COMP 4102A Term Project Report
Client Parking lot assistant

Phillip AMANYA (101030494)

Klaus CUMANI (101032429)

Mohamed CAMARA (101028735)

Devin JN PIERRE (100869993)

Abstract

The aim of this project was to provide a software that can analyse and process the availability of vacant parking spots. This project was implemented to provide small businesses with better management and security of their parking lots. During the implementation of our program, there were many challenges that we faced such as defining the background so that changes could be detected, in our case that would be empty parking spaces and occupied lots. This was specifically challenging to accomplish with a live video. Having the program running in a suitable time frame was also challenging due to the video having to be broken down into frames which takes a significant amount of resources.

Table of Contents

Introduction

The *Client Parking lot assistant* system created, uses various software packages and model training techniques to distinguish between objects and parking lot spaces from a bird's eye view. The underlying goal of the software is to use advanced **computer vision** techniques coupled with strong **mathematical functions** to analyse and process the availability of parking spaces. Use of *object detection, digital outlining* and *tensor flow modelling* techniques help facilitate the creation of this useful system and makes it a good candidate for being placed into a vision application. The application developed this term for COMP 4012A provides a solution to small businesses with parking lots to better help them manage and secure their parking lots by providing a custom software capable of detecting vacant or taken parking spots. Additionally, it is able to track parking lot activity. Initially the program was meant for real time tracking, however due computational limitations it was developed as a processing program.

Background

Background research was needed to develop the project to the full extent. Tremendous amount of research went into figuring out the fastest running object detection solution available that is coupled with machine learning. Problems were encountered when trying to use preprocessed training models, they focused on car detection from the back, front and sides. This is probably because of the craze in computer visions with autonomous driving. Spending time on learning how to train our own models was something that led us to realize that having a real time solution and having an easily trainable system was something that was not possible at the moment. Most real time object detection models are mathematically optimised, whereas the trainable solutions were bearbone and focused on accuracy over speed. Several key packages were identified and were chosen for our solution for the following reasons; Provide a workable trainable model that can easily be updated to make it more accurate over time, Run at a reasonable speed given basic processing power available for an affordable parking lot solution for small businesses. 'Detecto is a Python package that allows you to build fully-functioning computer vision and object detection models' (Deteco, n.d.) which is necessary to identify the objects within a complex scene. The use of Detecto will allow streamline and speedy detection to benefit the system when comparing multiple variations of an object. The use of Torchvision makes each video frame smaller and more condensed for optimized prediction (Torchvision, n.d.). Backgroundsubtractormog2 does operations faster than normal - for calculations considering Foreground detection, which is an essential component of this application. The purpose of Background Subtraction is to aid in detecting the numerous moving objects in videos from stationary and static cameras (Backgroundsubtractormog2, n.d.). The last goal outlined in our initial proposal: 'Add the ability to map out the shortest route for the client' - was chosen to be discarded after we realised that without it being a live solution adding the ability for a client to map out free parking spots was redundant as the current implementation can only serve the business as a security solution, or to be used to manually track which vehicles are parking in the parking lot and not entering the establishment. It was determined to be *infeasible* to give directions when the application works retroactively. Using a custom trained model was something that proved to be necessary because of the nature of already trained models focusing on autonomous driving with models trained from dashboard level. It was not difficult to train our model and using Detecto made that job easier. Having to first generate a decent test image we used 150 to train our model. We used a common program used for model training called labelImg. Examples can be found in the results section but having to highlight the object of interest and labeling it appropriately for every photo took over six hours. Each image had a corresponding xml file with the details. To finish we used googles online coding platform to take advantage of their accelerated GPU and trained the model (Bi. A, 2020). This is the model used later in the running of the program.

There already exists a parking lot software called intuVision® Parking. They offer a *paid* solution to aid businesses running a parking lot with special features like pedestrian detection and detecting parked cars in the loading area (intuVision, n.d.). Our *free* solution differs as it would be a customer solution, something that they interact with at the entrance when collecting their parking tickets. They would be able to see the parking lot and identify free spaces from a birds eye view which are colour coded appropriately. Unfortunately, in practice the *live* model is too computer intensive and to make use of the system efficiently, the application would need to be retroactive for the purposes of this demonstration. With more computing power and development time it could be extended into real-time.

Approach

Our parking lot solution as it stands holds more as a proof of concept and would need to be refined to be applicable for live streaming and parking lot monitoring. The way our program works is as follows. When applied to a new parking lot the developer or business owner would need to map out all the available parking spots that would be monitored. This is done by using the mouse to outline the corners of the parking spots given they are rectangular in shape. These coordinates on the image are saved in a numpy array and exported onto a file generated by the program. In the event the parking lot has already been mapped there won't need to be a need to remap the parking spots. One limitation comes as the spots are static, if the camera is moved the spots would need to be remapped. In a future implementation we could have used houghs transform to map out the straight lines in the parking lot and program the application to automatically detect parking spots. Or alternatively we could have a trained model to detect parking lots and have that be applied to any new parking lot and then extract the coordinates. After mapping a new parking lot the program goes on to load the trained model of cars taken from a height at which any parking lot camera might be. The program then takes each frame of the incoming video and analyzes it to determine where coordinate wise a car is. The program then uses background subtraction methods to highlight any moving objects in the frame from a base frame which was the previous frame. Initially without object detection this method led us to a lot of false positives. It would also stop detection of a car as soon as it had stopped moving, this happens because of the way background subtraction is implemented explained in detail below. Because we were losing track of an object every time it stopped moving for more than three frames it made us unhappy with using background subtraction alone even though it was perfect computationally for live videos. Once the program knows what frame is a car and where moving objects are, we just cross check that a moving object is a car, highlighting it red. We cross check any car that is inside a parking spot and highlight it blue. Finally we find empty parking spots and highlight them in white. The program does this processing for every frame in a video and outputs the solution to an output video file.

The reason we were losing tracked objects every time they came to a stand still for more than three frames is due to the nature of the implementation of background subtraction. Every new frame is pixel by pixel subtracted from the last frame.

$$P[F(t)] = P[I(t)] - P[B]$$

The result is a mask which is the intensity of pixel change, that is why when a vehicle comes to a stop the intensity change becomes smaller until the background subtractor considers the object part of the background. The Mean filter is used as a background by taking a series of preceding images and averaging them.

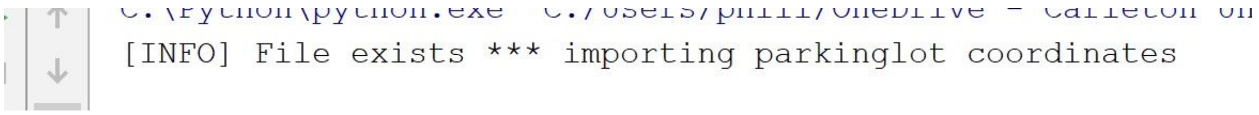
$$B(x,y,t) = \frac{1}{N}\sum_{i=1}^{N} V(x,y,t-i)$$

Where N is the number of preceding images taken from the video. The foreground is then obtained from subtracting the current image from the average of the images and thresholding it so that minute differences are not accounted for (Foreground Detection, n.d).

$$|V(x,y,t) - B(x,y,t)| > \mathrm{Th}$$

Results

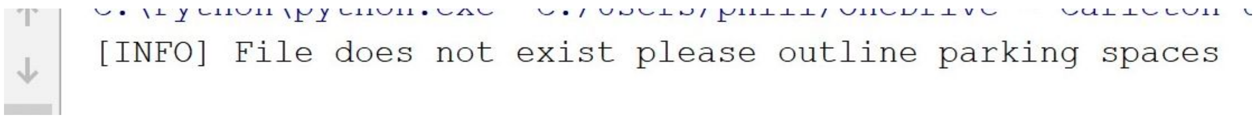
When you first run the program you will get one of two [INFO] messages. If the parkings coordinates exist you will get the following message.



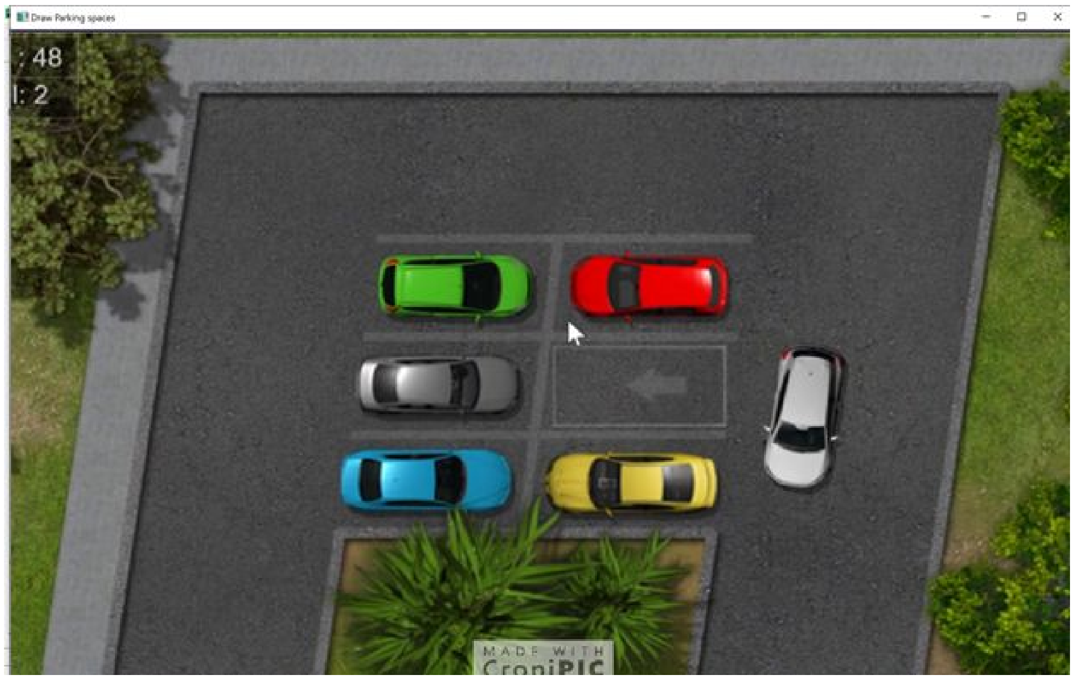
```
C:\Python\python.exe C:/Users/phili/OneDrive - Carleton Un
[INFO] File exists *** importing parkinglot coordinates
```

If the file does not exist you will get the following message

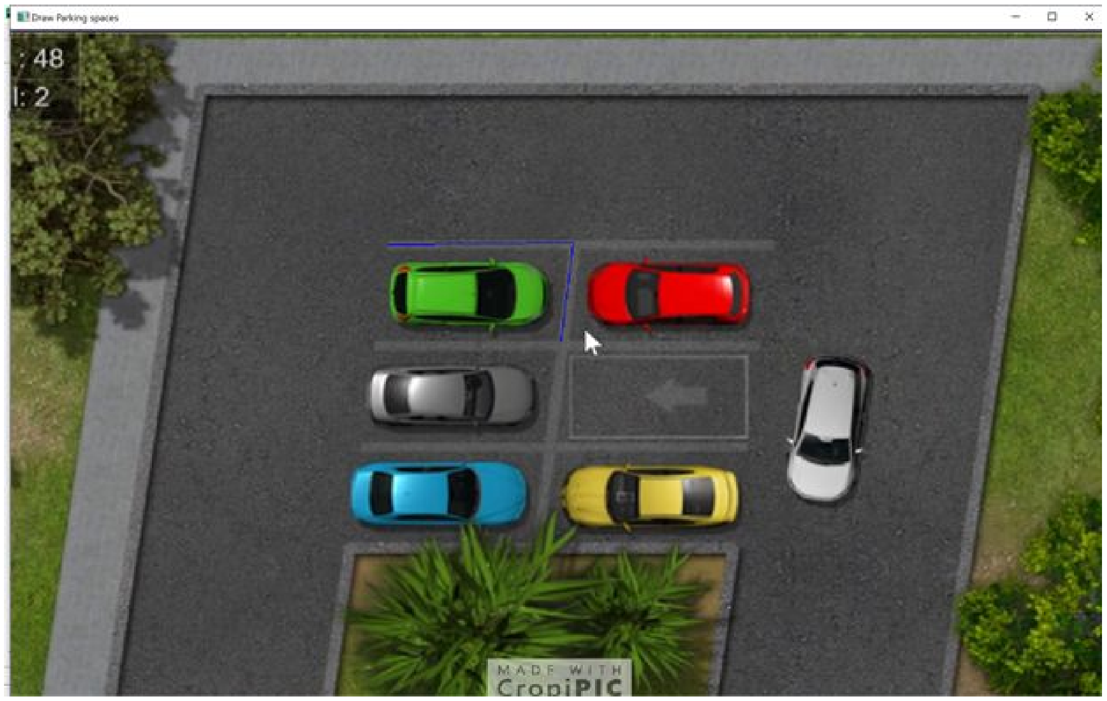
It will also open up the first frame of the video for the user to mark every parking spot.



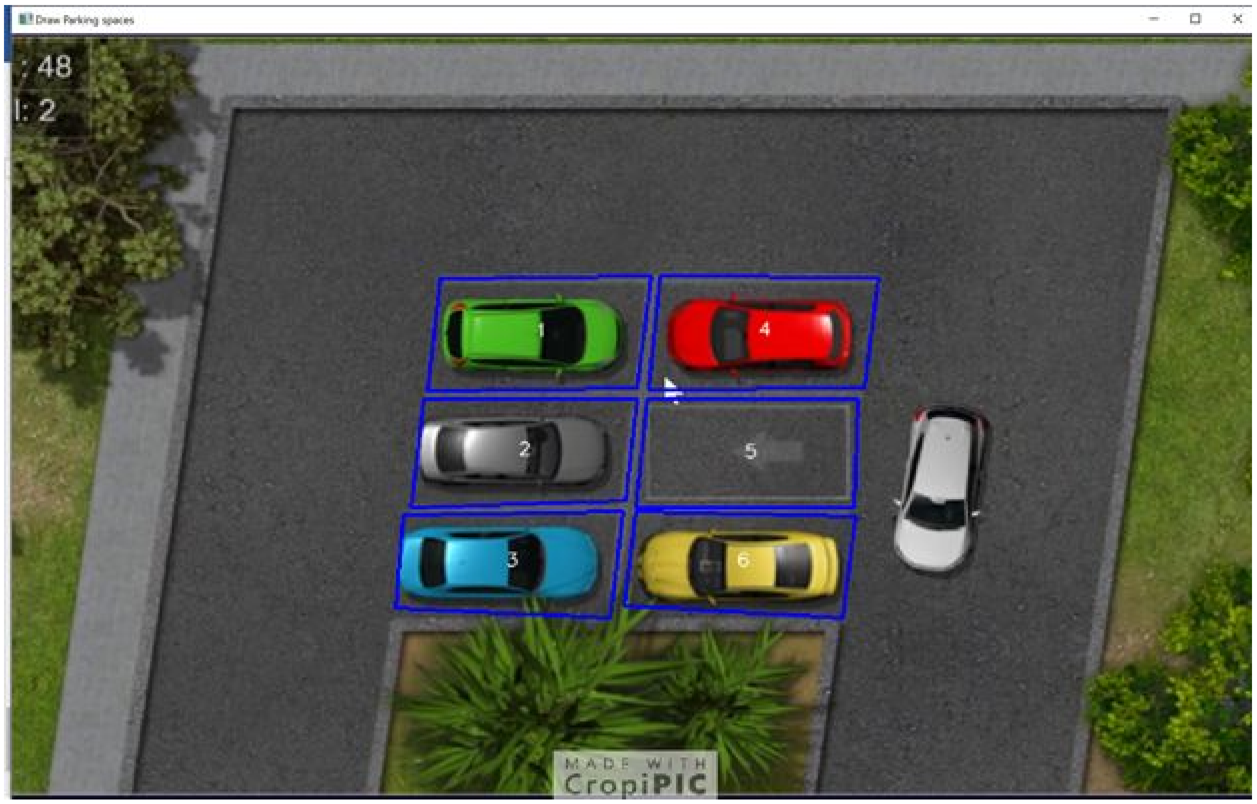
```
C:\Python\python.exe C:/Users/phili/OneDrive - Carleton U
[INFO] File does not exist please outline parking spaces
```



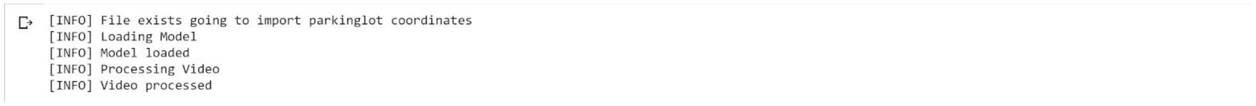
After you click on two corners of the parking spot a line will appear, the next corner will show two lines as follows.



When you have completed clicking all the corners of the parking lots they should be numbered and look as follows



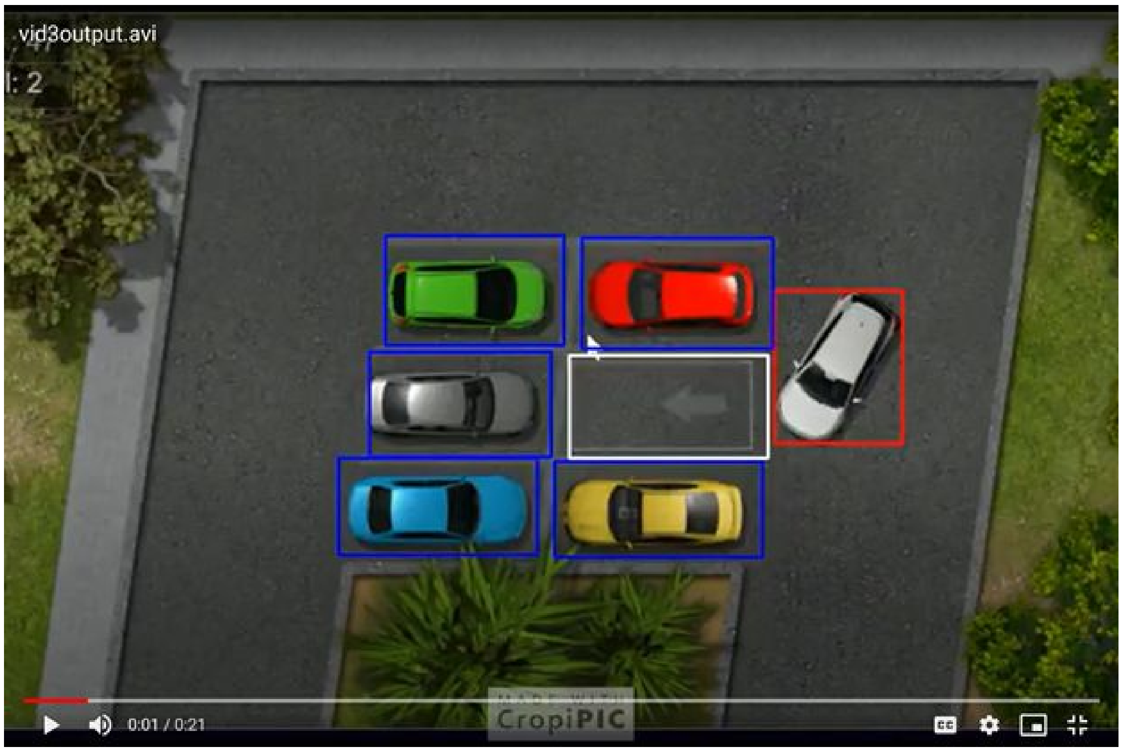
If there exists a parking lot map of the video then there will only be print statements of the status of the processing program

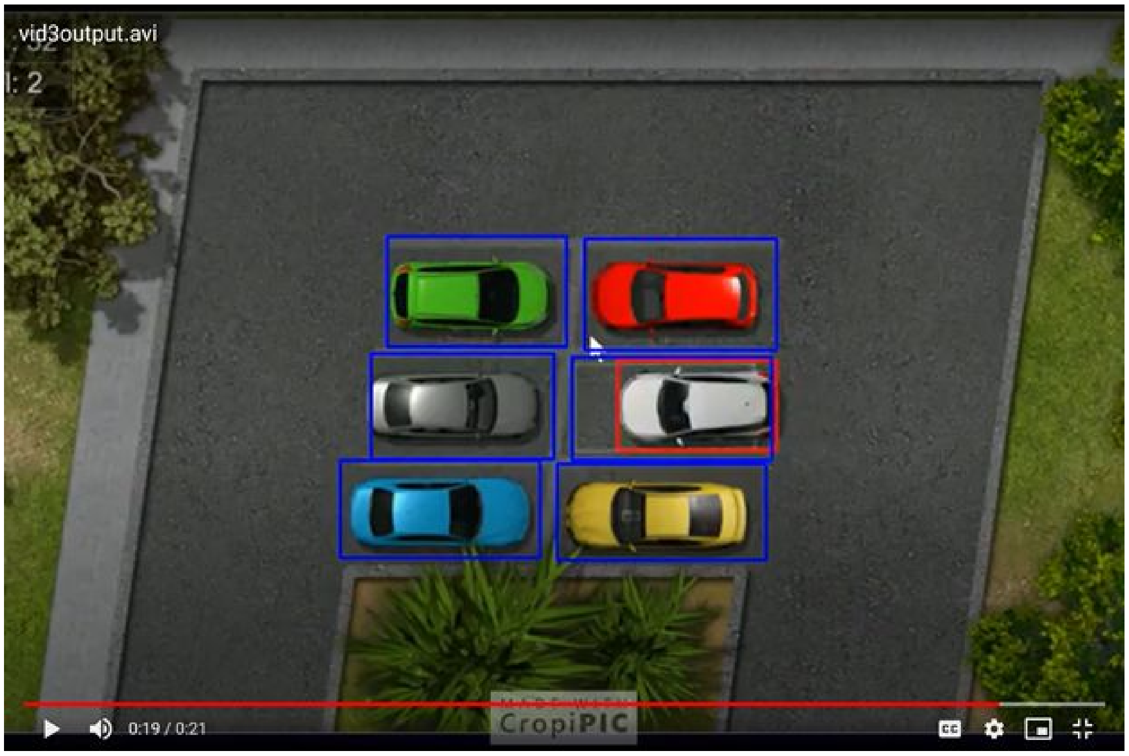


```
[INFO] File exists going to import parkinglot coordinates
[INFO] Loading Model
[INFO] Model loaded
[INFO] Processing Video
[INFO] Video processed
```

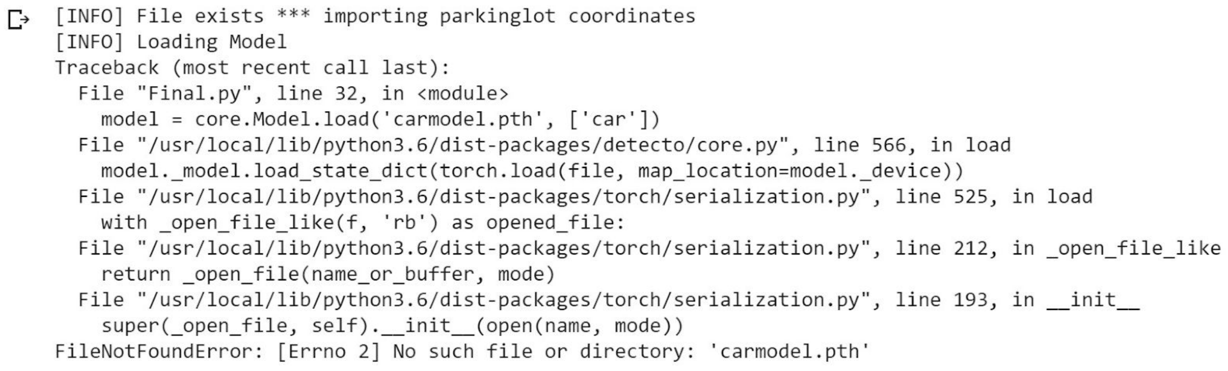
Once the program is done processing you should get an output file in the directory.

The program should show white parking spots that are empty, blue rectangles are taken. It should follow a car in a red rectangle.





If you get this failure message during run time you need to get the carmodle.pth from culearn, it was too big to upload to github with the code



```
[INFO] File exists *** importing parkinglot coordinates
[INFO] Loading Model
Traceback (most recent call last):
  File "Final.py", line 32, in <module>
    model = core.Model.load('carmodel.pth', ['car'])
  File "/usr/local/lib/python3.6/dist-packages/detecto/core.py", line 566, in load
    model._model.load_state_dict(torch.load(file, map_location=model._device))
  File "/usr/local/lib/python3.6/dist-packages/torch/serialization.py", line 525, in load
    with _open_file_like(f, 'rb') as opened_file:
  File "/usr/local/lib/python3.6/dist-packages/torch/serialization.py", line 212, in _open_file_like
    return _open_file(name_or_buffer, mode)
  File "/usr/local/lib/python3.6/dist-packages/torch/serialization.py", line 193, in __init__
    super(_open_file, self).__init__(open(name, mode))
FileNotFoundError: [Errno 2] No such file or directory: 'carmodel.pth'
```

Sample video was taken from a screen recording of game play on ‘Time to Park’. The tool was used to obtain a clear bird’s eye view (Time to Park, n.d).

List of Work

Equal work was performed by all members.

GitHub Page

https://github.com/PhillipAsiimwe/4102_ParkingLot_App

References

BackgroundSubtractorMOG2 - Class Reference. (n.d). Accessed Online Feb 05, 2020 at https://docs.opencv.org/3.4/d7/d7b/classcv_1_1BackgroundSubtractorMOG2.html

Bi. A (2020). Building custom-trained object detection models in Python. Accessed Online March 14, 2020 at https://towardsdatascience.com/build-a-custom-trained-object-detection-model-with-5-lines-of-code-713ba7f6c0fb

Detecto - Build fully-functioning computer vision models. (n.d). Accessed Online Feb 02, 2020 at https://pypi.org/project/detecto

Foreground Detection - From Wikipedia, the free encyclopedia. (n.d). Accessed Online March 11, 2020 at https://en.wikipedia.org/wiki/Foreground_detection

intuVision - Video Analytics Solution. (n.d). Accessed Online Feb 01, 2020 at https://www.intuvisiontech.com

Time to Park - Play Time to Park. (n.d). Accessed Online March 11, 2020 at https://www.crazygames.com/game/time-to-park

Torchvision - PyTorch, transforms are common image transformations. (n.d). Accessed Online Feb 03, 2020 at https://pytorch.org/docs/stable/torchvision/transforms.html