Report on Final Project Parking Lot Occupancy

Computer Vision UniPD

Kabir Bertan 2122545
Francesco Colla 2122543
Alessandro Viespoli 2120824

September 2024

Setup and instructions to run the code

OS: Linux (DEI VLab Virtual Machine)

The code is provided with a CMakeLists.txt file that will handle the linking of the OpenCV library files and the libraries created to solve the problem. To compile and run the code extract all the contents of the provided .zip file inside of a folder **including the dataset folder that contains important data for the program**. In a terminal execute the following commands:

- cmake .
- cmake --build .

This will create the executable **main**; to run it pass in the command line the path to the empty parking lots images and the target folder:

```
./main  ParkingLot_dataset/sequence0/frames  ParkingLot_dataset/sequenceN
```

The project is organized to have all the `.cpp` files inside the **src** directory, while their headers are inside the **include** directory. In **ParkingLot_dataset** there are all the images required or that might be given to the program. In **report_images** there are all the requested images, plus the extra ones used in this report. The program upon start will prompt a **WAIT** message for approximately 40 seconds. This is normal and happens while the segmentation model scans the template images. Wait a few seconds and the program resumes showing status **READY**. After the READY status has been reached, simply press a key to check the different images with non noticeable computational delay.

Overview

The program manages to complete all requested task in the project handout. Despite an initial wait due to the segmentation method, the program shows in order all the main requests one by one.
Our project is designed to always show the best possible outcome when talking about detecting the bounding boxes; in other words we only present to the user the final aggregation of the bounding boxes detected on each empty parking lot image present in the sequence0 folder. **In case someone would like to see how our program manages to find and correct the bounding boxes on a single empty parking lot image, it is necessary to only have one single image inside sequence0 directory**.
The flow of the main program is the following: identify the parking spaces on each empty image and aggregate them by making an average; for each sequence image extract the ground truth, apply the car segmentation, detect cars, calculate the metrics and compute the 2D map.
In the next sections, there will be more details regarding each task.

Task 1 - Parking space detection

In order to have our final estimate on the parking space localization we have created a series of methods that calculate, process and modify the initial starting lines up to the final bounding boxes. However, before describing how we have done so, we would like to briefly describe all approaches that we first have tried before coming with our final solution.

Our idea since the beginning was to pre-process the image to make the each line more clear followed by detection; the best combination of mask that we have reach was to combine the image pre-processing (grayscale conversion, contrast enhancing, bilateral filter) with Otsu histogram thresholding, and by eliminate high level of saturation on the S channel on the HSV version of the image.
Some other mask that we have tested are adaptive thresholding, Canny with find contours, Laplacian filters, Gabor filter, CLAHE, ad-hoc adaptive bilateral filter with the division of the image in tiles, histogram equalization + gamma correction. Then from the best mask we have tried to use both the standard and probabilistic Hough Lines transforms; by using the `HoughLines` method, lines were detected, but it was difficult to identify their origin, while with `HoughLinesP` we could find the exact line position. Although the lines were successfully highlighted, we were unable to fine-tune the methods to detect weaker parking lines without increasing their number beyond control. We have also tried to do a template matching approach but while achieving some results the time complexity to achieve them was just too high so we dropped the idea.

After many hours of research, we ended up finding a simple and efficient method that gave us the lines highlighted, like the `HoughLinesP` method, without relying on any mask or parameter; the method we used is called Line Segment Detector (**LSD**). LSD is capable of finding lines in a grayscale image without any parameter and external pre-processing. The method is solid even in bad lighting conditions, however when image conditions are poor some lines are weaker with respect to other images. For this last reason, not all bounding boxes are detected because they are removed during our process of line filtering, that is what we are going to describe later.

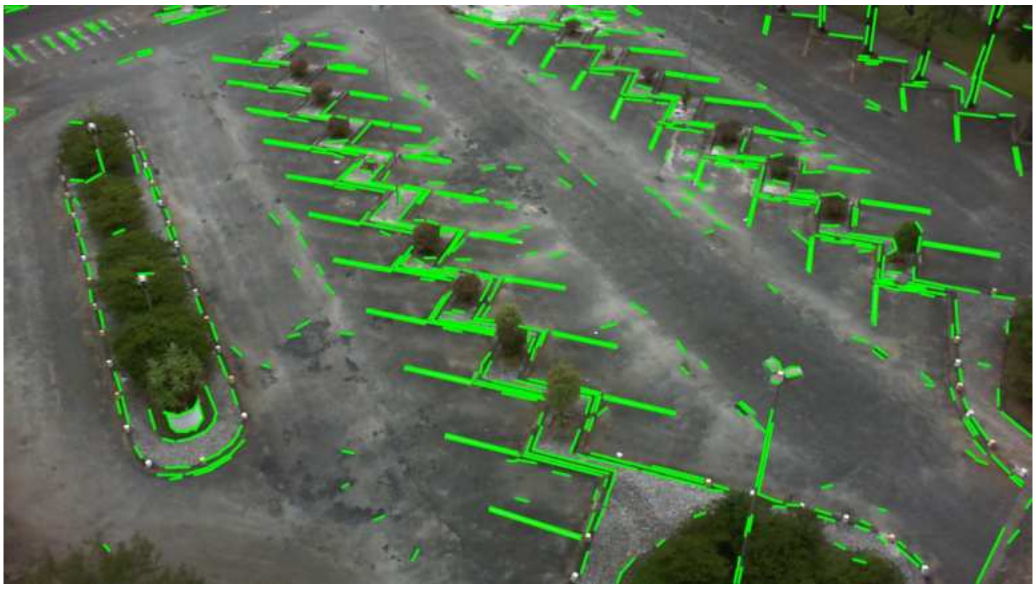


Figure 1: LSD lines

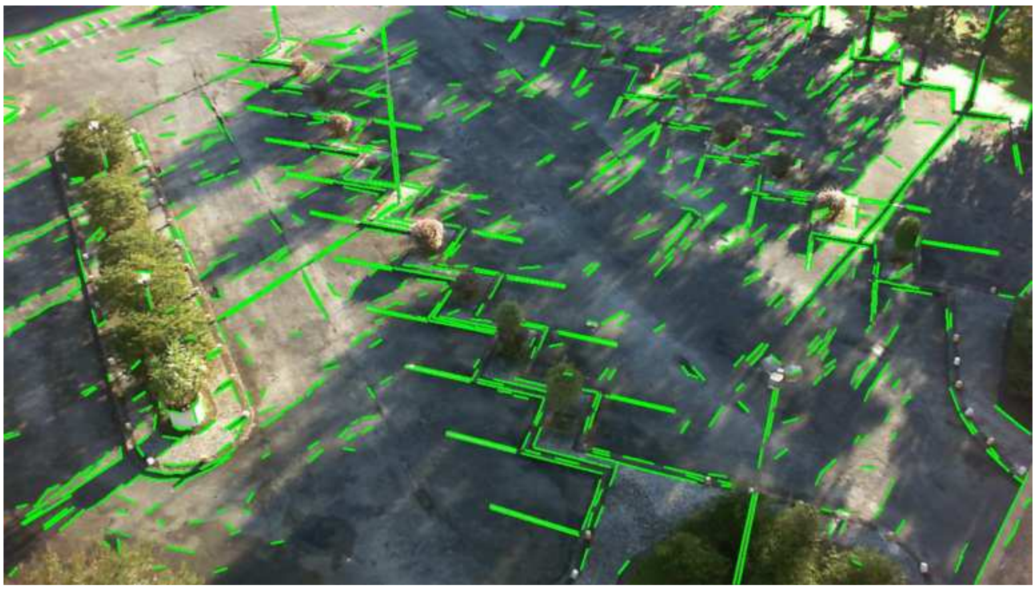


Figure 2: LSD lines

The founded lines are then **standardize** to our convention, meaning that the leftmost part of the line will always be my starting point. Next, they are passed through a **series of filters to keep only possible line candidates**. The filters applied are :

- Eliminate short lines.
- Remove lines with undesired angles.
- Discard lines in areas lacking sufficient white components.
- Exclude lines within the optional parking area.
- Eliminate lines that start or end near each other with similar angles, retaining the longest/strongest one.
- Remove perpendicular lines.

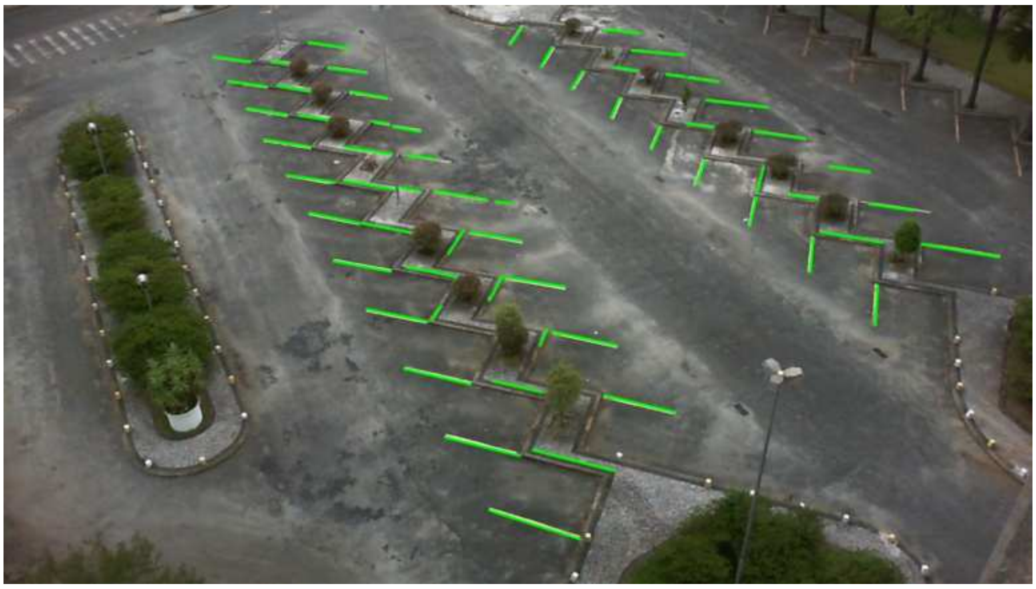


Figure 3: LSD lines after filtering

The next big step is to **find the right match** for each line in order to form a parking space. For this specific parking lot there are two main parking line angles, most lines have positive angles while few have a negative angles. From now on, we will describe the process for the lines with positive angles, however inside brackets there will be the version for the negative angle lines.
Given our parking lot topology, we came up with the idea that the distance between the start (end) of a line and the end (start) point of another line should be within a certain max distance. Therefore for each line based on its angle, we look at all the lines that have a similar angle, start-end (end-start) distance within range and not too distant in the y (x) axis.

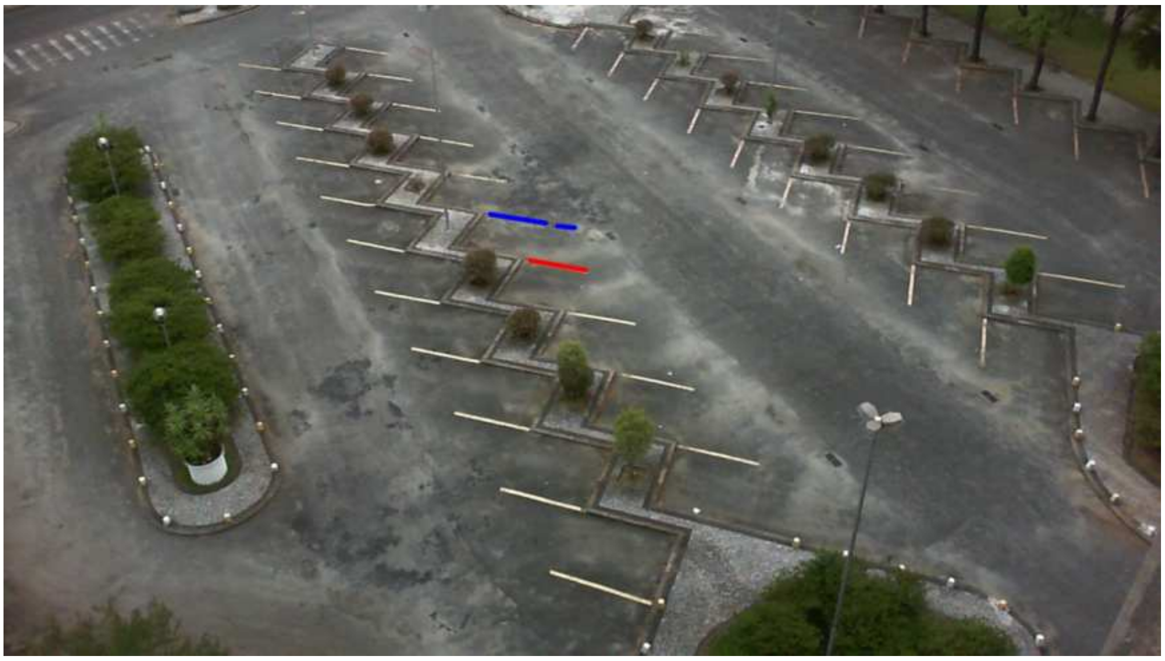


Figure 4: red line matched with the blue lines

For all possible matches, we keep only the one with the minimum start-end distance. By doing so, we are able to eliminate poor matches and even remove other useless lines.

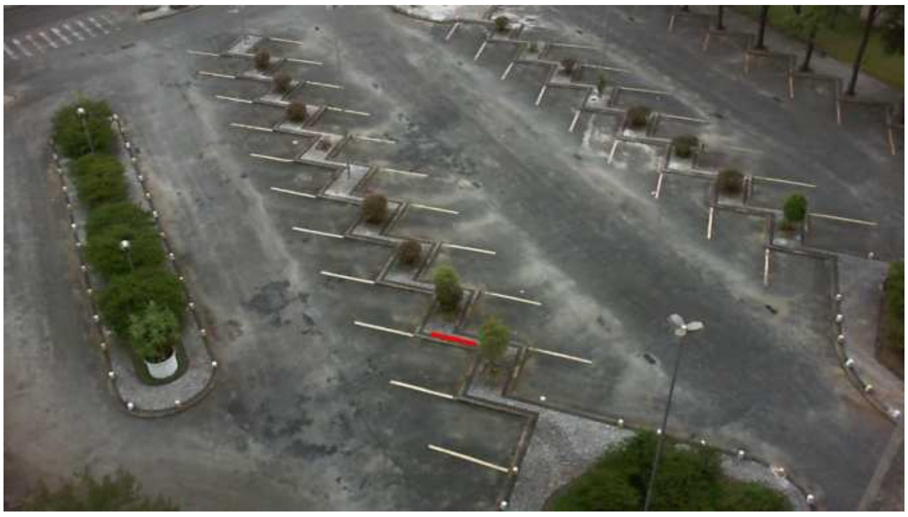


Figure 5: red line not matched with anything

From each pair of matched lines we take the maximum diagonal between the start-end or end-start distance and **build** `cv::RotatedRects` to first represent our bounding boxes.

One of the weakness of our method is that LSD is not capable of detecting some key "lines" that create some parking spaces; in particular there are 3 parking spaces that are made of one real parking line and raised concrete, which has the same color as the asphalt. In order to not hard code those parking spaces, we have created a method that find the location of the parking spaces above the critical ones and use the characteristics of them to build the rectangles for the critical parking spaces.

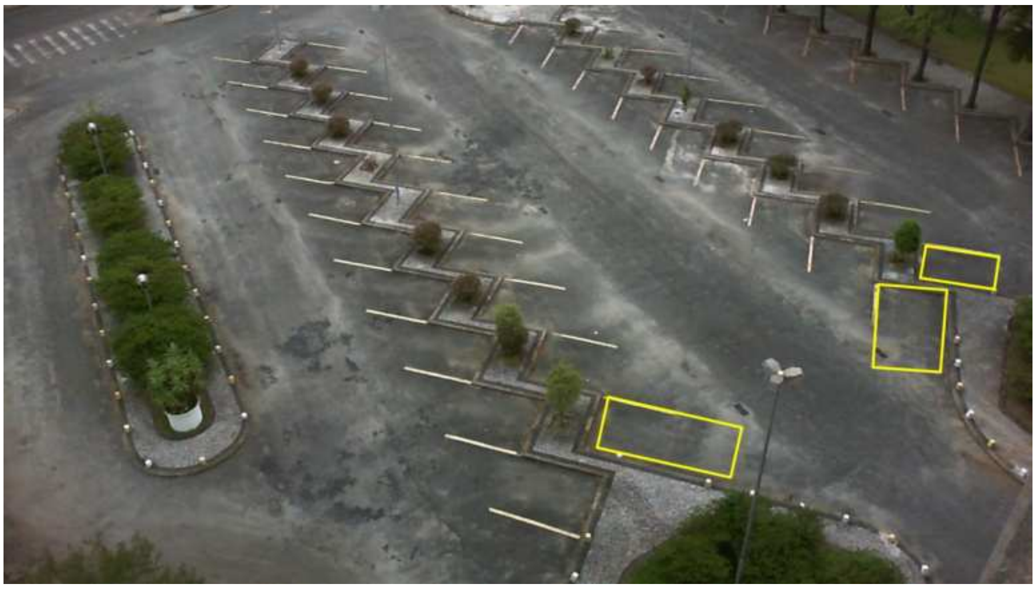


Figure 6: Deducted bounding boxes

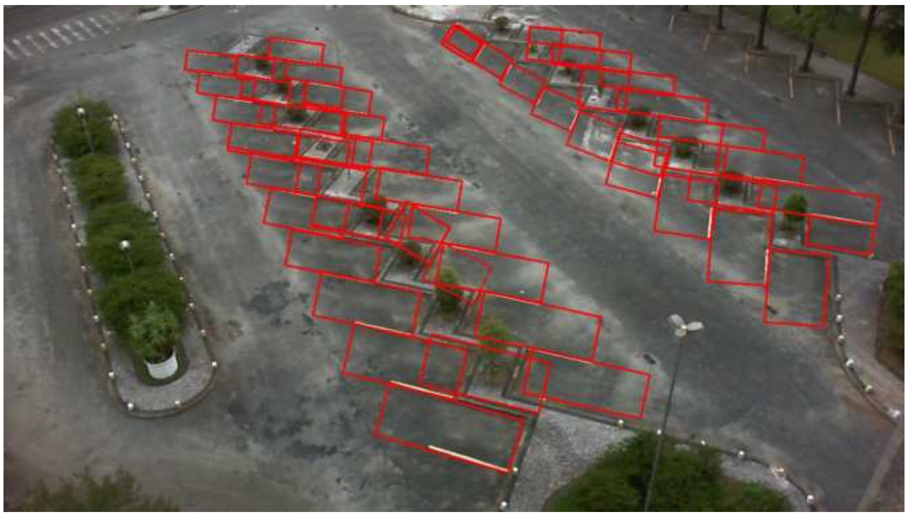


Figure 7: Early founded bounding boxes

After all the bounding boxes are build, we proceed to **remove outliers** in the generated rotated rects. The outliers that we seek and remove are :

- Rectangles between other rectangles : since a parking space has always one free side for a car to enter, we have thought of a method that checks if there is a rotated rect on the left and right.

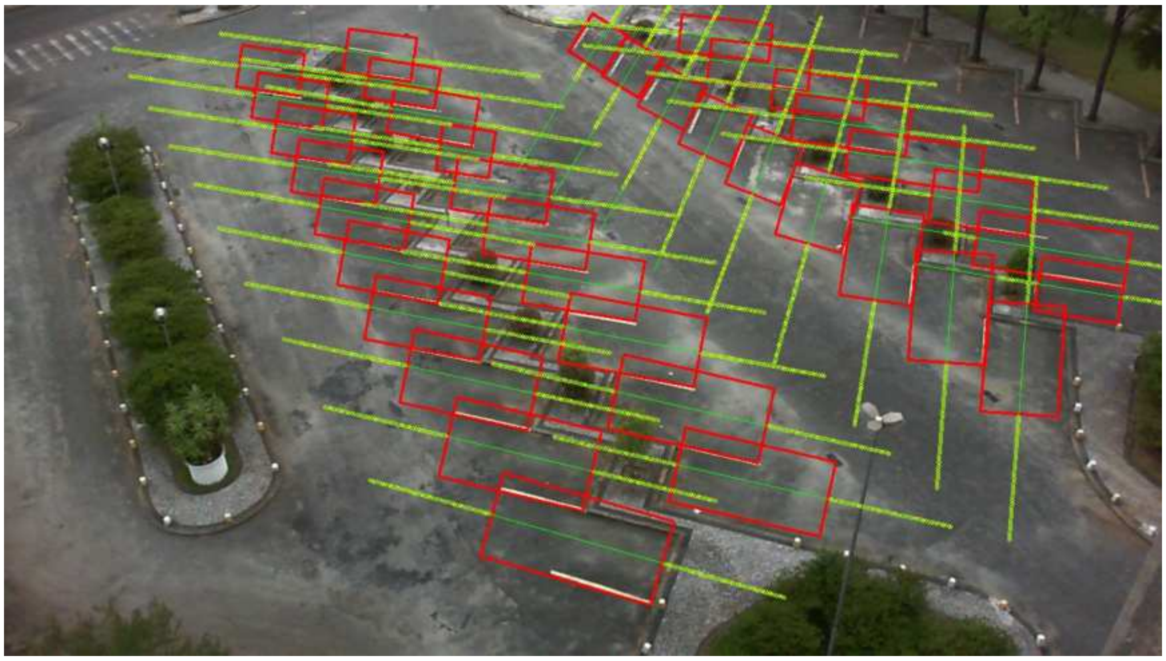


Figure 8: Deducted bounding boxes

- Rectangles with different aspect ratios.
- Overlapping rectangles : we keep the inner one.

As specified in the overview section, our program is designed to do all the aforementioned computations for each image inside the sequence0 directory.

After all the rotated rects are computed for all images, the program proceeds to return the final estimate of the bounding boxes by first **gathering the rotated rects that represent the same parking spaces** (look at the distance of the centers), then **make the average rotated rect** and finally adjust their position and dimension according to our specific parking lot (**perspective adjustment**). Despite this behaviour, our program is still capable of returning the detected parking spaces when only 1 image is present inside the sequence0 directory, therefore in line with the project requirements (just a design choice).

The perspective adjustment was not originally intended to be developed by us, however we were forced to do so because the position of the ground truth made the IoU few decimals below the 0.5 threshold. In the ground truth the bounding boxes are aligned not with the parking space but with the common position of the cars, so slightly offset from the position of the parking space. What we do is essentially move the center based on the free side of the parking space and, dynamically, the lower the y coordinates is, the more we increase the height (width).

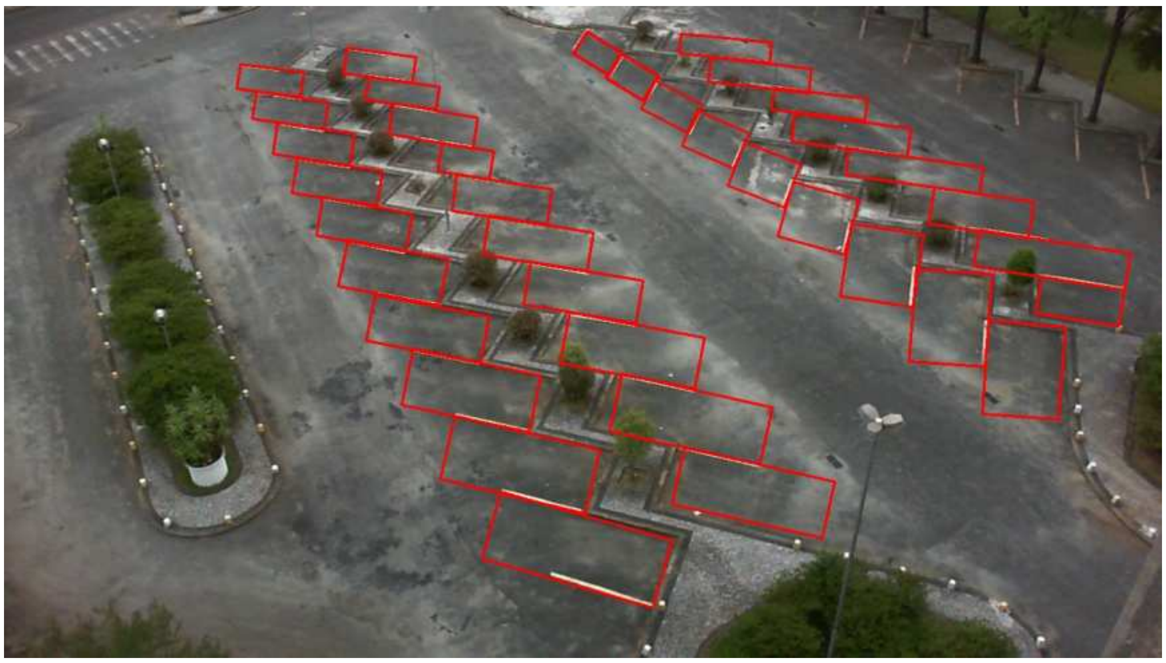


Figure 9: Average bounding boxes BEFORE perspective adjustment

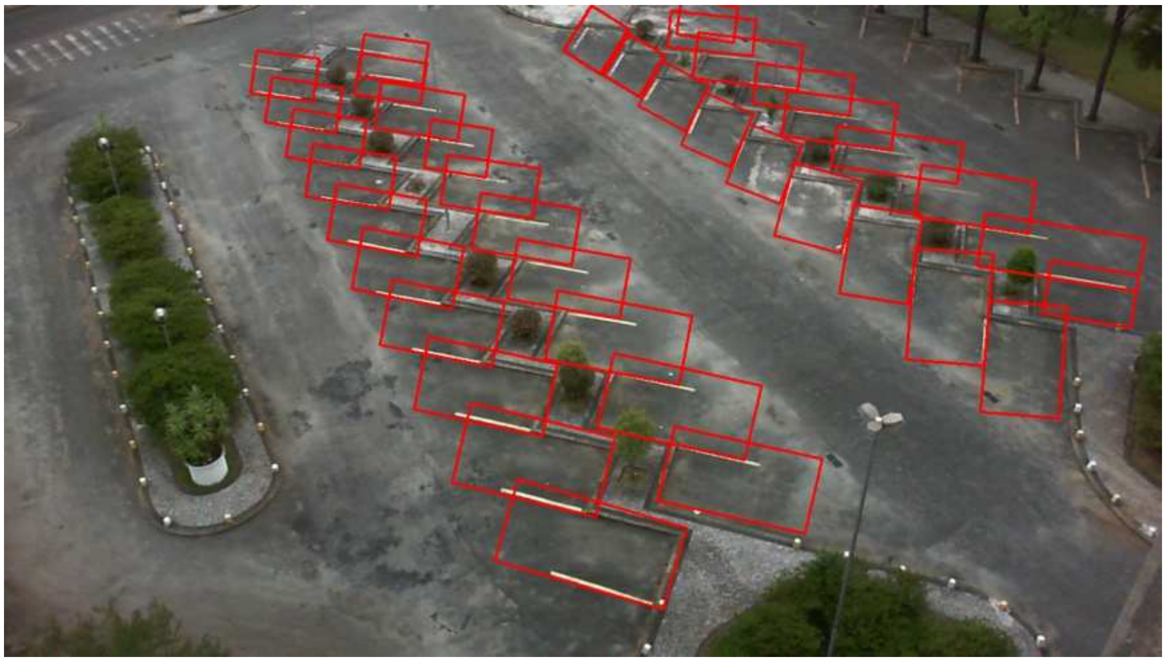


Figure 10: Average bounding boxes AFTER perspective adjustment

Lastly, the **enumeration of the bounding boxes**. To stay consistent with the paper enumeration, we first take the bounding box with the bottom right corner that has higher y and this will be the first of bounding box of the row; label and remove it from the vector. Then the connected or close bounding boxes are labeled and removed from the vector, until there are no more. Then the cycle is repeated for each row until we have 37 bounding boxes, that is the maximum of number of parking spaces we are allowed to detect. This method is not perfect, because if for instance one bounding box is missing then the whole labeling is shifted and non consistent with the ground truth. For this reason, we have tried our best to not rely on the parking space number for later use (not used for metrics, but used in the 2D map).

Is our program able to detect parking spaces in other parking lots? Except for the generation of the critical bounding boxes, perspective adjustment and labeling, LSD and the logic behind our filters are solid and with minor adjustment to thresholds and variables our program could handle other parking lots.

Task 2 - Car classification

Once the final bounding boxes are calculated and the segmentation method has done computing the mask, the car classification program can begin the processing. The class that handles this task is `ParkingLotStatus`.
In order to detect cars, for each bounding box we extract its region from the image then a cascade of controls check the presence of a car; if just one control is passed then we have a car, otherwise no car. The cascade of controls checks:

1. White pixel percentage inside the bounding box for the binary segmentation mask.
2. White pixel percentage inside the bounding box for saturation thresholding on the HSV version of the image (colored cars).
3. White pixel percentage inside the bounding box for thresholding darker areas in the RGB image (black cars).
4. Number of features by applying the filtering stage of mean shift segmentation and then detect features with SIFT.

The order of the controls is not casual, but well thought. The first control, uses a mask coming from the segmentation task because it is pretty precise to detect cars with different illumination and position changes; therefore if we see enough white pixel we are sure that it will not be a false positive. However this precision comes at the cost that some cars might not be detected; for this reason we also used other three controls for checking the presence of different car color schemes, in particular colored and black cars.
The final control is to essentially determine if the parking space is empty. The idea is the following; we first apply the filtering stage of mean shift segmentation in order to make the asphalt homogeneous and enhance differences made by shadows and colors, then we simply calculate the number of features detected by the SIFT algorithm; in this way no features are detected on the asphalt and we have seen that 35 features are enough to consider the parking space occupied.

Task 3 - Car segmentation

For this task we have spent a lot of hours for the research phase. Most of the recent literature moved to use deep learning based approaches making it very difficult to find ideas using traditional computer vision techniques. Before reaching the current level different approaches were analyzed; traditional methods like blob detection, Canny, feature matching, clustering and superpixels brought no or very poor results. The next step was the exploration of machine learning methods; the first idea explored was detecting an approximate bounding boxes around cars and then apply the GrabCut algorithm to segment them. An attempt of detecting cars was done with a Harr cascade, but the model included in OpenCV gave no results; manually training one model requires to manually annotate the images, a tedious and long process. A second attempt with machine learning was done with Support Vector Machines trained on PKLot and KITTI datasets but with no significant results.

The first acceptable results were obtained using a SIFT matcher (not used at the end), which matched the average of the sequence0 images to the one being analyzed, saving in a mask only the positions with no good matches (where there was a change). This created a decent mask that was then merged with a background subtractor; the merge was then used as a mask for GrabCut, giving a good first result in segmentation. However, this was only the first attempt until we discovered a better method.

We found out that is possible to use the MOG2 background subtractor even on static images, setting the learning rate to automatic during the training and to zero when applyed on the images after the training. This machine learning method trained once (at the beginning of the program execution) with images from the PKLot dataset and with the automatic learning rate adjustment gave impressing results. The MOG2 in order to produce good results doesn't need an huge amount of images, although using more images, instead of the only images from sequence0, lead to a more robust final model. For this reason, in order to increase the performance, we have added all the available images from PKLot dataset regarding this particular view and parking lot. We started analyzing the parameters that can be tuned on the MOG2; ending up putting them to:

- `history` = 500;
- `VarThreshold` = 14;
- `detectShadows` = true.

Others parameters like `backgroundRatio` or `nmixtures` were not useful and didn't help us to improve the results. We analyzed the results by either keeping the shadows or not, noticing that by detecting shadows and forcing those values from 127 (probable shadow or background) to 0 (background), the results were better than with the `detectShadows` parameter to off.

MOG2 is made to work with BGR, but we figured out that it was not capable to learn all the illumination changes, giving results that were not amazing in all the sequences; hence we preferred to combine (with a bitwise and) two MOG2 model trained separately with BGR and Lab version of the dataset. In the following, we analyze the whole process by using `2013-04-12_14_45_09_segmentation.png` from sequence5 as an example.

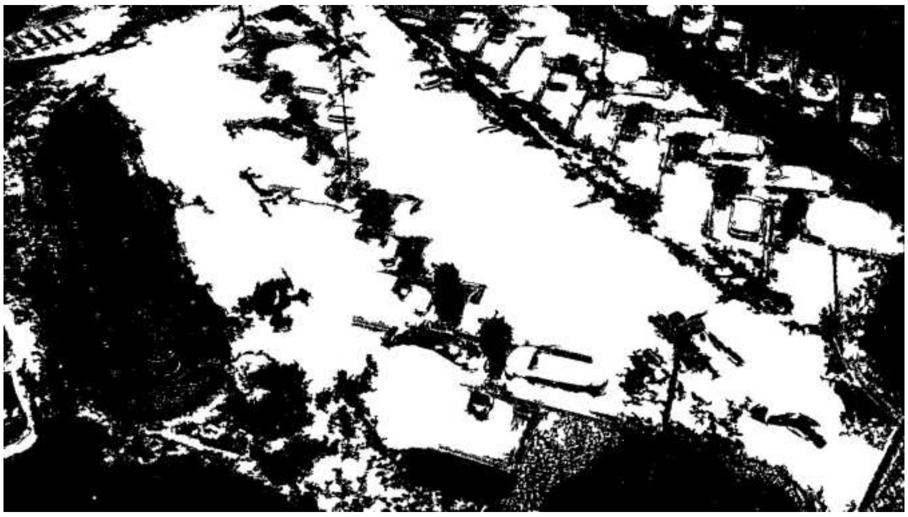


Figure 11: Result of MOG2 trained on LAB dataset

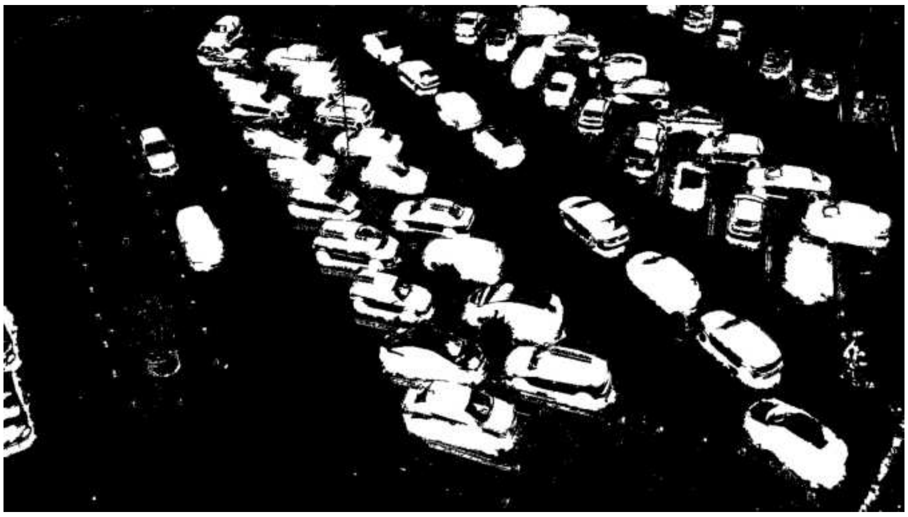


Figure 12: Result of MOG2 trained on BGR dataset

To make the masks cleaner, we thresholded the HSV versions of the image under analysis to remove the parts where the saturation was greater than 75 and thresholded the Lab version to detect the red cars (i.e. pixels with channel "a" value greater than 140).
By doing so, we were able to create two masks:

- One to remove elements that lowered the quality of segmentation while also inevitably removing red cars.
- One to reintegrate the removed red cars.

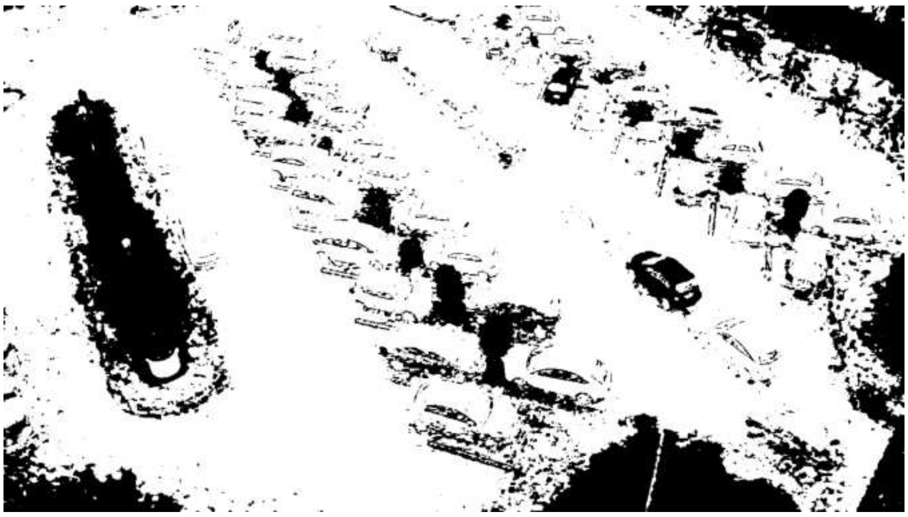


Figure 13: Mask of saturation that will be removed(black pixels)

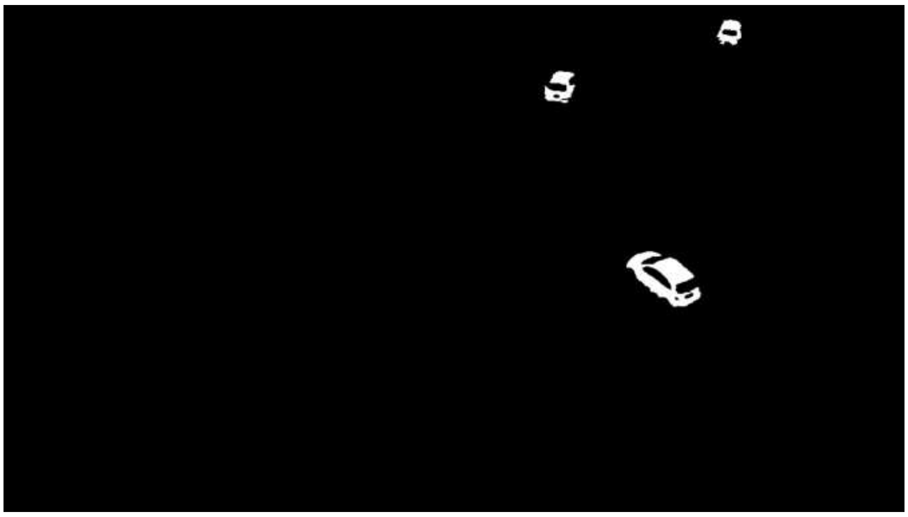


Figure 14: Mask containing red cars that will be reintegrated

This two masks are then applyed to cleaned results of mog2Lab and mog2BGR, obtaining after some cleaning and morphological operations:

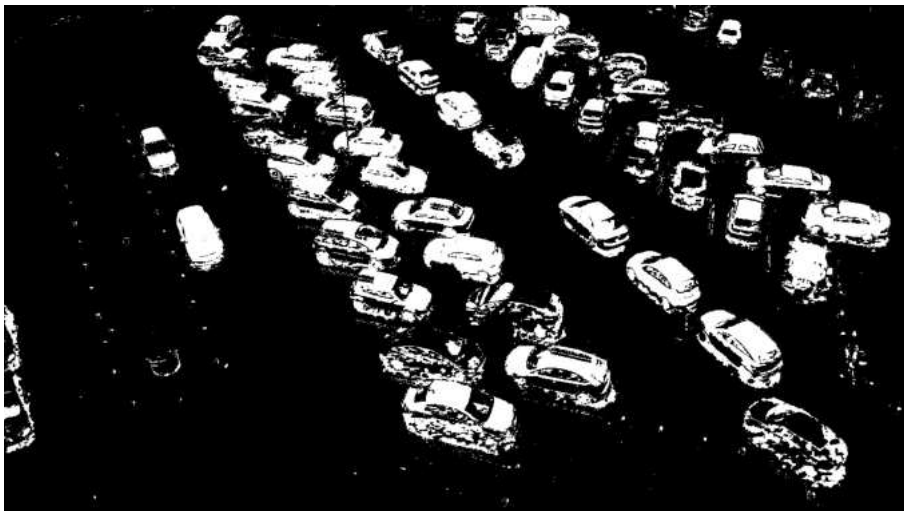


Figure 15: Mask containing Mog2BGR after removing saturation and reintegrating red cars

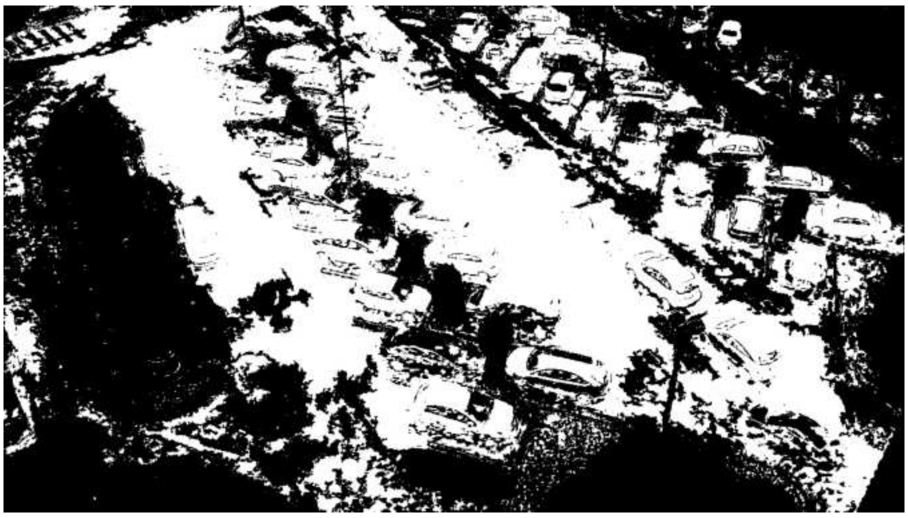


Figure 16: Mask containing Mog2LAB after removing saturation and reintegrating red cars

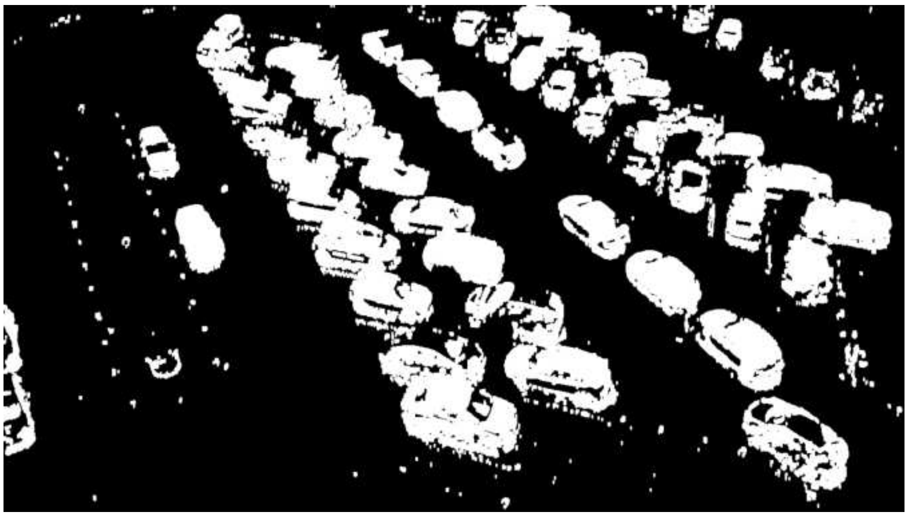


Figure 17: Mog2BGR Final Mask after morphological operations

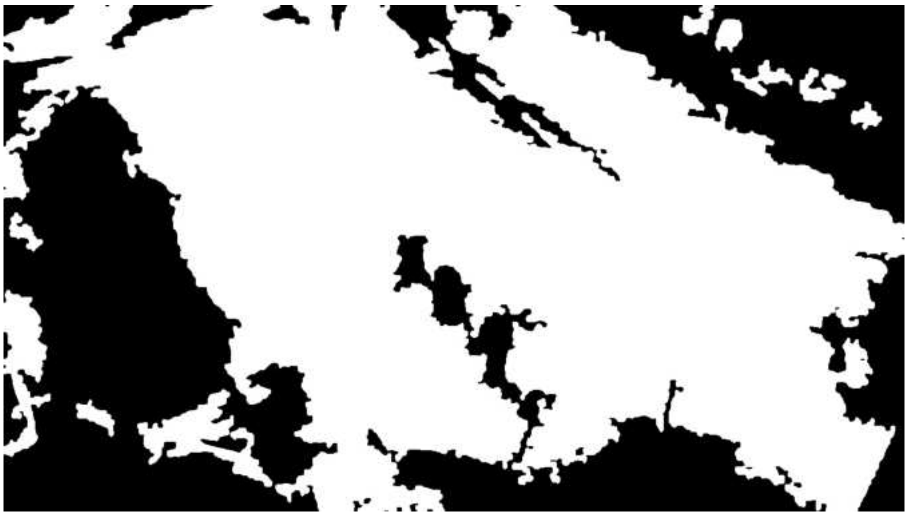


Figure 18: Mog2LAB Final Mask after morphological operations

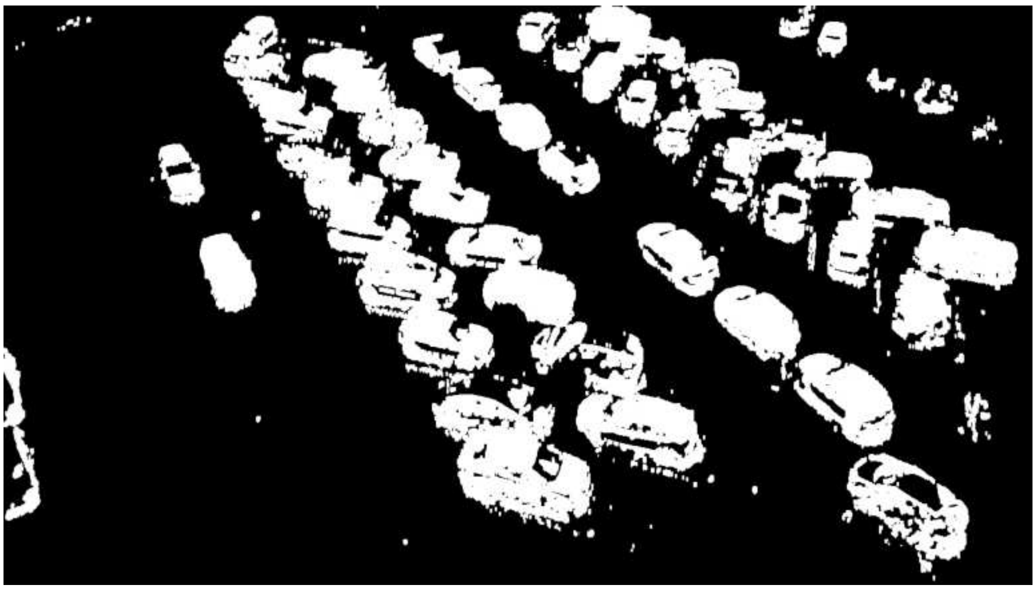


Figure 19: Result obtained combining the two masks

By combining them together using a bitwise and, we obtain more precision in some images; in some images there are people and noise that are not present in both masks, therefore when combined we obtain a complete noise and people removal.

Another useful idea that allowed us to increase the robustness to illumination changes was to perform a background subtraction by calculating the similarity between the image under analysis and those included in the training dataset, choosing for the background subtraction the one with the lowest absDiff. This background subtraction mask was combined with a bitwise and with the previous result, plus there is a reintegration with the mask of red machines (in some cases the mask of the background subtraction did not contain them and therefore with the AND operation they were lost).
After some final filling and cleaning procedures we obtained the final mask:

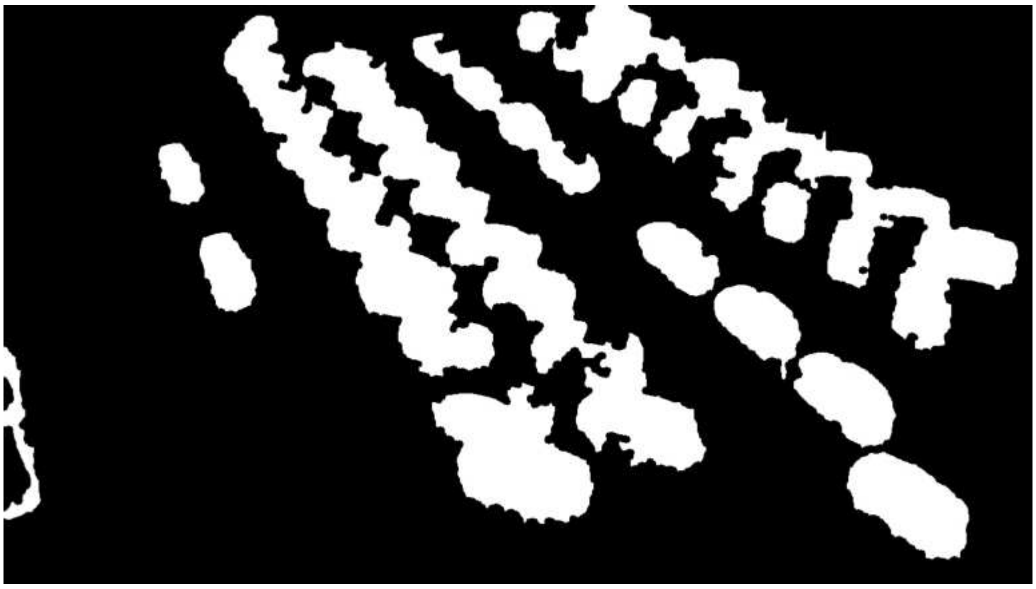


Figure 20: Final Mask

Segmentation is pretty robust and can be considered sufficiently generalised since it would be easy to adapt it to a different parking lot if training images are available.

Once the final binary mask is obtained, the bounding boxes of the parking spaces founded in task 1 are fetched and used to create another mask with the ROIs of the parking slots.

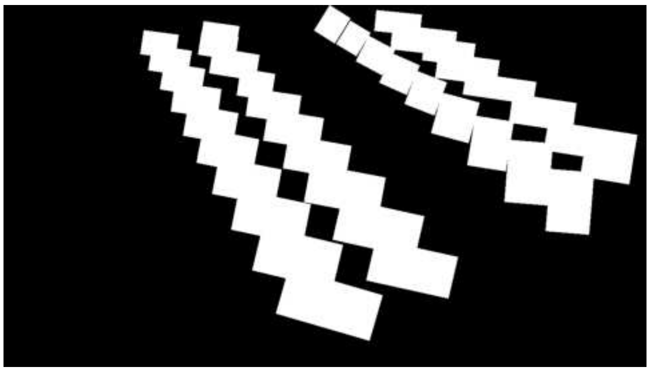


Figure 21: ROI of only parking lots

In order to determine if a segmented car is within a parking space, we look if the each connected components of the final mask intersect with the ROIs of the parking spaces. If there is a match then we know is inside a parking space (red color for all pixels of the connected component), otherwise outside (green color).
We also noticed that it was very difficult to segment correctly the cars in the optional area, so according to project specifications we used a black ROI to exclude it both from the segmentation mask and the ground truth used to calculate the metrics; after this the score raised significantly.

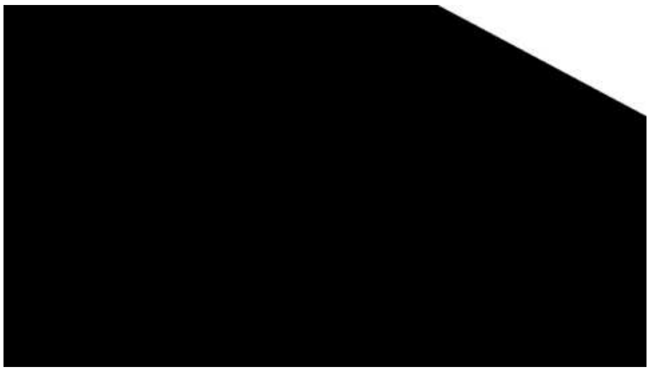


Figure 22: ROI of the optional area

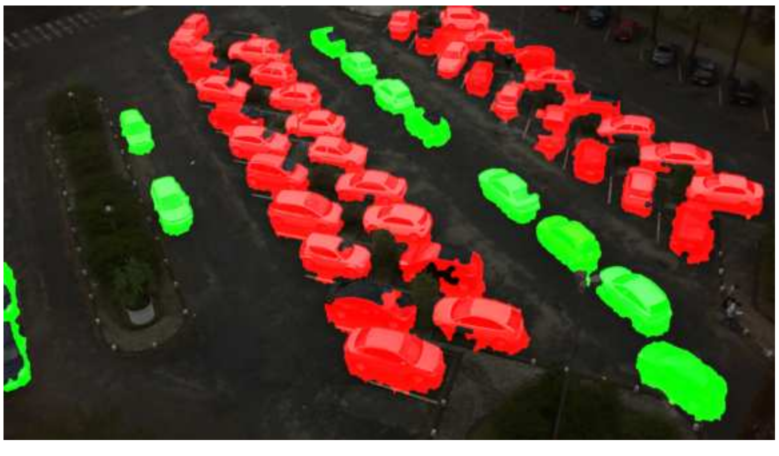


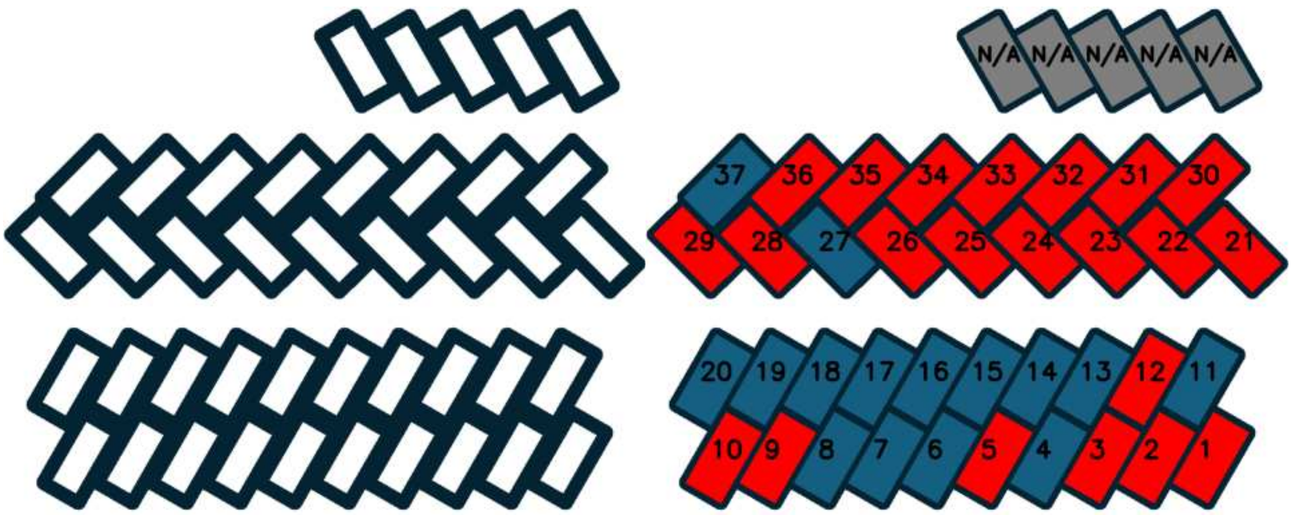
Figure 23: Final Segmentation

Task 4 - 2D Map

The map is created only using pixel operations and the parameters are tailored for the topography of the considered parking lot. During the construction of the image that represent the topography of the parking lot parameters like size and number of the parking slots, offsets, angles, single or double lines are passed to a dedicated function. With this kind of construction the map can be adapted to a different plan by changing parameters.

This method is exclusively graphical and does not perform any computation. Once the map has been drawn, it is required a vector of bounding box indexes of the busy parking spaces; the corresponding parking spaces will be red colored, while the other empty parking spaces will be blue.
Since the filling is index dependent and in the event of missing parking spaces, the labels with the number of the corresponding parking space are printed on the map making it readable in case of problems with the parking space detection. The ignored parking spaces are blocked with a grey filling and a ***N/A*** label, indicating they are not considered in this particular case.

Once the map has been fully drawn, it gets scaled and applied as an overlay on a fresh copy of the parking image, making it as similar as possible to the one presented in the handout.



(a) Initial skeleton (b) Filled map with labels

Figure 24: Map creation

Metrics

In order to compute the metrics, at computing time, we don't consider the optional parking row for both parking space detection and segmentation.

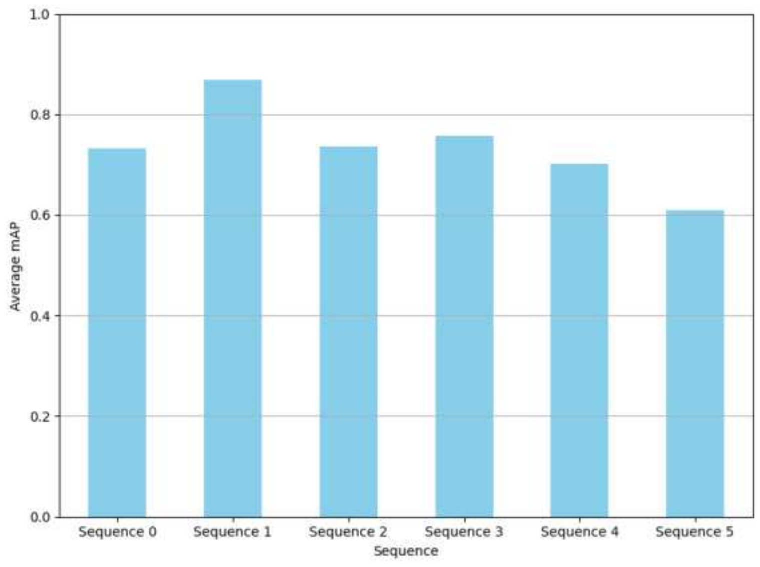


Figure 25: average mAP on all sequences

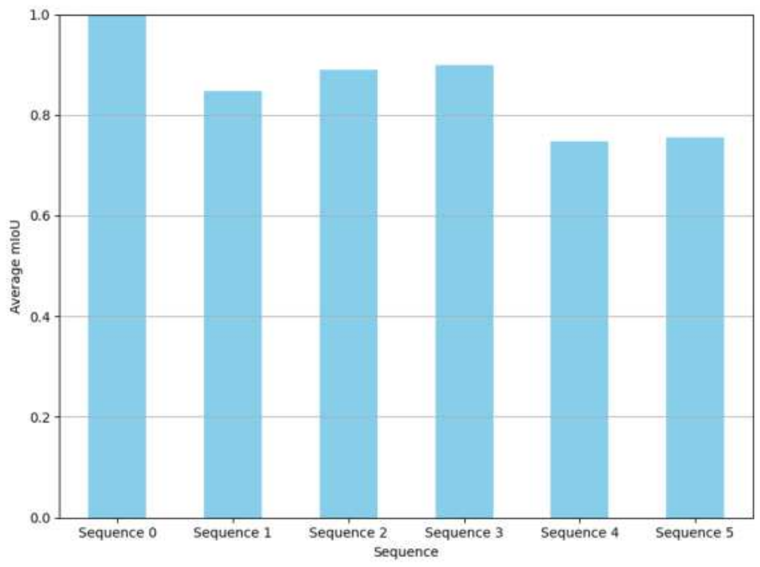


Figure 26: average mIoU on all sequences

Output images

As said before our program works with the final estimate of the bounding boxes. In order to see how our program behave with just one image it is necessary to have just one image inside the sequence0 directory. For the next session, the metrics values are calculated by not considering the optional parking row and by using the final merge of the bounding boxes. To have a better look at each image, please check the **report_images** folder.
Here are the detected bounding boxes for each sequence0 frame:

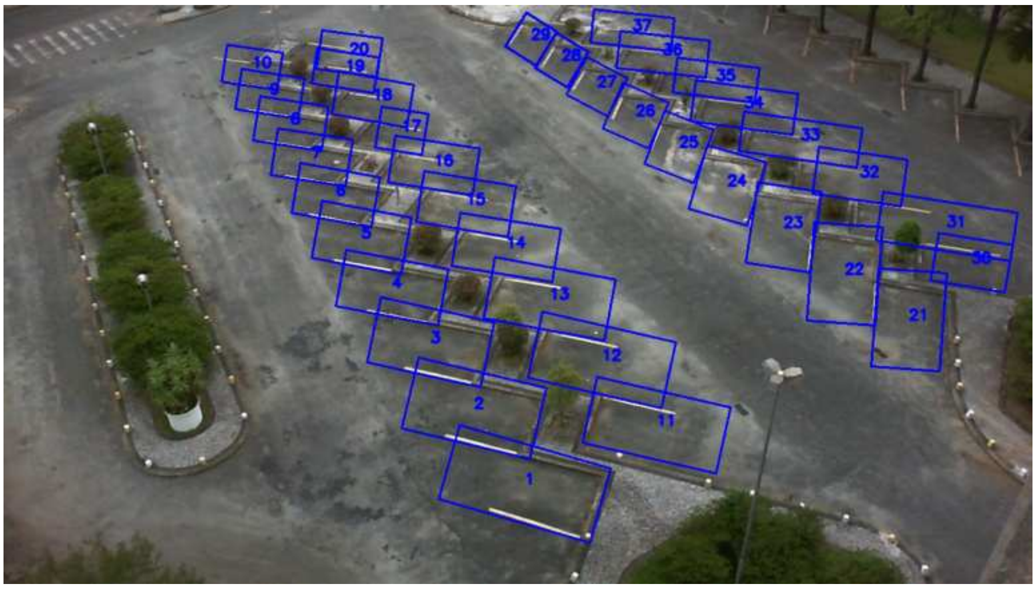


Figure 27: Detected bounding boxes on 2013-02-24_10_05_04.jpg

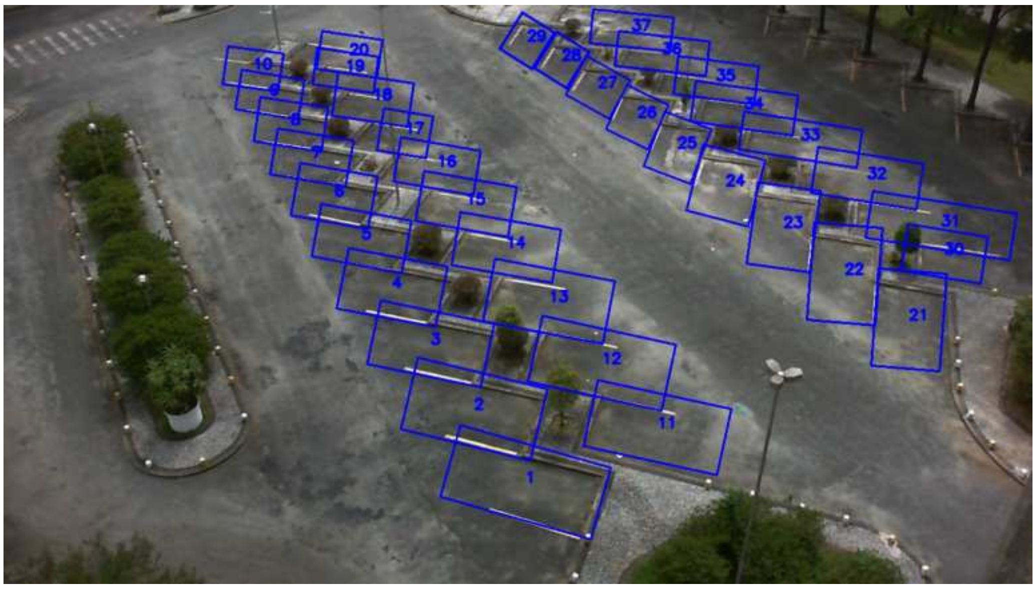


Figure 28: Detected bounding boxes on 2013-02-24_10_35_04.jpg

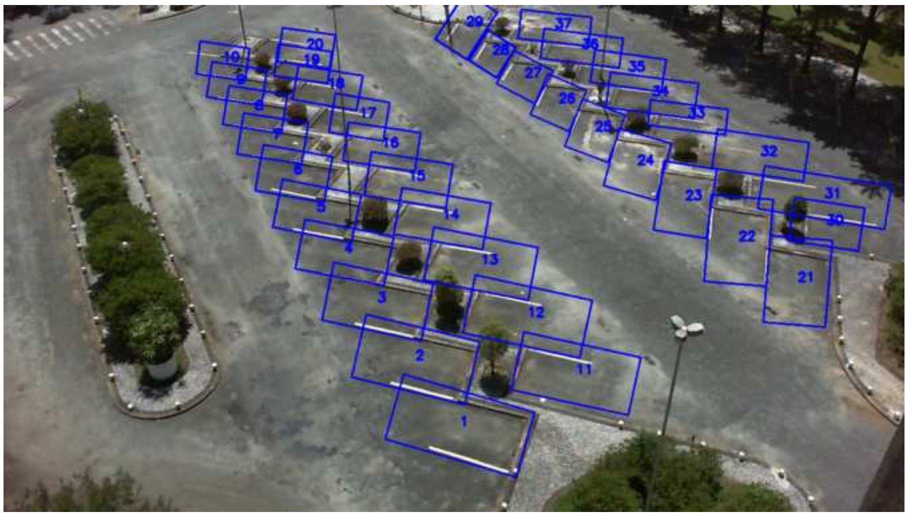


Figure 29: Detected bounding boxes on 2013-02-24_11_30_05.jpg

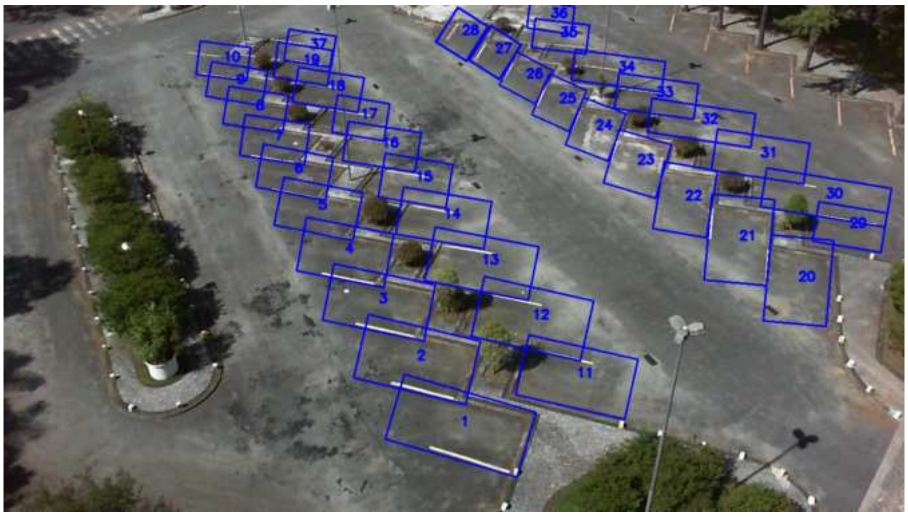


Figure 30: Detected bounding boxes on 2013-02-24_15_10_09.jpg

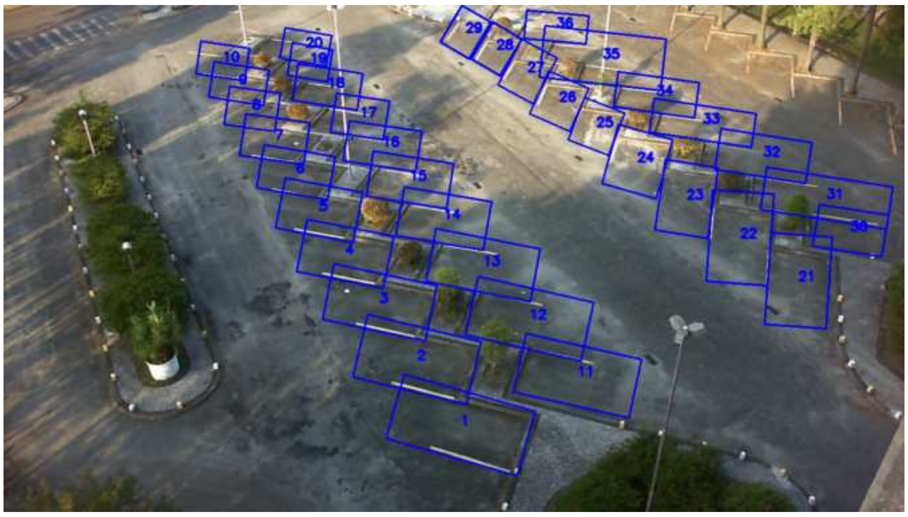
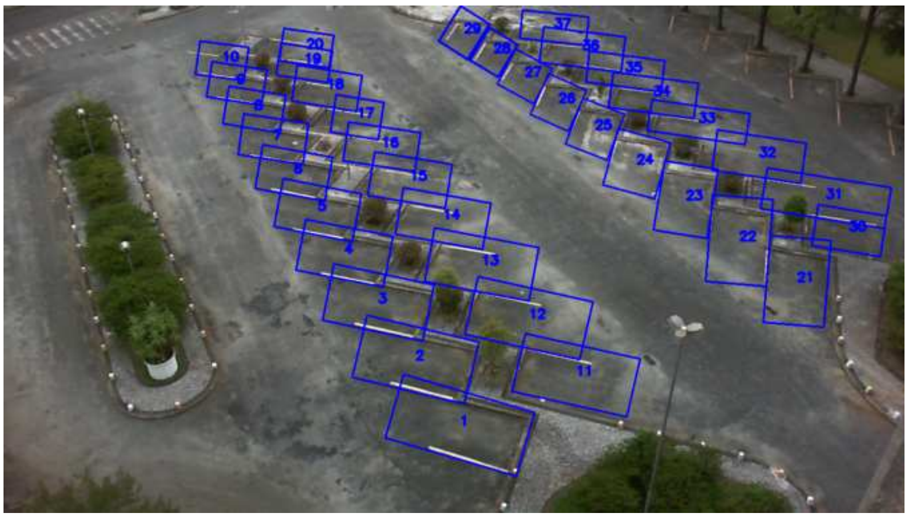
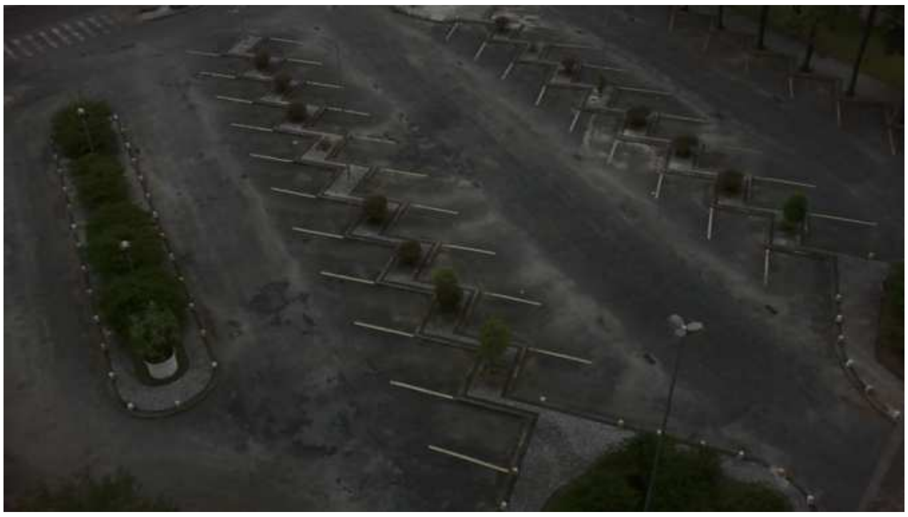


Figure 31: Detected bounding boxes on 2013-02-24_17_55_12.jpg

Sequence0



mAP = 0.772



mIoU = 1

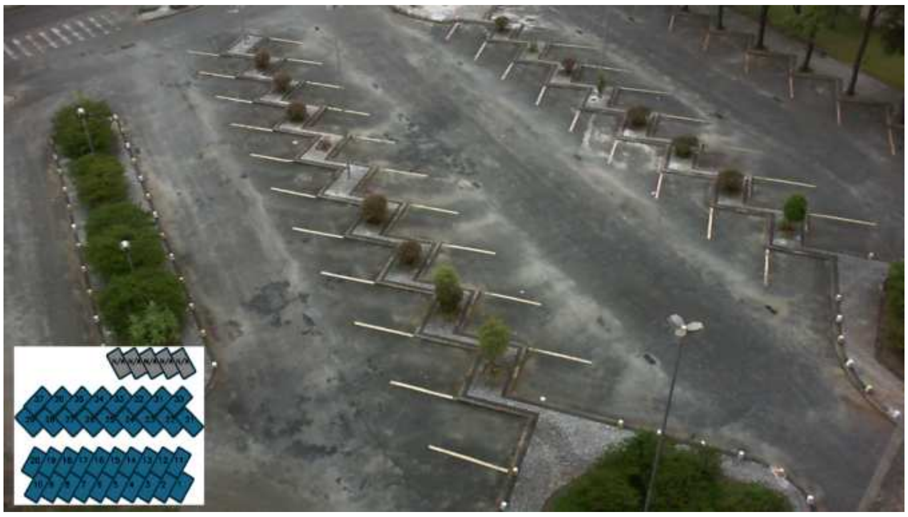
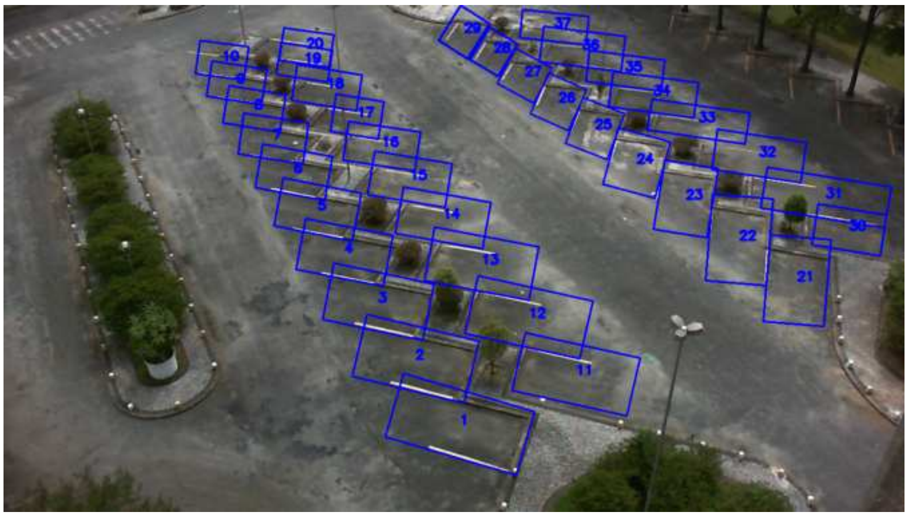
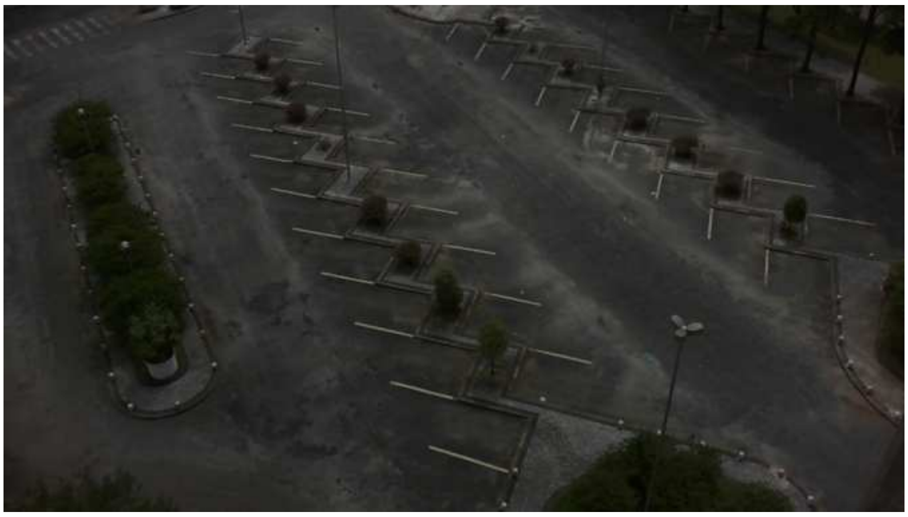


Figure 32: 2013-02-24_10_05_04.jpg



mAP = 0.772



mIoU = 1

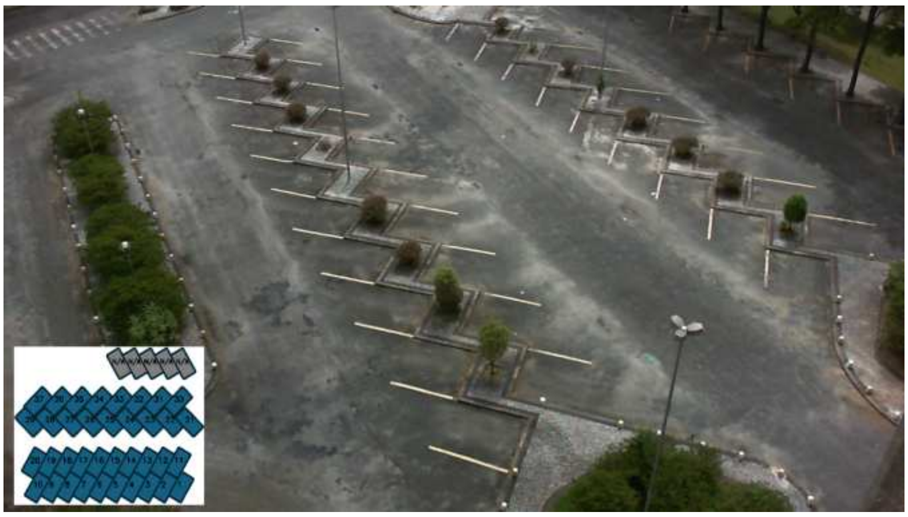
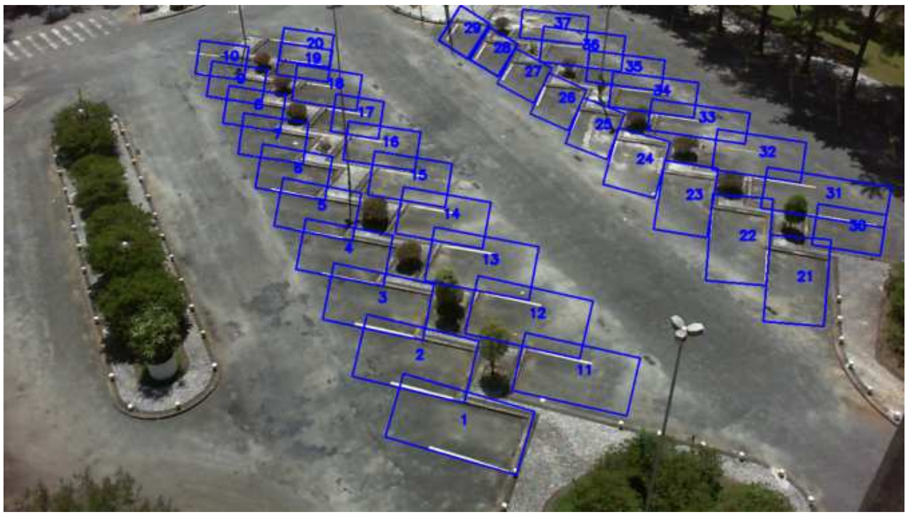


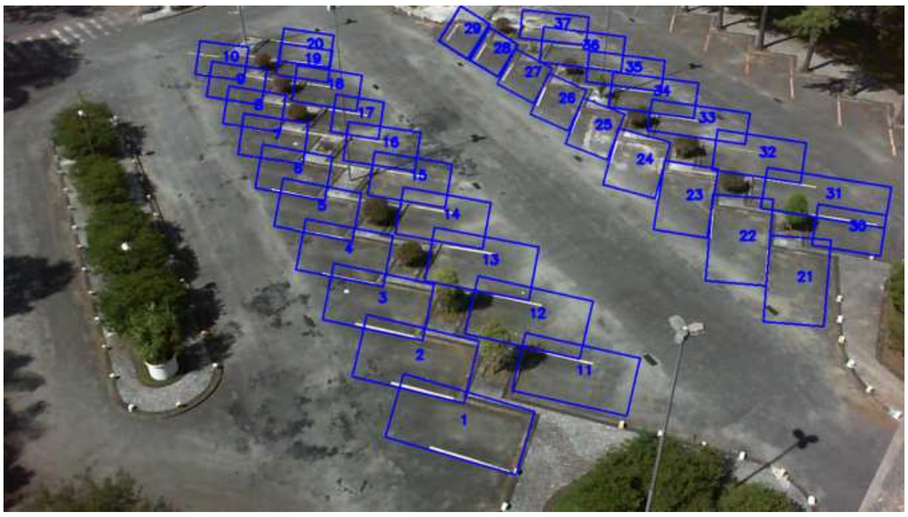
Figure 33: 2013-02-24_10_35_04.jpg



mAP = 0.772

mIoU = 1

Figure 34: 2013-02-24_11_30_05.jpg



mAP = 0.772

mIoU = 1

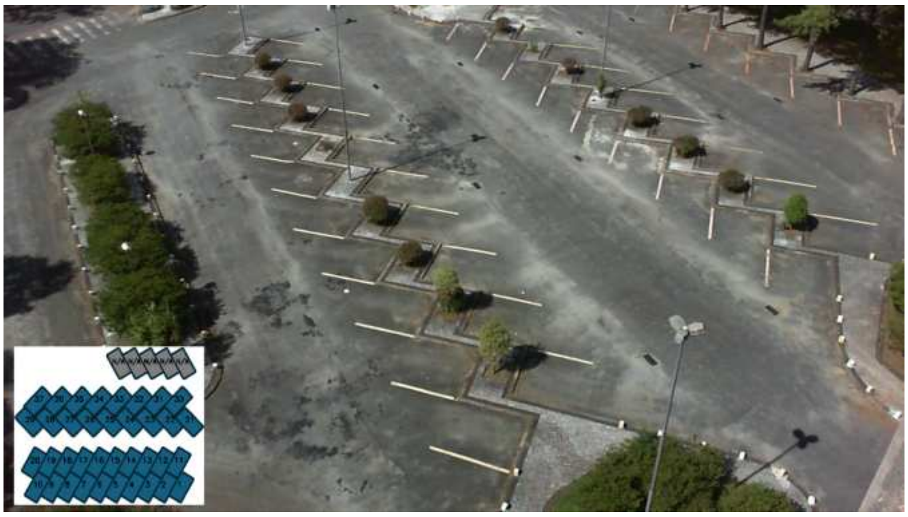
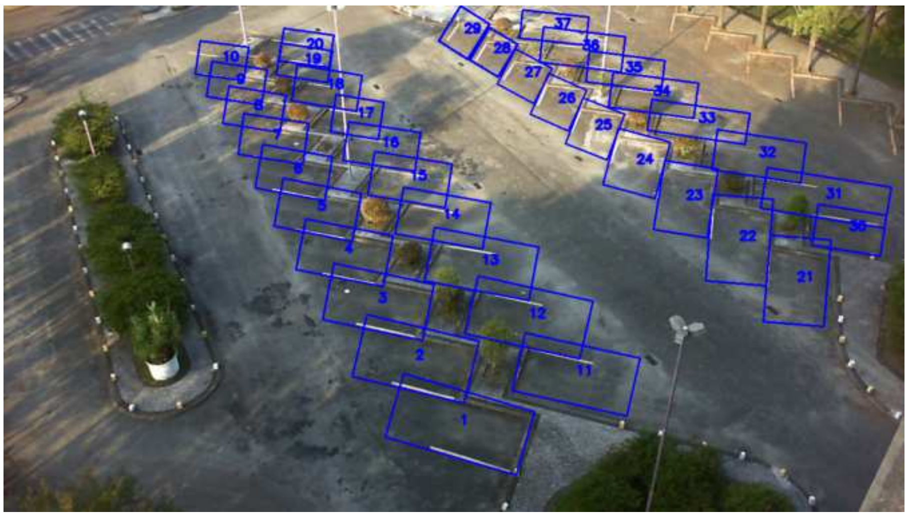
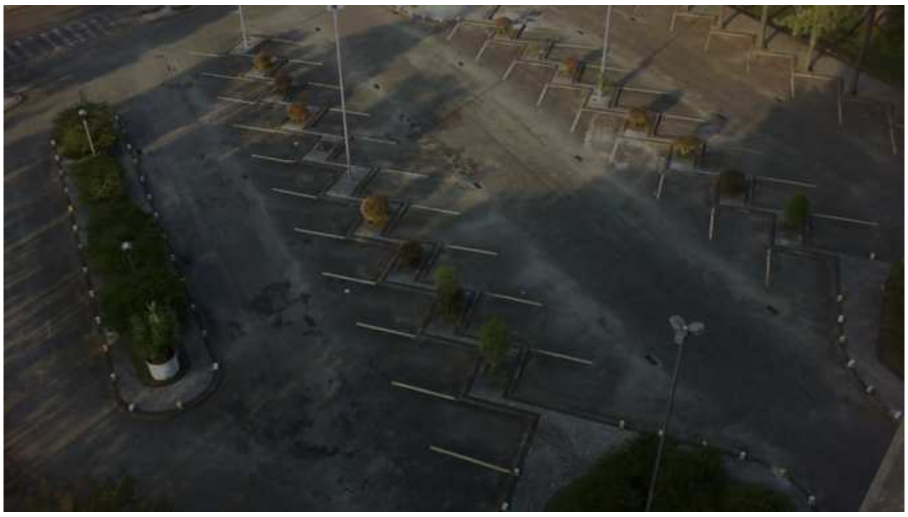


Figure 35: 2013-02-24_15_10_09.jpg



mAP = 0.772



mIoU = 1

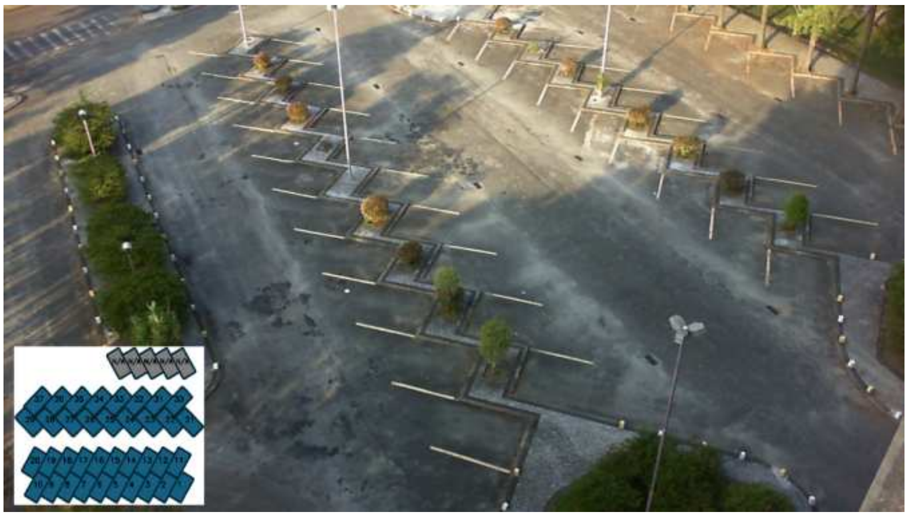
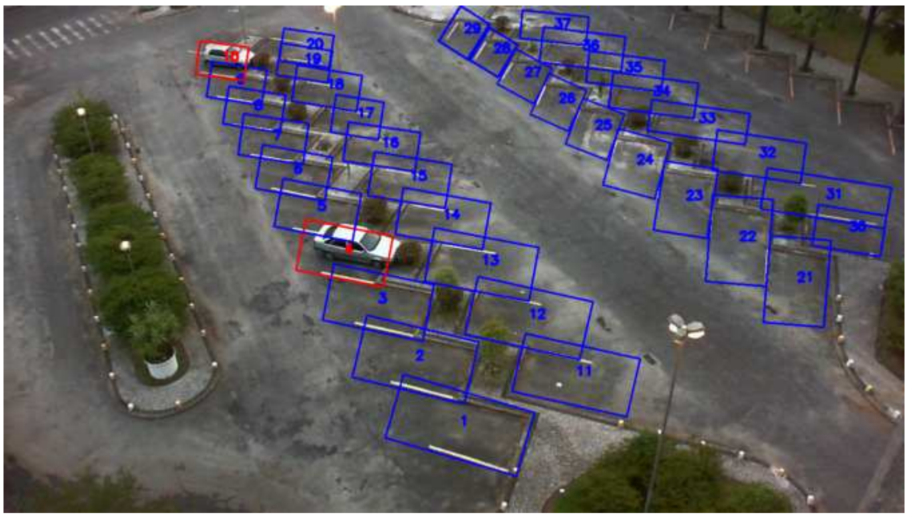
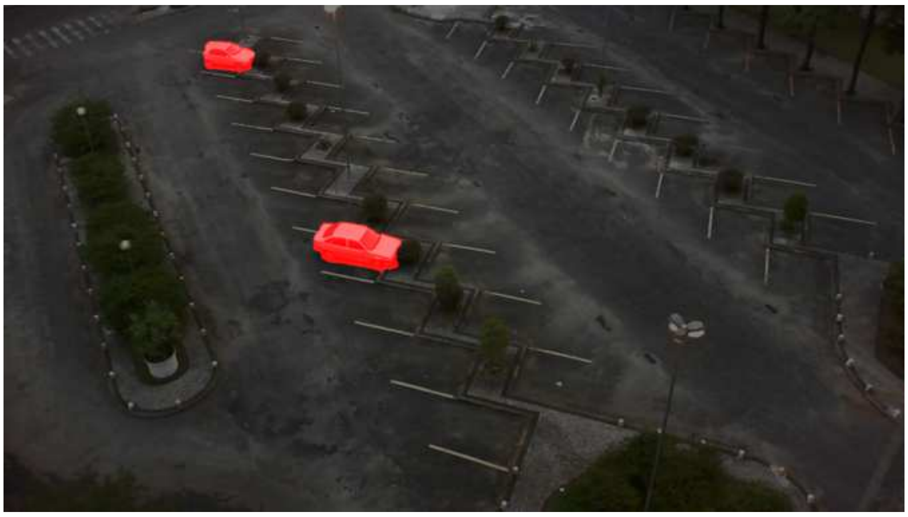


Figure 36: 2013-02-24_17_55_12.jpg

Sequence1


mAP = 0.883


mIoU = 0.918

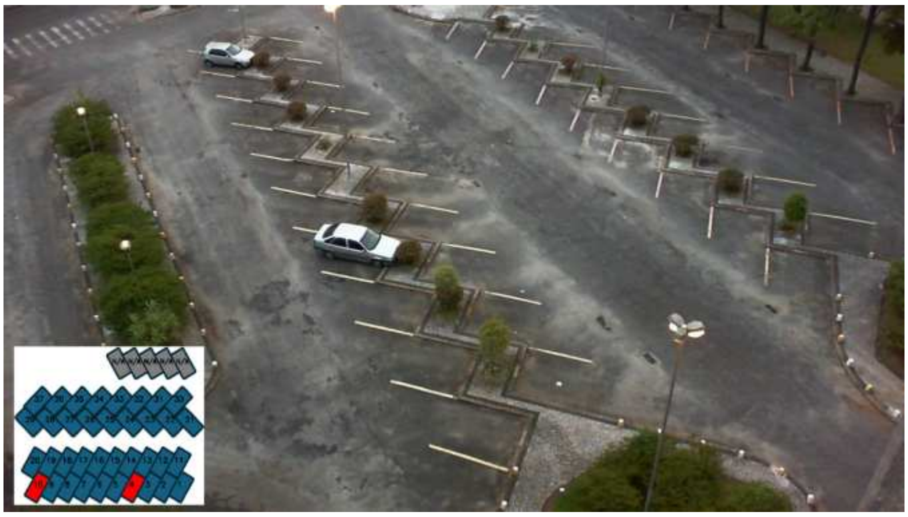
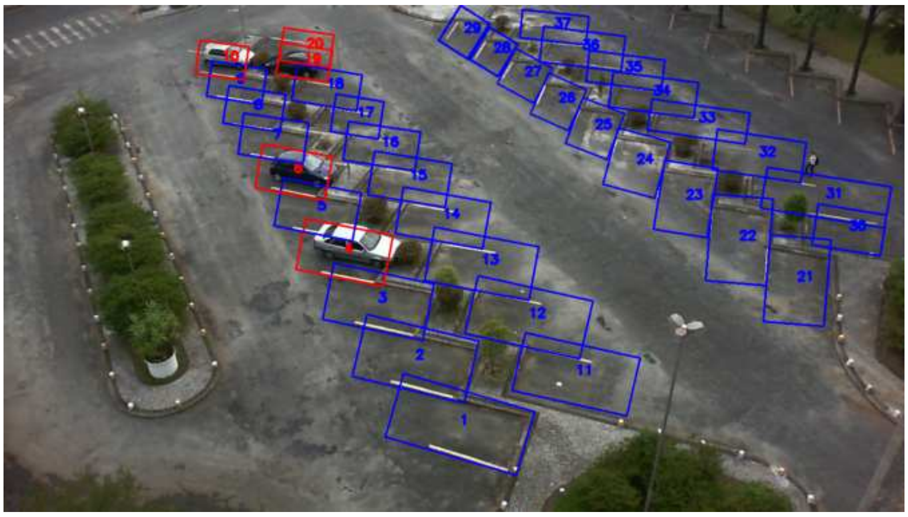
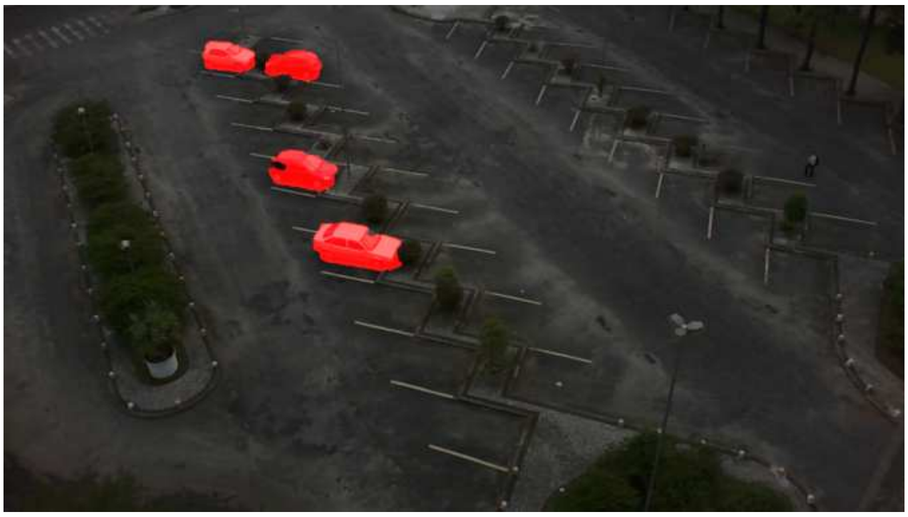


Figure 37: 2013-02-22_06_25_00.jpg


mAP = 0.888


mIoU = 0.903

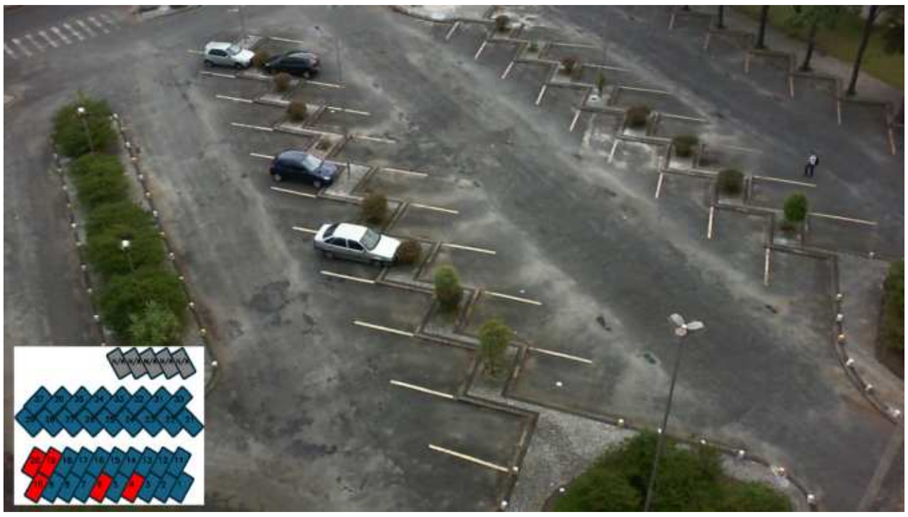
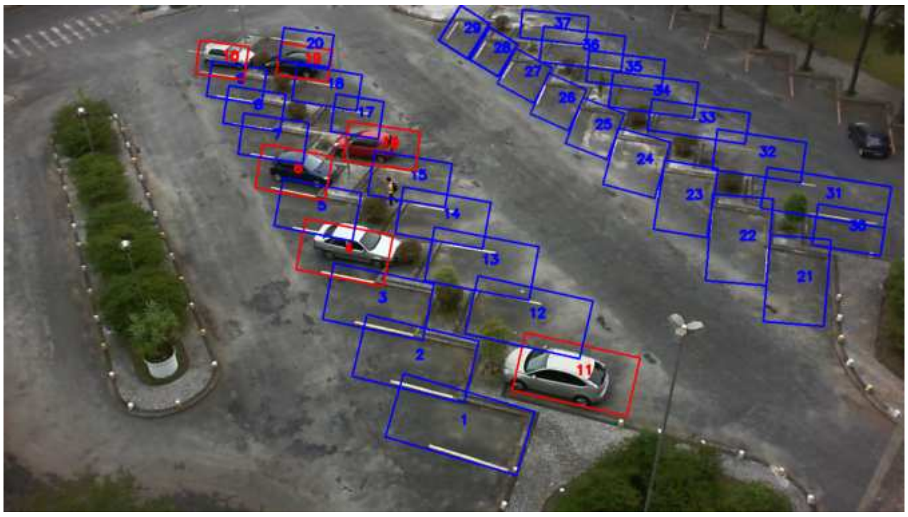
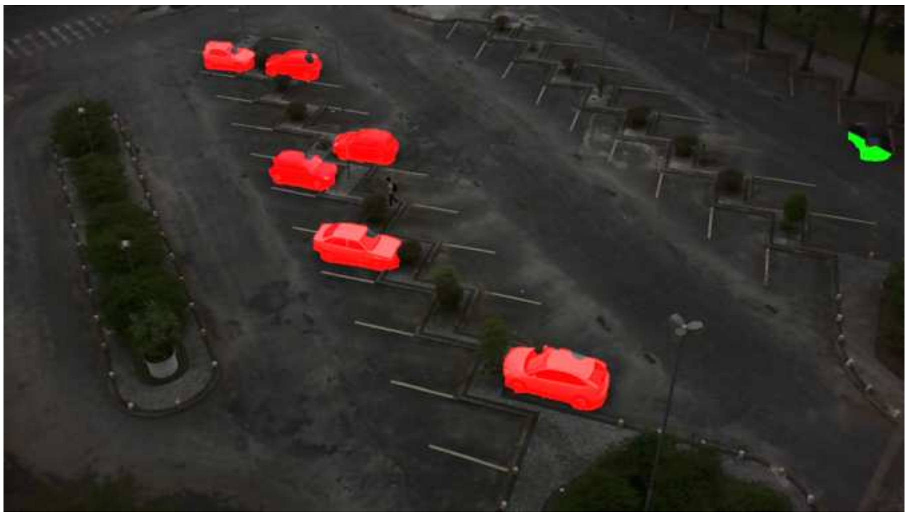


Figure 38: 2013-02-22_06_55_00.jpg



mAP = 0.877



mIoU = 0.600

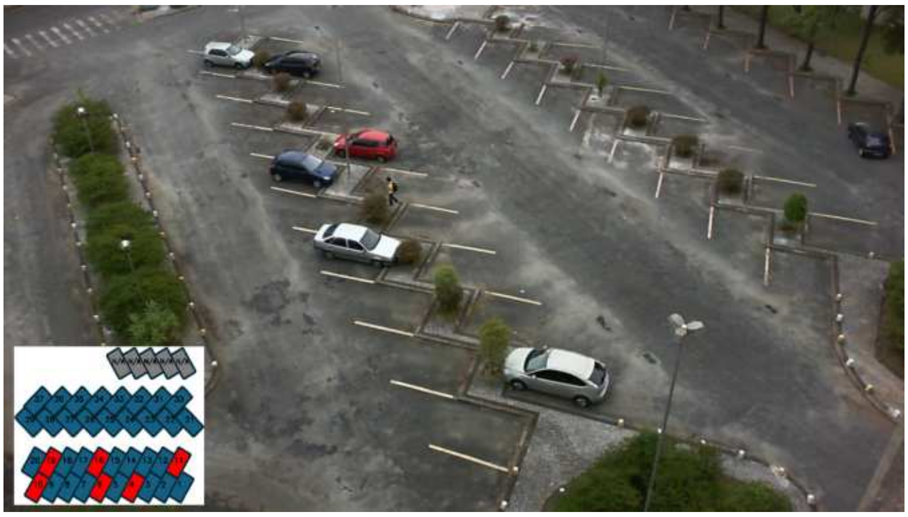
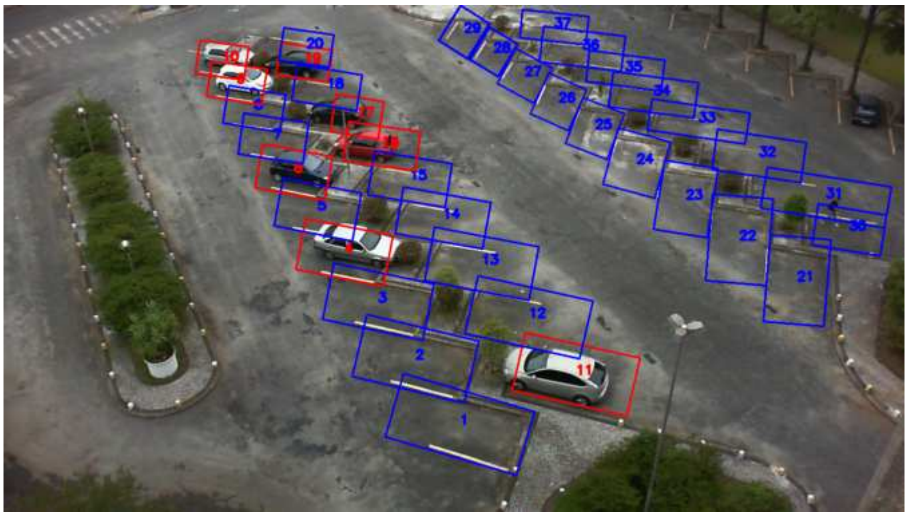
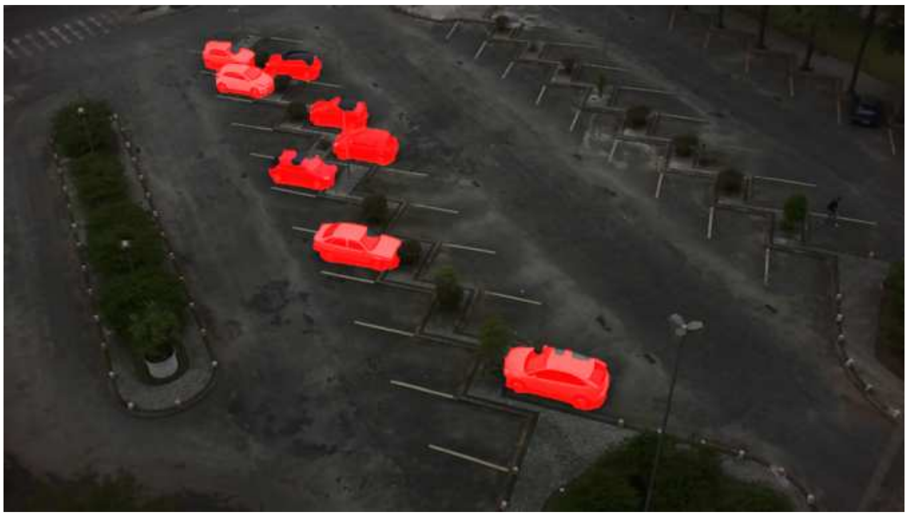


Figure 39: 2013-02-22_07_05_01.jpg



mAP = 0.874



mIoU = 0.900

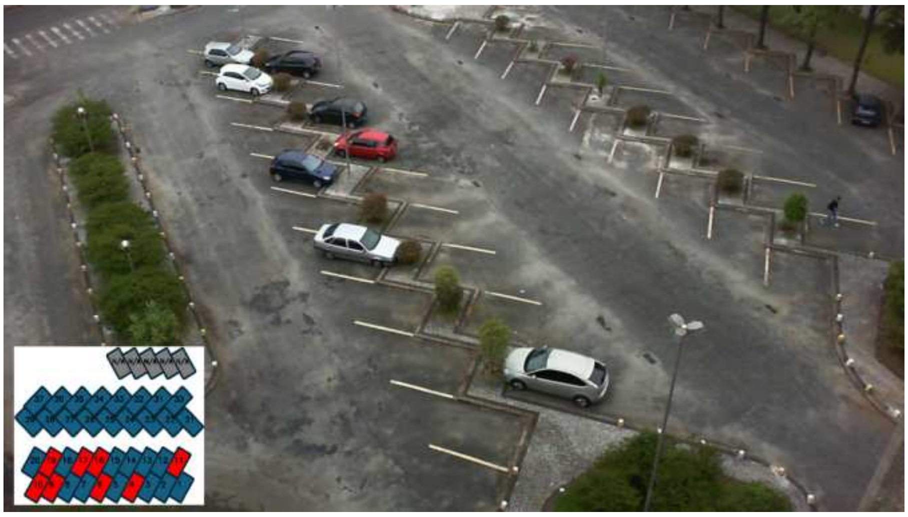
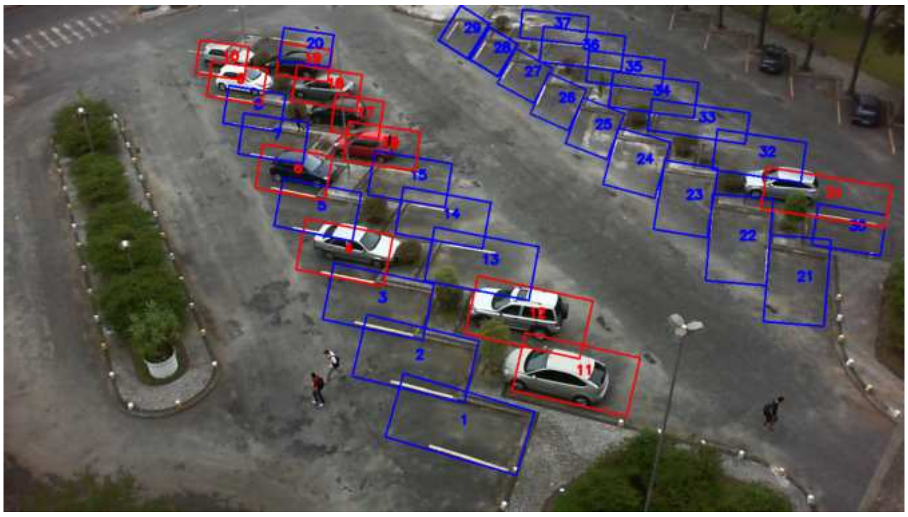
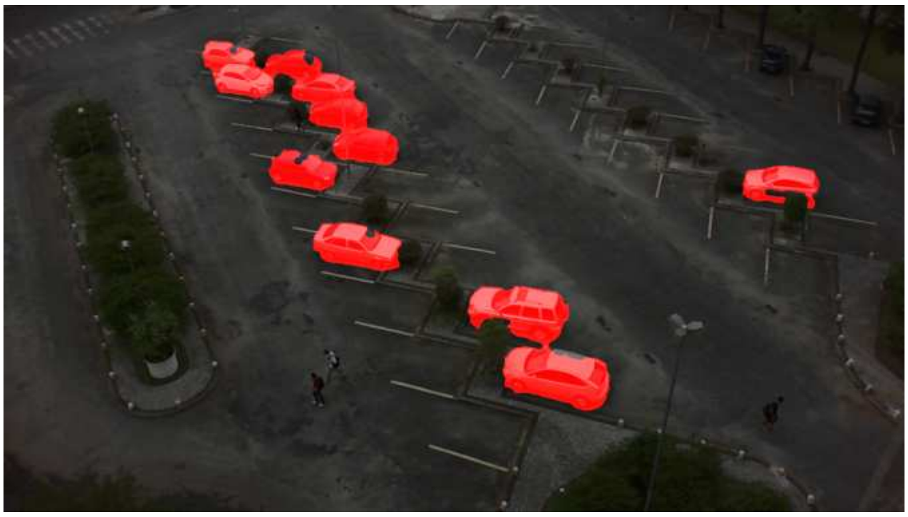


Figure 40: 2013-02-22_07_10_01.jpg



mAP = 0.824



mIoU = 0.912

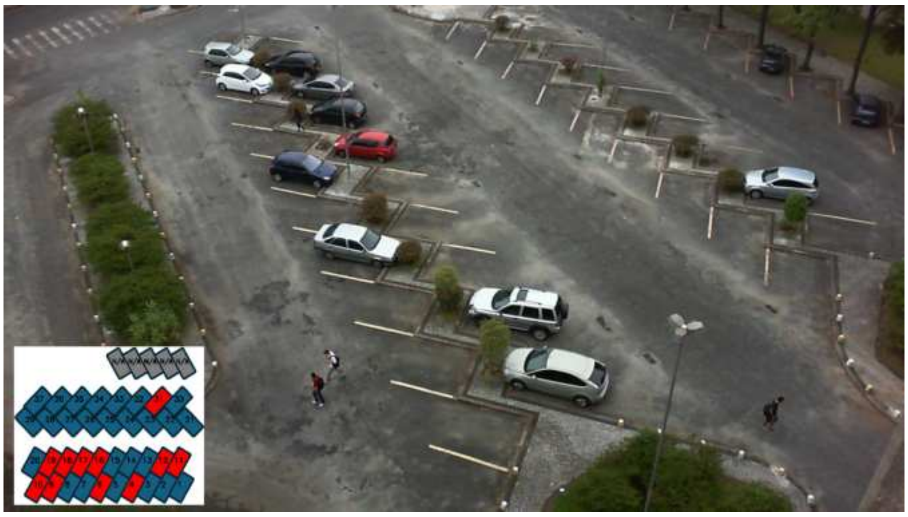
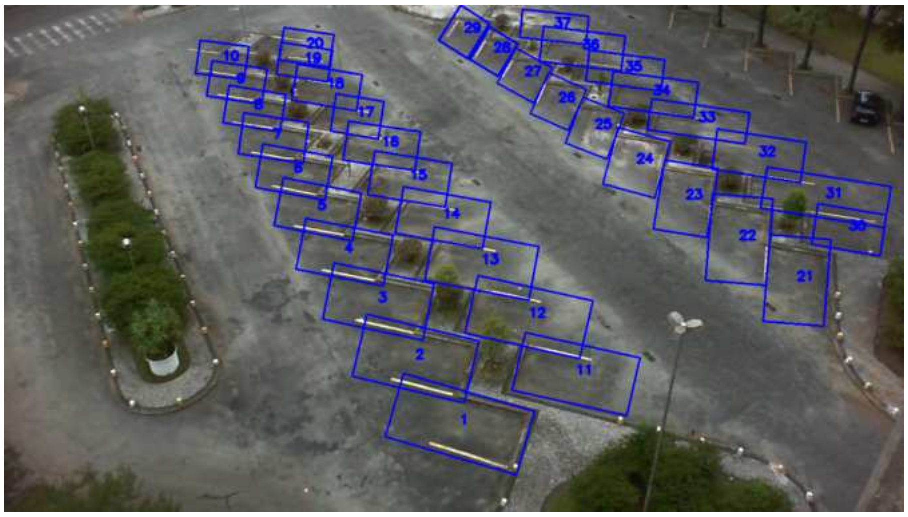
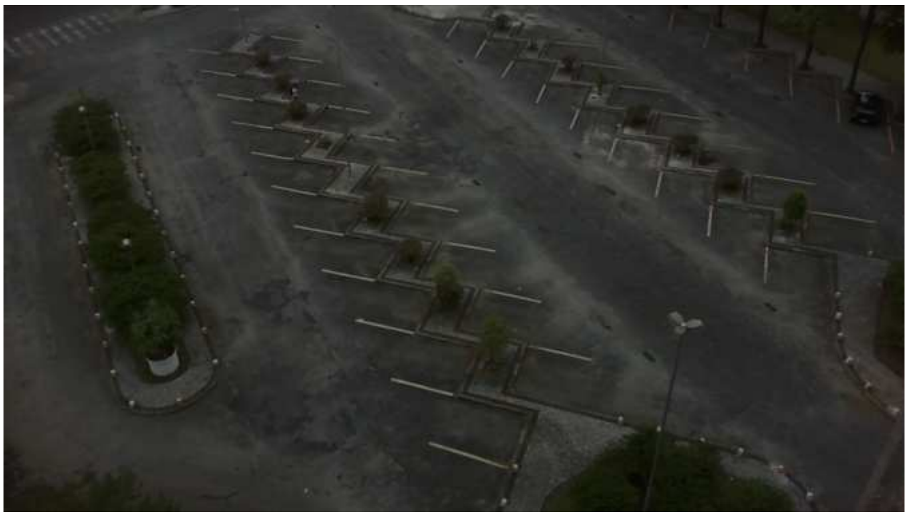


Figure 41: 2013-02-22_07_15_01.jpg

Sequence2



mAP = 0.772



mIoU = 1

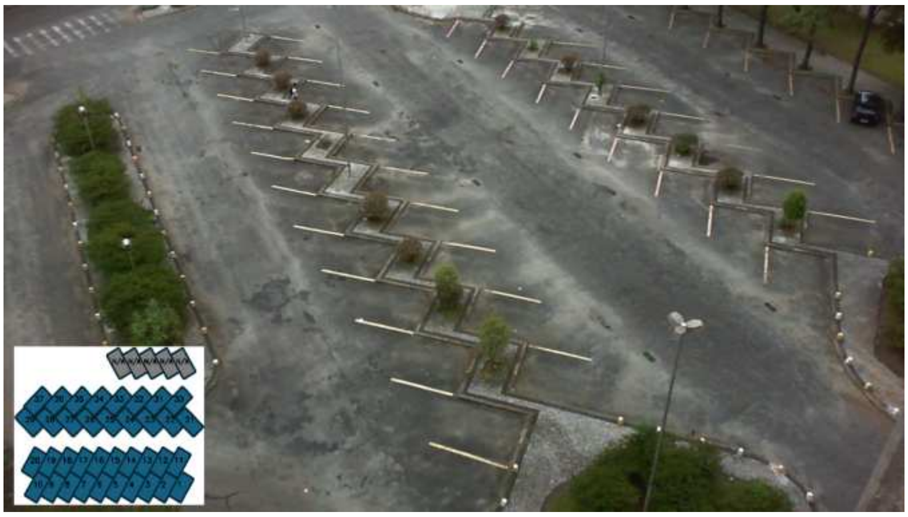
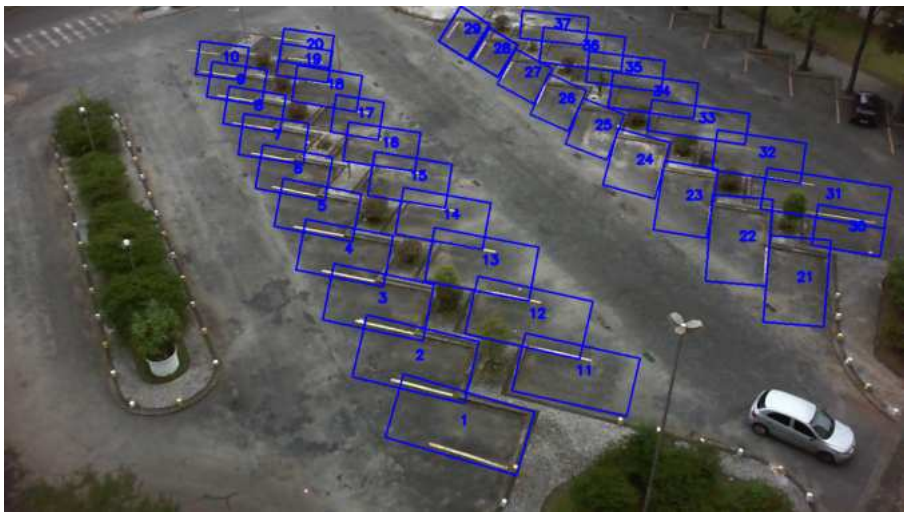
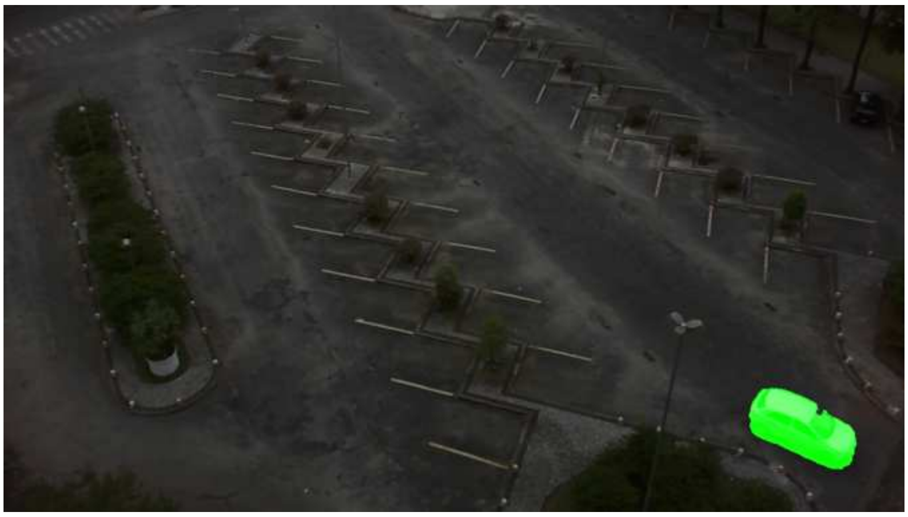


Figure 42: 2013-03-09_07_45_02.jpg



mAP = 0.772



mIoU = 0.919

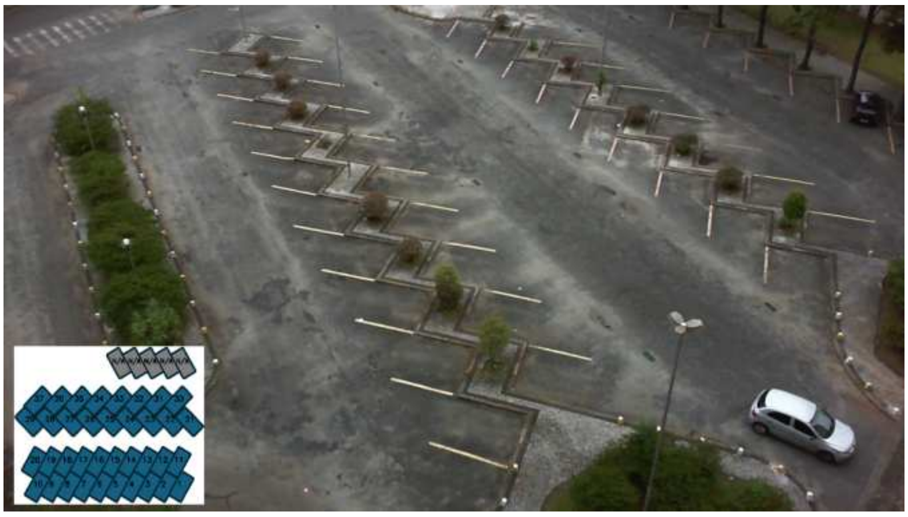
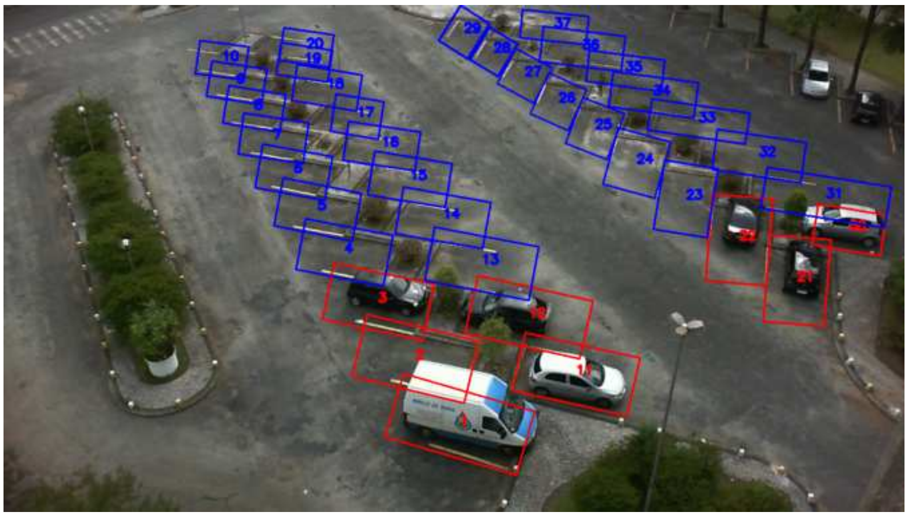
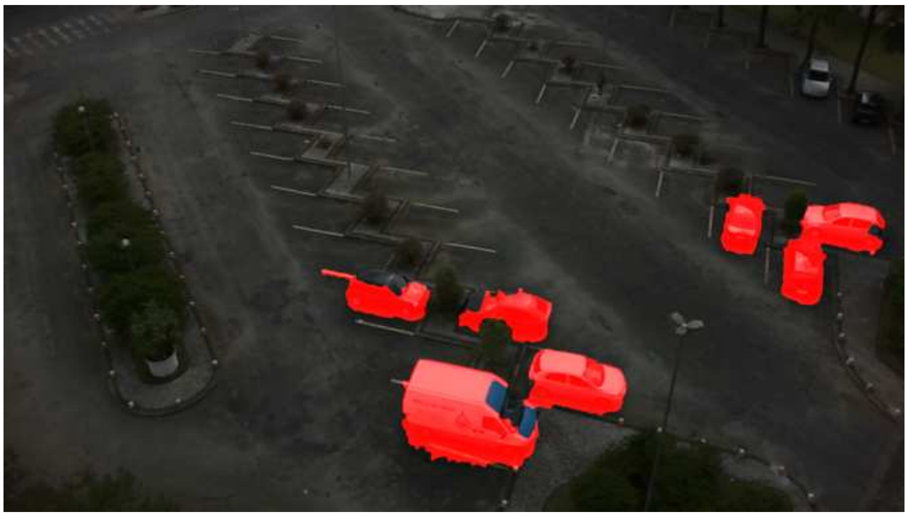


Figure 43: 2013-03-09_08_05_02.jpg



mAP = 0.722



mIoU = 0.843

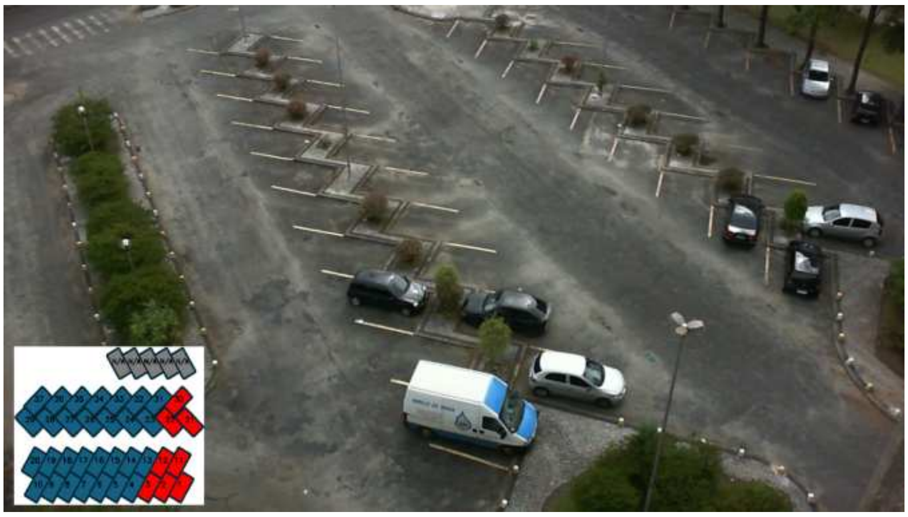
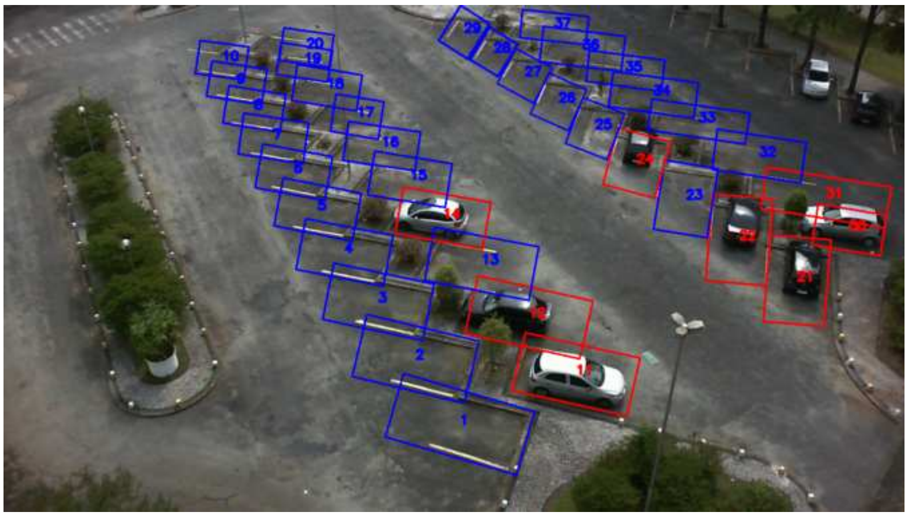
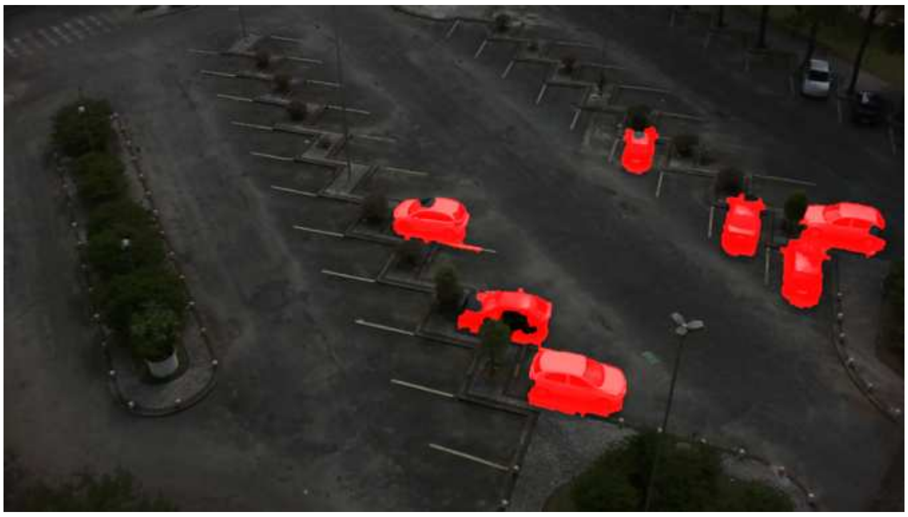


Figure 44: 2013-03-09_09_30_04.jpg



mAP = 0.739



mIoU = 0.853

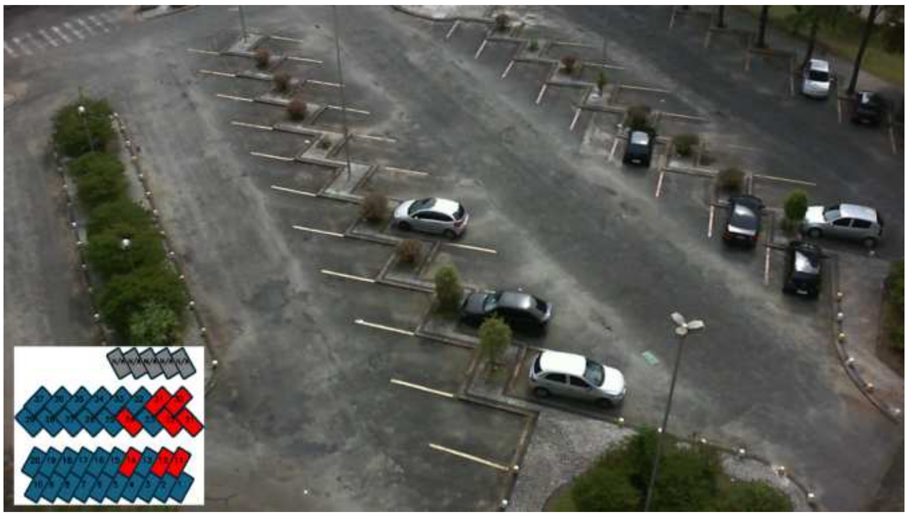
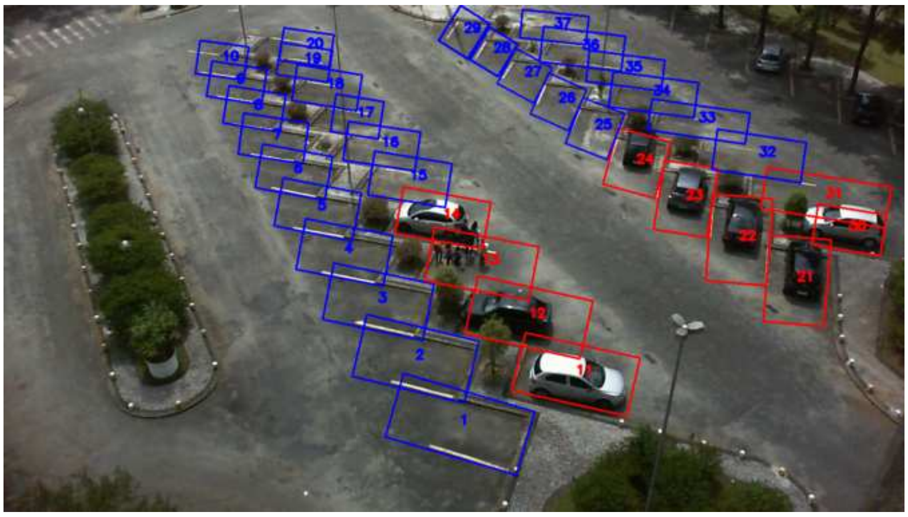
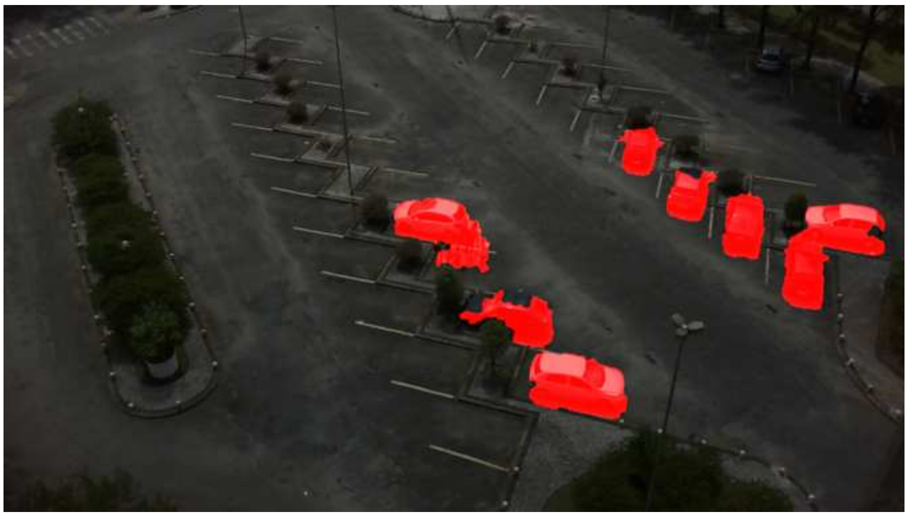


Figure 45: 2013-03-09_09_45_04.jpg



mAP = 0.677



mIoU = 0.834

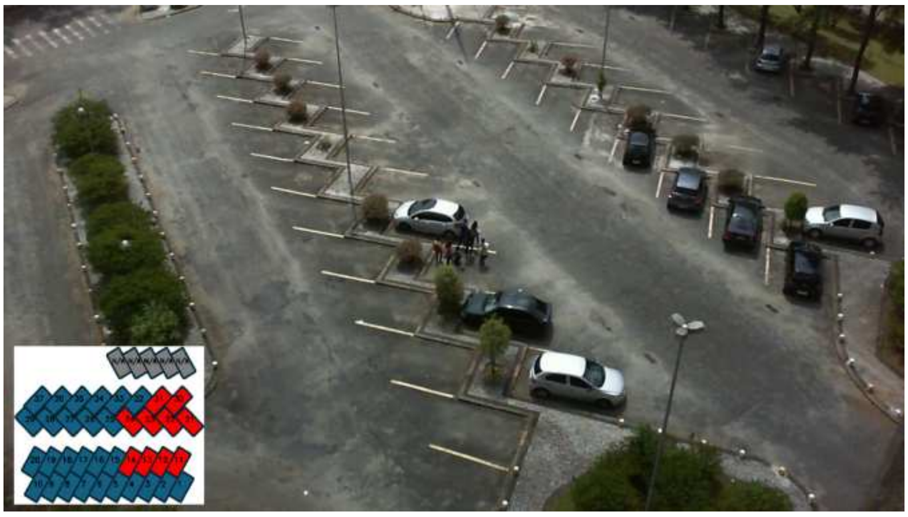
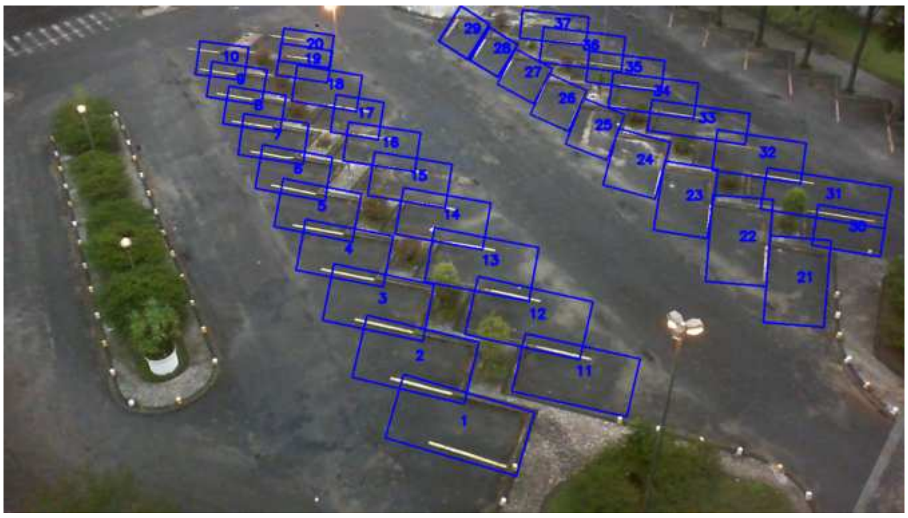
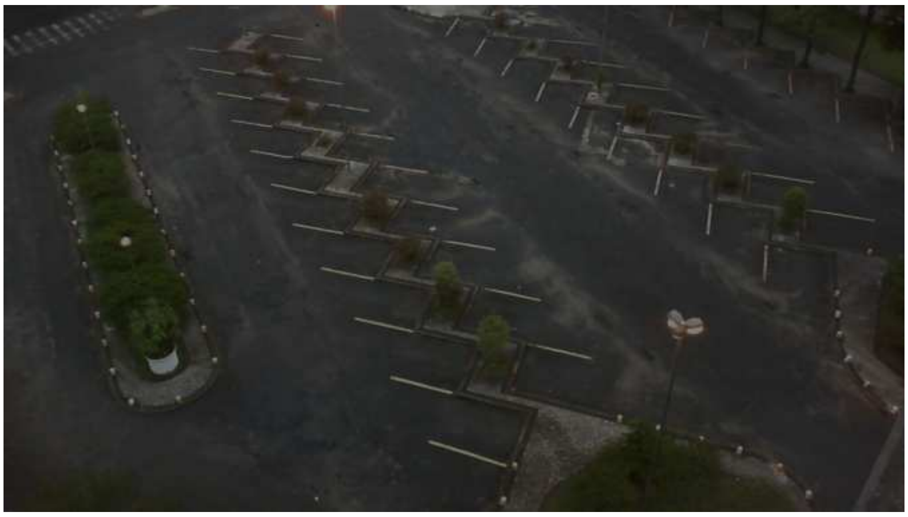


Figure 46: 2013-03-09_11_20_06.jpg

Sequence3



mAP = 0.772



mIoU = 1

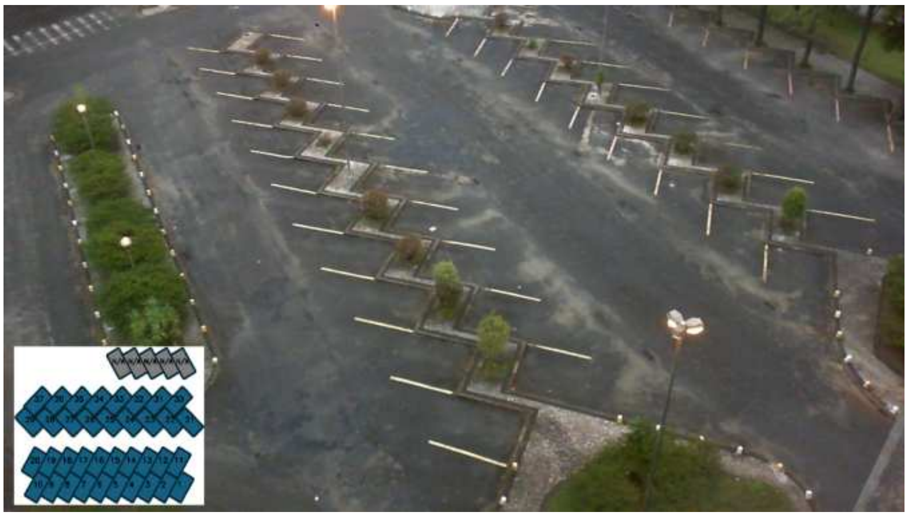
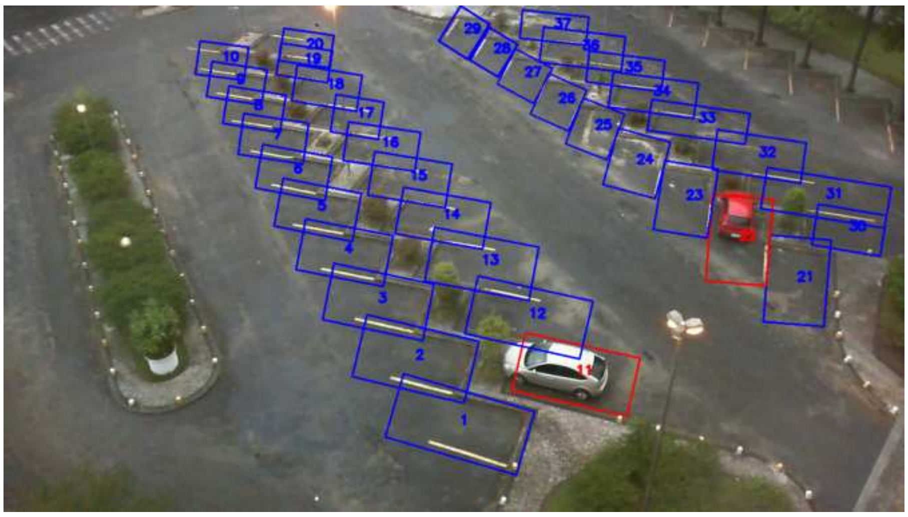
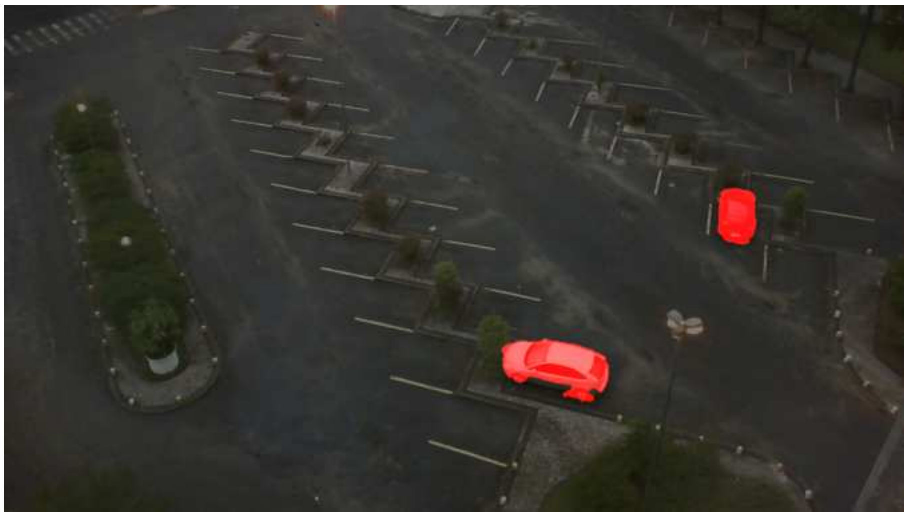


Figure 47: 2013-03-19_06_50_01.jpg



mAP = 0.705



mIoU = 0.924

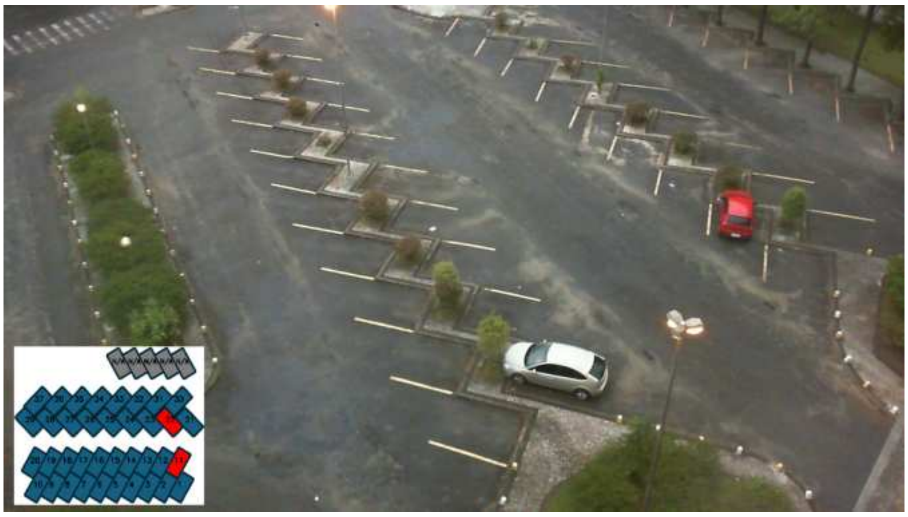
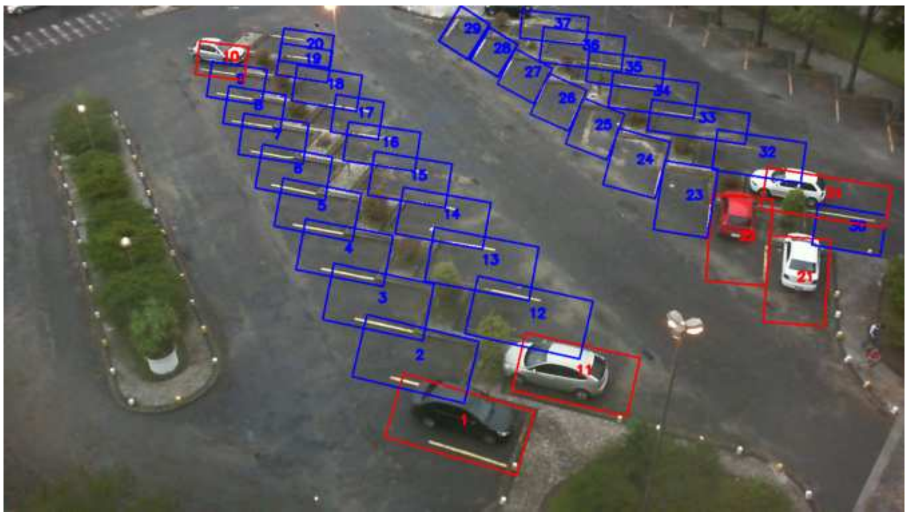
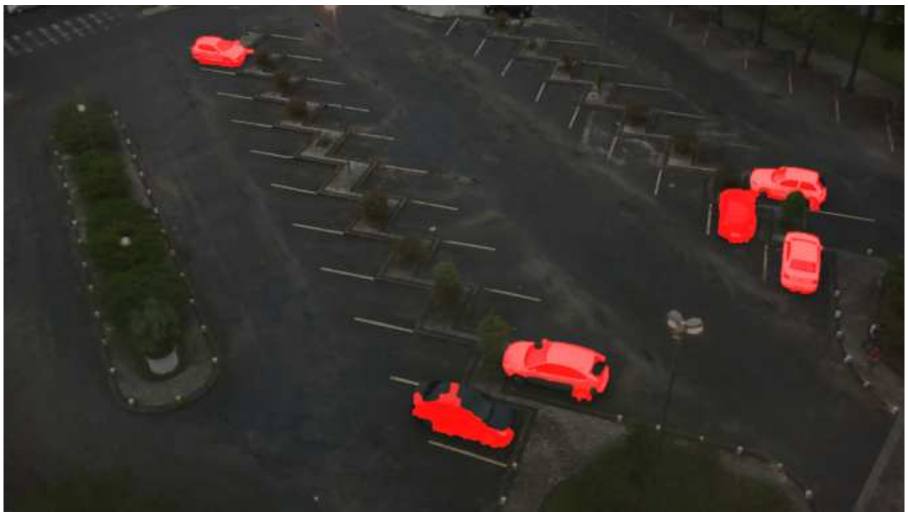


Figure 48: 2013-03-19_06_55_01.jpg



mAP = 0.752



mIoU = 0.852

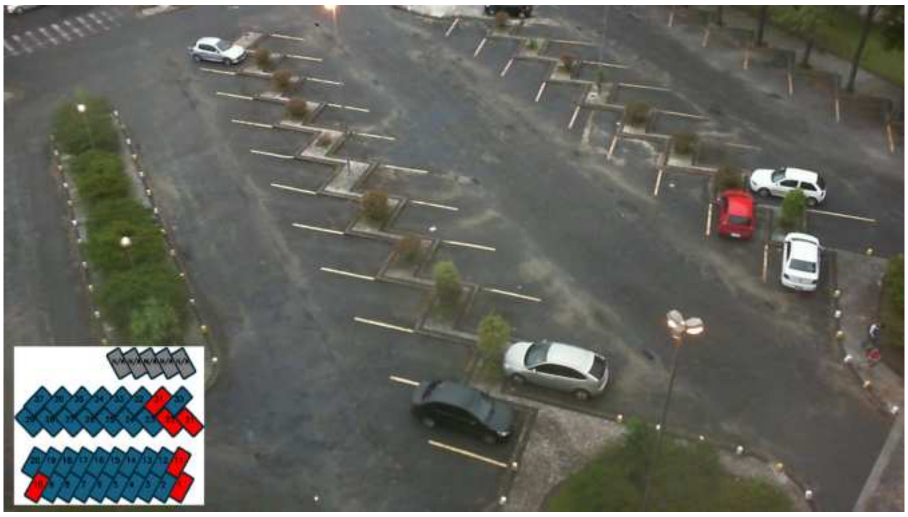
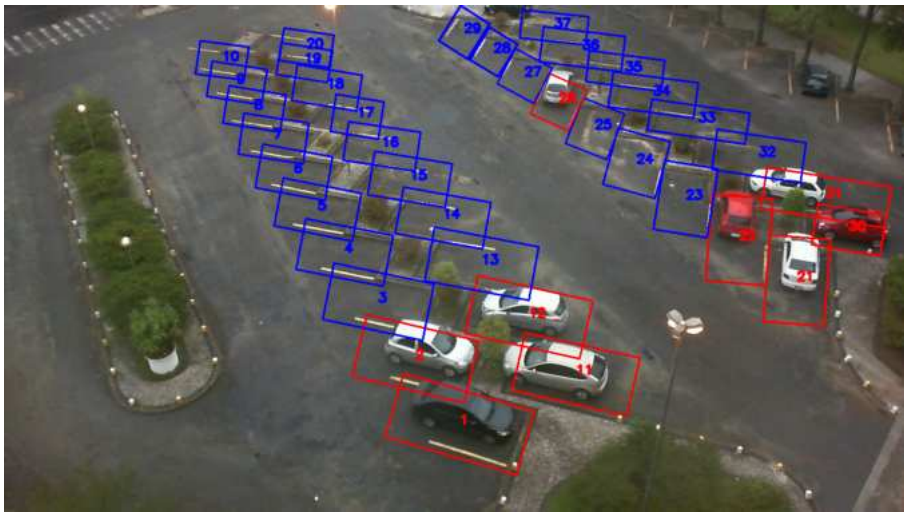
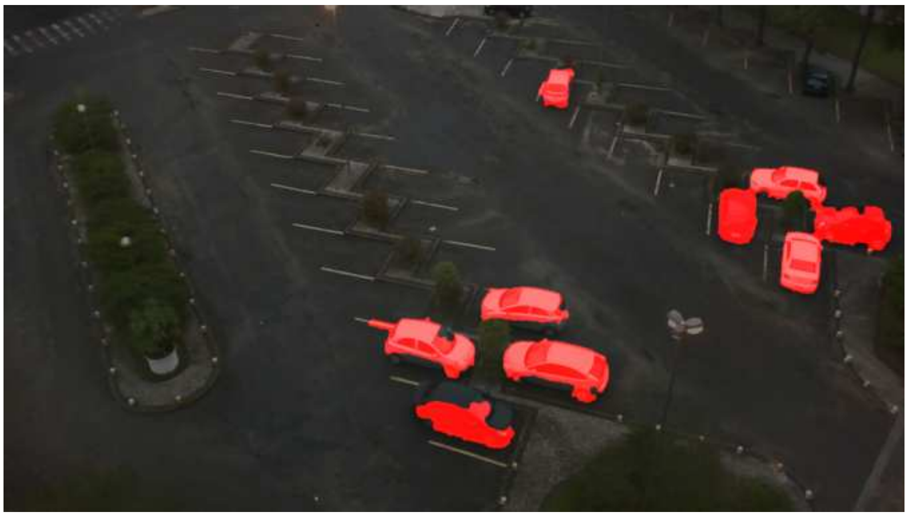


Figure 49: 2013-03-19_07_05_01.jpg



mAP = 0.734



mIoU = 0.853

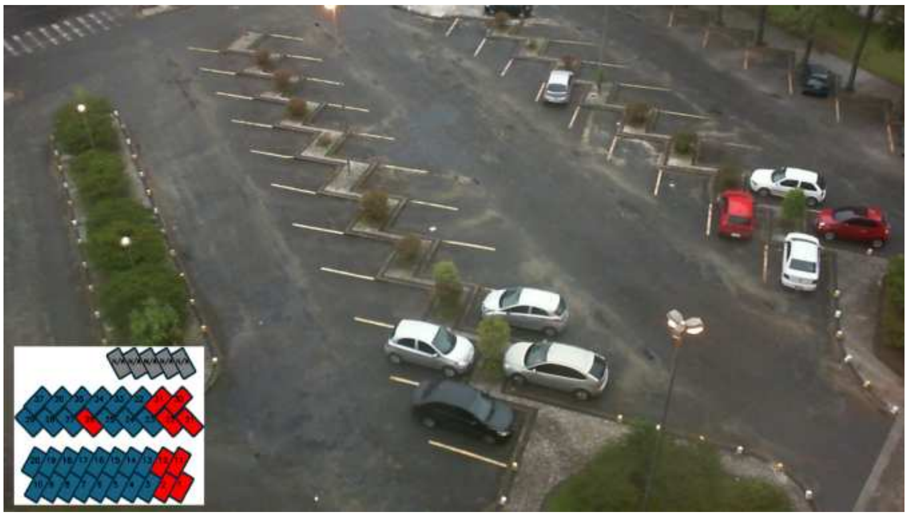
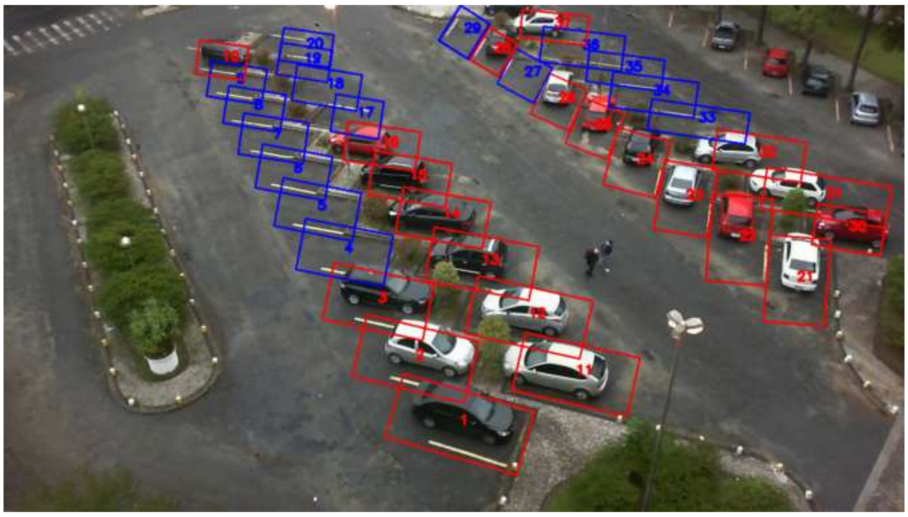
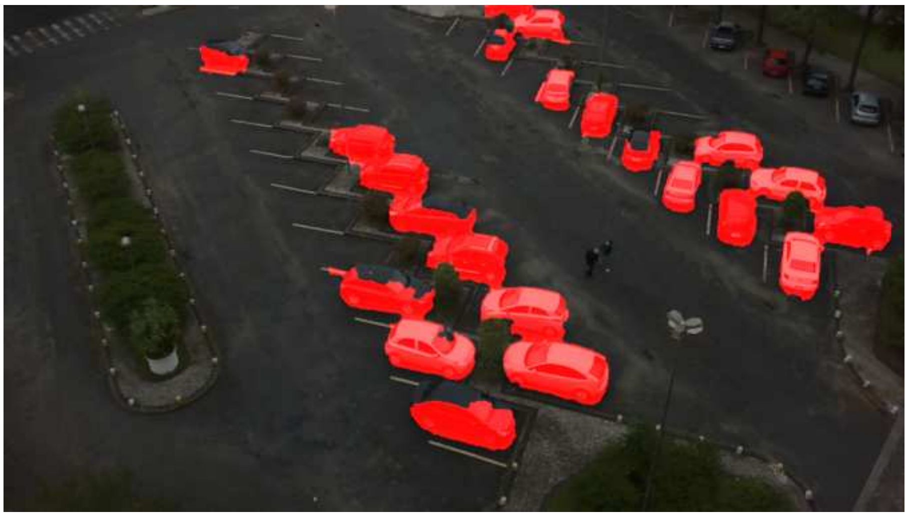


Figure 50: 2013-03-19_07_10_01.jpg



mAP = 0.822



mIoU = 0.865

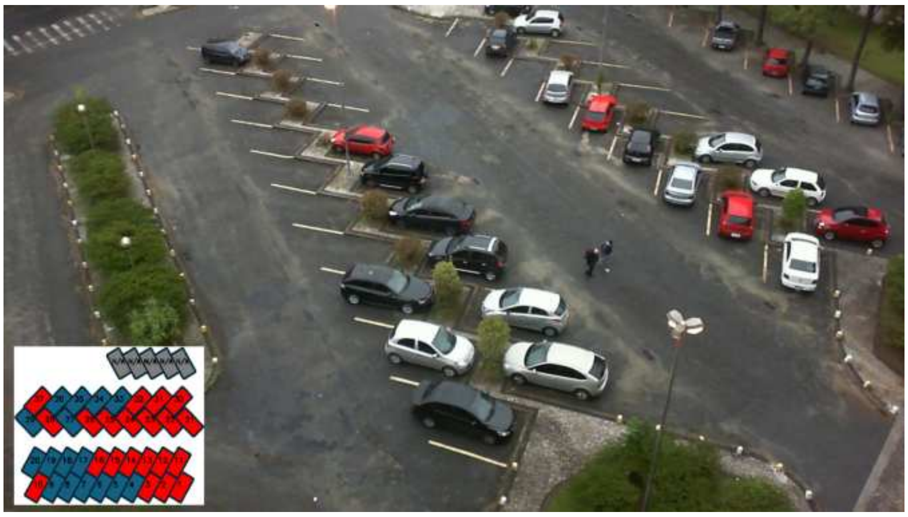
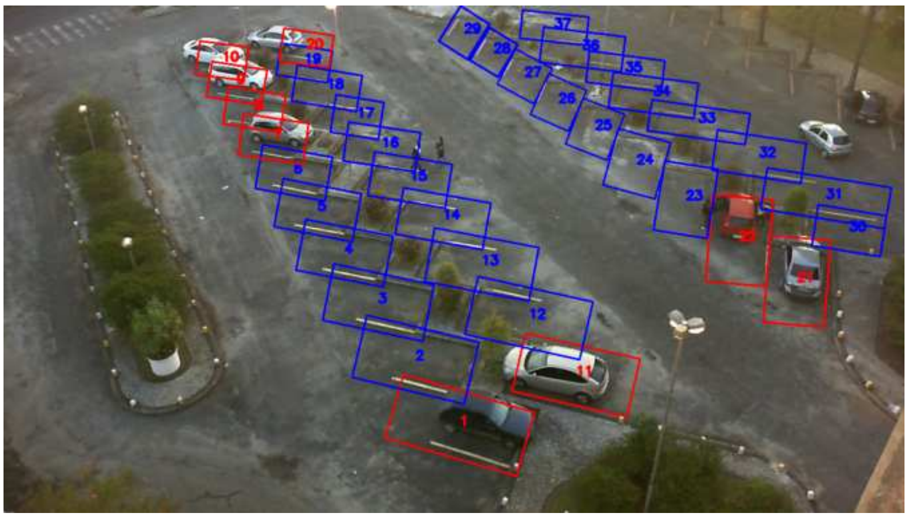
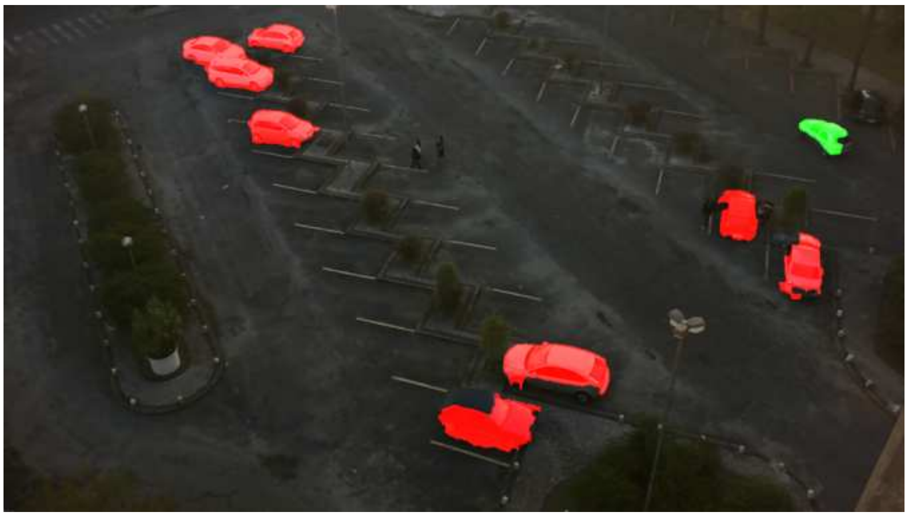


Figure 51: 2013-03-19_07_25_01.jpg

Sequence4



mAP = 0.714



mIoU = 0.803

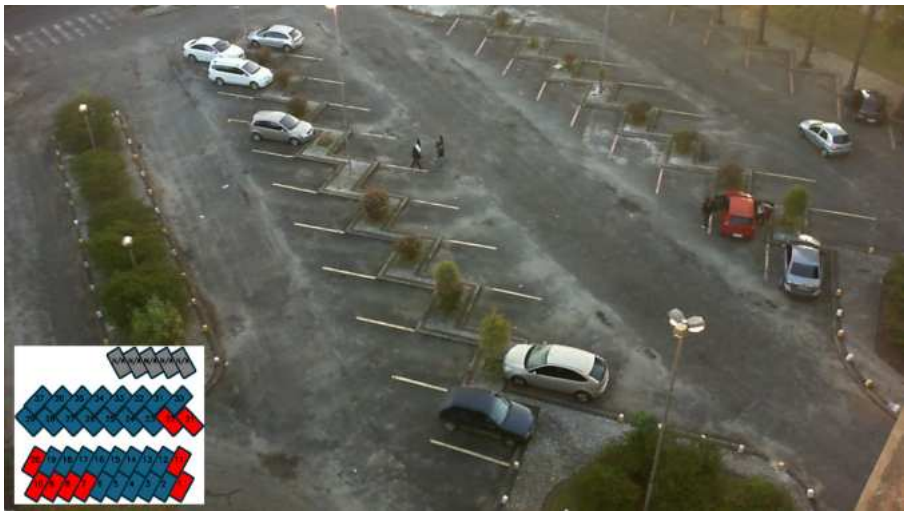
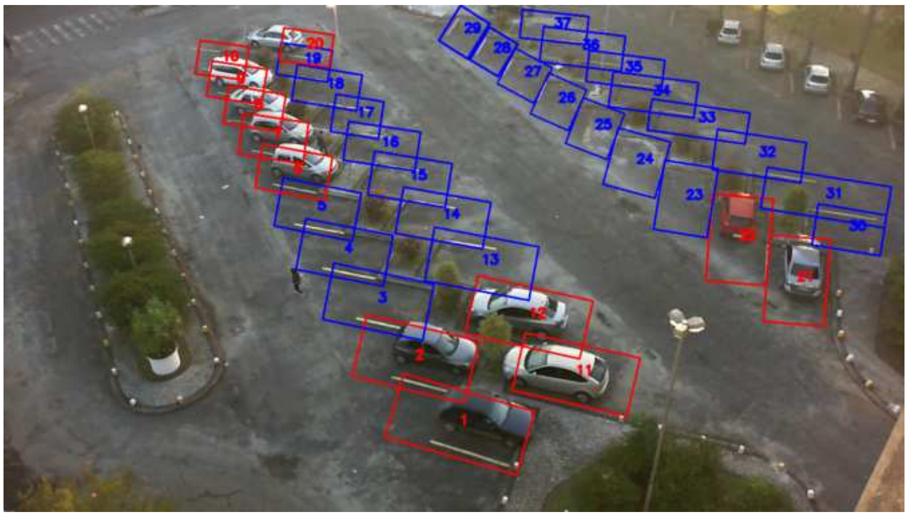
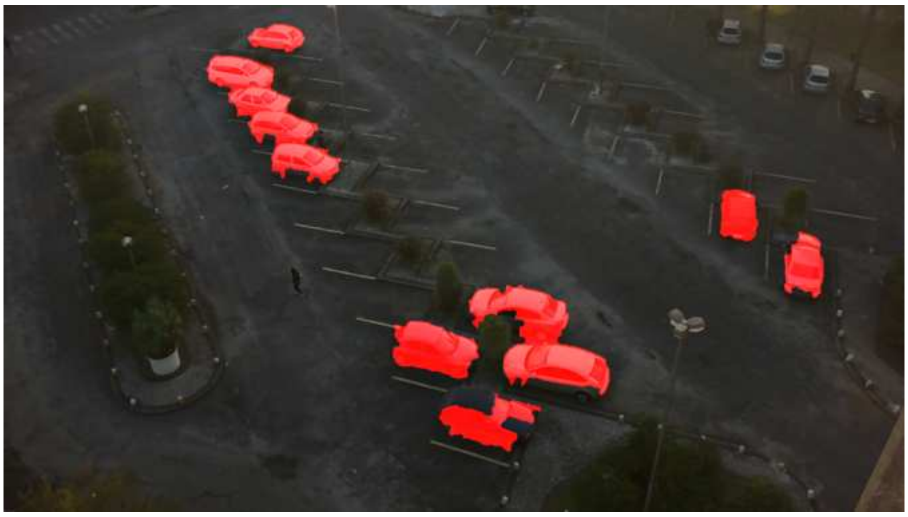


Figure 52: 2013-04-15_07_05_01.jpg



mAP = 0.806



mIoU = 0.872

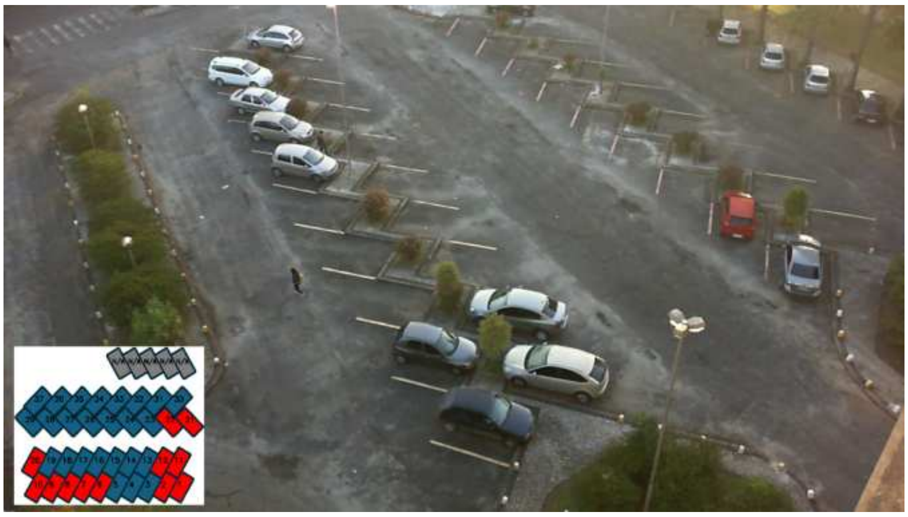
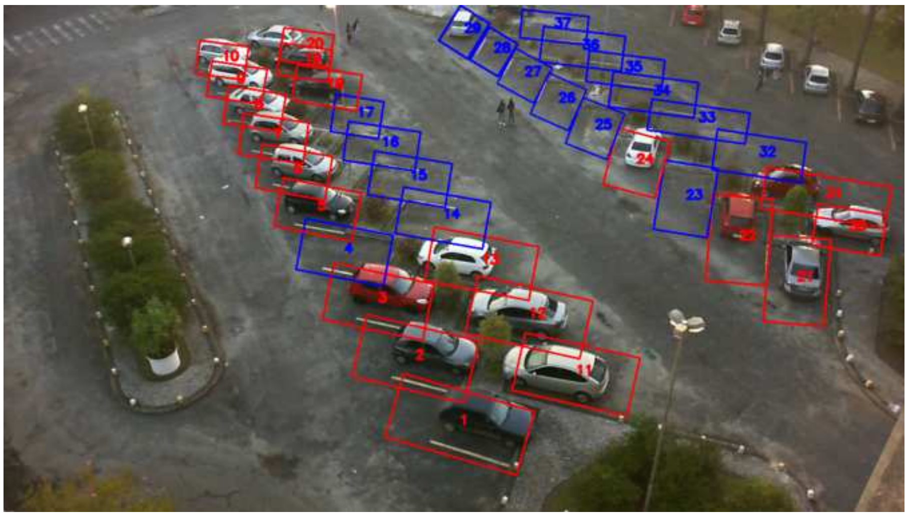
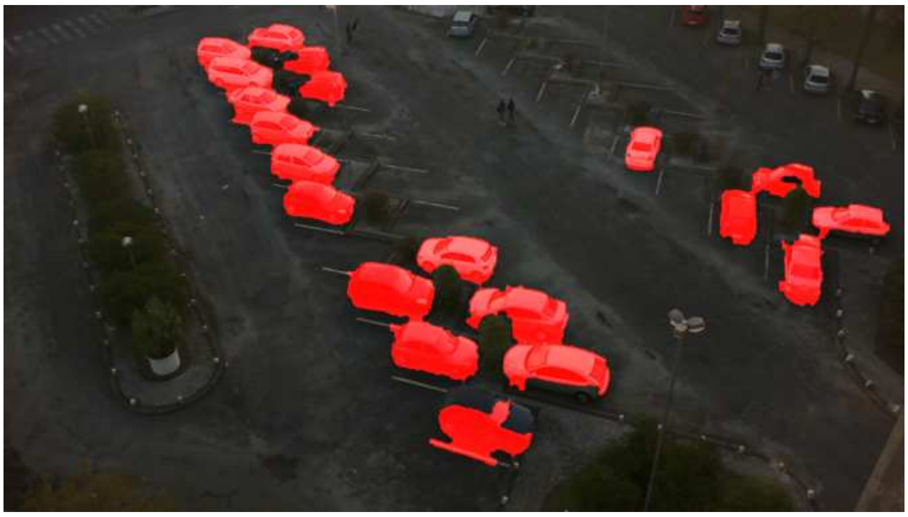


Figure 53: 2013-04-15_07_10_01.jpg



mAP = 0.839



mIoU = 0.863

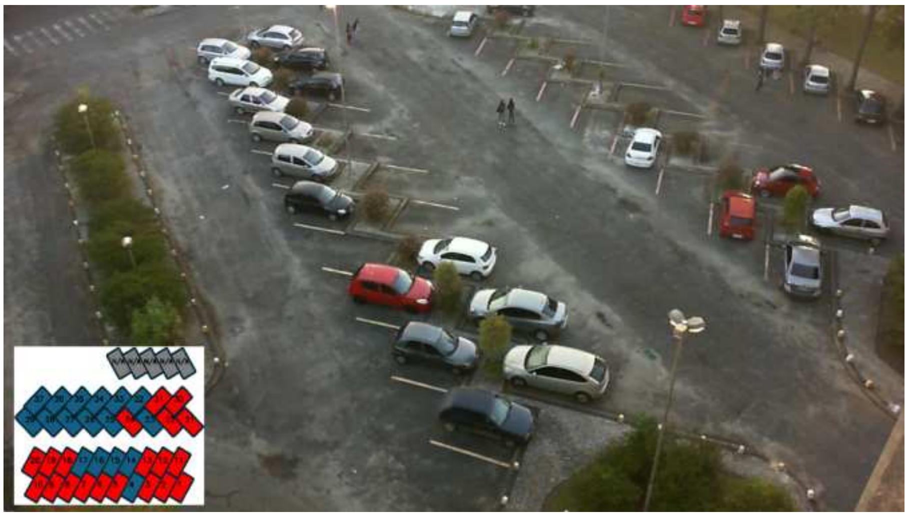
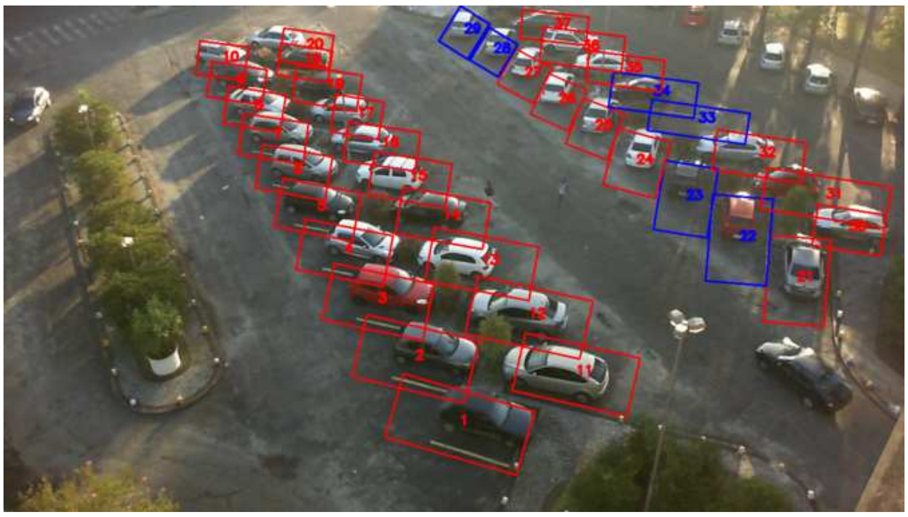
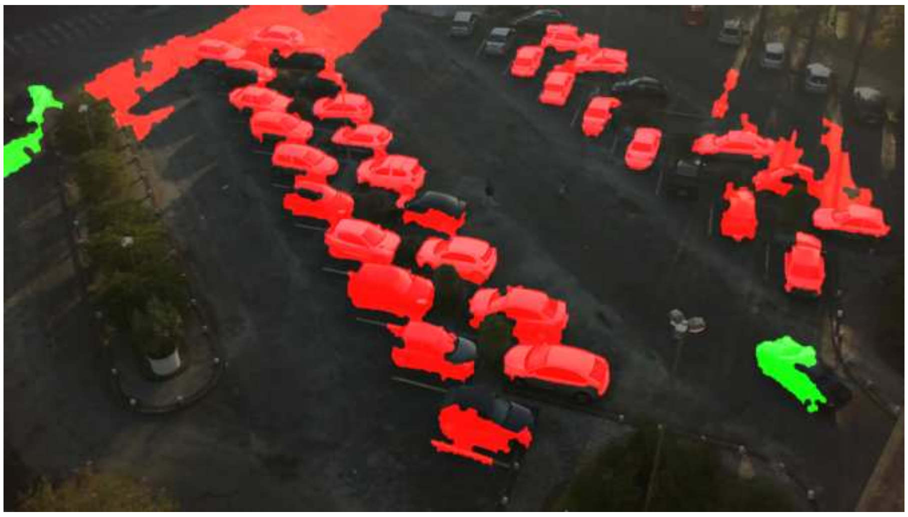


Figure 54: 2013-04-15_07_15_01.jpg



mAP = 0.451



mIoU = 0.620

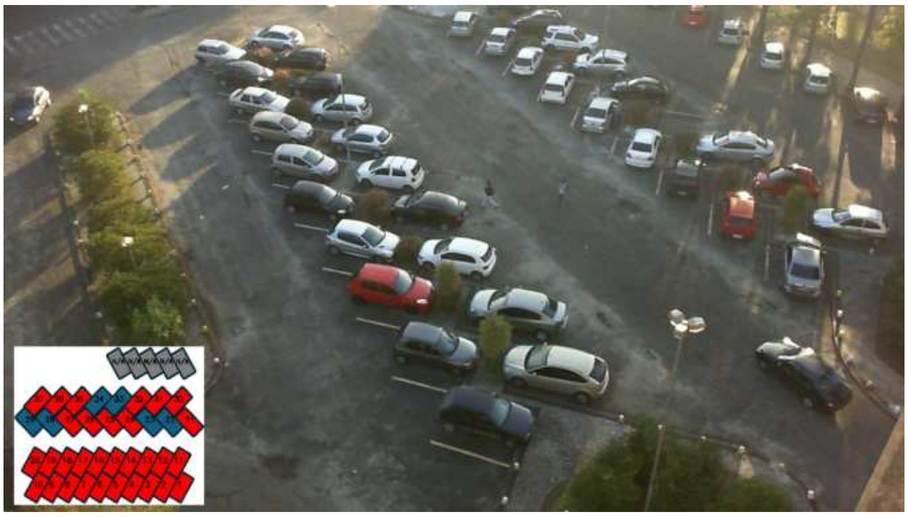
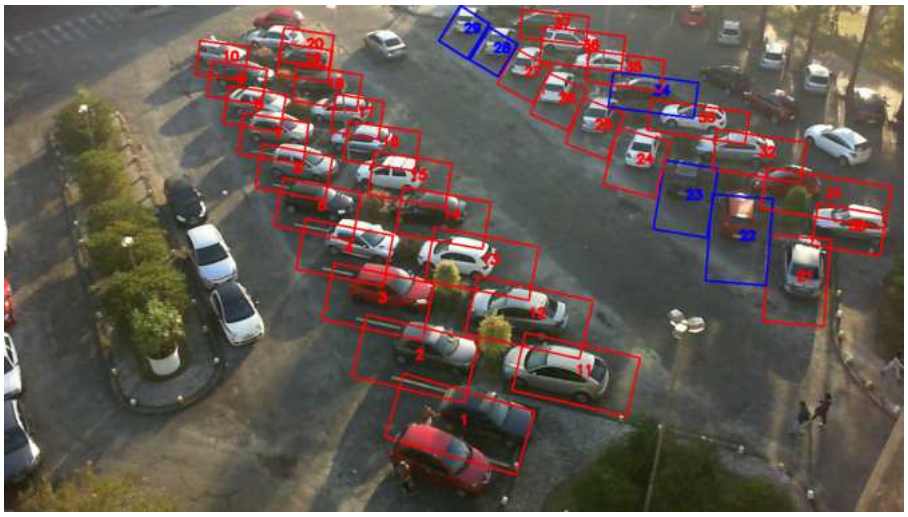
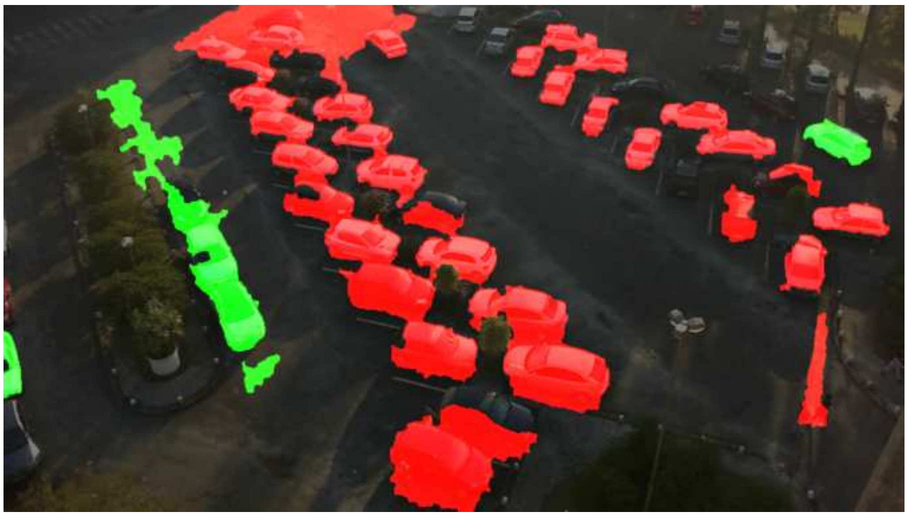


Figure 55: 2013-04-15_07_25_01.jpg



mAP = 0.695



mIoU = 0.577

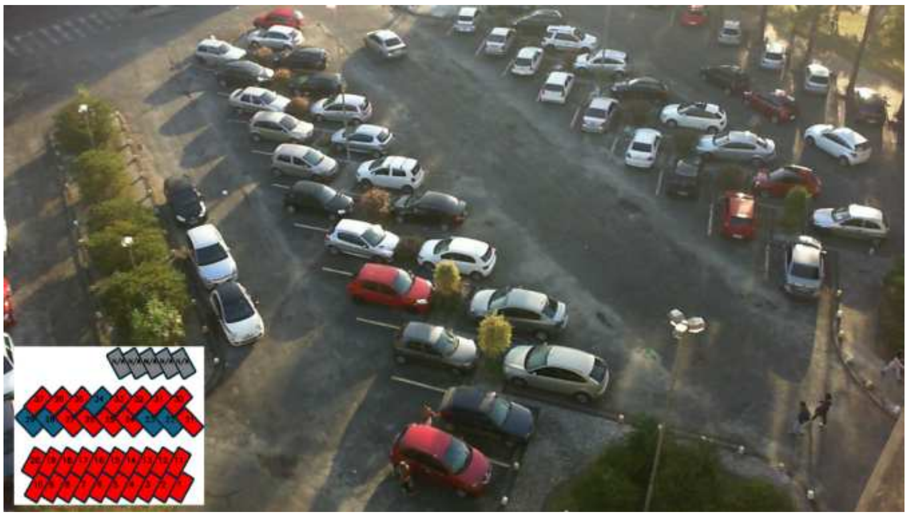
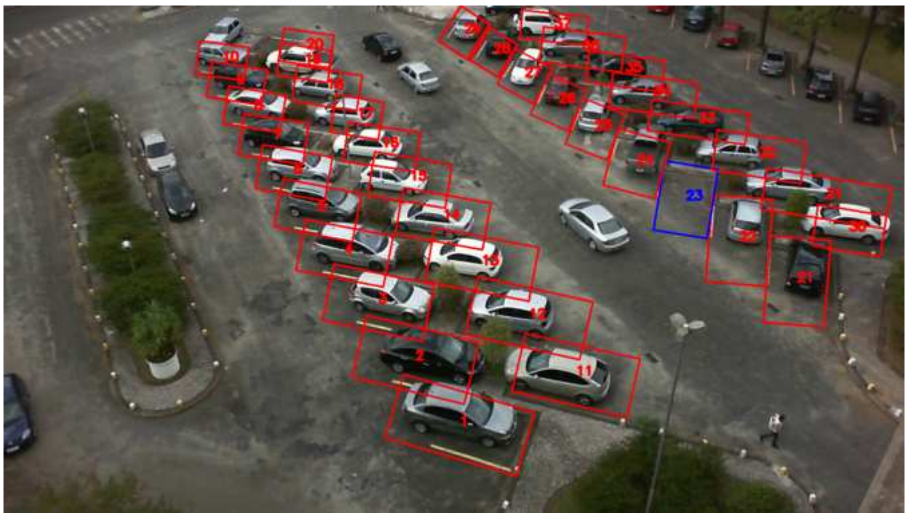
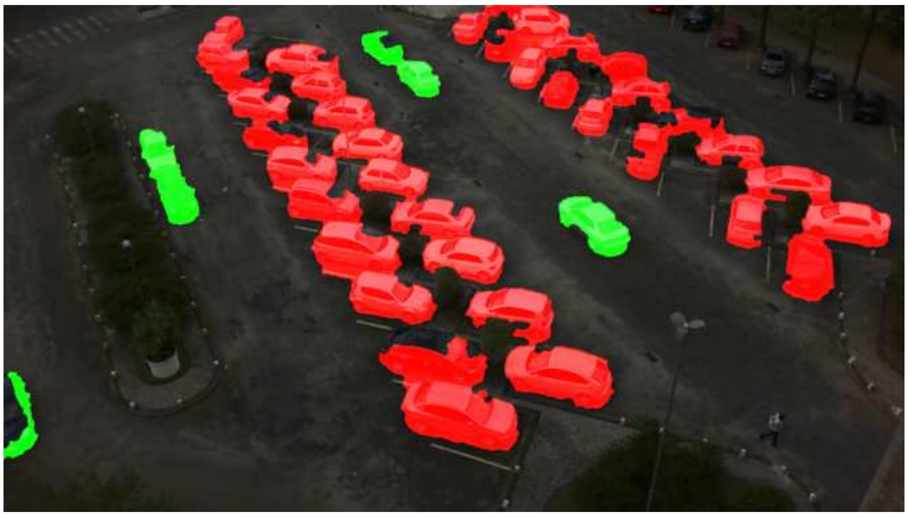


Figure 56: 2013-04-15_07_35_01.jpg

Sequence5



mAP = 0.703



mIoU = 0.807

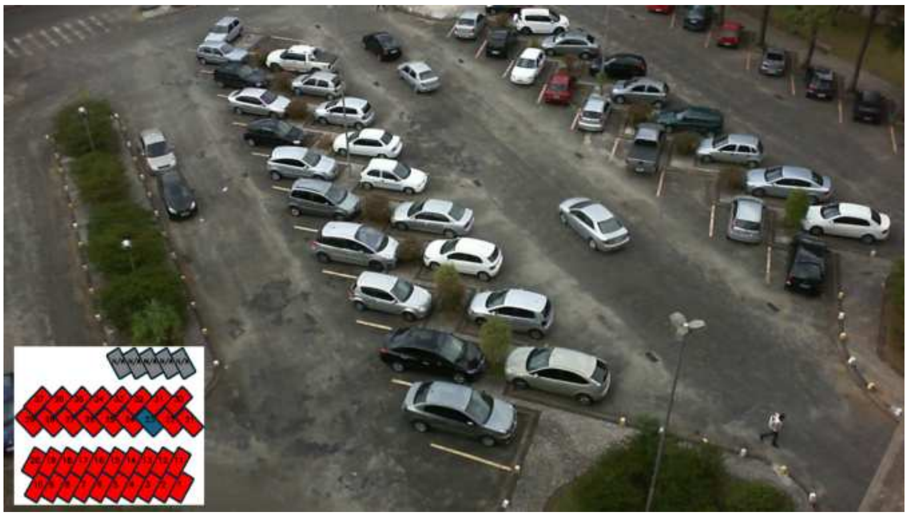
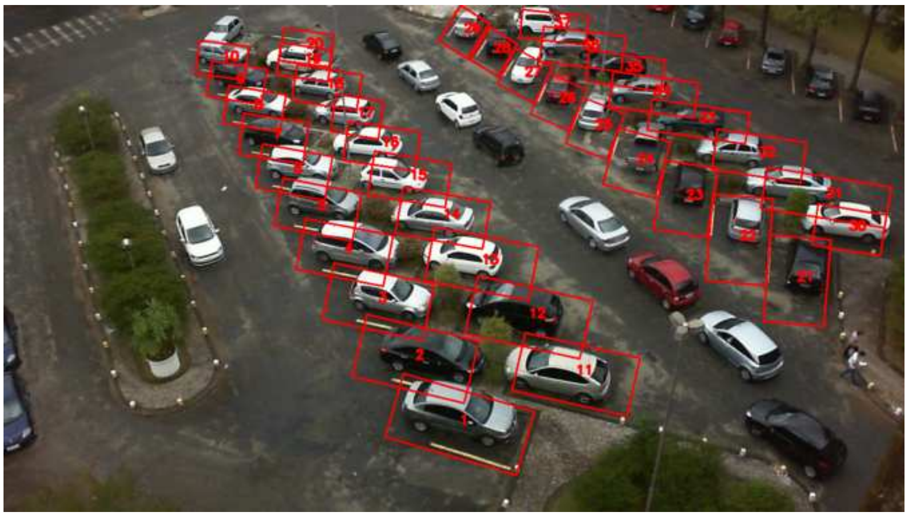
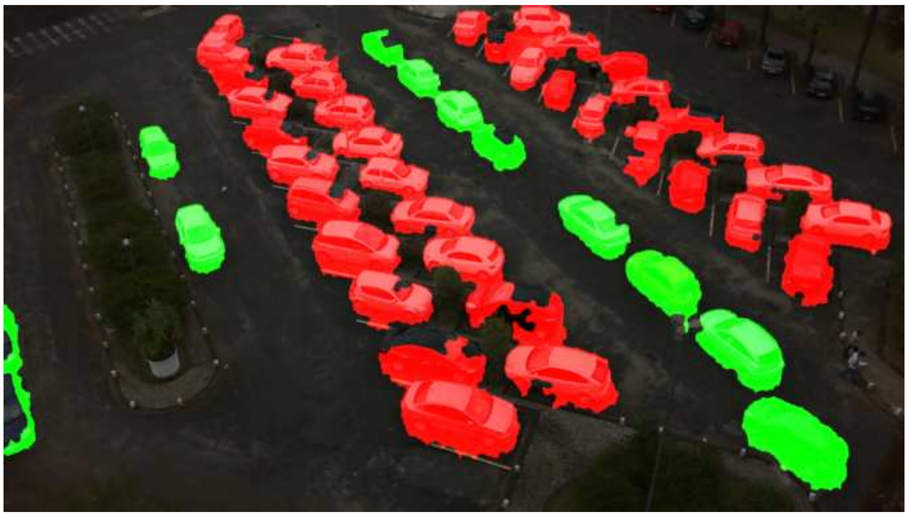


Figure 57: 2013-04-12_14_20_09.jpg



mAP = 0.431



mIoU = 0.798

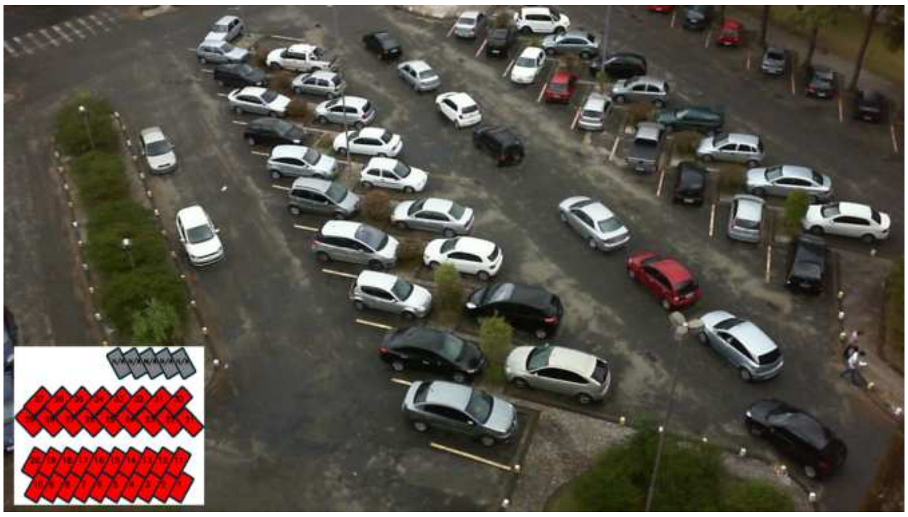
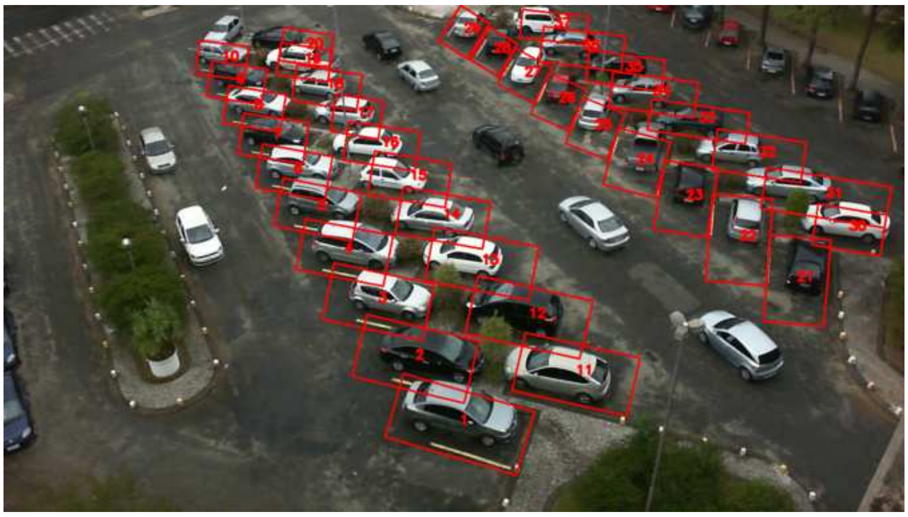
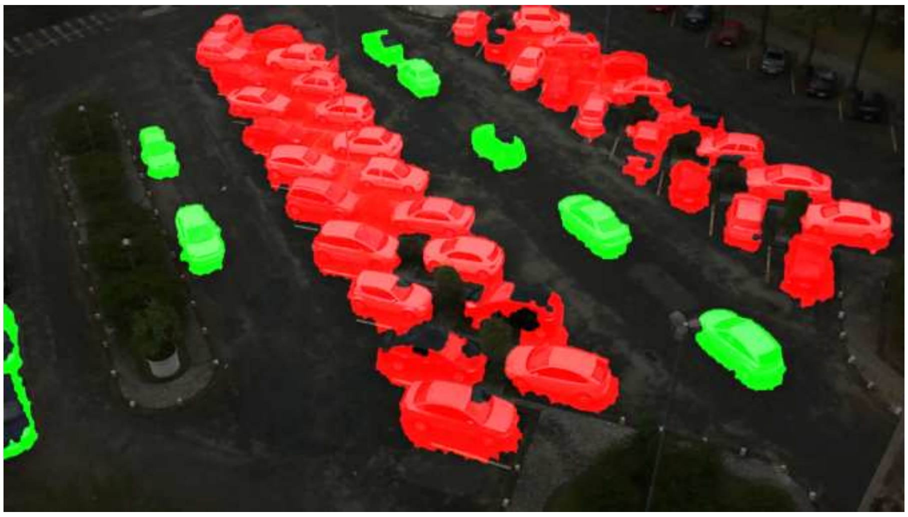


Figure 58: 2013-04-12_14_45_09.jpg



mAP = 0.772



mIoU = 0.767

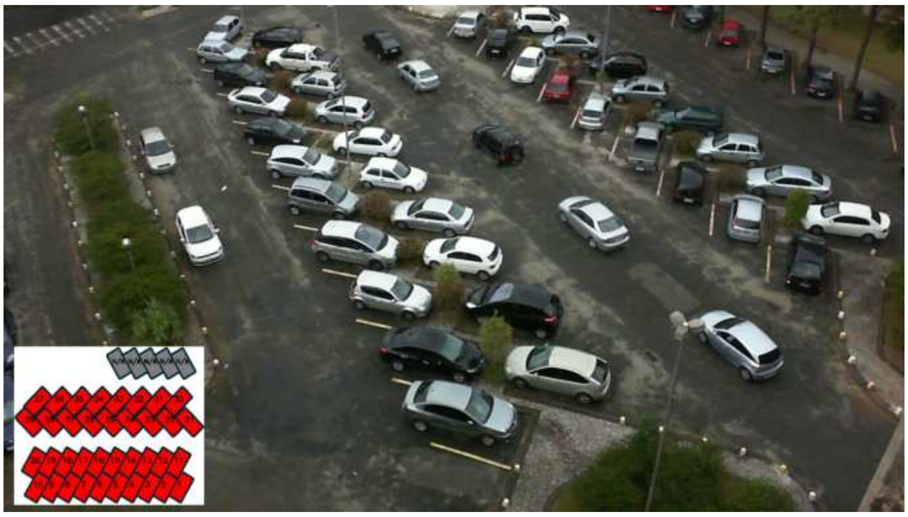
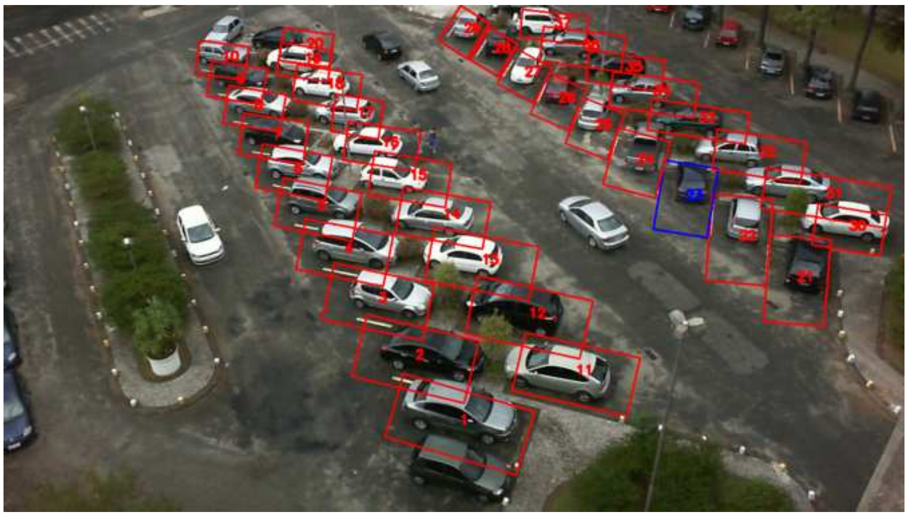
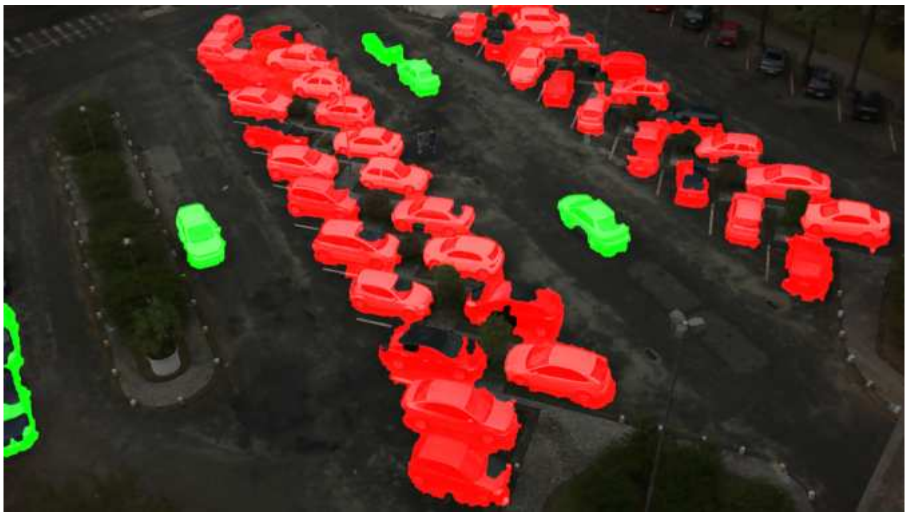


Figure 59: 2013-04-12_14_50_09.jpg



mAP = 0.771



mIoU = 0.703

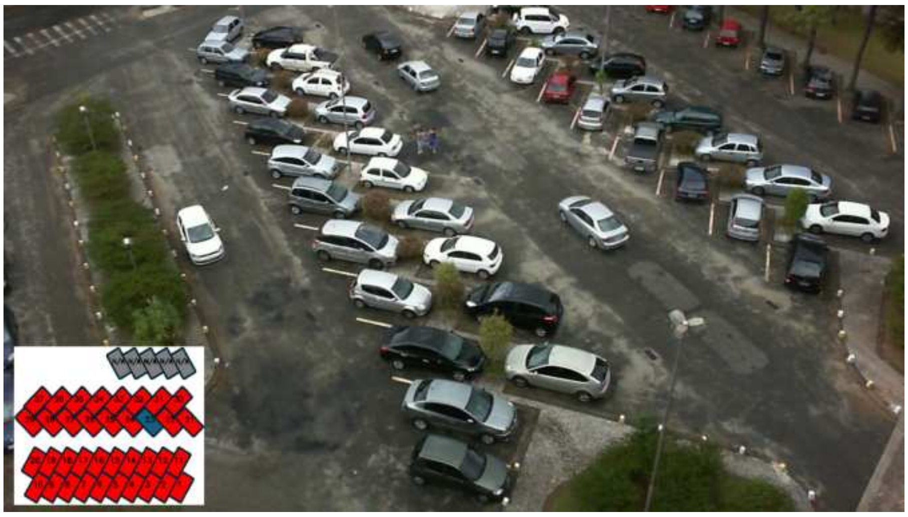
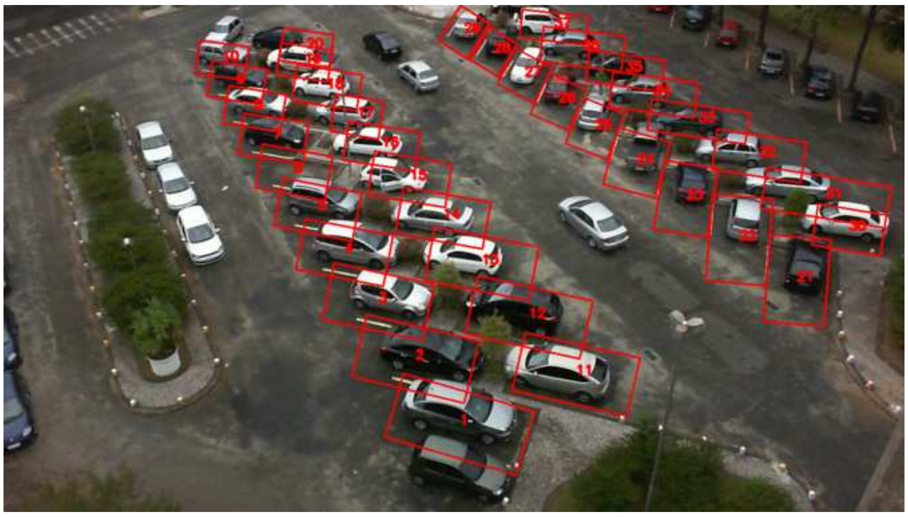
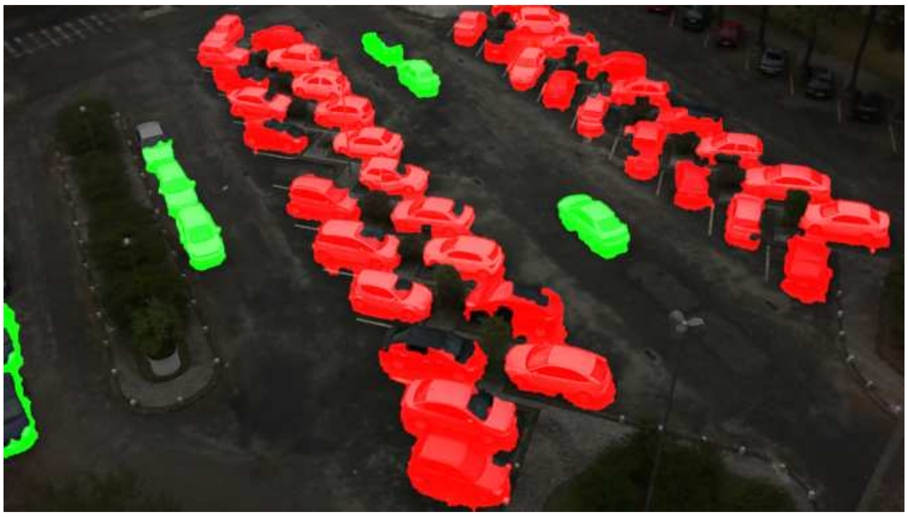


Figure 60: 2013-04-12_14_55_09.jpg



mAP = 0.371



mIoU = 0.698

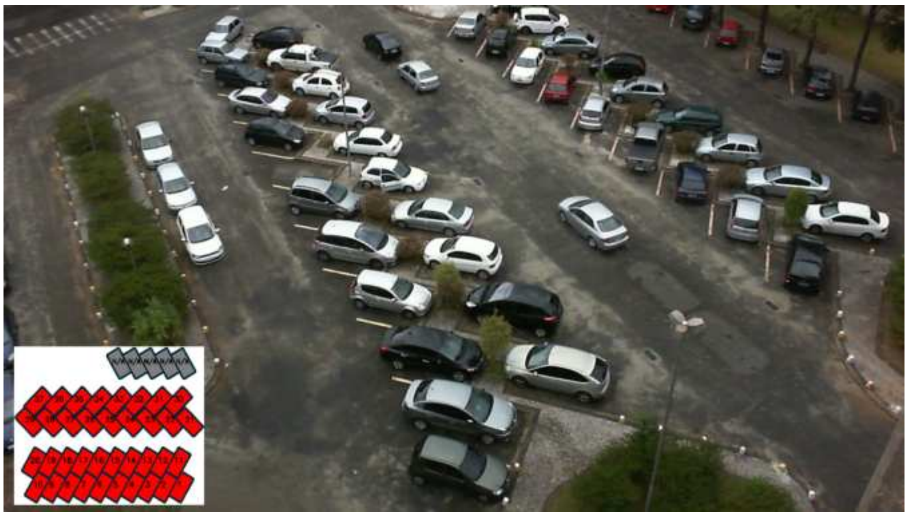


Figure 61: 2013-04-12_15_00_09.jpg

Contributions

1. **Kabir Bertan** - 150 ore
    - All files with author: XMLReader (hpp and cpp), Segmentation (cpp).
    - Minor adjustments on: ParkingSpaceDetector.cpp (functions to calculate caracteristics of a line, function to generate missing bounding boxes, function to standardize a line, function to find the bottom rigth corner of a rotated rect).
    - Ideas used: LSD for line detection, standardization of lines, how to create bounding boxes using two lines, generation of missing bounding boxes, use of a part of mean shift in parking lot status classification, Mog2 for segmentation, augmented dataset, saturation removal and red cars reintegration, background elimination using minimum abs diff.
    - Research on segmentation.
    - Research on OpenCV documentation
    - Research on line detection.
    - Testing on: Line detection(standard preprocessing + houghlines, adaptive bilateral using tiles, CLAHE, template matching, rotation and others), Segmentation(thresholding with BGR/HSV/Lab, template matching) and to improve mIoU for segmentation.
2. **Francesco Colla** - 148 hours
    - All files with author: Graphics (hpp and cpp), Segmentation (hpp).
    - Minor adjustments on: main.
    - Ideas used: machine learning approach in segmentation, background elimination, pixel operations map, putting together two MOG masks, segmentation testing.
    - Research on segmentation.
    - Research on line detection.
    - Testing of the code on the VLab virtual machine.
    - Testing on: background elimination, blob detection, feature matching, GrabCut, clustering, Sobel, super pixels, machine learning approaches for segmentation, Gabor filter for line segmentation.
    - Histogram representation of metrics.
3. **Alessandro Viespoli** - 154 hours
    - Code structure, behaviour and quality check.
    - All files with author: BoundingBox (hpp and cpp), ImageProcessing (hpp and cpp), Metrics (hpp and cpp), ParkingLotStatus (hpp and cpp), ParkingSpaceDetector (hpp and cpp) and main.
    - Minor adjustment on: Graphics (hpp and cpp), XMLReader (hpp and cpp), Segmentation (hpp and cpp).
    - Ideas used: line filters, all bounding box outliers removal criteria, final policy for merging founded bounding boxes, cascade of controls behind car detection.
    - Research on line detection.