IMAGE PROCESSING PROJECT REPORT

Samet Taspinar(st89), Emre Durmus (ed1385)

1. ABSTRACT

Separation of background and foreground concept is used in many applications. One common application is finding the interesting parts of the long videos recorded by surveillance cameras. Two basic methods to separate the background and the foreground are *Mean Subtraction* and *Mixture of Gaussians* methods. These methods do not deal well with illumination changes, shadow effect, and the dynamic backgrounds in the scene. One of the most succesful method for this problem is *Robust Principal Component Analysis* (rPCA) method. In this project, we modified *Incremental Principal Component Pursuit* algorithm which is a derivative of *rPCA* with two different methods, and compared our results with the results of *Mean Subtraction*, *Mixture of Gaussians* and the original *Incremental Principal Component Pursuit*. We saw that our results look visually much more satisfying than the results of the other methods.

Index Terms— foreground detection, background subtraction

2. INTRODUCTION

CCTV and surveillance cameras have widely been used by governments, private companies, and private homes for security purposes as well as for traffic analysis. However, when an incident occurs at an unknown time, it is difficult and laborious for analysts to identify the time as they must watch many hours of video footage. For this purpose, the separation of background and foreground information in such cameras has been an important issue that can help security to speed up obtaining the desired partial information from the video.

Common challenges for background modeling are illumination changes, shadow effect, and the dynamic background. Illumination of the scene may differ under different weathers, or during the different times of the day. If the scene is indoor environment, the light in the environment may be on or off. Shadow of the moving objects may also be detected as foreground object. Dynamic backgrounds such as the shaking leaves of a tree may also be detected as foreground.

Two common methods to separate the background and the foreground are *Mean Subtraction* and *Mixture of Gaussians* methods. These methods do not overcome these challenges. Although *Mixture of Gaussians* model is more adaptive to the background changes than the *Mean Subtraction*, as seen in Figure 1, the *Mixture of Gaussian* model suffers from these challenges.

Another method for separating the background and the foreground is *robust principal component analysis* (rPCA). rPCA is considered as state-of-the-art method for the background and the foreground separation, however, it has high computational cost and memory usage. Since rPCA processes the frames as batches which makes it hard to use it for non-batch/online situations.

In this project, we modified *incremental principal component pursuit* (incPCP) algorithm which is a derivative of rPCA. *incPCP* computes rPCA incrementally which makes it to be both memory and computation efficient. This method also able to quickly adapt the changes in the background. Original output of this method was not satisfying for our datasets. That's why we modified incPCP with two different methods and achieved visually more successful results.

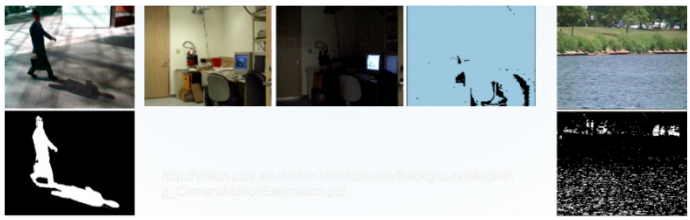


Fig. 1: Examples for Mixture of Gaussians Model

3. BACKGROUND

In this project, we focused on 3 different method for Background-Foreground Separation problem, namely, *Mean Subtraction, Mixture of Gaussian, and Incremental Principal Component Pursuit (incPCP) which is a derivative of Robust Principal Component Analysis (rPCA)*. We have implemented the first two techniques in python, and we have improved the initial implementation of incPCP in Matlab. Since the initial two of them are already well established techniques, we have implemented and analyzed these techniques and didn't try to improve them.

3.1. Mean Subtraction

In this technique, the average of all pixels are obtained using matrix addition and division. Equation 1 show how the average of the frames obtained. Using equation 1, the pixel by pixel average of the whole video is obtained and this model is used for entire foreground extraction process. In this equation, *BG* is background model, *N* is number of frames and *V(i)* is each frame.

$$BG = \frac{1}{N}\sum_{i=1}^{N} V(i) \tag{1}$$

Equation 2 shows the subtraction of i^{th} frame from the background model and its absolute value is obtained. In this equation,

FG(i) stands for foreground of i^{th} frame. The pixel values in foreground which are less than the preset *threshold* of 20 are set to 0 and the brightness of the frames are not guaranteed to be very precise in each frame. In Figure 2, you can see the gray-scaled original frame, and in Figure 3, you can see the foreground found by *Mean Subtraction* method. Although this is a static camera, as you can see, many background pixels were detected as foreground.

$$FG(i) = |BG - V(i)| \tag{2}$$

Fig. 2: Gray Scaled Original Frame

Fig. 3: Foreground by using Mean Subtraction

3.2. Mixture of Gaussians

In this model, the value in each pixel is modeled as mixtures of Gaussian. Then, this model determines the background model based on the persistence and the variance of each mixture[5]. As the new frames come, the model checks the mixtures of the pixels, and if there is a match with a background mixture, then it is considered as background, otherwise, it is considered as foreground.

Equation 3 calculates the probability of seeing the current pixel value. In this equation, t represents the frame number. K is the number of mixtures used which typically is between 3 and 5. $w_{i,t}$ represents the weight of the i^{th} Gaussian. $\mu_{i,t}$ represents the mean of the i^{th} Gaussian. $\sigma i, t$ represents the covariance matrix of the i^{th} Gaussian. η represents the probability density function of Gaussian[5]. η can be seen in Equation 4.

In Figure 4, you can see the original frame, and in Figure 5, you can see the foreground found by *Mixture of Gaussians* method.

$$P(X_t) = \sum_{i=1}^{K} w_{i,t} * \eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \tag{3}$$

$$\eta(X_t, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X_t - \mu_t)^T \Sigma^{-1} (X_t - \mu_t)} \tag{4}$$

Fig. 4: Original Frame

Fig. 5: Foreground by using Mixture of Gaussians

3.3. Incremental Principal Component Pursuit (incPCP)

IncPCP is an improvement in Robust Principal Component Analysis(rPCA). To understand incPCP, it is important to understand rPCA. rPCA is a modification of the widely used statistical procedure of principal component analysis (PCA) which works well with respect to grossly corrupted observations. A number of different approaches exist for rPCA, including an idealized version of rPCA, which aims to recover a low-rank matrix L_0 from highly corrupted measurements [1].

The rank of a matrix indicates the number of linearly independent components whereas the sparsity stands for the number

of zero elements in a matrix. Equation 5 shows the main formula of rPCA in which neither the rank of the low rank matrix nor the sparsity is known. However, this procedure aims to recover both matrices approximately or precisely if possible.

$$M = L_0 + S_0 \tag{5}$$

On the other hand, incPCP is an extension of rPCA in which the goal is the same, to achieve the separation of low rank and sparse matrices. To do this Equation 6 has been used and a Matlab implementation of the algorithm has been shared[4]. The benefit of incPCP compared to rPCA is that it doesn't require computationally expensive the initialization stage. It doesn't require to compute whole PCA in each stage, instead, it adds new frame to the already calculated set, and *incrementally recalculates* the changes which makes the algorithm quickly adapt to the changes in the new frame. The initialization step takes less than a second and for up to 480p videos, the program can run on real time according to the paper[4].

$$argmin \left\|L\right\|_* + \lambda * \left\|S\right\|_1 s.t. D = L + S \tag{6}$$

where *D* stands for a static video.

Equation 6 can easily be applied to background subtraction problem when the video is static. Because many frames will be approximately equal to the background and most of the frames will be sparse after background subtraction, the low-rank matrix can be considered as background, and the sparse matrix as the foreground.

However, solving Equation 6 directly is computationally expensive. To solve this problem Rodriguez et al. proposed solving the following equations[3]:

$$L_{k+1} = argmin \left\|L + S_k - D\right\|_F s.t. rank(L) = t \tag{7}$$

$$S_{k+1} = argmin \left\|L_{k+1} + S - D\right\|_F + \lambda * \left\|S\right\|_1 \tag{8}$$

The equation 7 is the computationally expensive part of the entire code, however, it can be efficiently computed using Lanczos method when t is small using PROPACK library[2].

The pseudo code for this algorithm is as follows[2]:

Algorithm 1

$initialization$
$S_1 = 0, D = inputvideo, rank = 1(initialrank)$
for $k = 1, 2, outerLoops$ **do**
    solve Eqn. 7 with t = rank (save singular values to *v*)
    if $v_{rank} / \sum k = 1 rank > \tau$ **then**
        $++rank$
        solve Eqn. 8
    end if
end for

Consider a frame $V_{\text{r, c, d}}$ where *r* is the number of row, *c* is columns, and *d* is dimensions or channels. By vectorizing $V_{\text{r, c, d}}$, we can obtain $V_{\text{r} * \text{c} * \text{d, 1}}$ vector which has $r * c * d$ rows and 1 column.

Since Mean Subtraction and Mixture of Gaussian are basic techniques for this research topic, we have implemented them and found the results in Figures 2, 3, 4, and 5. Our main contribution is on incPCP, therefore, we explain it in the *Experiment Section*

4. EXPERIMENT

For this project, our project TA, Chenge Li, has given us two sets of images which were obtained from two different videos. These videos are used for testing and comparing our results with her results. For the sake of ease for explanation, we will call these videos as *sunny.avi* and *cloudy.avi* both of which are videos recorded by traffic cameras during sunny and cloudy weathers, respectively.

4.1. Pre-processing

The *sunny.avi* is not a static video where the hand jitters of the video takers caused the results to be bad that most of the video was considered as foreground.

4.1.1. Video Extraction

Because the original video was not stable, we needed needed to obtain it from the images. We used *ffmpeg* to create video that we give the *sorted frames* as input. We used the output video for stabilization.

4.1.2. Video Stabilization

a) ffmpeg Stabilization: *ffmpeg* has *deshake module* which is used for stabilization. This module finds the SURF features between two consecutive frames and the rotation, translation, scaling (*i.e. affine parameters*) between them. After finding these parameters, it zeros out the operations by inverse matrix of the original affine matrix. For example, if the second image is rotated 0.1 degrees, *deshake module* rotates the second image -0.1 degrees and then closes the black regions by symmetry. Based on the experiments, this solution was found not to be very effective.

b) YouTube Stabilization: After some research we found that YouTube stabilizer has the best stabilization technique available. However, the drawback of this stabilizer is it crops the video by approximately $5 - 10\%$ from all the edges. In our discussion with Chenge, we agreed this is an acceptable issue for the project. *sunny.avi* was uploaded to, stabilized, and then downloaded from YouTube.

4.2. Improvements in incPCP

For the sake of brevity, we will, mostly, use *cloudy.avi* video frames in this section, however, we will add some figures from *sunny.avi* as well. Since the original code is implemented in Matlab, all our work is done in Matlab.

4.2.1. Step 0: Default View

In this step, we setup the environment and run the code for the videos. The original code returns RGB image of both L and S matrices and the background of the sparse matrix is set to gray (*i.e.* 0 *in sparse matrix is set to gray in RGB image*). The resulting S and L matrices have $r*c*d$ rows and 1 column. Therefore, it is necessary to convert these matrices to original dimensions. The implementation [4] contains a module which does the normalization of the sparse and low-rank matrices.

Another important parameter in the original code was the *history* parameter which, by default, is set to 200 that indicates the maximum number of frames to is used to calculate the L and S matrices from the *SVD*. Figure 6 shows the 20^{th} frame of the *cloudy.avi* whereas Figure 7 shows the output sparse matrix of the incPCP which corresponds to foreground.

Fig. 6: 20^{th} frame of cloudy.avi

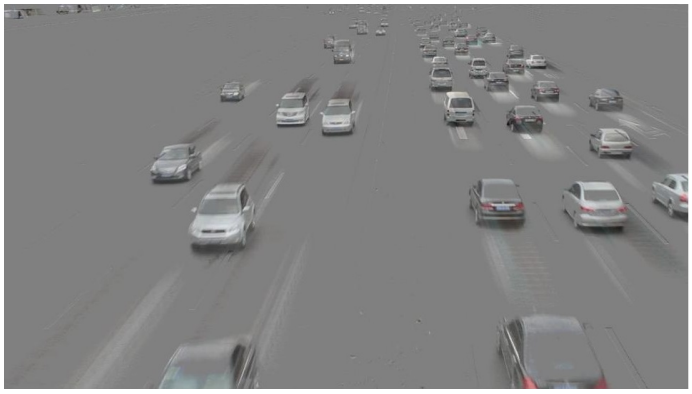
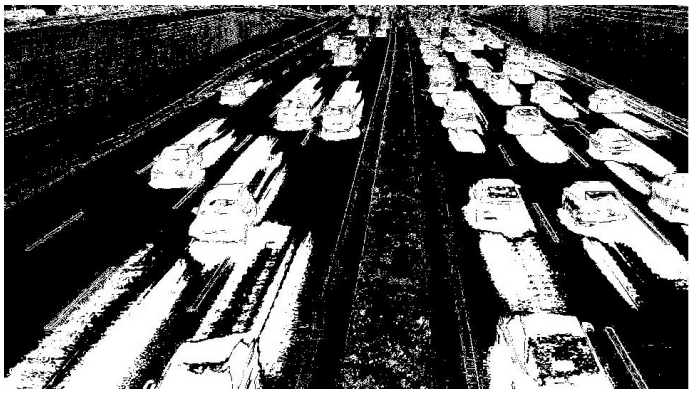


Fig. 7: Output sparse matrix of default incPCP

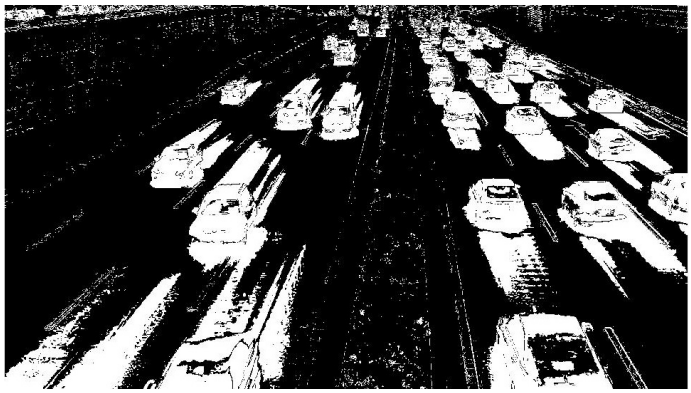
The output in Figure 7 is misleading; since it is an RGB image, the slow-motion (ghost) effects are not clearly seen in the output. Because the default history parameter of incPCP is quite high, the ghost effects remain in the entire sparse output.

4.2.2. Step 1: Shorter History, Black and White Output

To avoid the misleading output of the default incPCP code, we converted the sparse matrix from RGB into BW. To do this, all the non-zeros in the 3-channel S matrix are mapped to 1, and the $zeros$ are mapped to 0 in BW image. To decrease the ghost effect, we decreased the history parameter to 50 from 200. This changes helped us to have slightly less motion effect, and to clearly see the *real output* of the incPCP.



(a) S in BW space, History = 50



(b) S in BW space, History = 10

Fig. 8: Different History Values

Figure 8a shows the output image in black and white color space when the history parameter is 50 . In this frame (*i.e.* 20^{th} frame of the cloudy), approximately 31% of the sparse matrix is non-sparse. This is a very crucial problem that need to be fixed. Furthermore, there is non-continuity in the moving objects. On the other hand, Figure 8b shows the figure when *history* was set to 10 frames. In this case, 24 percent of the entire frame is non-sparse which shows the effect of the *history* parameter.

4.2.3. Step 2: Thresholding

In this step, we added a threshold value to sparse matrix; instead of mapping all the non-zero values to 1, we decided to map only the values whose absolute value is above threshold, τ which was set to 0.35. This value is experimentally found by deciding on between discontinuity of the moving objects vs. ghost effect. However, a further research in a big data can be done to see the optimal threshold value.

Figure 9 shows the output after the operations mentioned above. The main problem in this figure is discontinuity in the moving objects, moreover, there are still small connected components in the figure.

Fig. 9: Thresholding

4.2.4. *Step 3: Median Filter, Connected Components*

We used median filter to get rid of some of the small dots and then, we found the connected components in the image. This connected components corresponds to moving objects. Figure 10 shows the connected components in the image.

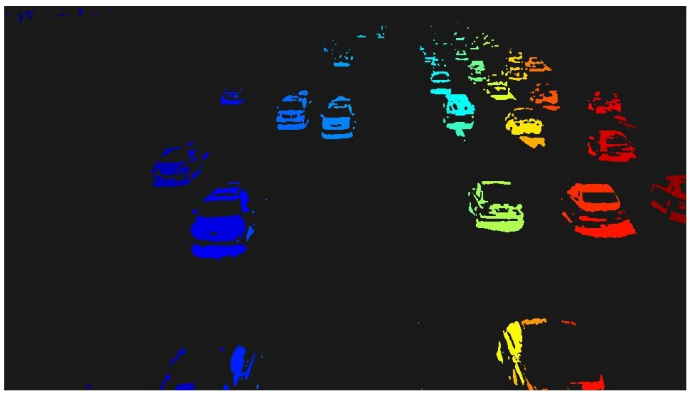


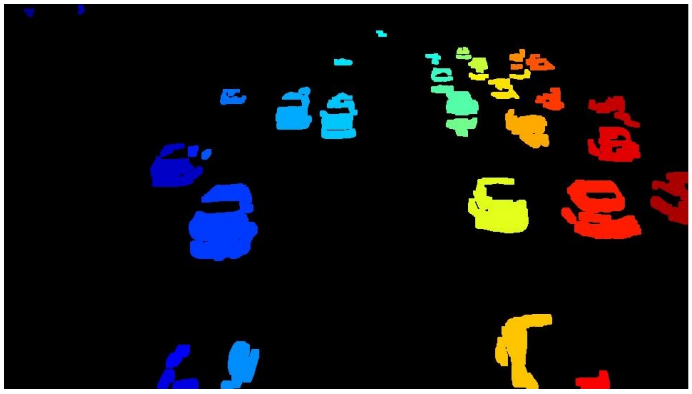
Fig. 10: Connected components

4.2.5. *Step 4: Frame Division, Eroding, Dilating*

In this step, our aim is to remove small connected components and create a better structured set of moving objects. However, objects which are further away from the camera are very small in comparison to closer objects. Thus, we implicitly divided the frames into 4 different horizontal blocks. In the uppermost block, the dilation and eroding parameters were smaller than the bottom sub-frame. Table 1 and Table 2 show the erode and dilate parameters for each of the blocks.

Figure 11a and Figure 11b show the resulting connected components after erode and dilate operations, respectively. It is important to note that the vertical dilation is higher than horizontal dilation, this is because in most cases, there are horizontal discontinuity in the cars which is observed in many frames. To avoid combining the cars next to each other, we were conservative on horizontal dilation.

(a) After Eroding



(b) After Dilating

Fig. 11: Dilate and Erode Operations

4.2.6. *Step 6: Adding Bounding Box*

Since the connected components, in previous step, are visually appealing and relatively good, we add bounding boxes to all the connected components in the real frame. Figure 12a shows the resulting 20^{th} frame with bounding boxes.

Figure 12b shows another frame in which the implementation is not as successful as the previous figure. It can be seen there are many *inner boxes* which needs to be adjusted.

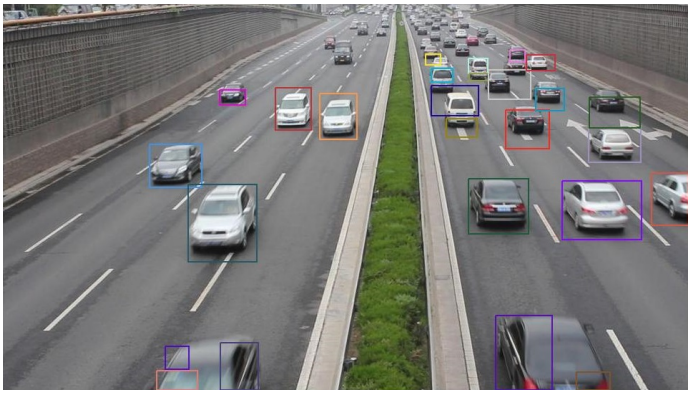
4.2.7. *Step 7: Removing Inner Rectangles*

Since we know the position of each bounding box, we decided to eliminate the inner bounding boxes. Given a set of rectangles, we removed all the bounding boxes which lie inside another rectangle. We set a experimental threshold of 60%, which indicates if 60% of one rectangle is inside another one, that rectangle is removed.

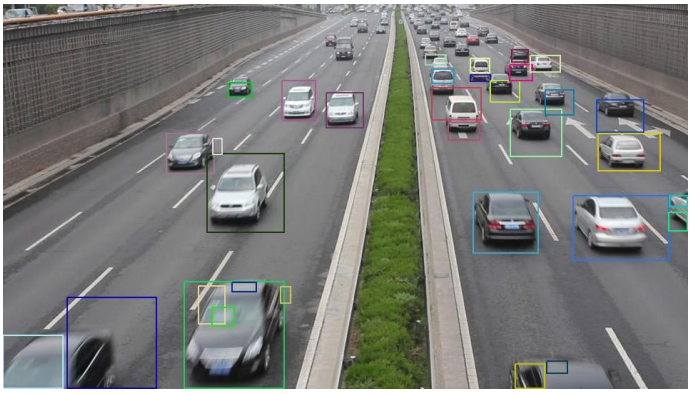
Figure 13 shows the output images after removing vertical boxes. However, we realized another problem, namely, uncombined neighboring bounding boxes.

4.2.8. *Step 8: Combining Vertically Neighboring Bounding Boxes*

Similar to previous step, we found all the consecutive neighboring bounding boxes, and then combined them. The combination

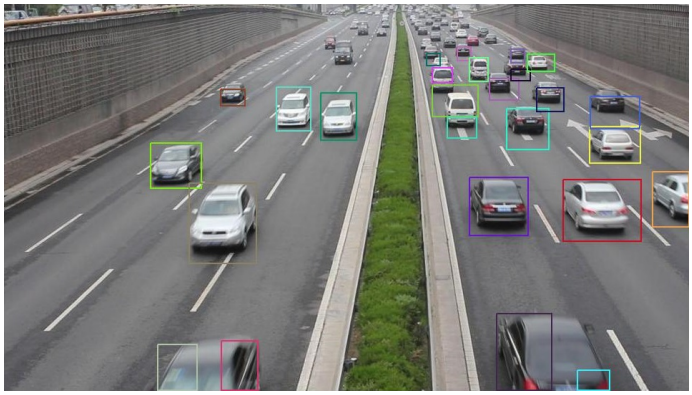


(a) After Adding Bounding Boxes

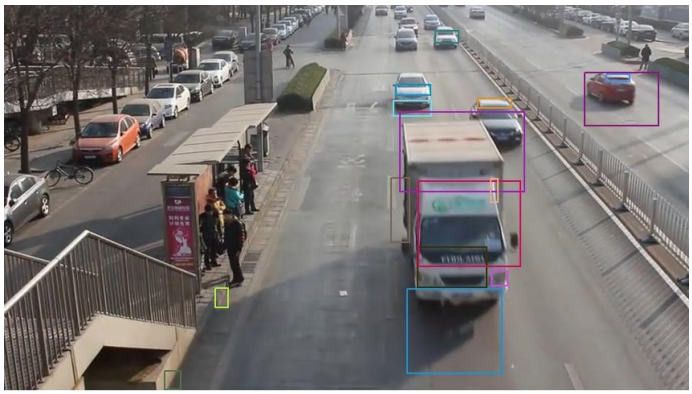


(b) Problem: Inner boxes

Fig. 12: Adding bounding boxes



(a) After removing inner boxes



(b) Problem: Consecutive Vertical Boxes

Fig. 13: Removing inner boxes

is done vertically, and, again, we used 60% parameter for combining. If top/bottom edge of smaller box is less than 60% of the corresponding edge of bigger box, we didn't combine, otherwise, we combined them. Also, if horizontal distance between them is more than 5 pixels, we didn't combine them.

Figure 14 shows the output of this step in which the bounding boxes over the truck has been combined.

4.3. Method 2

In this method, we used a different approach than the first method. Basically, we found the individual blobs, and group them together. In this method, we gave 10 as the history parameter to overcome the shadows.

4.3.1. Step 1: Binarizing the Sparse Matrix

In the first step, we binarized the sparse matrix which is the original output of the incPCP algorithm. Original sparse matrix has negative and positive values. We observed that it is using negative values for the black color regions in the original frame. We got the absolute values of this matrix, then we made the pixels with intensities below 0.35 black, and the rest is mapped to white. Figure 15 shows the original sparse matrix, and Figure 16 shows the binarized version.

4.3.2. Step 2: Dividing the Frame

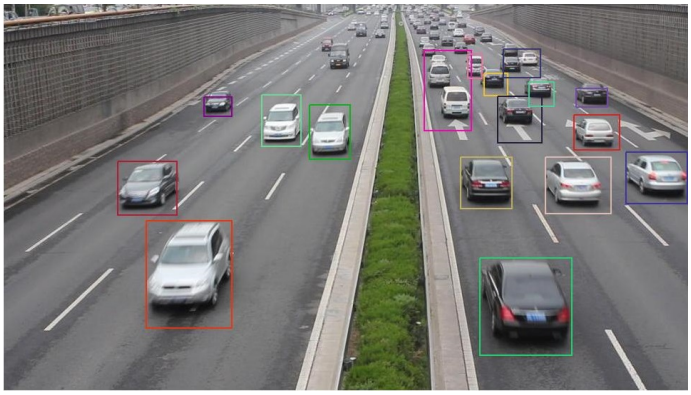
In the second step, we applied median filter to remove small white dots in the binarized frame. Then, we horizontally divided the frame into 4 equal-sized pieces. The sizes of the cars and the distances between the cars get smaller as the cars get further from the camera, that's why we need to work individually on these separate pieces with different appropriate thresholds. Figure 17 shows the divided pieces of the frame.

4.3.3. Step 3: Eroding, Dilating

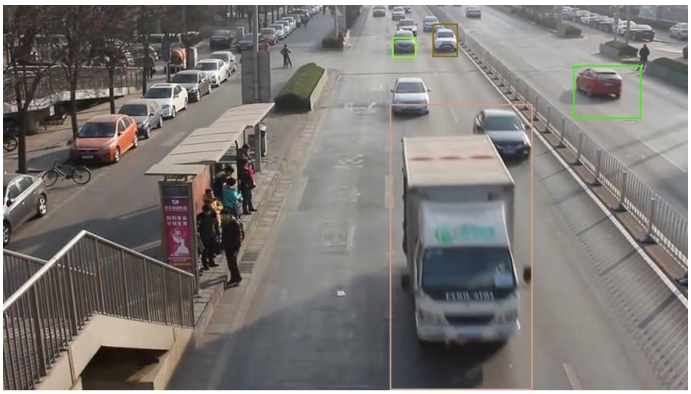
In this step, we applied Matlab *imerode* and *imdilate* functions to the individual pieces with different shapes and parameters to remove the small white parts and to cover the some of the holes in the cars respectively. Figure 18 shows the result of *imerode* and the *imdilate* functions. The parameters and the shapes used for each piece of the frame can be seen on Table 1 and Table 2. We numbered the pieces from top to bottom.

4.3.4. Step 4: Finding Blobs and Centroids

After eroding and dilating the pieces, we found all blobs and their boundaries in the frame by using *bwboundaries* function. After founding the blobs, we used *regionprops* function to find their centroids. Figure 19 shows the centroids of the blobs by red dots.



(a) Final Image



(b) After Combining the Vertical Neighbours

Fig. 14: Final Image

Fig. 15: Original Output

Table 1: Erode Parameters

Piece Number	Erode Shape	Parameter
1	square	1
2	square	2
3	square	3
4	square	4

Fig. 16: Binarized Output

Fig. 17: Divided Frame

Table 2: Dilate Parameters

Piece Number	Dilate Shape	Parameter
1	rectangle	[2 1]
2	rectangle	[5 2]
3	rectangle	[8 3]
4	rectangle	[10 4]

Fig. 18: Eroded and Dilated Frame

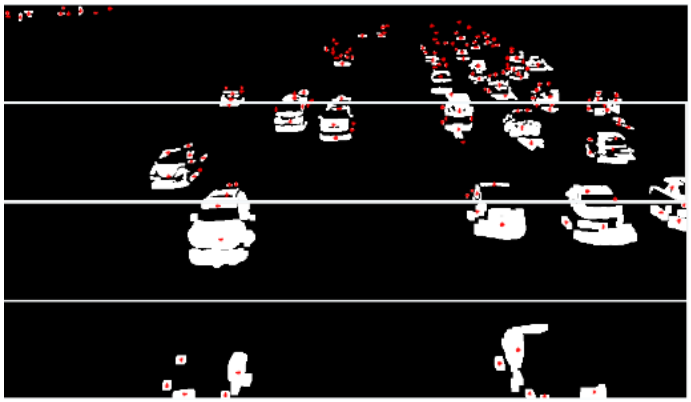


Fig. 19: Blobs and Centroids

4.3.5. *Step 5: Grouping the Blobs*

We created distance matrix between each centroids by finding the *Euclidian Distance* between each pair of the centroids. Then, we grouped the blobs if the distance between their centroids are below the threshold. Figure 20 shows the centroids of the groups by blue dots and the individual blob centroids by red dots. The thresholds used for each individual piece can be seen on Table 3.

Table 3: Distance Thresholds

Pieces	Distance threshold
1	10
2	55
3	90
4	150

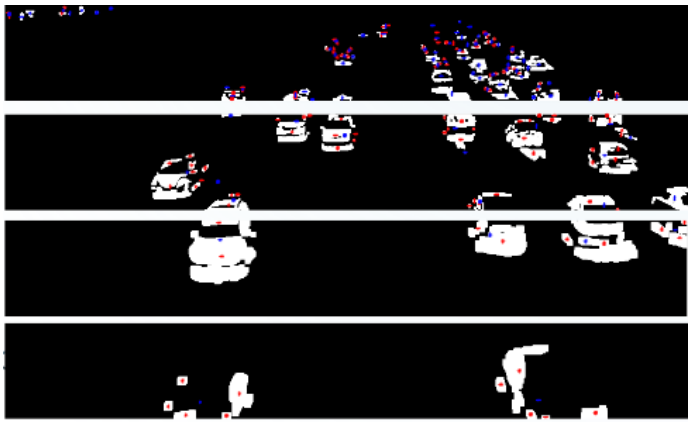


Fig. 20: Groups and Centroids

4.3.6. *Step 6: Adding Boxes*

After creating the groups, we found the group boundaries by using the boundaries of the blobs returned from the *bwboundaries* function. Then, we places bounding boxes according to these boundaries. Figure 21 shows the bounding boxes for each group.

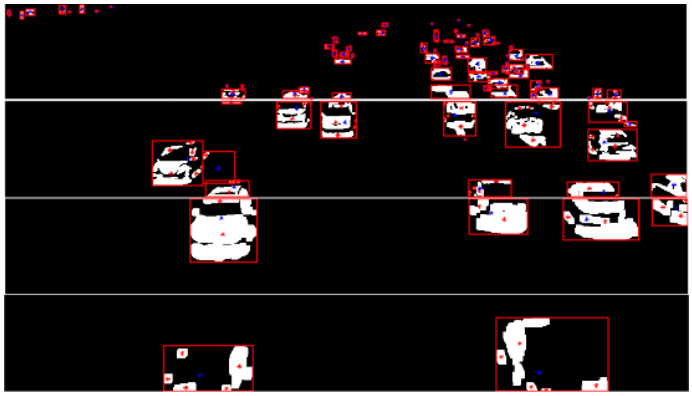


Fig. 21: Bounding Boxes for the Groups

4.3.7. *Step 7: Fixing the Boxes*

In Figure 21, you can see that if the groups belonging to the same car are in different pieces of the fragment, they have separate bounding boxes because we worked in different pieces individually. In this last step of this method, we examined the rectangles across the consecutive pieces of the frame. If we found that they are adjacent to each y direction and they are close enough on the x direction, we created one big rectangle that covers the adjacent boxes. You can see the fixed boxes on Figure 22.

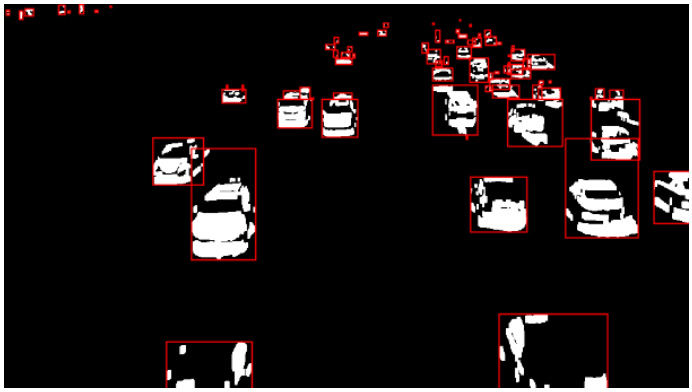


Fig. 22: Fixed Bounding Boxes

4.3.8. *Step 8: Placing Boxes on Original Frame*

In the last step, we placed the rectangles we created on the original frame. Figure 23 shows the final result of the *Method 2*.

5. CONCLUSION

In this project, we implemented background and foreground separation algorithms by using *Mean Subtraction*, *Mixture of Gaussians*, and *Incremental Principal Component Pursuit* methods. We observed that although incPCP has the best background

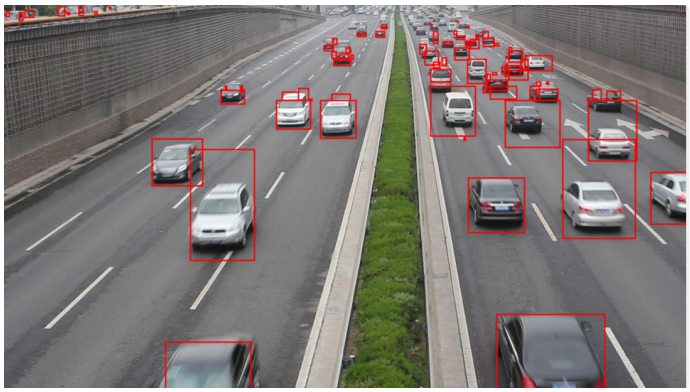


Fig. 23: Final Result Method 2

adaptability, it was not giving satisfactory results for our datasets. Then we modified incPCP with two different methods, and we were able to improve it and get visually better results without sacrificing online processing capability of the method.

We believe that there are still areas that we can improve the results. One of them is adding time association in our model. We know that same car will be in couple of consecutive frames before getting out of the camera's vision. So, by using this heuristic, we can make our model more efficient and more robust. Another one is that we can add histogram checking to our method 2 by grouping the individual blobs. When we find our groups by distance matrix, we can check the original frame to see if the histograms of the blobs are close to each other. Adding histogram checking may add some overhead to our model, but it can make our grouping more robust because usually each car has only one color on their body. Also, the parameters we used in this project were experimentally obtained. These parameters can also be optimized to improve the result.

6. APPENDIX

The videos:

- sunny.avi
- cloudy.avi

The python codes:

- mean_subtraction_gmm.ipynb

The Matlab codes:

- incPCP_method1 - incPCP_method2

References

[1] Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):11, 2011.

[2] Rasmus Munk Larsen. Propack-software for large and sparse svd calculations. *Available online. URL http://sun. stanford. edu/rmunk/PROPACK*, pages 2008–2009, 2004.

[3] Paul Rodriguez and Brendt Wohlberg. Fast principal component pursuit via alternating minimization. In *Image Processing (ICIP), 2013 20th IEEE International Conference on*, pages 69–73. IEEE, 2013.

[4] Paul Rodriguez and Brendt Wohlberg. A matlab implementation of a fast incremental principal component pursuit algorithm for video background modeling. In *Image Processing (ICIP), 2014 IEEE International Conference on*, pages 3414–3416. IEEE, 2014.

[5] Chris Stauffer and W. Eric L. Grimson. Adaptive background mixture models for real-time tracking. *in Proceedings. 1999 IEEE Computer Society Conference on Computer Vision an Pattern Recognition (Cat. No PR00149). IEEE Comput. Soc. Part Vol. 2, 1999*.