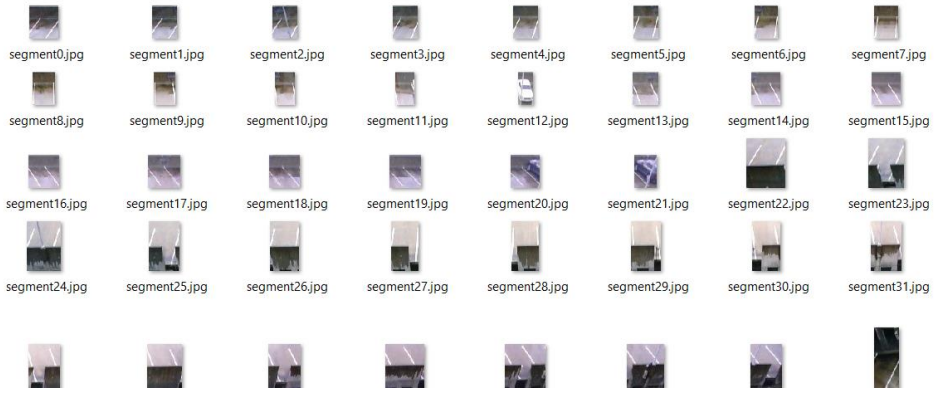


Figure 16. Cropped images of predefined bounding boxes

The segmented images were then used to test our classifier. All of them worked well.

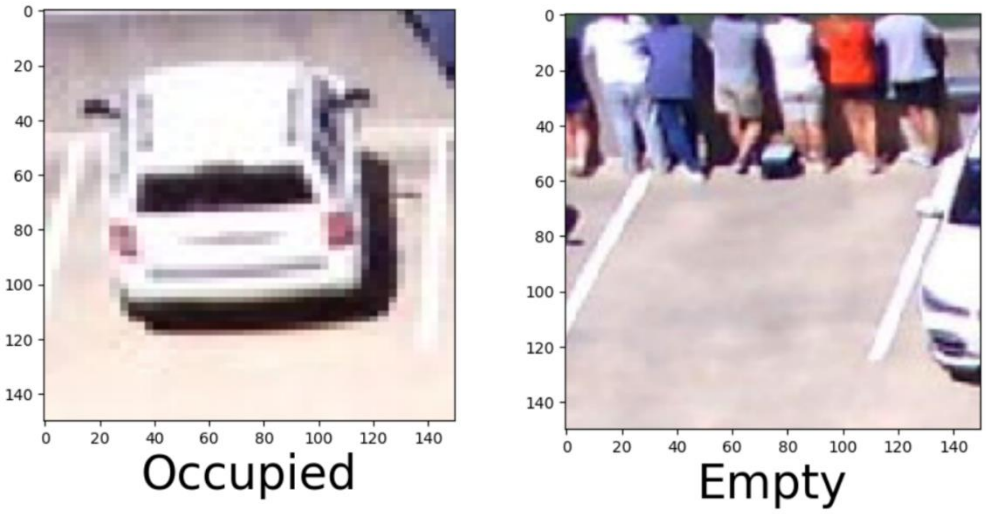


Figure 17. Result of classifier on selected images

Finally, we created a script to test the overall detection system with our hardware system in the CRC parking lot. We showed all those segmented parking slots with their corresponding bounding boxes, and set the empty slots to be green, and the occupied slots to be blue.

Also, we tested the connection between our model and the SQL database, which successfully updated every 30 seconds.

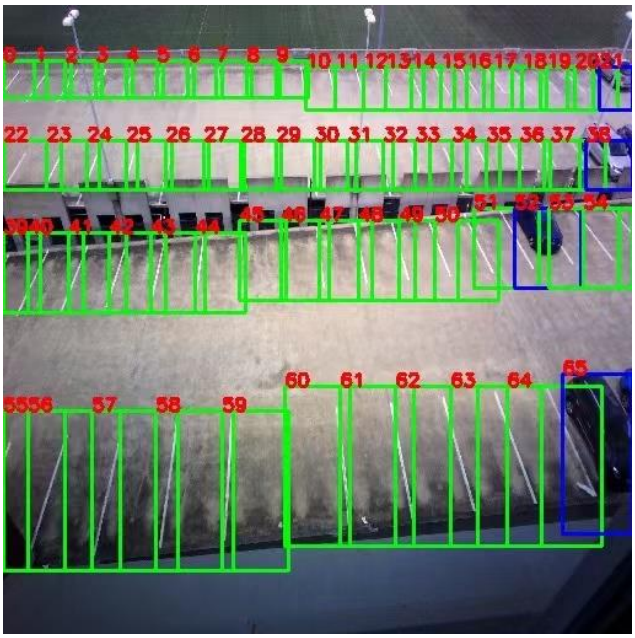


Figure 18. Empty lot detection with blue as occupied and green as vacant

After several rounds of testing, we found that our detection system works with 100% accuracy if all the parking spots are clearly visible. If there are some shadow areas, it might have some errors. The overall accuracy is around 95%, which is higher than our requirement.

5.3. Navigation Algorithm

The A* algorithm was used to select the shortest path from the entrance to each individual parking lot.

To test this algorithm, we first wrote code to rectify for our photos to simulate the top-down view from Google Maps. Then, the A* algorithm identified the desired path to the certain spot with a red line.

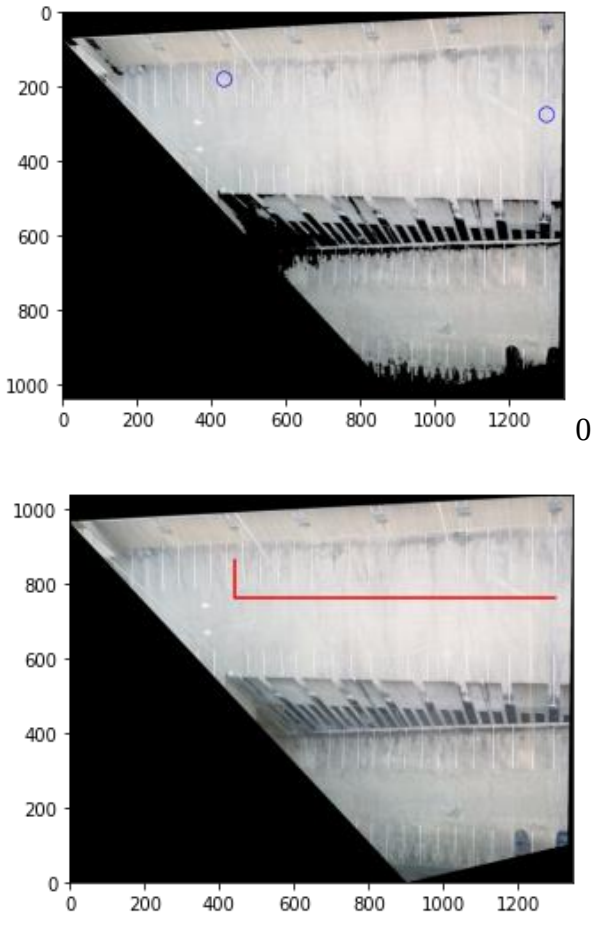


Figure 19. A* Pathfinding algorithm with starting and ending points in blue circle and path in red

Finally, we used the pathfinding algorithm to sort the "Lot IDs" in the database in order of closest unoccupied to farthest unoccupied, then closest occupied to farthest occupied. From here the web application can query the closest parking spot from the top of the list to display the map shown in **Figure 20** (also **Figure 9**).

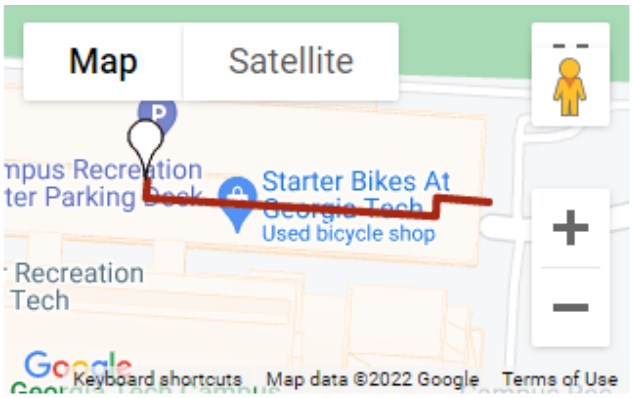


Figure 20. Path shown on Google Maps (to park)

5.4. User Interface

Initial tests of the user interface began with designing the pages of our web application with React on the a localhost (and later pushed to Netlify to host the site), starting with the landing page. As more pages were added, we ensured that navigation between the pages were smooth.

After populating the SQL database with the relevant parking lot information, a backend API was written to integrate data exchange between the user and the database through API calls. This allowed for retrieval of the nearest parking lot, nearest handicap parking lot and updating occupancy status once users confirm that they have parked in the assigned parking spot. These API calls were individually tested prior to integration with the user interface to facilitate troubleshooting.

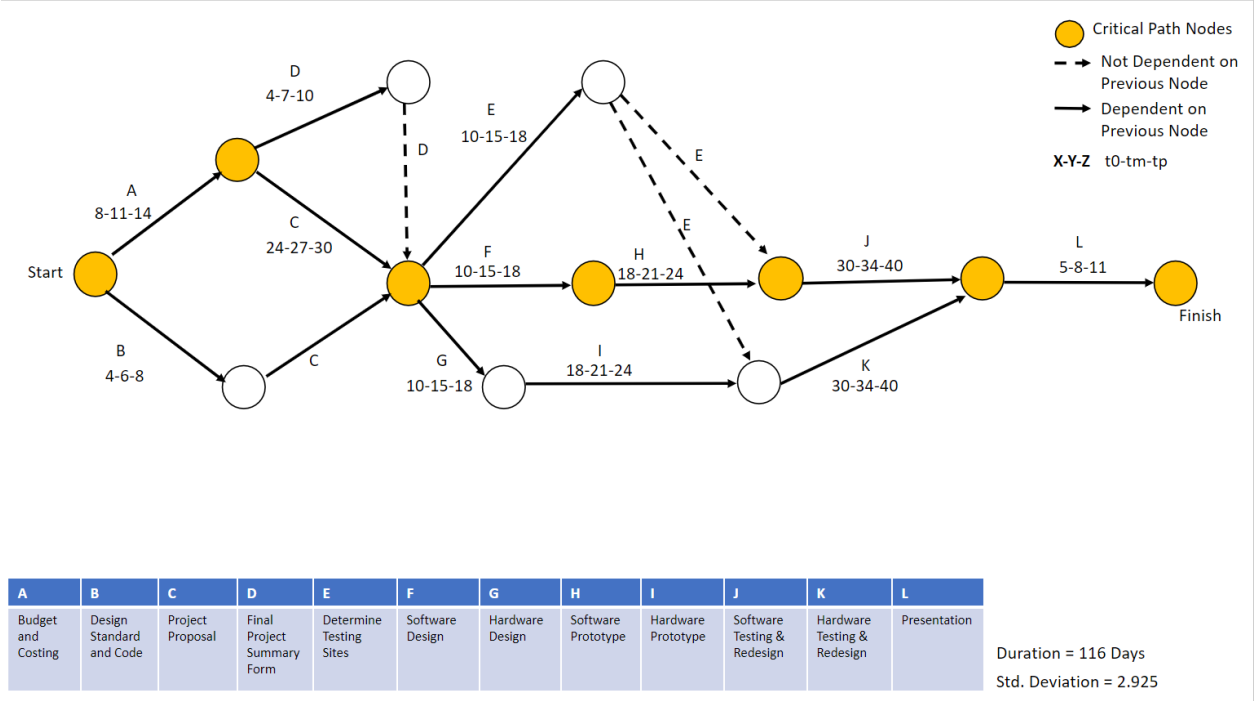
5.5. Overall Demonstration

The following demo (also linked in Appendix A) summarizes the experience on the UI and the execution of the classification and segmentation algorithms.

https://drive.google.com/file/d/1vpijONvjzTPr2089D6ewCtY7oModstjp/view?usp=sharing

6. Schedule, Tasks, and Milestones:

The following PERT, **Figure 21**, depicts the team's original task, estimate duration, and longest duration from before design began.



A	B	C	D	E	F	G	H	I	J	K	L
Budget and Costing	Design Standard and Code	Project Proposal	Final Project Summary Form	Determine Testing Sites	Software Design	Hardware Design	Software Prototype	Hardware Prototype	Software Testing & Redesign	Hardware Testing & Redesign	Presentation

Duration = 116 Days
Std. Deviation = 2.925

Figure 21. Original PERT chart

The first GANTT table, **Figure 22**, was the team's projected schedule before the start of the second semester.

Semester 2 (Subject to Change)	01/10/22	04/29/22	80d			Not Started	
Determine Testing Site(s)	01/10/22	01/28/22	15d		0%	Not Started	Wei Xiong Toh
Determine Criteria	01/10/22	01/14/22	5d		0%	Not Started	Yunchu Feng
Shortlist Potential Carparks	01/17/22	01/21/22	5d	40	0%	Not Started	Faiza Yousuf
Rank Potential Carparks	01/24/22	01/28/22	5d	41	0%	Not Started	Kelin Yu
System Design	01/10/22	01/28/22	15d		0%	Not Started	All
Hardware	***01/10/22***	***01/28/22***	***15d***		***0%***	***Not Started***	***Kelin Yu***
Camera/Drone Setup	01/10/22	01/20/22	9d		0%	Not Started	Kelin Yu
Camera/Drone Calibration	01/21/22	01/28/22	6d	45	0%	Not Started	Yunchu Feng
Software	***01/10/22***	***01/28/22***	***15d***		***0%***	***Not Started***	***Wei Xiong Toh***
Dataset Refining	01/10/22	01/14/22	5d		0%	Not Started	Faiza Yousuf
Image Processing Algorithm	01/14/22	01/20/22	5d		0%	Not Started	Wei Xiong Toh
UI/Mobile App	01/20/22	01/28/22	7d		0%	Not Started	Faiza Yousuf
System Prototype	01/31/22	02/28/22	21d	43	0%	Not Started	All
Hardware	***01/31/22***	***02/28/22***	***21d***		***0%***	***Not Started***	***Yunchu Feng***
Camera/Drone Setup	01/31/22	02/15/22	12d		0%	Not Started	Kelin Yu
Camera/Drone Calibration	02/16/22	02/28/22	9d	53	0%	Not Started	Yunchu Feng
Software	***01/31/22***	***02/28/22***	***21d***		***0%***	***Not Started***	***Faiza Yousuf***
Dataset Refining	01/31/22	02/04/22	5d		0%	Not Started	Faiza Yousuf
UI/Mobile App	01/31/22	02/28/22	21d		0%	Not Started	Faiza Yousuf
Image Processing Algorithm	02/07/22	02/28/22	16d	56	0%	Not Started	Wei Xiong Toh
System Test/Redesign	03/01/22	04/15/22	34d	51	0%	Not Started	All
Hardware	***03/01/22***	***04/15/22***	***34d***		***0%***	***Not Started***	***Kelin Yu***
Camera/Drone Setup	03/01/22	04/15/22	34d		0%	Not Started	Kelin Yu
Camera/Drone Calibration	03/01/22	04/15/22	34d		0%	Not Started	Yunchu Feng
Camera/Drone Test	03/01/22	04/15/22	34d		0%	Not Started	Faiza Yousuf
Software	***03/01/22***	***04/15/22***	***34d***		***0%***	***Not Started***	***Wei Xiong Toh***
Image Processing Algorithm	03/01/22	04/15/22	34d		0%	Not Started	Wei Xiong Toh
Dataset Refining	03/01/22	04/15/22	34d		0%	Not Started	Faiza Yousuf
UI/Mobile App	03/01/22	04/15/22	34d		0%	Not Started	Faiza Yousuf
Presentation	***04/15/22***	***04/26/22***	***8d***		***0%***	***Not Started***	***Faiza Yousuf***
Website	04/15/22	04/26/22	8d		0%	Not Started	Faiza Yousuf
Powerpoint	04/15/22	04/26/22	8d		0%	Not Started	Yunchu Feng
Video Demo	04/15/22	04/26/22	8d		0%	Not Started	Kelin Yu
Design Notebook	01/10/22	04/29/22	80d		0%	Not Started	All

Figure 22. Original Second Semester GANTT

The GANTT chart, **Figure 23,** organizes the team's tasks according to the subsystem, the task leader, and the time it was completed.

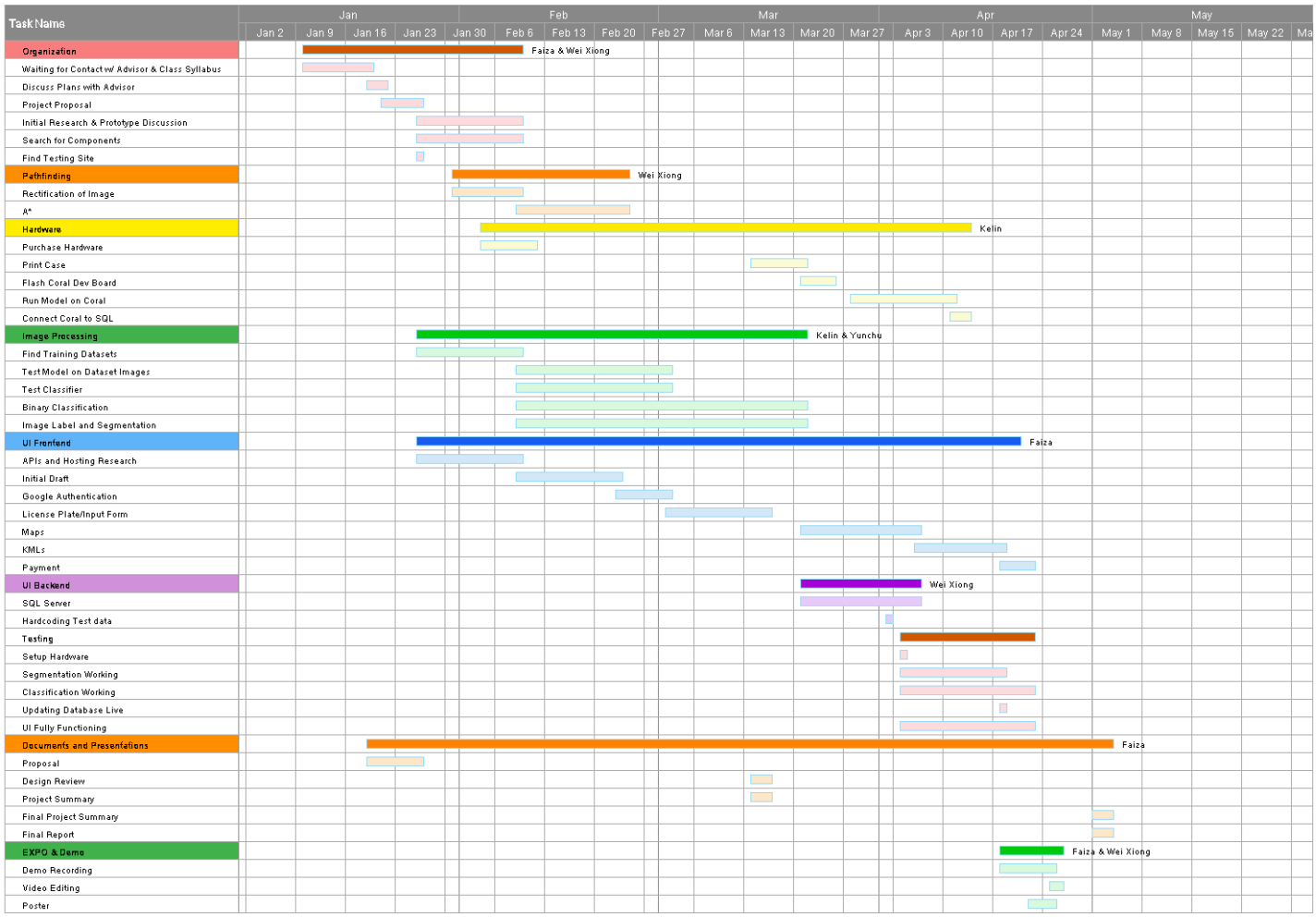


Figure 23. Final Gantt Chart (see in Appendix A)

7. Marketing and Cost Analysis

7.1 Marketing Analysis

The target market consists of venue owners with garages and parking lots in busy areas wanting to enhance the parking experience of their customers. This parking guidance system is not a new concept, with more than a few companies just in the Metro Atlanta area. Hub Parking, for example, will install hardware in every single parking lot and intersection to guide the driver. With sensors and LCD screens everywhere in the garage, the setup will cost around $10,000, and 20,000 for larger parking lots [10]. With iValet, only a single camera was used, which will save the hypothetical cost around $1000 - $3000. The camera cost can

be scaled according to how many are used in a given parking lot. An example of the current iValet budget is listed in Table 4.

Table 4. Components Cost for Prototype

Current Parts List with Cost and Citations			
Part	Part Type	Cost	Citation
Google Coral Dev Board	Dev Board	$132.99	[3]
Google Coral Camera	Camera	$21.99	[4]
SD Card	Memory/Storage	$6.19	NA
Tripod	Mount for Board and Camera	$24.99	NA
Printed Coral Case	Case for Board and Camera	NA – Printed at HIVE Makerspace	NA
Webhosting on Netlify	UI	$11.99/month (only had personalized domain for month of April)	[11]
Total Cost		$198.15	

7.2 Cost Analysis (Budget)

Four engineers completed the current iteration and development of iValet. The total labor hours for the iValet and Labor cost are calculated in Table 5. Data for hourly payment is the average entry cost found online [12]. The table has been separated into the meeting, data, model training, software, and hardware portion with the total cost coming to around $8,565 for the labor cost.

Table 5. Estimated Hours per Teammate

estimates are from https://www.salary.com/						
Yunchu:	Meetings	Data	Model Training	Testing	Path Finding	Total Hour
Hour:	15	6	30	10	2	63
Salary/Hour:	$30	$35	$45	$30	$35	
Total Cost:	$450	$210	$1350	$300	$70	$2380
Faiza	Meetings	Data	UI	Testing	Path Finding	
Hour:	15	3	30	10	2	60
Salary/Hour:	$30	$35	$32	$30	$35	
Total Cost:	$450	$105	$960	$300	$70	$1885
Wei Xiong	Meetings	Data	Model Training	Testing	Path Finding	
Hour:	15	30	2	10	15	72
Salary/Hour:	$30	$35	$45	$30	$35	
Total Cost:	$450	$1050	$90	$300	$525	$2415
Colin	Meetings	Data	Hardware	Testing	Path Finding	
Hour:	15	1	30	10	4	60
Salary/Hour:	$30	$35	$32	$30	$35	
Total Cost:	$450	$35	$960	$300	$140	$1,885
Complete Total						$8,565

The statistics over five years of production and service of iValet are given in with the following assumptions. First, with each year the business will expand, meaning the number of sales will increase. For example, the first increase will be five more sales in the third year then ten more in the fourth. Revenue will also be increased from $11,724 to $16,414 accordingly. Second, after each year, it is assumed iValet will run into certain design issues and maintenance issues with each redesign, costing around $15,000 due to engineering and part usage. Third, all parts are kept at a stable current market price with no volatility. These sales numbers are gathered online from some similar businesses like parking Guidance System, LLC, and Hub Parking. Twenty units in the first year is below average for most auto-guidance system services. Also, most guidance systems services are subscription-based, meaning their customers will remain high each year.

Table 6. Recurring Costs

	Number/ Cost /Salary per yr.	Year 1		Year 2		Year 3		Year 4		Year 5
Sales Volume (units)	20		20		25		35		45	
Unit Price	$468.99		$468 .99		$468 .99		$46 8 .99		$46 8 .99	
Sale Revenue		$9,379 .80		$9,37 9 .80		$11,724 .75		$16,414 .65		$21,1 04 .55
Non-re Cost	$20	$400	$20	$400	$20	$500	$20	$700	$20	$900
1. Research and Dev (based on industry estimate)										
Redesign		$15,00 0		$15,0 00		$15,000		$15,000		$0

Engr Change Order		$10,000		$10,000		$10,000		$10,000		$0
2.Production										
Total Parts	$186.16	$3723.20		$3723.20		$4654		$6515.60		$8377.20
Google Coral Dev Board	$132.99	$2,659.80		$2,659.80		$3,324.75		$4654.65		$5984.55
Google Coral Camera	$21.99	$438.80		$438.80		$549.75		$769.65		$989.55
SD Card	$6.19	$123.80		$123.80		$154.75		$216.65		$278.55
Tripod	$24.99	$499.80		$499.80		$624.75		$874.65		$1124.55

Profits are shown in Table 7. Profits are calculated based on the number of sales each year, minus the cost of that year. The cost consists of marketing, packaging, support and maintenance, and distribution. Even though the total profit is very similar, we are increasing the sales each year. This is because more parts and labor are included. iValet is looking to break even in 5 years, and if subscription-based maintenance is deployed, the profit will be even higher.

Table 7. Estimated profits

	Number /Cost/Salary per yr.	Year 1	Year 2	Year 3	Year 4	Year 5
3. Packaging	$10,000	$200,000	$200,000	$250,000	$350,000	$450,000
4. Marketing	$15,000	$300,000	$300,000	$375,000	$525,000	$675,000
5. Sales	$20,000	$400,000	$400,000	$500,000	$700,000	$900,000
6. Distribution	$20,000	$400,000	$400,000	$500,000	700,000	$900,000
7. Support	$15,000	$300,000	$300,000	$375,000	$525,000	$675,000
Total Cost/Year		$43,760	$43,760	$48,450	$57,829	$42,209
Overhead Total	150	$65,639	$65,639	$2,465,639	$86,744	$63,314
Adjusted Cost		$109,399	$109,399	$2,514,089	$144,573	$105,523
Cost/Unit		$468.99	$468.99	$468.99	$468.99	$468.99
Total Profit/Yr.		$21,880	$21,880	$2,417,190	$28,915	$21,105

8. Conclusion & Current Status

What was Delivered:

The system is able to determine the occupied status of sixty-six parking spots at the CRC with 95% to 100% accuracy, as well as provide static maps to user to park in each spot, relocate their parked car at this spot, then exit the parking lot from their lot.

Potential Improvements:

- The current design needs pre-defined images of parking spaces, so it cannot be used in unknown areas immediately.
- The team attempted to use another segment-based algorithm, Mask R-CNN, which can work dynamically, in distinct locations, but it performed poorly. It can be retrained with a larger dataset to get better performance.
- Classifier performance varies based on lighting conditions. An improved dataset comprising images taken from the Coral camera will be useful to ensure more accuracy.
- Implement a zoning system in the SQL database and UI form to allow users to select zones they prefer to park in large venues (e.g., proximity to specific seats in a large stadium)
- Integrate images from multiple cameras for a larger field of view as the current system only includes 66 parking spaces
- The attempted geolocation method on the UI to help users dynamically move on the navigation screen (as opposed to a static path) can be unreliable depending on the browser, an alternative method to improve the navigation must be researched

9. Leadership Roles

The leadership roles are listed as follows in Table 8.

Table 8. Roles of team members

Role	Description	Team Member
Design Lead	In charge of keeping the team to the design schedule and tracking what deliverables were met.	Wei Xiong Toh
Software Lead	Leads and directs tasks related to the computer vision, pathfinding, and user interface.	Wei Xiong Toh
Hardware Lead	Leads tasks related to setting up camera and processor in testing environments.	Kelin Yu
Testing Lead	In charge of delegating and tracking all testing processes.	Yunchu Feng & Wei Xiong Toh
Documentation	Keeps track of all documentation including notes, deliverables, etc.	Yunchu Feng
Webmaster	Will head design and content of the project website.	Faiza Yousuf
Expo Coordinator	Ensures the project is presentable at the expo (necessary PowerPoints, video footage, monitors, posters, etc.)	Faiza Yousuf & Wei Xiong Toh

10. References

[1] INRIX, “Searching for Parking Costs Americans $73 Billion a Year,” *Inrix*. https://inrix.com/press-releases/parking-pain-us/ [Accessed May 02, 2022].

[2] “Home,” *PlacePod*. https://placepod.com/ [Accessed May 02, 2022].

[3] “Dev Board,” *Coral*. https://coral.ai/products/dev-board/ [Accessed May 02, 2022].

[4] “Camera,” *Coral*. https://coral.ai/products/camera/ [Accessed May 02, 2022].

[5] Learning Center. 2021. *What is OSI Model | 7 Layers Explained | Imperva*. [online] Available at: <https://www.imperva.com/learn/application-security/osi-model/> [Accessed 22 November 2021].

[6] Hack Reactor, “What is JavaScript used for?,” Hack Reactor, 26-Aug-2021. [Online]. Available: https://www.hackreactor.com/blog/what-is-javascript-used-for. [Accessed: 22-Nov-2021].

[7] "ISO/IEC/IEEE International Standard for Systems and software engineering -- Content management for product life-cycle, user, and service management documentation," in ISO/IEC/IEEE 26531:2015 (E) , vol., no., pp.1-60, 15 May 2015, doi: 10.1109/IEEESTD.2015.7106441.

[8] “ISO/IEC/IEEE Draft International Standard - Systems and Software Engineering-- Life Cycle Management-- Part 2: Guidelines for the Application of ISO/IEC/IEEE 15288 (System Life Cycle Processes). IEEE, 2018, pp. 1–69.”

[9] Chanco Schiffer Law, LLC. 2021. Georgia’s Video Recording Laws. [online] Available at: [Accessed 22 October 2021].

[10] Kenneth and P. Articles, “Parking guidance systems,” *PARKING GUIDANCE SYSTEMS, LLC.*, 02-Mar-2020. [Online]. Available: https://parkingguidancesystems.com/. [Accessed: 07-Oct-2021].

[11] “Plans and Pricing,” *Netlify*. https://www.netlify.com/pricing/ (accessed May 02, 2022).

[12] Salary.com, “Unlock the power of pay,” Salary.com, 01-Apr-2021. [Online]. Available: https://www.salary.com/. [Accessed: 22-Nov-2021].

Appendix A

Project Website

https://eceseniordesign2022spring.ece.gatech.edu/sd22p37/

Project Demo Video

https://drive.google.com/file/d/1vpijONvjzTPr2089D6ewCtY7oModstjp/view?usp=sharing

Final Presentation

https://docs.google.com/presentation/d/1iCwHxiddyIrrpUa8-g1fNoHMb90EUD5qlaEiYZfCxuM/edit?usp=sharing

EXPO Poster

https://docs.google.com/presentation/d/16W4DV4rCXY2BUOCvoMsAo1zAl-zIN6Gf/edit?usp=sharing&ouid=104356297121719412631&rtpof=true&sd=true

Design Review Presentation

https://docs.google.com/presentation/d/1qZ9jTlgd1fxjsgbVpeiUAxHusKAFGBr54G2pZNemn98/edit?usp=sharing

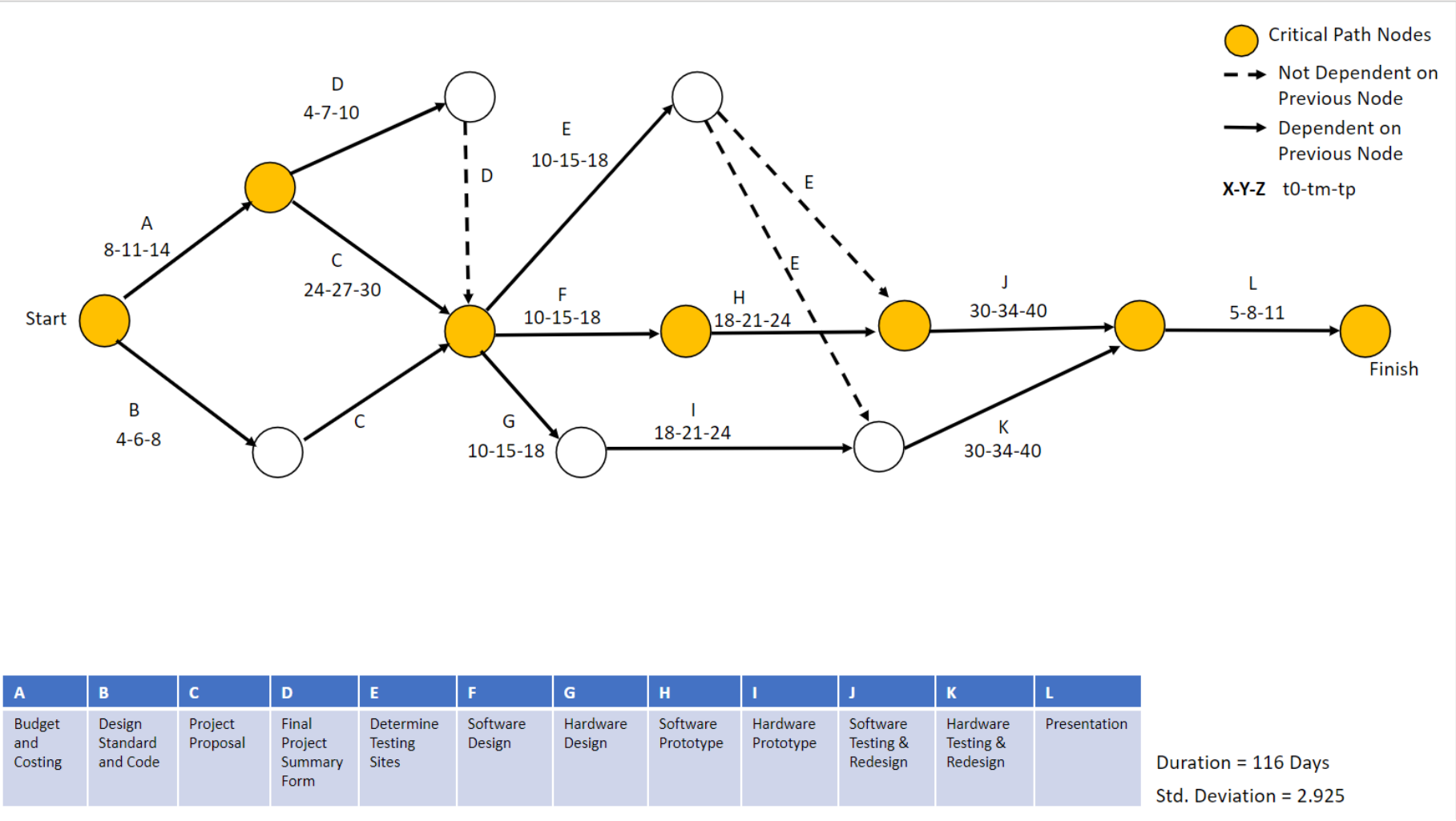
Proposal Presentation

https://docs.google.com/presentation/d/1g54UxryRkfnSOuQ-JeoBnaMlSywAq_rm5Q9VdJ5JtDI/edit?usp=sharing

QFD

		Engineering Requirements							Benchmarks	
		Max height of cameras	Accuracy of detection	Ideal algorithm run time	Maximum area of vision (size of parking lot	Camera Frame Rate	Camera Resolution	Cost	Daylight	Night/Artificial Light
General Requirement	Parking lot is visible	X			X	X	X		X	X
	System Maintains power			X	X					
	User Interface is coherent		X	X						
	Maintains communication with app			X	X					
	Minimal power consumption (cameras)			X	X	X	X			
	Neatly installed	X			X			X		
	Can be replacated to reasonable variety of lots	X	X	X	X	X	X		X	X
	Units	meters	%	seconds	m^2	Frames/sec	Megapixels	Dollars		
		15	90	300	10000 (estimate)	24	2	700		
		Engineering Targets								

		Engineering Requirements							Benchmarks	
		Max height of cameras	Accuracy of detection	Ideal algorithm run time	Maximum area of vision (size of parking lot)	Camera Frame Rate	Camera Resolution	Cost	Daylight	Night/Artificial
General Requirements	Parking lot is visible	X			X	X	X		X	X
	Maintains power			X	X					
	User Interface is coherent		X	X						
	Maintains communication with app			X	X					
	Minimal power consumption (cameras)			X	X	X	X			
	Neatly installed	X			X			X		
	Can be replacated to reasonable variety of lots	X	X	X	X	X	X		X	X
	Units	meters	%	seconds	m^2	Frames/sec	Megapixels	Dollars		
		15	90	300	10000 (estimate)	24	2	700		
		Engineering Targets								

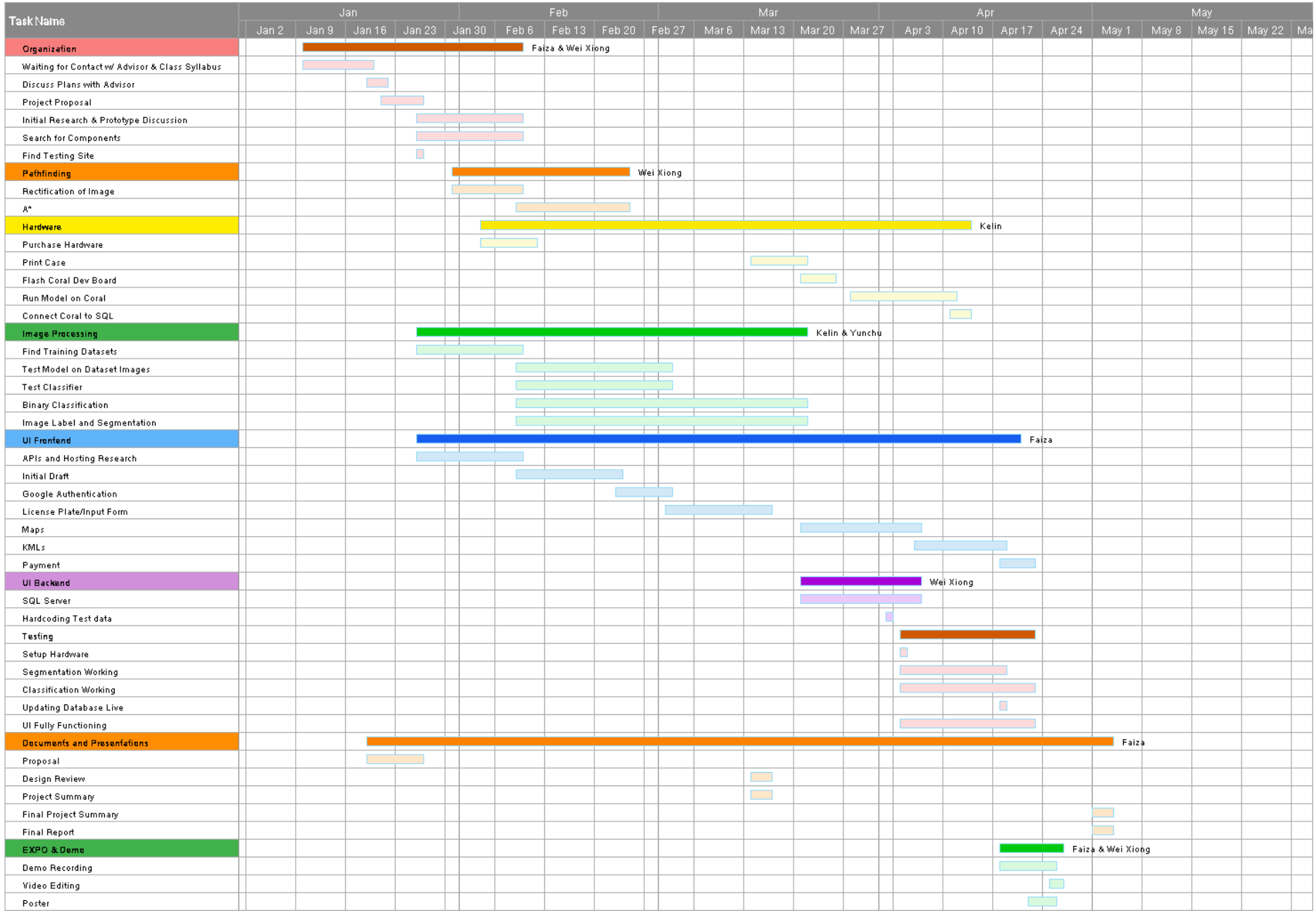


A	B	C	D	E	F	G	H	I	J	K	L
Budget and Costing	Design Standard and Code	Project Proposal	Final Project Summary Form	Determine Testing Sites	Software Design	Hardware Design	Software Prototype	Hardware Prototype	Software Testing & Redesign	Hardware Testing & Redesign	Presentation

Duration = 116 Days

Std. Deviation = 2.925

GANTT


Task Name
Jan
Feb
Mar
Apr
May
Jan 2
Jan 9
Jan 16
Jan 23
Jan 30
Feb 6
Feb 13
Feb 20
Feb 27
Mar 6
Mar 13
Mar 20
Mar 27
Apr 3
Apr 10
Apr 17
Apr 24
May 1
May 8
May 15
May 22
1 Organization
2 Waiting for Contact w/ Advisor & Class Syllabus
3 Discuss Plans with Advisor
4 Project Proposal
5 Initial Research & Prototype Discussion
6 Search for Components
7 Find Testing Site
8 Pathfinding
9 Rectification of Image
10 A*
11 Hardware
12 Purchase Hardware
13 Print Case
14 Flash Coral Dev Board
15 Run Model on Coral
16 Connect Coral to SQL
17 Image Processing
18 Find Training Datasets
19 Test Model on Dataset Images
20 Test Classifier
21 Binary Classification
22 Image Label and Segmentation
23 UI Frontend
24 APIs and Hosting Research
25 Initial Draft
26 Google Authentication
27 License Plate/Input Form
28 Maps
29 KMLs
30 Payment
31 UI Backend
32 SQL Server
33 Hardcoding Test data
34 Testing
35 Setup Hardware
36 Segmentation Working
37 Classification Working
38 Updating Database Live
39 UI Fully Functioning
40 Documents and Presentations
41 Proposal
42 Design Review
43 Project Summary
44 Final Project Summary
45 Final Report
46 EXPO & Demo
47 Demo Recording
48 Video Editing
49 Poster
Faiza & Wei Xiong
Wei Xiong
Kelin
Kelin & Yunchu
Faiza
Wei Xiong
Faiza
Faiza & Wei Xiong