1. Method

This part of the report details the process that was followed to process images of parking from three cameras namely left camera, center camera and right camera. The objective of processing these images was as follows:

- Segment region of free parking spaces from the overall parking space that is in view of the image from either of the three cameras
- Identify distances of parked cars with each other
- Allocate parking lots to the segmented free parking spaces. Each parking lot is roughly considered to be 2.5m wide
- Assign geo-coordinates to regions of parking spaces in the image form either of the three cameras
- Calculate total number of parked cars in the image
- Calculate number of free parking lots available for parking in the image
- Calculate total number of parking lots (free + occupied) as viewed in the image from either of the three cameras
- Send update notifications about the change in number of free or occupied parking lots to a phone.

In order to achieve the objectives, images from the cameras were collected and they comprise the dataset for this project. The images were then processed using opencv in c++ and python both to achieve the desired objectives. Furthermore, object detection for cars in the image was achieved using Keras-retinanet running on the tensorflow framework. Following are the details of the implementation that was developed to achieve aforementioned objectives.

1.1. Dataset

The dataset was a collection of images in the daytime, evening time and night time of the views from the three cameras. Each image is 1280 pixels in width and 720 pixels in height and was a colored image stored in ".jpg" format. A total of 197 images from all timings of all the three cameras were labeled and cars as well some non-car objects were labeled and the labels for each image were stored in VOC-format in a ".xml" file. It is important to note that the labels are of two types only namely 'car' and 'not car'. This was a choice that was taken because the object of interest in our project were cars (including vans, trucks, motorcycle etc) and not other objects like trees, lamps, cycles etc. having only two labels makes a case for a better object detection as the neural network does not have learn weights for multiple classes. The labeling of data gave us 8678 labels in 197images images from all timings of all the three cameras. This constitutes the training dataset for further parts of the project. An example of these labels is as follows:

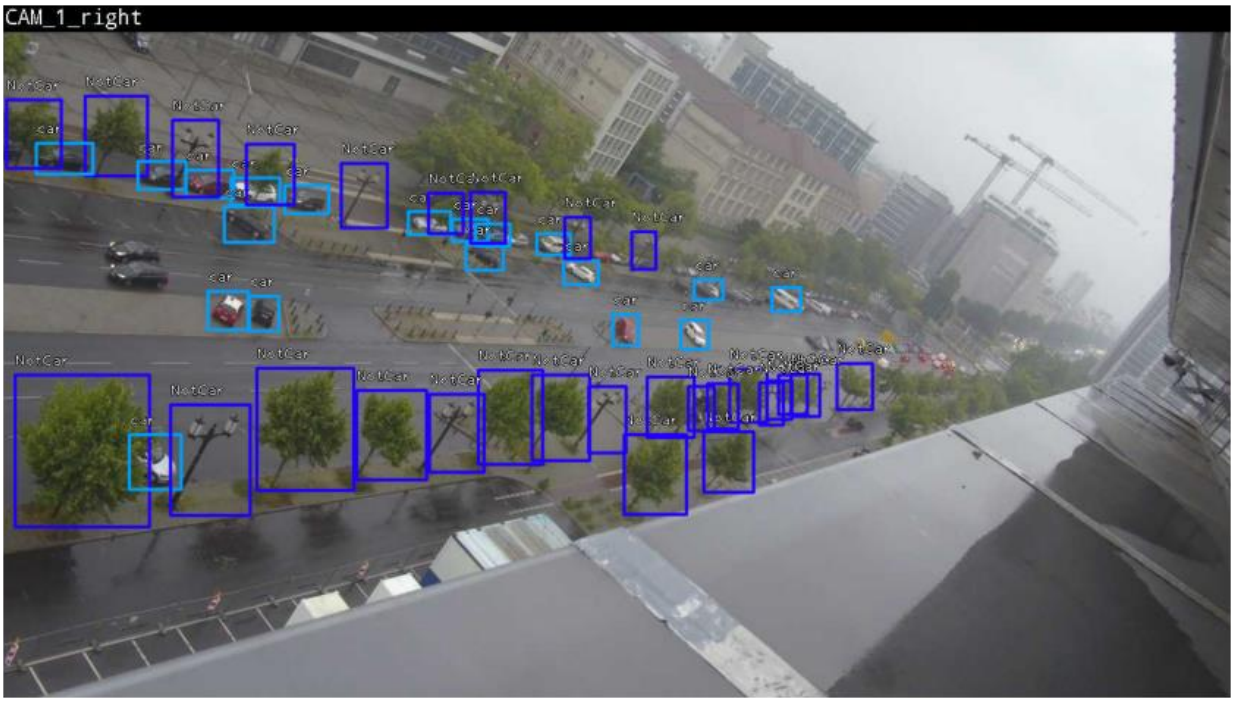


Figure 1 example labeled image right camera rainy weather

1.2. Neural network for car detection

Many different options were available for object detection of the cars in the images in our dataset. We chose to go with neural networks because they are powerful machine learning techniques that are effective for this purpose. In order to implement a neural network we chose RetinaNet[1] which is a neural network in itself. RetinaNet is one of the best one-stage object detection models that has proven to work well with dense and small scale objects. For this reason, it has become a popular object detection model to be used with aerial and satellite imagery. The view of the parking are from either of the three cameras that were used for this project is a far off view. That is why we made a choice to use this RetinaNet as aerial or satellite imagery is also a far off view of objects. A general overview of this neural network is that there are four major components of a RetinaNet model architecture (Figure 3):

a) Bottom-up Pathway - The backbone network (e.g. ResNet) which calculates the feature maps at different scales, irrespective of the input image size or the backbone.

b) Top-down pathway and Lateral connections - The top down pathway upsamples the spatially coarser feature maps from higher pyramid levels, and the lateral connections merge the top-down layers and the bottom-up layers with the same spatial size.

c) Classification subnetwork - It predicts the probability of an object being present at each spatial location for each anchor box and object class.

d) Regression subnetwork - It's regresses the offset for the bounding boxes from the anchor boxes for each ground-truth object.

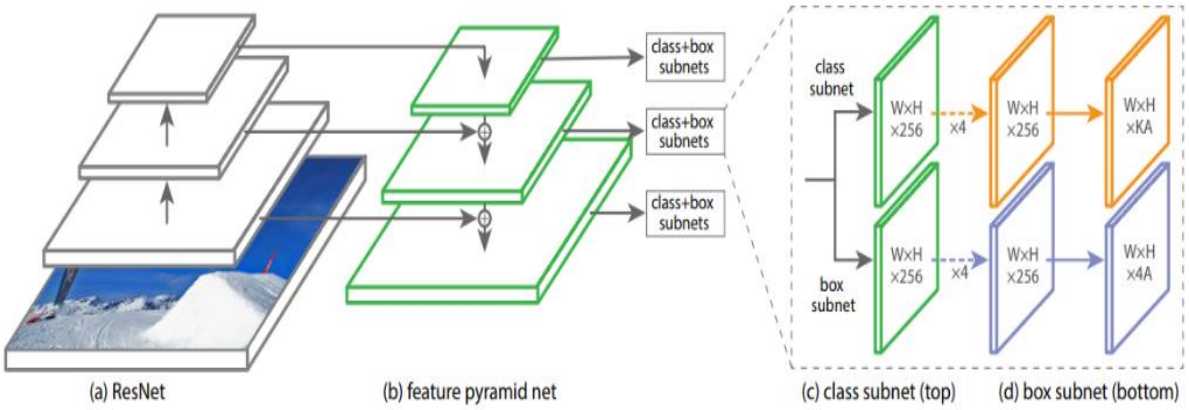


Figure 2 architecture of RetinaNet

A python [2] implementation of this neural network was used in default configuration i.e. the backbone of the neural network was kept as 'resnet50' . Residual Network (ResNet) is a Convolutional Neural Network (CNN) architecture which was designed to enable hundreds or thousands of convolutional layers. The backbone we used had 50 layers hence 'resnet-50'.

1.3. Training the neural network

Three different neural networks were trained. Each one for images of each camera. The training was done online using google colab [3] and the trained models were then downloaded for use. The model for center camera images was trained on 69 images, the model for left camera images was trained on 72 images while The model for right camera images was trained on 56 images where each image had multiple labels. For the two classes of labels/objects that were to be detected, each model was trained to have a classification loss of less then 0.15.

1.4. Region of interest

In views of each of the cameras, fixed region of interest was identified manually which constituted the region where we would assign parking lots. Following images can explain this part. The blue region in the images was hard-coded to be the parking areas of interest in any of the following images:

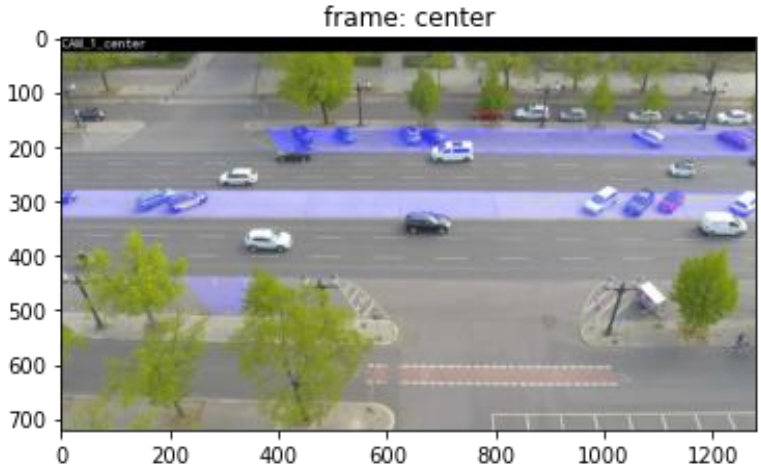


Figure 3 region of interest center

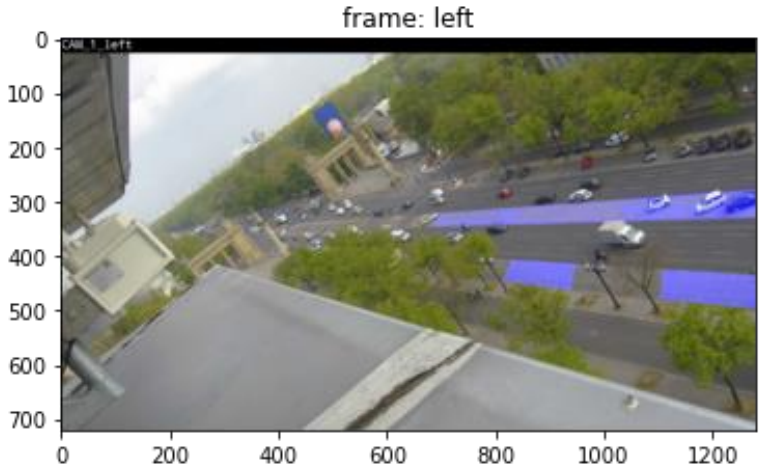


Figure 4 region of interest left

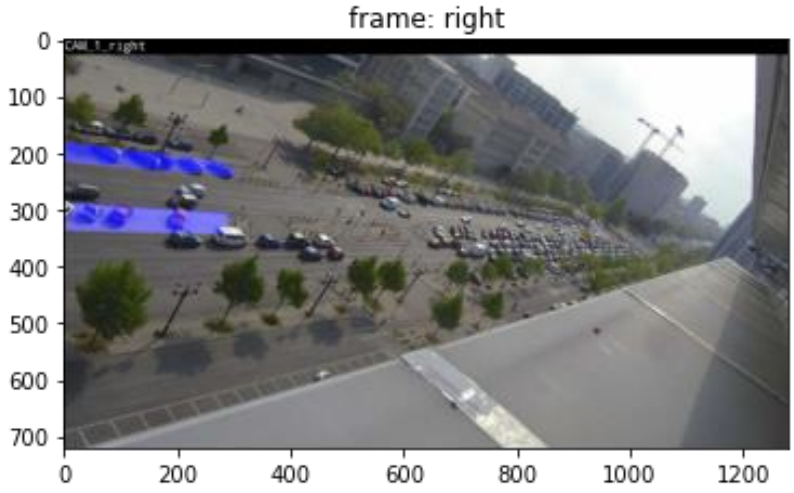


Figure 5 region of interest right

1.5. Template matching to identify which image

The images in the dataset are recorded by a camera. The camera puts in the image a text stating which camera's image is the image. It can be seen on the top _left corner of any image in our dataset. A template matching technique was used to identify which camera's image is being processed by the project in running time.

Figure B : the three templates used to identify camera of the image

The region of the image that these template match is the top left corner of the image as can be seen in figure 6.

1.6. Segmentation of free spaces in the parking region

The trained model were used to identify cars in the image. We then moved to identify if there are cars that are detected in any of our region of interest. The areas in our region of interest where a car was not detected was considered to be a free parking area and that is how it was segmented from the total parking region that was in the view of either of the three cameras. The green region in the following images identify our segmented region. Please note that object detection by using the keras-retinanet was essential in order to get these results.

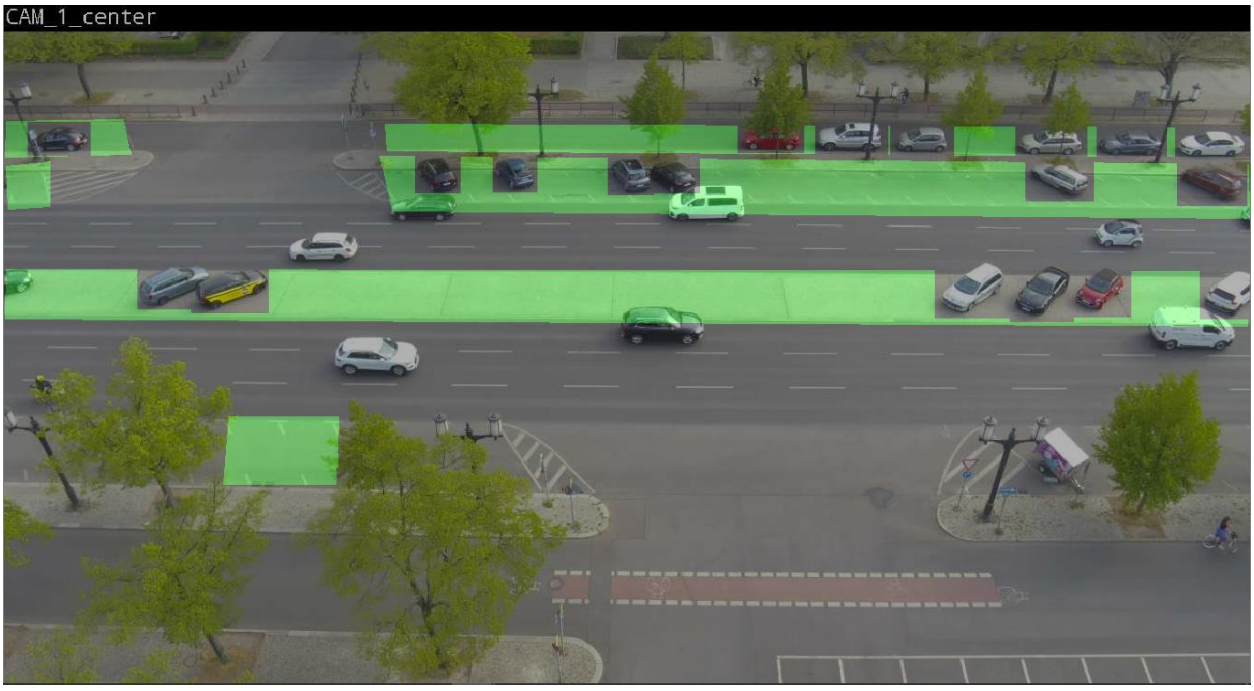


Figure 6 segmented parking region

1.7. Point correspondences

Besides the original view of the cameras, each camera had a second view of the parking. Using the second view allowed to generate point correspondences that could be used for a 3d scene reconstruction of the parking. The second view of each camera was acquired by accessing the live camera using TU Berlin's von connection and downloading images of the changed angle for each camera. In order to generate the point correspondences, corners were detected in each view using SURF feature detection in OpenCV. The corners detected in either of the views were then matched with each other using brute force matcher. The coordinates of these points in the images serve as the basis for converting these points into 3D points which can then be used to identify distances between objects in the images.

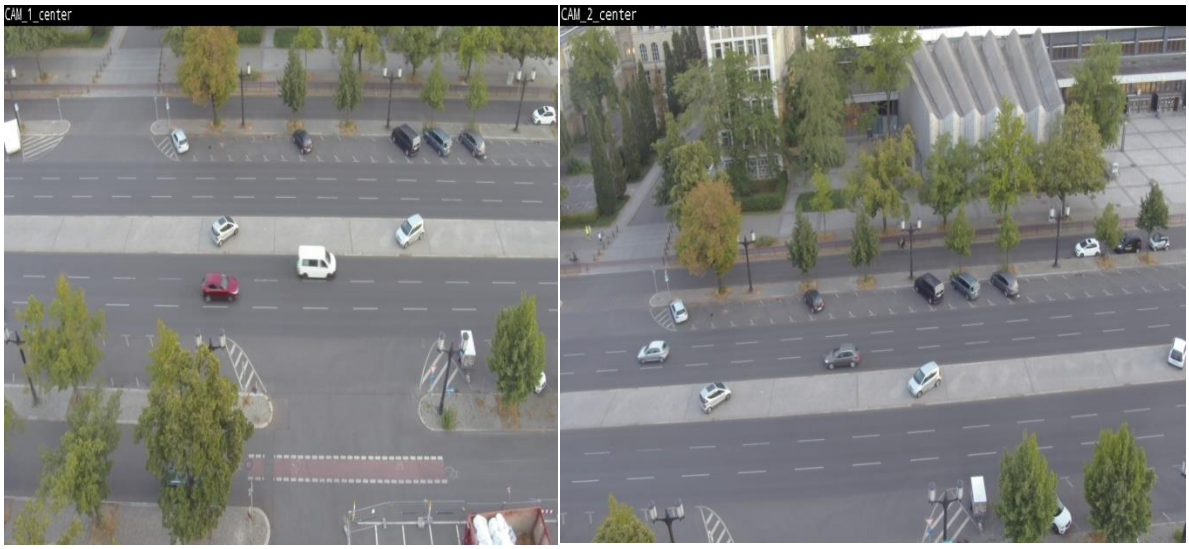


figure A:: two views of the center camera

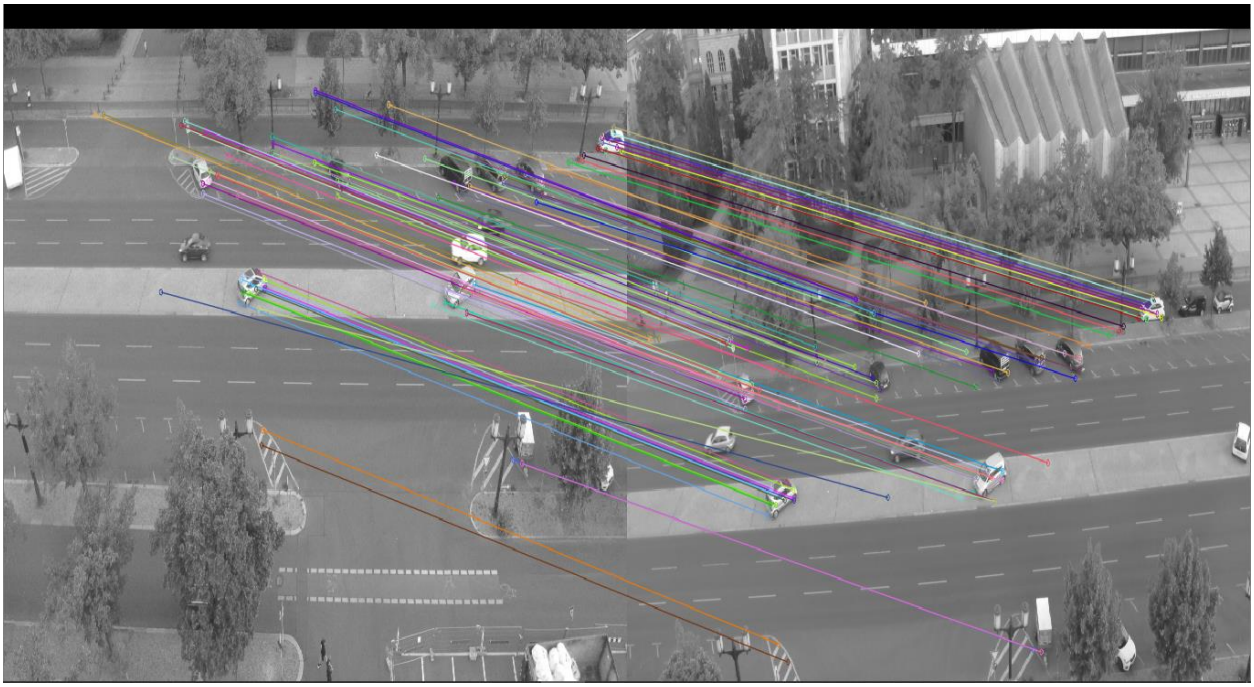


Figure 7 point correspondences between two views of center camera

1.8. Identifying distances

Besides the 2D point correspondences, we were provided with the camera matrix (intrinsic parameter) and distribution coefficients for each camera. These were acquired by calibrating the three cameras on chess board images.

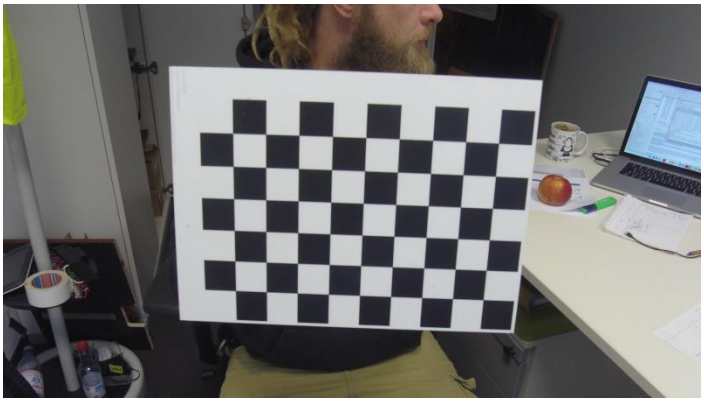


Figure 8 chess board for center camera

using the 2D point correspondecnes and the intrinsic parameters of the caliberated camera, the fundamental matrix, essential matrix and epipolar geometry between the two images was determined and the2D points were triangulated to represent points in 3D. These points were however only points that were in the view of the two views in each camera as shown in figure A. For our purposes we needed distances between objects that were visible in just one view of each camera. So from the distances obtained via triangulation and their respective 2D point in the first angle of each camera, a planar reconstruction of the entire image of the first angles of each camera was achieved using a simple

homography (perspective) transformation. The matrix for this transformation was just a homography matrix between the 3d points that were triangulated and the normalized homogenous coordinates of the 2d points that were used in the triangulation. This gave us relative distances up to scale for the entire region in the first angle views of each camera. Note that the distances were relative to the coordinates of the first angle of each camera and not absolute distances in meters. This was because no actual correspondence of real 3D distances in the parking regions were made. Rather the 3D points were calculated using point correspondences that existed between the similar region of the two different viewing angles of each camera. Because the similar region was not of the entire image, we were forced to find the perspective transformation between the 2d points of only the first angle and the triangulated 3d point. In that way we could assign 3d coordinates to points in the entire first view of each of the cameras. These distances are not absolute distances in meters rather they are relative to the view of the second angle of any camera with respect to the first and they are only true up to a scale. A result of the approach is as under:

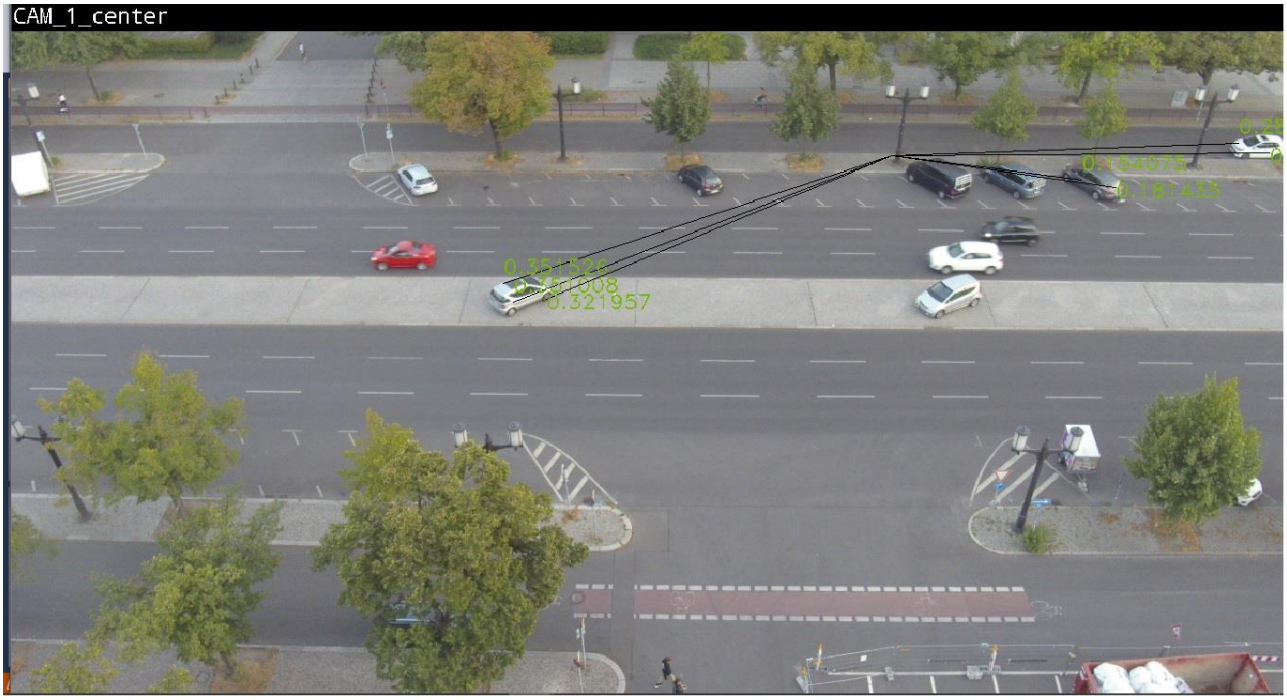


Figure 9 distances between objects

1.9. Free/occupied parking space

If in an area in the region of interest as shown in figure 3,4 and 5, a car was detected, then that region was marked as an occupied space in the overall parking area. Other areas were considered to be free for parking. In the following image, lots that are drawn in red were detected to be occupied.

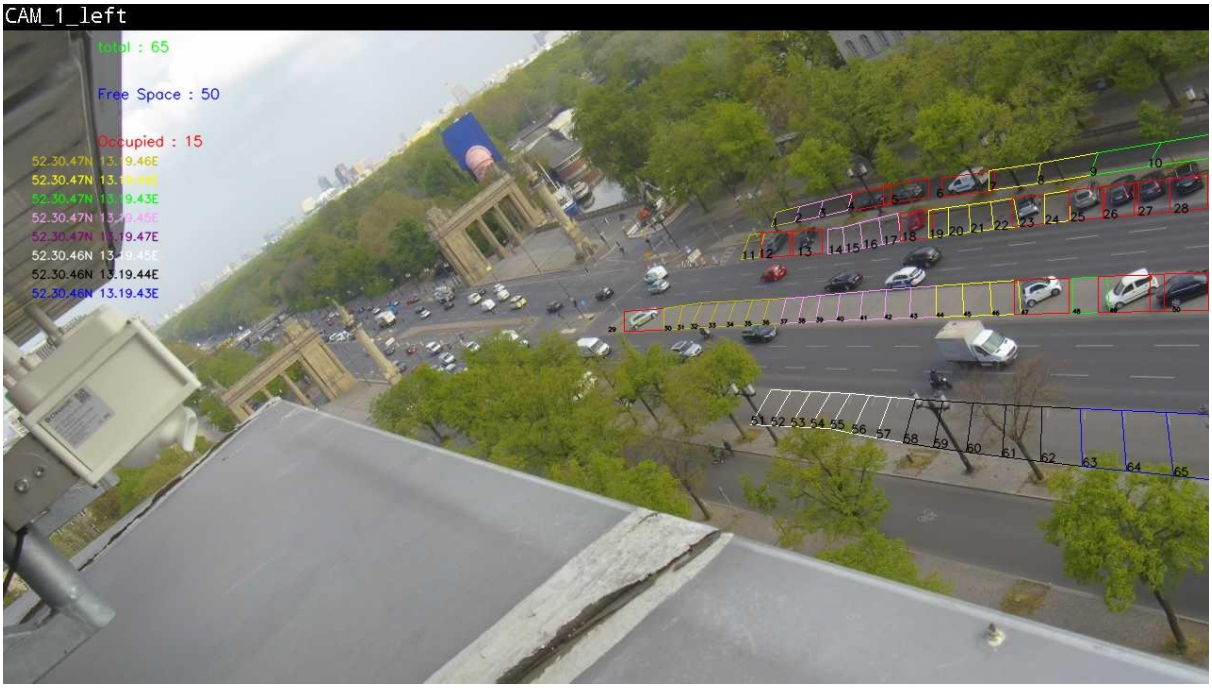


Figure 10 occupied parking lots drawn in red color

1.10. Allocation of parking lots

In the images of each camera, coordinates were manually put that represented regions in one which one car be parked. The width of such regions (parking lots) was roughly made of enough pixels to accommodate one car as seen in the image. Having known these coordinates for each of the center, left and right camera, we allocated free spaces where ever the car was not detected and moved to the next coordinate. But whenever a car was detected, then the coordinates for the next parking space were adjusted with respect to the bounding box (as given by the Neural network) of the detected car. In this way we were able to traverse the coordinates in the region of interest, assign parking lots and dynamically adjust parking lots in the areas where car is detected. See figure 10

The allocation of parking lots is totally dynamic. This can further be seen in the result image of figure 11 and 12. Observe in the parking lot 16 in figure 11. A car is detected and the neighboring parking lots of lot number 17,18,19 and so on are not occupied. In figure 12, for the same car in the same parking lot when the bounding box changes because of car driver getting into or out of the car, the parking lots 17,18,19 and so on actually shift their position to adjust for the dynamically changing bounding box of the car at parking lot 16.

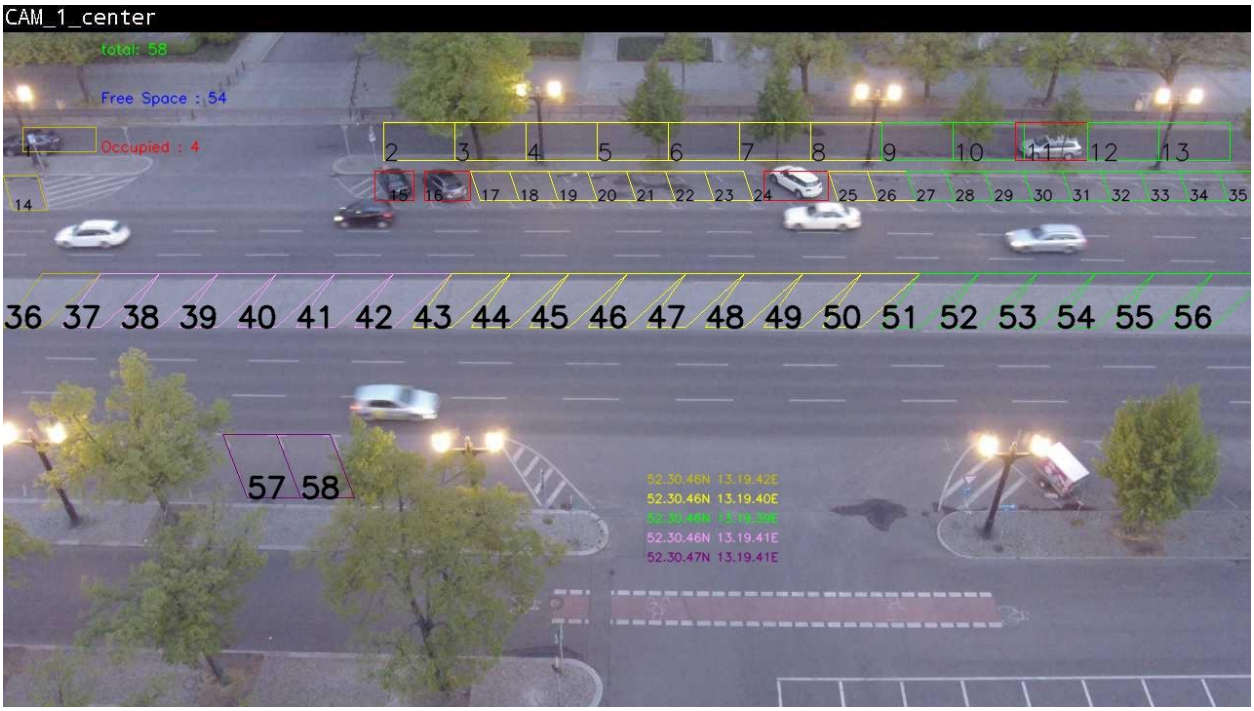


Figure 11observe bounding box of parking lot 16

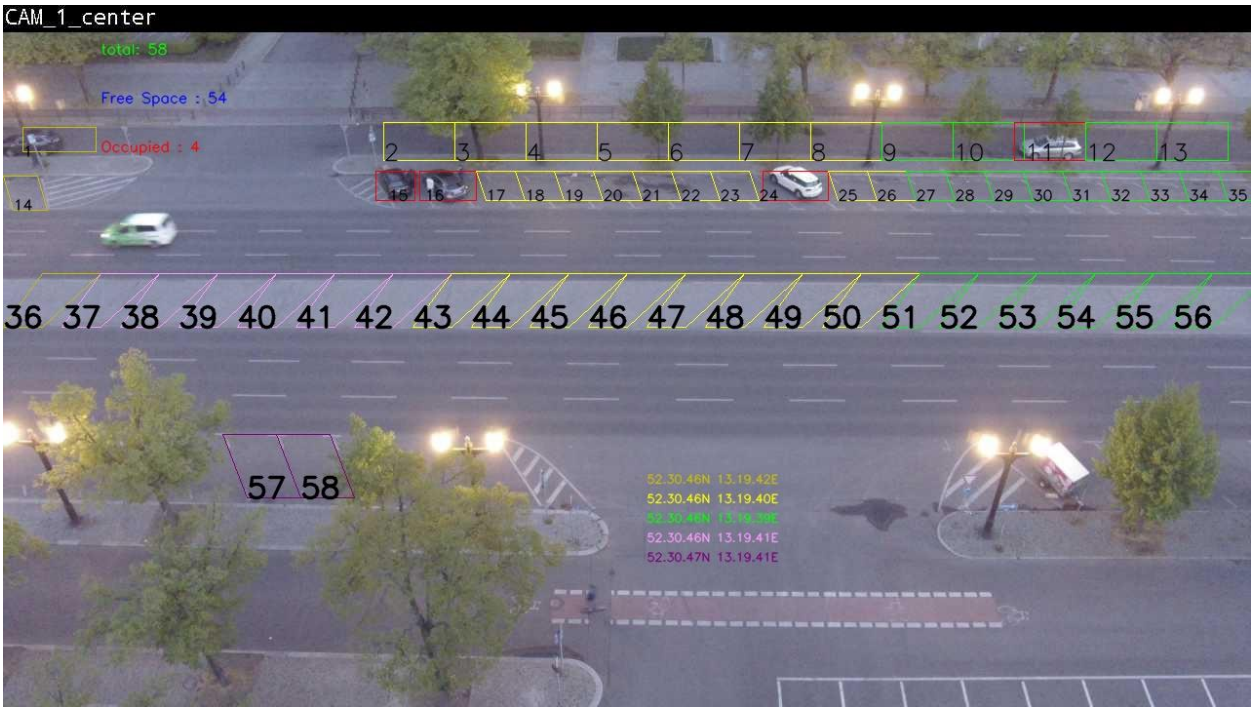


Figure 12 with change in dimension of lot 16, the neighboring parking lots also dynamically shift their positions

1.11. Assigning Geo coordinates

Regions in the image were referenced by viewing the same areas of Straße des 17. Juni, Berlin on google earth[4] and corresponding geo coordinates were noted manually. A reference image from google earth is as under. It shows the value of geo-coordinates in the bottom right corner.

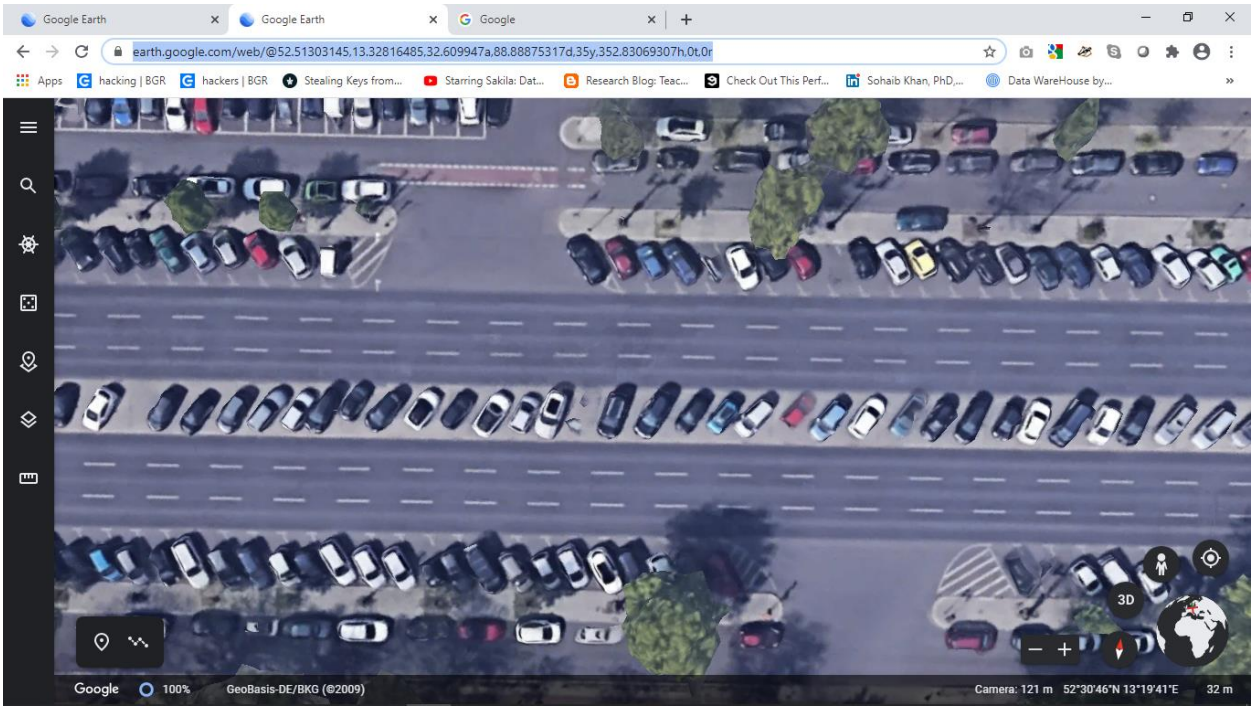


Figure 13 google earth reference image for center camera

The regions that are visible on google earth were manually matched with the view in the images of either of the three cameras and a general observation was made of the geo-coordinates value that is given by google earth. The changes in the geo-coordinate values were noted for regions and in the images from cameras in the dataset, the corresponding pixel locations were marked. In this way a relation of geo-coordinate values and pixel values in the camera images was developed. The geo-coordinates are shown in result images in different colors. See figure 10

1.12. Phone notification

Using the Twilio[5] API in python, SMS sending feature was developed which allows the program to send messages to one registered phone number. The idea behind this implementation was to send regular updates at 10 seconds intervals so that the information about number of free and occupied parking spaces could be send as an output of the program besides the processing of images.

2. Results

2.1. Free/occupied space

For an assessment of results a criteria of errors in the results was defined. The number of errors in all the images that were assessed is taken as the performance of the project. The criteria of an error is:

- Not detecting a car parked in the region of interest in either of the three images is an error.
- Detecting one car more then once is an error.
- Marking a free space as parked or vice versa is an error
- Overlapping bounding boxes of free space areas is an error

Because the assessment is subjective and manual by observing the processed image manually, certain outputs/results were not considered an error:

- Missing identification of cars or parking spaces that are occluded by neighboring cars or passing by cars or trees etc is not considered an error
- Overlapping of bounding boxes of parked cars with each other is not considered an error

An example image of a final processed image is discussed n figure below for understanding:

Figure 14 error image

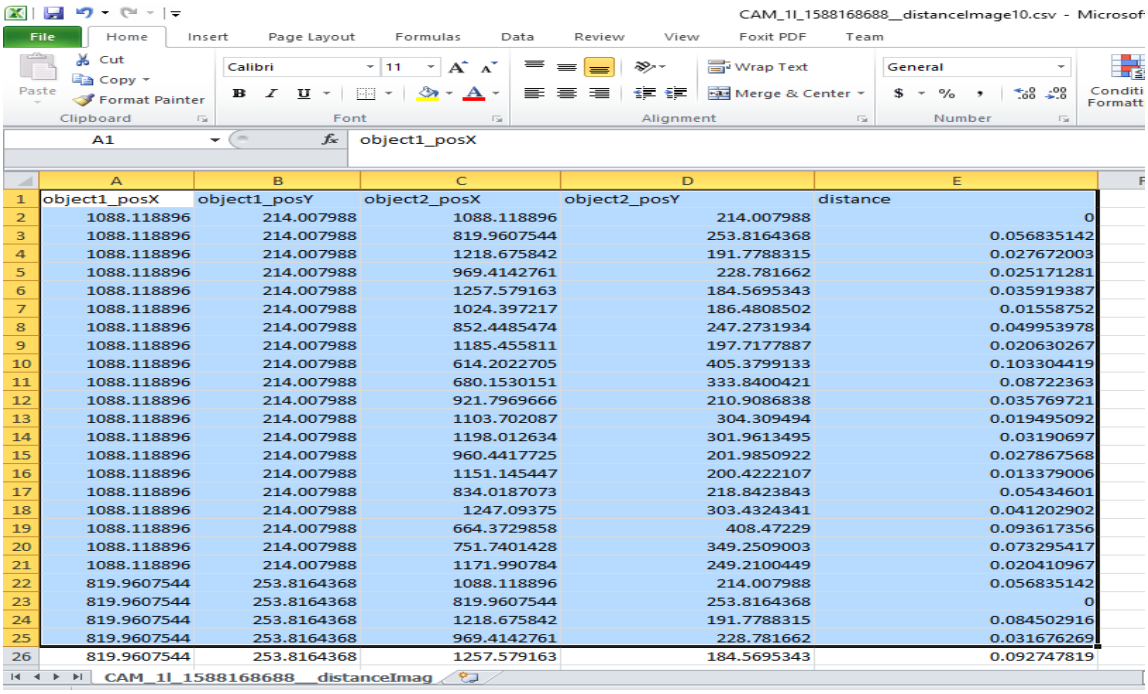
In figure 12, the yellow bounding box is considered an error because a place could have been identified between parking lot 5 and 6. The reason for the error is that the bounding boxes of the two parked cars is too big.

The region marked by the green box in figure 12 is not considered an error because that parking area is completely occluded by the bounding boxes of the cars.

Based on this subjective assessment a total of 27 images were analysed and 11 errors were found. Note that a measure of the correctly identified free parking space or parked space was not made.

2.2. Relative distances

The processing of each image resulted in measuring of distances which are measured up to a scale and are not real distances as mentioned in section 1.8. these distances were recorded in a csv file as an output of the project. A screenshot of a csv file with distance values is included in the report below



CAM_1l_1588168688__distanceImage10.csv - Microsof

File Home Insert Page Layout Formulas Data Review View Foxit PDF Team

Cut Copy Format Painter Paste Clipboard

Calibri 11 Font

Wrap Text Merge & Center Alignment

General Number

A1 object1_posX

	A	B	C	D	E
1	object1_posX	object1_posY	object2_posX	object2_posY	distance
2	1088.118896	214.007988	1088.118896	214.007988	0
3	1088.118896	214.007988	819.9607544	253.8164368	0.056835142
4	1088.118896	214.007988	1218.675842	191.7788315	0.027672003
5	1088.118896	214.007988	969.4142761	228.781662	0.025171281
6	1088.118896	214.007988	1257.579163	184.5695343	0.035919387
7	1088.118896	214.007988	1024.397217	186.4808502	0.01558752
8	1088.118896	214.007988	852.4485474	247.2731934	0.049953978
9	1088.118896	214.007988	1185.455811	197.7177887	0.020630267
10	1088.118896	214.007988	614.2022705	405.3799133	0.103304419
11	1088.118896	214.007988	680.1530151	333.8400421	0.08722363
12	1088.118896	214.007988	921.7969666	210.9086838	0.035769721
13	1088.118896	214.007988	1103.702087	304.309494	0.019495092
14	1088.118896	214.007988	1198.012634	301.9613495	0.03190697
15	1088.118896	214.007988	960.4417725	201.9850922	0.027867568
16	1088.118896	214.007988	1151.145447	200.4222107	0.013379006
17	1088.118896	214.007988	834.0187073	218.8423843	0.05434601
18	1088.118896	214.007988	1247.09375	303.4324341	0.041202902
19	1088.118896	214.007988	664.3729858	408.47229	0.093617356
20	1088.118896	214.007988	751.7401428	349.2509003	0.073295417
21	1088.118896	214.007988	1171.990784	249.2100449	0.020410967
22	819.9607544	253.8164368	1088.118896	214.007988	0.056835142
23	819.9607544	253.8164368	819.9607544	253.8164368	0
24	819.9607544	253.8164368	1218.675842	191.7788315	0.084502916
25	819.9607544	253.8164368	969.4142761	228.781662	0.031676269
26	819.9607544	253.8164368	1257.579163	184.5695343	0.092747819

CAM_1l_1588168688__distanceImag

Figure 15 distance values as an output of the proeccesing of images

3. Recommendations

3.1. Distance correction

As the distances that were calculated are only up to a scale, they need to be corrected to represent real life absolute distances. An approach is proposed herewith which recommends the implementation of complete scene reconstruction rather than the planar reconstruction that has been in this project. For scene reconstruction purpose, following steps should be followed:

- Use the already learned intrinsic parameters of the cameras.
- Observe distances from actual measurement from google earth of the same area as is viewed by the any of cameras.

- Record the corresponding pixel coordinate of the observed points for which distance was measured using google earth.
- Generate the point correspondences of the recorded pixel coordinates in the adjoining second view of each camera. See figure A and 7
- Complete a scene reconstruction using the intrinsic parameter of the camera the 2 sets of 2D points in the two views and their corresponding 3D points in the bundle adjustment algorithm.

A complete scene reconstruction will allow for measuring of absolute distances between objects that are in the view of the two views of any camera.

3.2. Segmentation of individual cars

In order to improve the precision of allocation of parking lots, rather only detection of cars, the body of the car can be segmented out from the image itself using a neural network that is trained to segment cars in an image. This will help especially where the car is parked or viewed at an angle from the camera instead of being viewed as a horizontally placed or vertically placed object. An approach of segmenting objects (parked cars) in the image will reduce the parking lots or cars that misidentified because of occlusion by the bounding box of a neighboring car/object.

3.3. Deployment on live camera

If a direct access to the camera is deployed alongside the project, then the project can be used to run on the live feed of the image. Currently the program on our computer completely processes around 0.25-0.167 frames per seconds. But on a GPU enabled machine the performance will be faster and the project could be used on the live feed from any of the cameras.

References:

[1] https://arxiv.org/abs/1708.02002

[2] https://github.com/fizyr/keras-retinanet

[3] https://colab.research.google.com/

[4] https://earth.google.com/web/@52.51303145,13.32816485,32.609947a,88.88875317d,35y,352.83069307h,0t,0r

[5] https://www.twilio.com/sms