CS6640 A2

Jake Bergquist

9/16/2018

All code was developed based on the same video I used for assignment 1. I have included that same video file again in a zip folder called video.zip for this assignment.

1) Develop the CS6640_Laws and explore its use to identify semantic regions in the images from the video with kmeans.

To develop the CS6640_Laws function I first listed out all of the laws arrays and convoluted them according to the following using the conv function:

L7 = [1 6 15 20 15 6 1]
E7 = [-1 -4 -5 0 5 4 1]
S7 = [-1 -2 1 4 1 -2 -1]
W7 = [-1 0 3 0 -3 0 1]
R7 = [1 -2 -1 4 -1 -2 1]
O7 = [-1 6 -15 20 -15 6 -1]

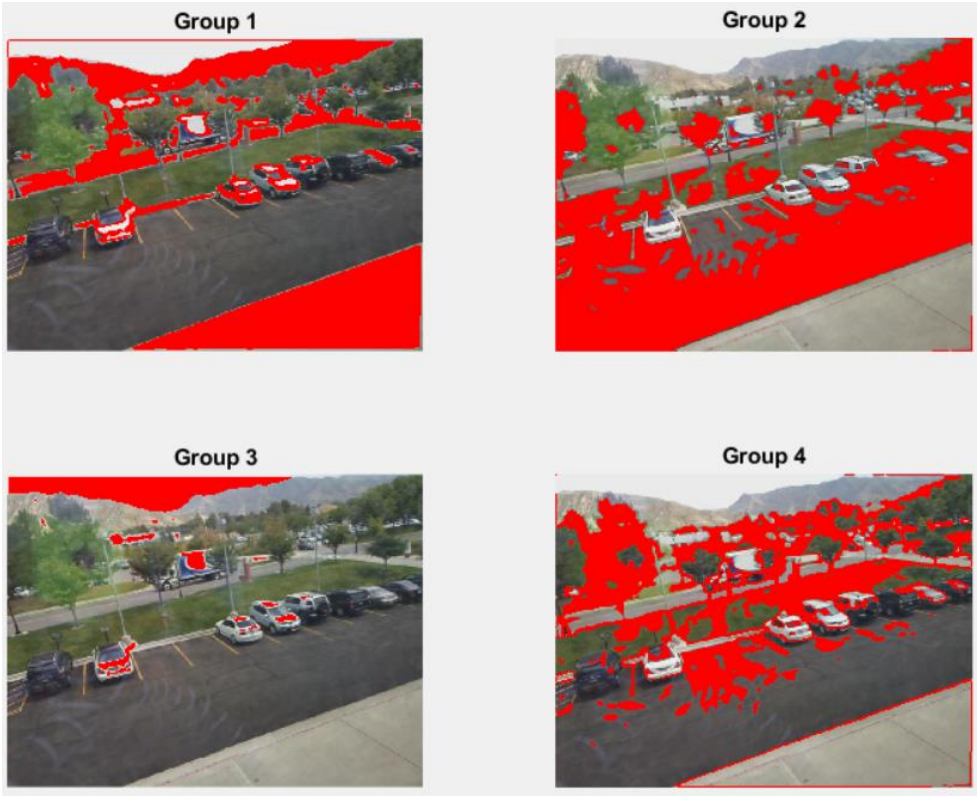
i. L7,L7
ii. L7,E7
iii. L7,S7
iv. L7,W7
v. L7,R7
vi. L7,O7
vii. E7,E7
viii. W7,R7
ix. W7, O7
x. 7 x 7 mean

After doing so I then used the conv2 function to convolve the filter matrices with the input image. The excess edges were trimmed and then filtered again with the means filter (x). The resulting outputs were then inserted in order to the columns of the output m*n X 10 textures array.

After developing the CS6640_Laws I ran the function on frame 2 from my video (**Figure 1**) and utilized kmeans() with k = 4. Shown in **Figure 2** are the four groupings produced by this grouping.

Figure 1

Figure 2



If we observe we an see that there are some sections that are reasonably well segmented such as the sidewalk in the foreground (group 1) and the ski (group 3). Group 1 also shows isolation of the road in the middle ground. Additionally group 2 and 4 combined would give good isolation of the asphalt. Lets now try looking at reducing the number of groups.

Figure 3

With k = 3 groups we find that the sections isolated by the textures filters are the asphalt, grass, and trees (Group 1), the sky and light reflective car sections (Group 2), and the pavement, street and other smooth textures in Group 3. It would seem that dark and rough textures fell into Group 1. Finally lets try to parse our what the texture field saw into more fine categories. By upping to k = 8 groups we get **Figure 4** where we can see that while the textures filters can see some delineations such as the sidewalk and asphalt it also has a hard time connecting the entire grass and asphalt. However it is possible to see how groupings such as 6 7 and 8 could be used to delineate the cars from the parking lot. Additionally group 2 shows highlighting of mostly white cars and the sky, which could identify the cars potentially.

Figure 4

2) Using quadratic and harris operator to calculate registration transforms for the video frames

As an opening statement I would like to say that it is VERY VERY unlikely that the video frames would need registrations. Circumstances that would cause this would be the glass in front of it flexing due to wind or temperature, or problems with the camera, or if the camera was bumped or moved. The camera captures the images and accounts for any distortions within the camera.

With that in mind:

First I developed the CS6640_Harris() function apply the Harris operator to a given input grayscale image. To calculate this operator for a given image I first computer the gradient of the image using the gradient function. This produces a gradient dx and dy of the image. By again using the gradient function on each of these, the dx and dy, we obtain dxx dxy and dyx dyy. Using these we can produce the Harris matrix for each pixel using the column and row values (c,r) for each pixel according to **Equation 1**.

Equation 1: H = [dxx(r,c), dxy(r,c); dyx(r,c), dyy(r,c)]

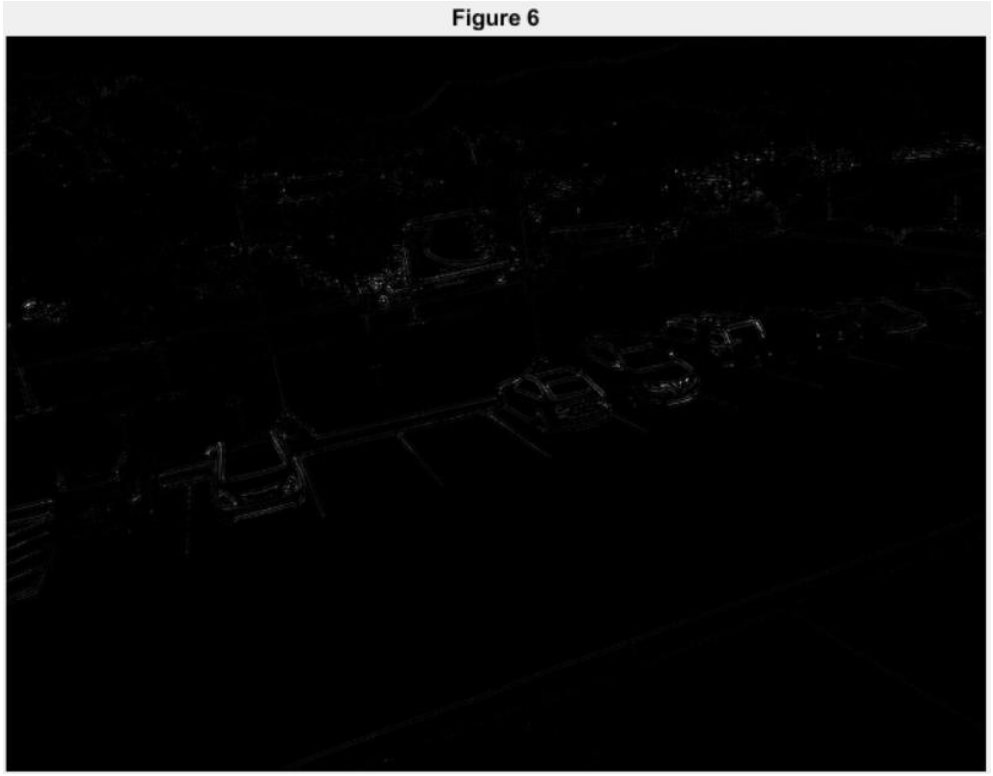
With the Harris matrix determined for each pixel we then compute the Harris value for each pixel according to **Equation 2**.

Equation 2: HarrisValue = 0.05*trace(H)^2

This value is then assigned at the same pixel location as the pixel used to calculate it. The resulting image is then normalized by subtracting the minimum value, then dividing by the maximum. This normalizes the values to be between 0 and 1. Then the values are multiplied by 255 to scale them to be between 0 and 255 so that we can visualize it as an image.

First we use the grayscale image of Frame 2 of our video, shown in **Figure 5**, for our Harris function. This produces **Figure 6**.

Figure 5

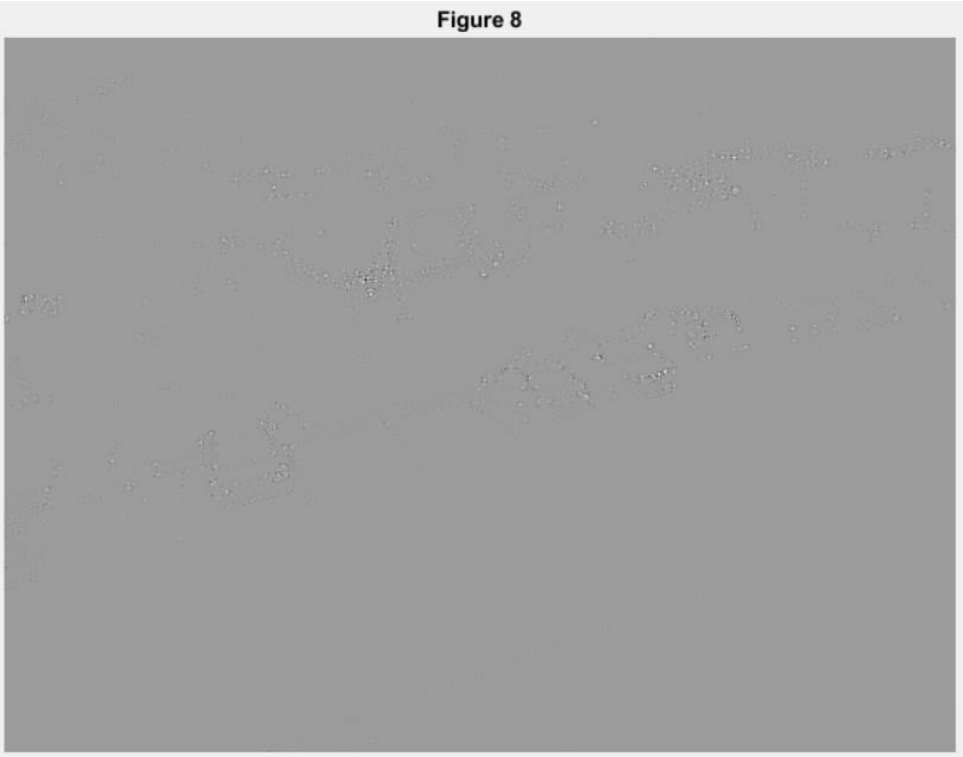


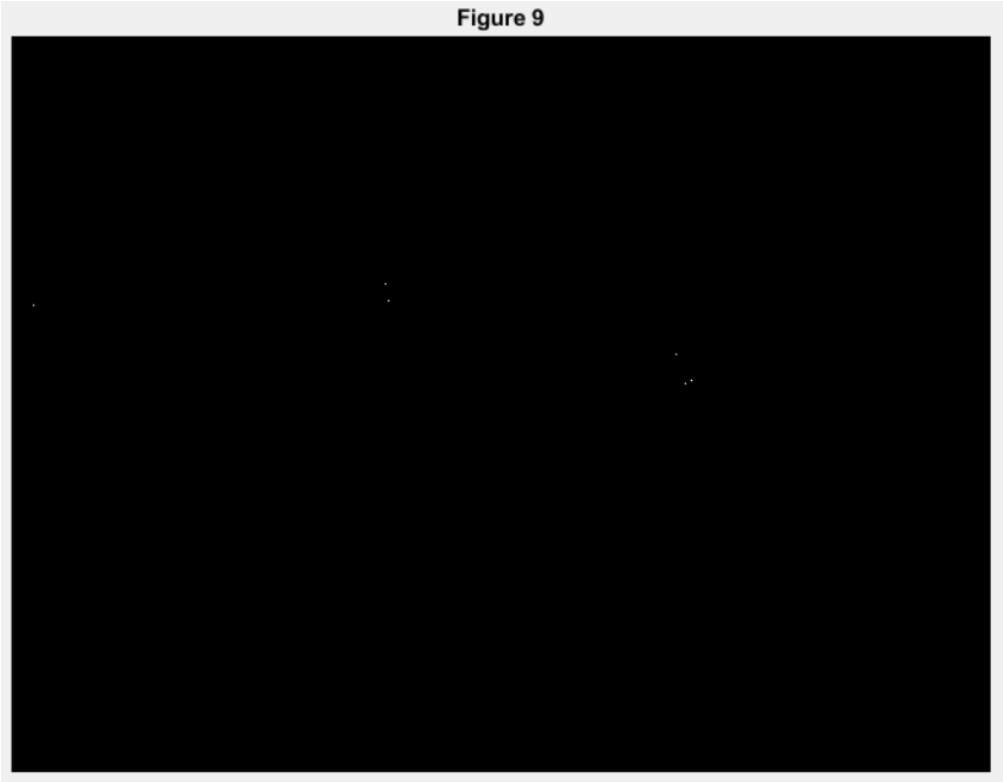
We can see in **Figure 6** the major features and many minor features of the image have been outlined by the Harris operator. These outlines clearly show the shapes of both the parked cars and the moving one on the road. It could be feasible to use these outlines to mark the edges of an area of interest that was determined via other methods. Then a flood fill could be used to fill in the area of interest.

If we threshold **Figure 6** at 120 and mark those points we find what the Harris operator would determine as potential correspondence points. Markers are placed over these locations in **Figure 7**.

Figure 7

However, it was later announced that the proper formula for calculating the Harris Value was not the one shown in **Equation 2** but rather the one shown in **Equation 3** below. Utilizing this calculation (which is the one presented in the final CS6640_Harris() function) on the grayscale **Figure 5** we get **Figure 8**. This image can be hard to interpret but if we look closely we can see that there are some bright spots in particular around the tight corners of the edges of the parked cars. If this image is then thresholded at H_im > 230 we get **Figure 9** which shows several more clearly highlighted areas. To make this more clear these highlights are dialiated by 5 and overlayed on the original gray image (**Figure 10**).





In **Figure 10** we see highlighted the areas that were super threshold. Unfortunatly two of them are on a moving vehicle on the road, and thus they would make bad correspondence points for registration. Of the remaining four points, all seem usable, although three are grouped rather close to each other (on the van in the foremost parking lot) which is undesireable. Overall there are simply not enough correspondence points to perform a

quadratic or affine registration, thus the threshold will have to be lowered. However four is enough points for a rigid transformation as that only requires solving for four constants.

Figure 10

Figure 11

Figure 11 shows the same Frame 2 but with the Harris threshold at 200. This gives us a few more correspondence points to choose from. By performing the same operation on Frame 3 of our video we can

identify the matching correspondence points to input into our quadratic or affine registration function. **Figure 12** shows the result of the same operation on frame 3 and **Figure 13** and **Figure 14** show the matching correspondence points between the two images (with a red circle around them for visibility). What should be noted (re iterating my point about how this video does not really need registration) is that the coordinates for the corresponding correspondence points are exactly the same for each set. Thus the transform computed for both the quadratic and the affine should be the identity transform.

Figure 12

Figure 13: Frame 2

Figure 14: Frame 3

When we look at the computed q and A for our quadratic and affine respectively we get the following:

q: [−0.00 1.00 0.00 − 0.00 0.00 0.00 0.00 0.00 1.00 − 0.00 0.00 0.00]

$$A: \begin{bmatrix} 1.000 & -0.000 & 0.000 \\ -0.000 & 1.000 & 0.000 \\ 0 & 0 & 1.000 \end{bmatrix}$$

For the quadratic transformation we can see that these q values result in X' = X and Y' = Y, while for the affine transformation this is the identity matrix. When we view the resulting registered image of frame 2 (**Figure 15** for quadratic and **Figure 16** for affine) we see that the images are practically unchaged from their original state (**Figure 14**). There is some slight perturbance at the left edge of **Figure 15** but this is likely due to floating point error, which can also account for the -0.00 values in both resulting matrices.

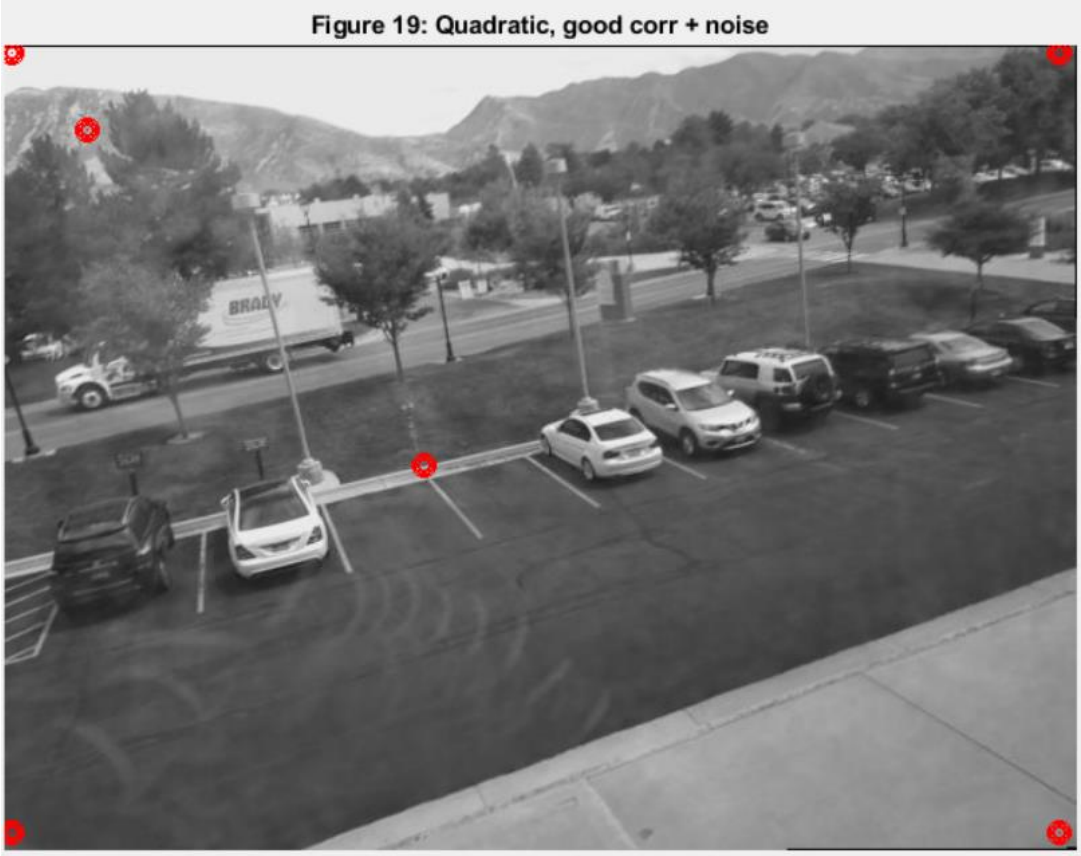
Figure 15

Figure 16

If we are to add a bit of noise to the correspondence points to make them not perfect we result in different transformations. By randomly adding or subtracting 1 to blur the correspondence points we get much poorer transformations in the regions not covered by the correspondence points. In the quadratic transformation seen in **Figure 17** (with red circles added to show correspondence points used) we can see that the image registered well in the area between the correspondence points, but not outside of them. The affine transformation however (**Figure 18**) responded much more robustly to the perturbed correspondence points, and only shows minimal issues on the edges of the image, if any. The unruly nature of the quadratic registration could perhaps be fixed by using more, and better correspondence points.

Figure 17: Quadratic, corr + Noise

Figure 18: Affine, corr + Noise



As can be seen in **Figure 19** when we pick correspondence points that span the entire image the constraining of the registration is much better to result in a better quadratic transformation, even with noisy correspondence points. However, in all it would not seem that the use of registration is not necessary in our videos, as the affine and quadratic transformations we get from these are very close if not identical to the identity transformations.

3) Developing CS660_Tracks() to track car movement in the videos.

In order to develop this function I first read in a video and extracted the frames from that video using the vid.getframe() function. I then reused my CS6640_Background function (copied it in as a sub function) to get the background for the video. Then for each frame I subtracted the grayscaled background from the grayscaled frame. This difference image was then thresholded into a binary image at a threshold value equal to half the max intensity of the difference image. I then used imerode() with a circular kernel [0 1 0; 1 1 1; 0 1 0] to reode away any errant small sections that made it past the threshold. This took care of any movement of the trees or other small amounts of background movement. What was left were major moving components. I then used the bwconcomp() function to identify connected components of super threshold pixels. The largest of these would be any moving vehicles seen in the frame. In order to try to account for things getting in the way of the moving cars (such as the trees blocking the view) I first calculated which connected components were the largest and second largest. I then calculated the Euclidean distance between the centroids of these components. If this distance was below a reasonable threshold (20 pixels) I considered these two components to be part of the same moving body and calculated the centroid of the two together. If the distance was greater I used the larger

component and ignored the smaller one. It should be noted that in this way, if two cars were present, the larger of them would be traced, and not the smaller. I then used the centroid position of my moving vehicle to place a marker using insertMarker(). The resulting image was saved in an m x n x 3 x k matrix where m x n x 3 is the RGB image and k is the frame of the video. The centroid was also saved in the tracks output variable with the first column being x values and the second column being y values of the centroids. Finally the frames were converted to uint8 and saved as a movie object via the immovie() function, which takes the m x n x 3 x k matrix as the input. The output tracks for my video are shown in **Figure 20** plotted on the background image attained via CS6640_Background.

Figure 20: tracks

For my particular video there were a few cars moving. First there was a large white truck, followed by a bus, and two small cars. The two small cars were too small to be recognized as they were moving in the screen at the same time as the bus. The large truck (seen in **Figures 10:19**) was relatively easy to track, although the algorithm did still struggle when a large part of the truck was behind the trees, as can be seen in **Figure 21**. This resulted in a jumpy placement of the centroid and as such a jumpy track in the video. Another interesting note was that the large black windows of the bus made it exceedingly difficult to track, as they did not stand out on the background of the black road. As can be seen in the threshold of the difference image in **Figure 22** and the difference image in **Figure 23**, the bus from frame 16 (**Figure 24**) was difficult to track. This could potentially be addressed by assessing the textures of the bus and following the moving edge of it in addition to using this difference method to outline its shape. In all, while the tracks may seem jumpy, I am satisfied with how the function was able to track the movement of the largest vehicles in the video I used.

Figure 21: Truck behind trees

Figure 22: Threshold of difference

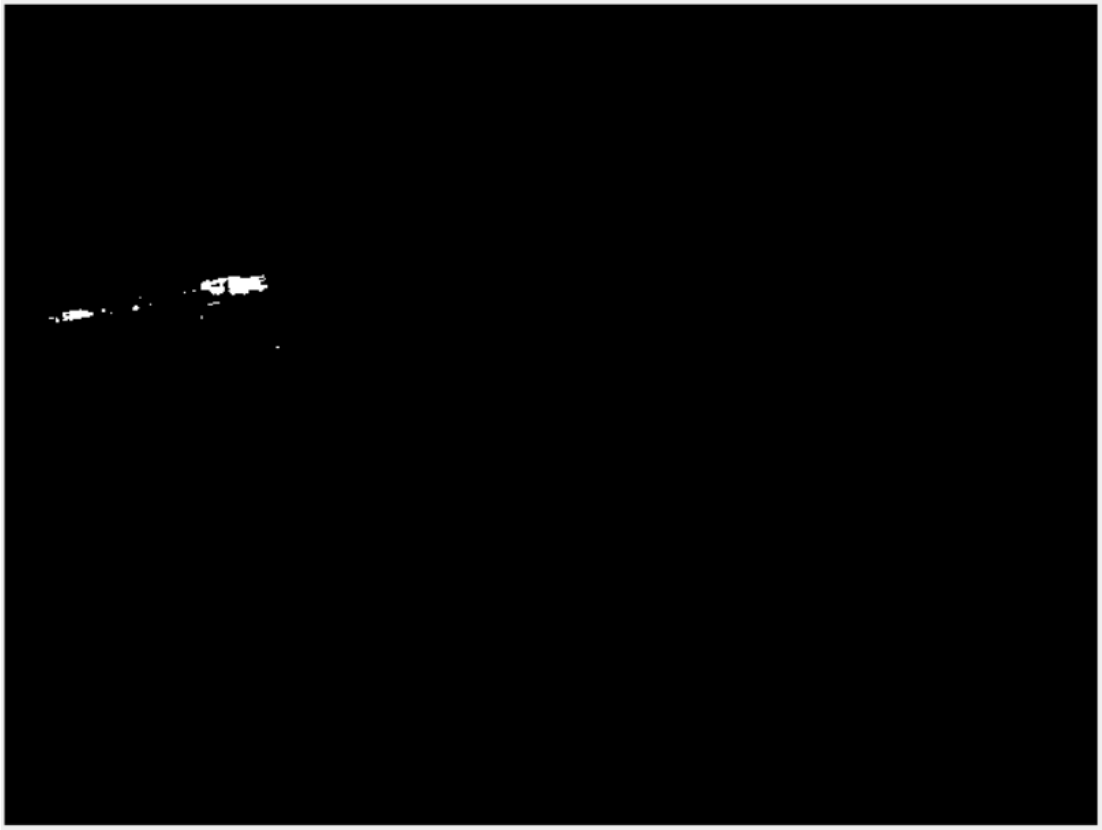


Figure 23: Differeance from Background

Figure 24: Frame 16: Bus