



iValet Final Presentation

Faiza Yousuf, Wei Xiong Toh, Kelin
Yu, Yunchu Feng



Introduction

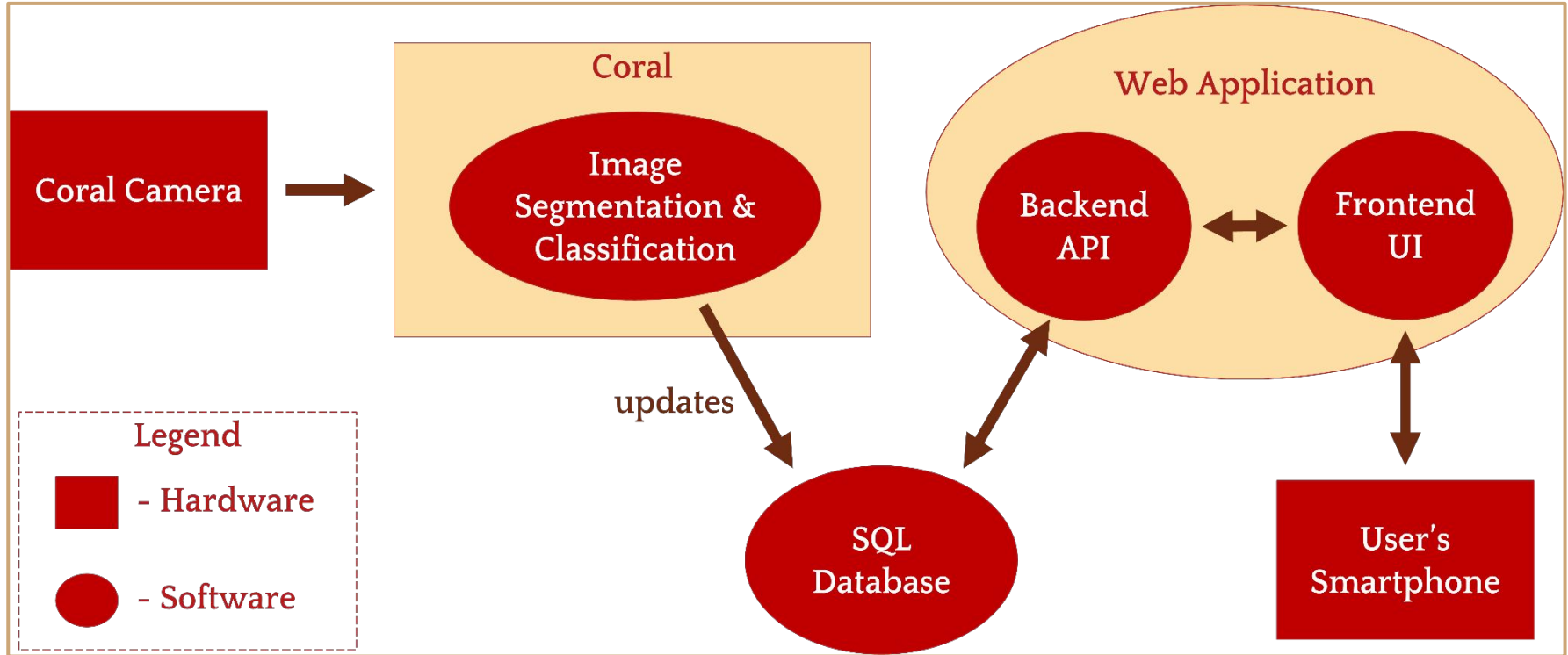
Drivers spend **17h per year** on average searching for parking. The estimated cost of the wasted time, fuel and emissions produced by these drivers amount to **\$345 a year**.

iValet aims to alleviate this problem by directing drivers to the nearest empty parking spot once they enter the parking lot.

Cost

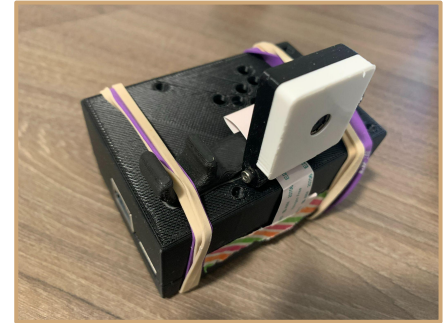
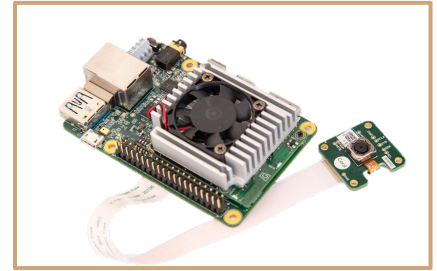
Item	Cost
Google Coral	\$132.99
Coral Camera	\$21.99
SD Card	\$6.19
Tripod	\$24.99
Printed Coral Case	NA
Total	\$186.16

System Diagram



Hardware

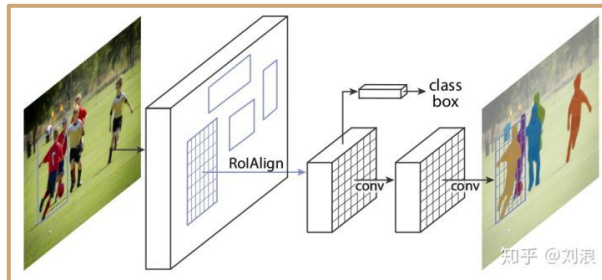
- Google Coral
 - Runs Mendel Linux (Debian derivative)
 - Supports TensorFlow Lite
 - Mainly compatible with Python and C++
- Coral camera
 - 87.6° field of view
 - 2582 x 1933 active array size
 - 50/60 Hz lumination
- Printed coral case
 - 3-D printed box and frame
 - Stabilizing the camera within a 90° angle



First Approach - Segmentation

Mask R-CNN:

Framework for object instance segmentation.



```
def load_dataset(self, dataset_dir, is_train = True):
    self.add_class("parking", 1, "ParkingOccupied")
    self.add_class("parking", 2, "ParkingEmpty")
    if is_train:
        img_dir = os.path.join(dataset_dir, 'train/images')
        labels_dir = os.path.join(dataset_dir, 'train/labels')
    elif not is_train:
        img_dir = os.path.join(dataset_dir, 'test/images')
        labels_dir = os.path.join(dataset_dir, 'test/labels')
    for filename in os.listdir(img_dir):
        image_id = filename[:-4]
        img_path = os.path.join(img_dir, filename)
        label_path = labels_dir + '/' + image_id + '.xml'
        self.add_image('parking', image_id = image_id, path=img_path, annotation=label_path)
```

Loading dataset

```
def draw_image_with_boxes(filename, boxes_list, class_list):
    data = plt.imread(filename)
    plt.figure(figsize=(20,14))
    plt.imshow(data)
    ax = plt.gca()
    for box, cls in zip(boxes_list, class_list):
        y1,x1,y2,x2 = box
        width, height = x2 - x1, y2 - y1
        if cls==1:
            rect = patches.Rectangle((x1,y1), width, height, fill=False, color='blue')
        elif cls==2:
            rect = patches.Rectangle((x1, y1), width, height, fill=False, color = 'green')
        ax.add_patch(rect)
    plt.show
```

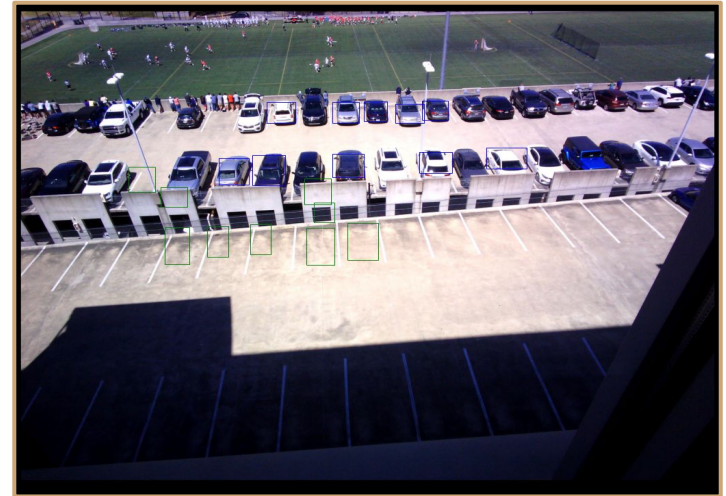
Building bounding box

First Approach - Segmentation

Performance of CRC is bad, so we use another approach for it.



Static Image test

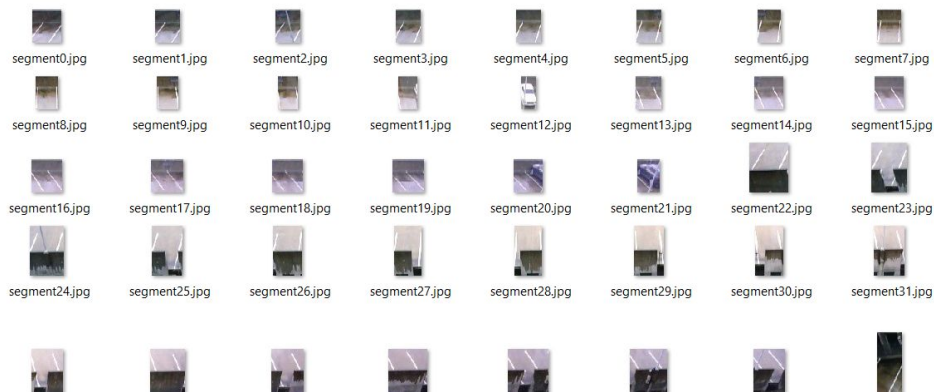


CRC test

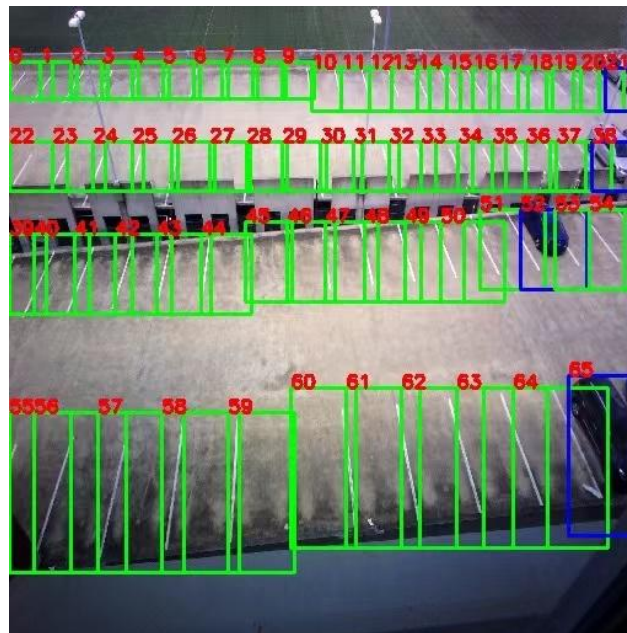
Segmentation

Pre-defined image:

Restoring vertice of each slots in an array.



Segmented parking slots



Results after classifying

Segmentation

```
a=[]
a.append((45,85,0,35))
a.append((45,85,25,55))
a.append((45,85,50,80))
a.append((45,85,75,105))
a.append((45,75,100,130))
a.append((45,85,125,155))
a.append((45,85,150,178))
a.append((45,85,173,202))
a.append((45,85,198,225))
a.append((45,85,222,248))
a.append((45,85,246,270))
a.append((45,85,270,293))
a.append((45,85,293,311))
a.append((45,85,311,342))
a.append((45,85,332,367))
a.append((45,85,356,391))
a.append((45,85,377,414))
a.append((45,85,398,437))
a.append((45,85,422,459))
a.append((45,85,442,480))
a.append((45,85,465,500))
a.append((45,85,485,512))
```

```
print("starting loop\n")
cap = cv2.VideoCapture(0)
while(True):
    print("looping\n")
    ret, frame = cap.read()
    resize = cv2.resize(frame, (512,512), interpolation=cv2.INTER_NEAREST)
    rect = cv2.resize(frame, (512,512), interpolation=cv2.INTER_NEAREST)
    for i in range(66): ##loop each predefined parking lot
        print("segmenting " + str(i))
        x,x1,y,y1 = a[i][0],a[i][1],a[i][2],a[i][3]
        new_frame = resize[x:x1,y:y1] ##grab the new image
        new_frame = cv2.resize(new_frame, (150,150), interpolation=cv2.INTER_NEAREST)
        val = predictImage(new_frame,i)
        if val[0][0] >=1: ##Occupied
            command = '''UPDATE "parking_lot" SET "empty" = 0 WHERE "lot_id"={}.format(i)
            rect = cv2.rectangle(rect,(y,x), (y1, x1), (255,0,0), 2)
            rect = cv2.putText(rect, str(i), (y,x), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 2)
            cv2.imwrite(str(i)+'.jpg',new_frame)

        else:##free
            command = '''UPDATE "parking_lot" SET "empty" = 1 WHERE "lot_id"={}.format(i)
            rect = cv2.rectangle(rect,(y,x), (y1, x1), (0,255,0), 2)
            rect = cv2.putText(rect, str(i), (y,x), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 2)
        cur.execute(command)
        con.commit()
    cv2.imwrite('output7'+'.jpg', rect)
    print("database updated")
    time.sleep(30)
```

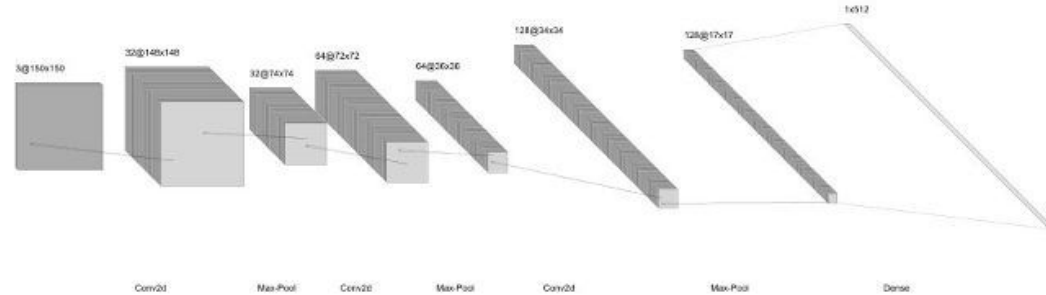
```
segmenting 0
segmenting 1
segmenting 2
segmenting 3
segmenting 4
segmenting 5
segmenting 6
segmenting 7
segmenting 8
segmenting 9
segmenting 10
segmenting 11
segmenting 12
segmenting 13
segmenting 14
segmenting 15
segmenting 16
segmenting 17
segmenting 18
segmenting 19
segmenting 20
segmenting 21
segmenting 22
segmenting 23
segmenting 24
segmenting 25
segmenting 26
segmenting 27
segmenting 28
```

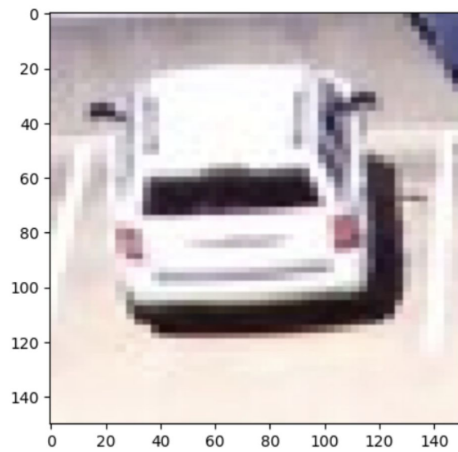
Classification

Using TensorFlow 2.0.0 CNN

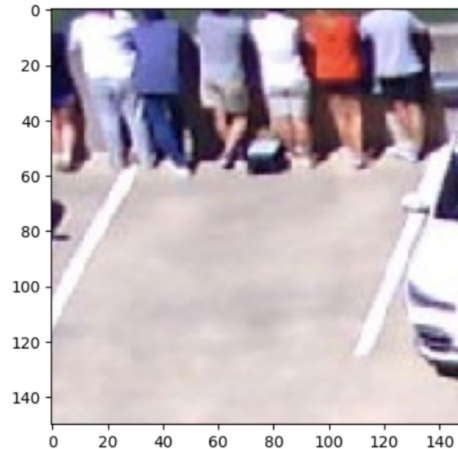
3 Convolution Layer with Max-Pool with Dense Layer

Binary output: 0-Empty, 1-Occupied.





Occupied



Empty

```
# Convolutional layer and maxpool Layer 1
model.add(keras.layers.Conv2D(32,(3,3),activation='relu',input_shape=(150,150,3)))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Dropout(0.5))

# Convolutional layer and maxpool Layer 2
model.add(keras.layers.Conv2D(64,(3,3),activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Dropout(0.5))

# Convolutional layer and maxpool Layer 3
model.add(keras.layers.Conv2D(128,(3,3),activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Dropout(0.5))

# This layer flattens the resulting image array to 1D array
model.add(keras.layers.Flatten())

# Hidden layer with 512 neurons and Rectified Linear Unit activation function
model.add(keras.layers.Dense(512,activation='relu'))
model.add(keras.layers.Dropout(0.5))

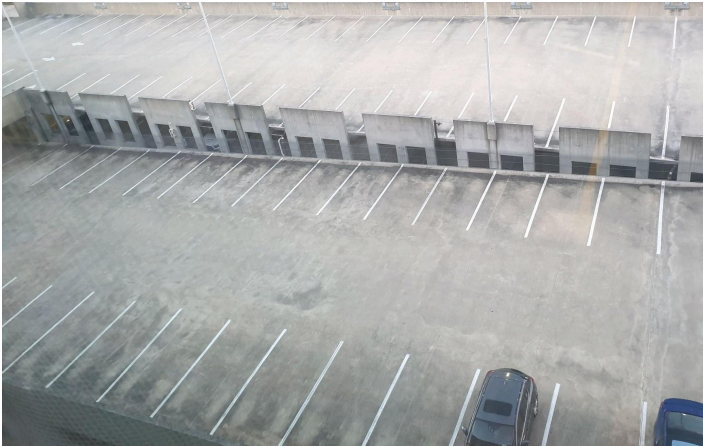
# Output layer with single neuron which gives 0 for empty or 1 for occupied
#Here we use sigmoid activation function which makes our model output to lie between 0 and 1
model.add(keras.layers.Dense(1,activation='sigmoid'))

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

Image Rectification

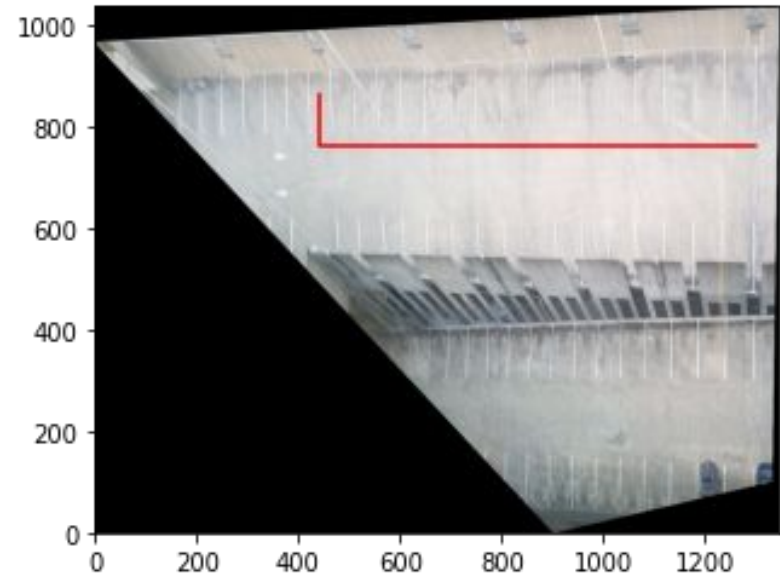
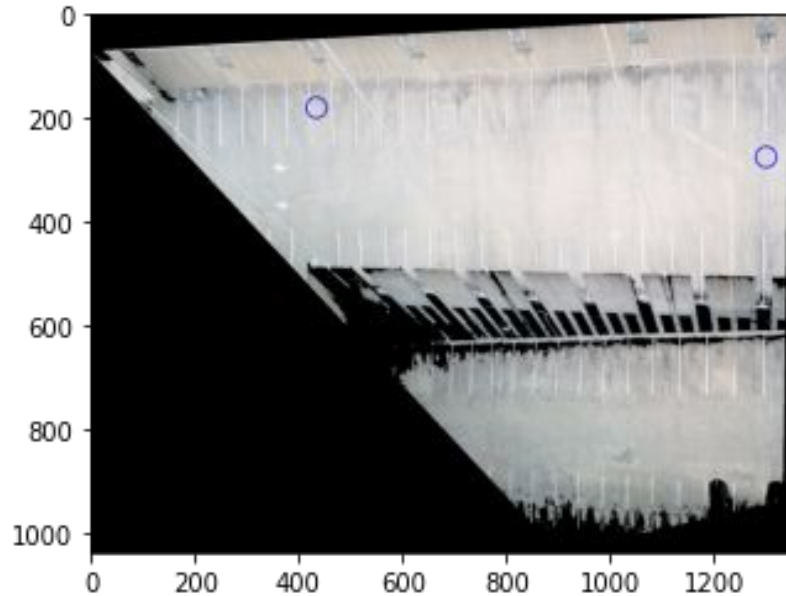
Principle of vanishing points

RANSAC algorithm to identify vanishing points



A* Pathfinding Algorithm

Image mask to identify obstacles



Distance Calculation

Search Google Maps

☰

🔍

📍

🛒 Groceries

🍴 Restaurants

🏠 Takeout

🏨 Hotels

⛽ Gas

🏪 Pharmacies

Sign in

301.16 ft

250.00 ft

200.00 ft

150.00 ft

100.00 ft

50.00 ft

P

Campus Recreation Center Parking Deck

📍

Starter Bikes At Georgia Tech Used bicycle shop

Measure distance

Click on the map to add to your path

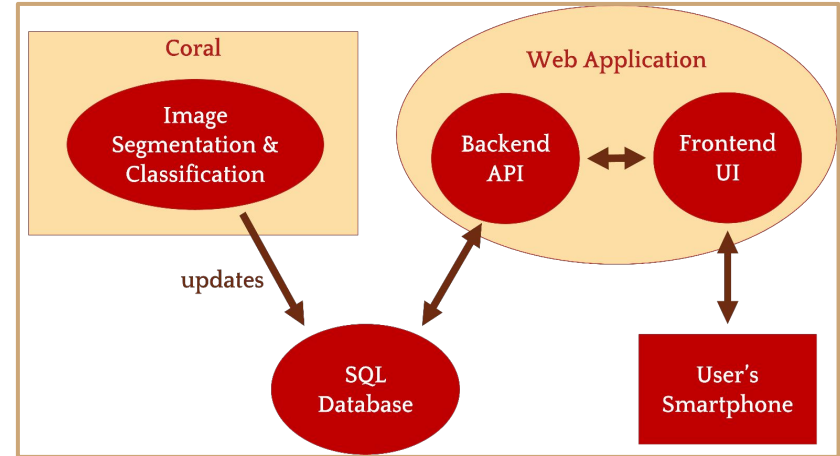
Total distance: 301.16 ft (91.79 m)

Layers

Imagery ©2022 Maxar Technologies, Sanborn, Map data ©2022 United States Terms Privacy Send feedback 20 ft

SQL Database

- Postgres Database Schema
 - Lot_id - Lot number (0 to 65 at the CRC)
 - Empty - 0 or 1 to show if this lot_id is occupied
 - Distance - distance this lot is from the entrance, used to sort database in order of closest to farthest available, then closest to farthest occupied
 - License Plate - input from the user interface, currently only used to log who parked where, at what time, and for how long
 - Handicap - 0 or 1 marks if this spot is handicap or not
 - Time Parked - datetime variable used to track the time the person started and ended parking.

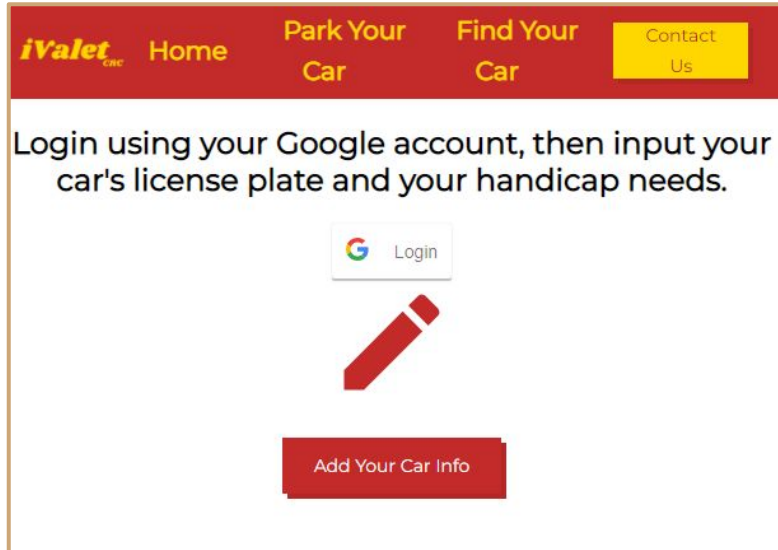


User interface - Tools/Libraries

- Frontend - React
 - Google Maps/Navigation - `@react-google-maps/api`
 - Google Authentication - `react-google-login`
 - Payment (Stripe) - `@stripe/react-stripe-js`
 - Navigation Indicator - `react-navigator-geolocation` (Unreliable in multiple browsers)
- Backend - Express & Node.js
 - SQL - PostgreSQL
 - Payment - Axios - `axios`
 - Rest API based in Angular.js, capable of intercepting and canceling requests, built-in client-side protection against cross-site request forgery

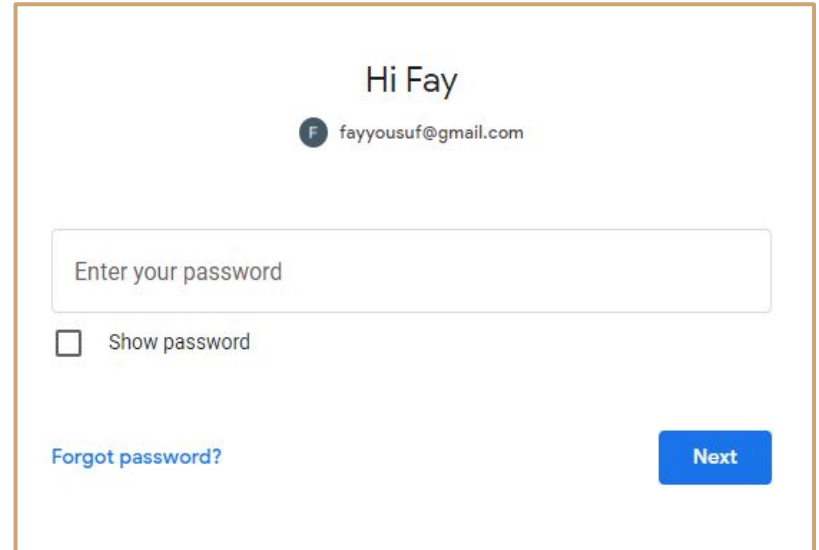
User Interface - Landing Page

- To avoid users needing to download an app, ideally a QR code at the lot entrance would lead them to this landing page



The screenshot shows the landing page for iValet. At the top, there is a dark red navigation bar with the iValet logo on the left and four menu items: 'Home', 'Park Your Car', 'Find Your Car', and 'Contact Us'. Below the navigation bar, the main content area has a white background. It starts with the text 'Login using your Google account, then input your car's license plate and your handicap needs.' Below this text is a 'Login' button with the Google logo. Underneath the button is a red pencil icon. At the bottom of the main content area is a red button labeled 'Add Your Car Info'.

Landing Page



The screenshot shows a Google login page. At the top, it says 'Hi Fay' next to a profile picture and the email address 'fayyousuf@gmail.com'. Below this is a large text input field with the placeholder text 'Enter your password'. Underneath the input field is a checkbox labeled 'Show password'. At the bottom left, there is a blue link that says 'Forgot password?'. At the bottom right, there is a blue button labeled 'Next'.

Google Login

User Interface - User Inputs



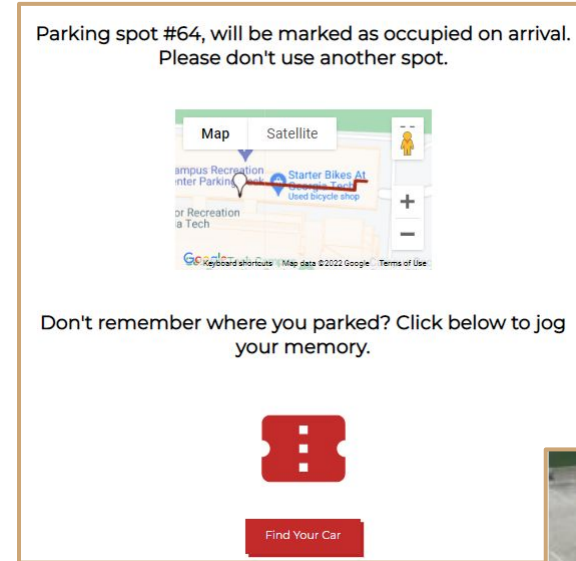
The screenshot shows the iValet website's navigation bar with links for Home, Park Your Car, Find Your Car, and Contact Us. Below the navigation bar is a form titled "Input Details" with a text input field for "Licence plate" containing the value "pop345", a checkbox for "Handicap parking", and a "Submit Info" button.

Navigates from "Add Your Car Info"

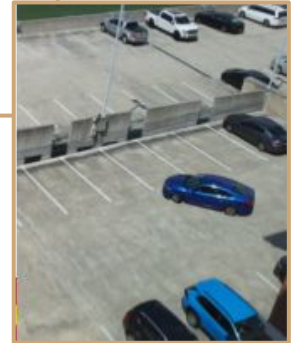
- Users are able to input their license plate and handicap needs.
 - (If Handicap is not available, routed to the closest parking spot)
- License plate info is currently used for logging history only (who parked in what spot, at what time)
 - Could be utilized to double check which spots users actually parked in the future

User Interface - Navigation

- Three navigation screens
- 1) Parking your Car - shows a route from the entrance to the appropriate spot
 - 2) Find your Car - when the user returns to the parking lot, a map from the entrance of the building to their car
 - 3) Exit the Lot - after the user has paid, they will be shown a map navigating them from their spot to the lot exit

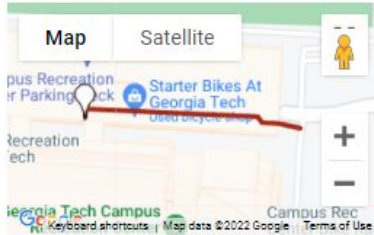


Map to initially park



User Interface - Navigation

Here's your car! Be careful of oncoming traffic.



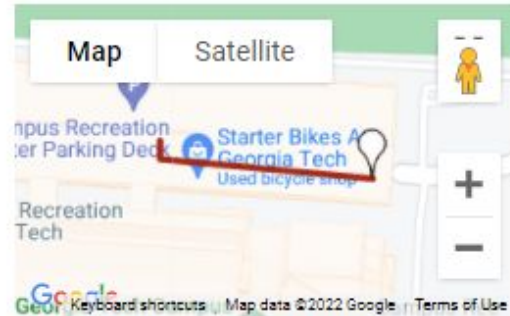
Before you go, we'll have to charge you a fee for parking with us.



Pay Before You Go!

Map to find parked car

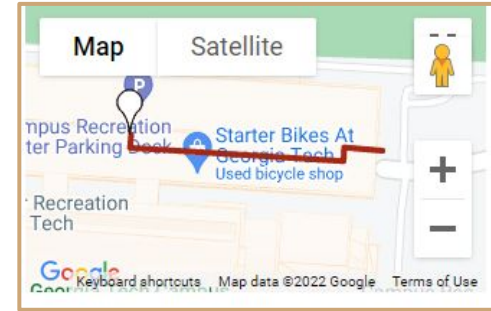
Thank you for parking with us! Exit the Parking Lot.



Map to exit parking lot after paying

User Interface - Navigation

- Each embedded map is centered on the CRC latitude and longitude coordinates in Google Maps
- Each parking lot is assigned three .KML files, ParkCrc#, FindCrc#, ExitCrc#
- KML is a custom route drawn in Google maps.
- The routes are accessed by the raw address of the .KML on GitHub
 - <https://raw.githubusercontent.com/Robuddies/iValetUpdate/backend/KMLs/FindCrc52.kml>



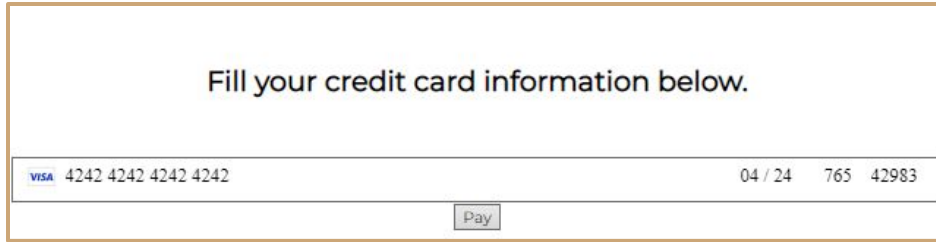
ExitCrc39.kml	Add files via upload
ExitCrc40.kml	Add files via upload
ExitCrc41.kml	Add files via upload
ExitCrc42.kml	Add files via upload
ExitCrc43.kml	Add files via upload
ExitCrc44.kml	Add files via upload
ExitCrc45.kml	Add files via upload
ExitCrc46.kml	Add files via upload
ExitCrc47.kml	Add files via upload
ExitCrc48.kml	Add files via upload
ExitCrc49.kml	Add files via upload

User Interface - Payment



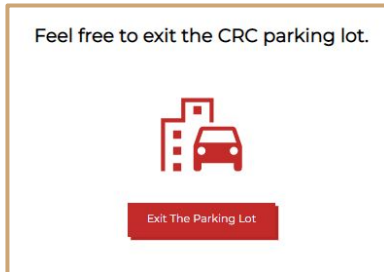
The screenshot shows the top navigation bar of the iValet CRC website with links for Home, Park Your Car, Find Your Car, and Contact Us. Below the navigation bar, the calculated payment amount is displayed as "\$2" with a "Pay" button underneath.

Calculated payment amount



The screenshot shows a form titled "Fill your credit card information below." with a text input field containing a Visa card number "4242 4242 4242 4242", an expiration date "04 / 24", and a CVV "765 42983". A "Pay" button is located at the bottom of the form.

Credit Card input



The screenshot shows a prompt to exit the parking lot with the text "Feel free to exit the CRC parking lot." and a red icon of a parking lot with a car. Below the icon is a red button labeled "Exit The Parking Lot".

Prompt to exit the lot - navigates to exit map

- Payment amount is scaled by CRC costs and how long the user has been logged (SQL query)
- After payment is processed, (Pay button) users will be taken to the Exit Navigation screen

Challenges

- Training the model with existing, online datasets caused inaccuracies when testing at the CRC (differences in lighting, intensity of shadows, etc.), required a lot of fine tuning once CRC testing began
- The trained Mask R-CNN model didn't work well with parking lots at CRC because of different angles and insufficient trained data. Using pre-segmented images with a classifier to solve this problem
- Coral was incompatible with Microsoft SQL Server (driver issue), had to switch to PostGreSQL

Future Work & Current Drawbacks

- The current design needs pre-defined images of parking spaces, so the it cannot be used in unknown areas immediately.
- We attempted to use another segment-based algorithm, Mask R-CNN, which can work in different places, but it does not work well. We can retrain it 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 (e.g proximity to seats in a large stadium)
- Integrate images from multiple cameras for a larger field of view.
- Current geolocation method on the UI to help users on the navigation screen can be unreliable, need to research another method