CS6640 A5

Jake Bergquist

11/7/2018

1) To begin with I used my CS6640_MM function to extract my segmentation of the moving vehicles. I modified my MM function to also return an internal structure that has the filled out segmentation, as the movie I usually output is only an outline of the moving vehicles. I then used the bounding box of a single moving vehicle from a frame to select a subset of the car to refine. I then used a pewit and sobel filters to emphasize verticle and horizontal lines. Before filtering the segment outside of the original segmentation was set to 0, and after edge detection, the edge between the Original segmentation and the) area was removed (set to 0). I then added these filter results and binarized them. The resulting binary edges was dialated to connect nearby region, then eroded twice to get rid of small, superfluous edges. This resulting edge binary was used to trim the original segentation as an improvement. The areas were then compared between this segmentation and a hand drawn segmentation. The areas were compared by pixel volume, the edges were compared by eye.

Figure 1 shows the background for this video and **Figure 2** shows frame 11, with two cars. The car in the middle will be the one of our focus. **Figure 3** shows the background subtract image, identifying two moving things. **Figure 4** shows the movie frame for frame 11 where a few moving things are outlined. **Figure 5** shows the binary mask of the segmentation of that frame. We will focus on the car in the middle, seen in **Figure 6**. **Figure 7** shows us zoomed in on the car of interest, cropped to a little outside the bounding box of our original segmentation. **Figure 8** shows the lines within the original segmentation according to our filters. The car outline can be seen relatively easily. **Figure 9** shows the original segment (top) and the trimmed one (bottom). When compared to **Figure 10**, the hand segmentation, we can see that the trimmed segment is improved, including less non car area, but there is still significant improvement to be made. The are of the hand segment is 1430 pixels. The original segment has an area of 2886 pixels. Our trimmed segment has 2168 pixels. While this is closer, it is still quite a bit larger than our original (about 1.5 size, compared to the original segment which is 2x the hand segment). In the line comparison our trimmed segment is still rather blocky and does not follow the curves of the car too well, but the box is tighter on the car than the original. In the line image (**Figure 8**) we can see that the back edge of the car and the wheels are distinct. By querying for these shapes we could follow the curves better. Using active contours on this would also likely tighten in around these curves.

Figure 1: Grayscale Background Image

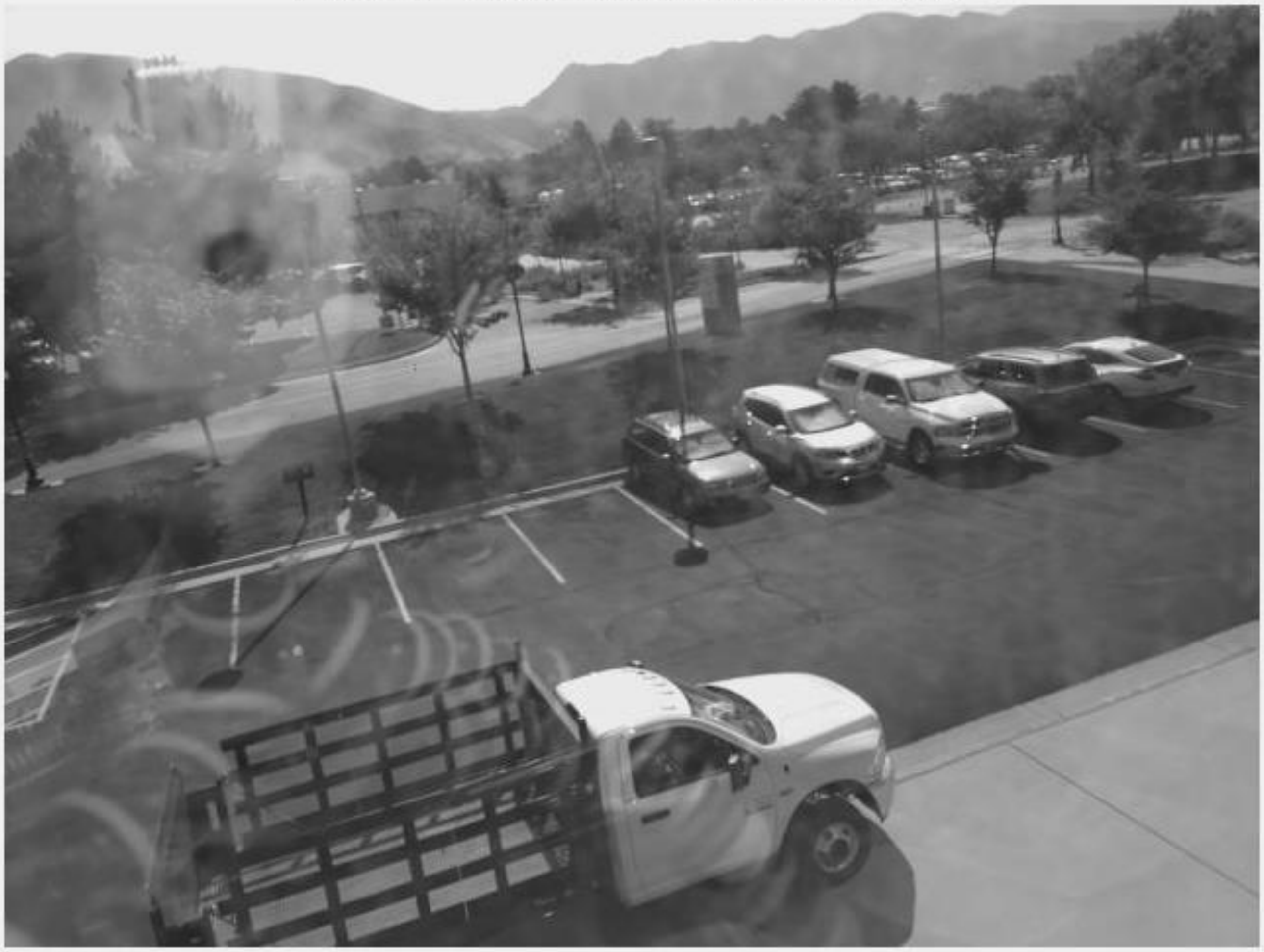


Figure 2: Grayscale Frame 11

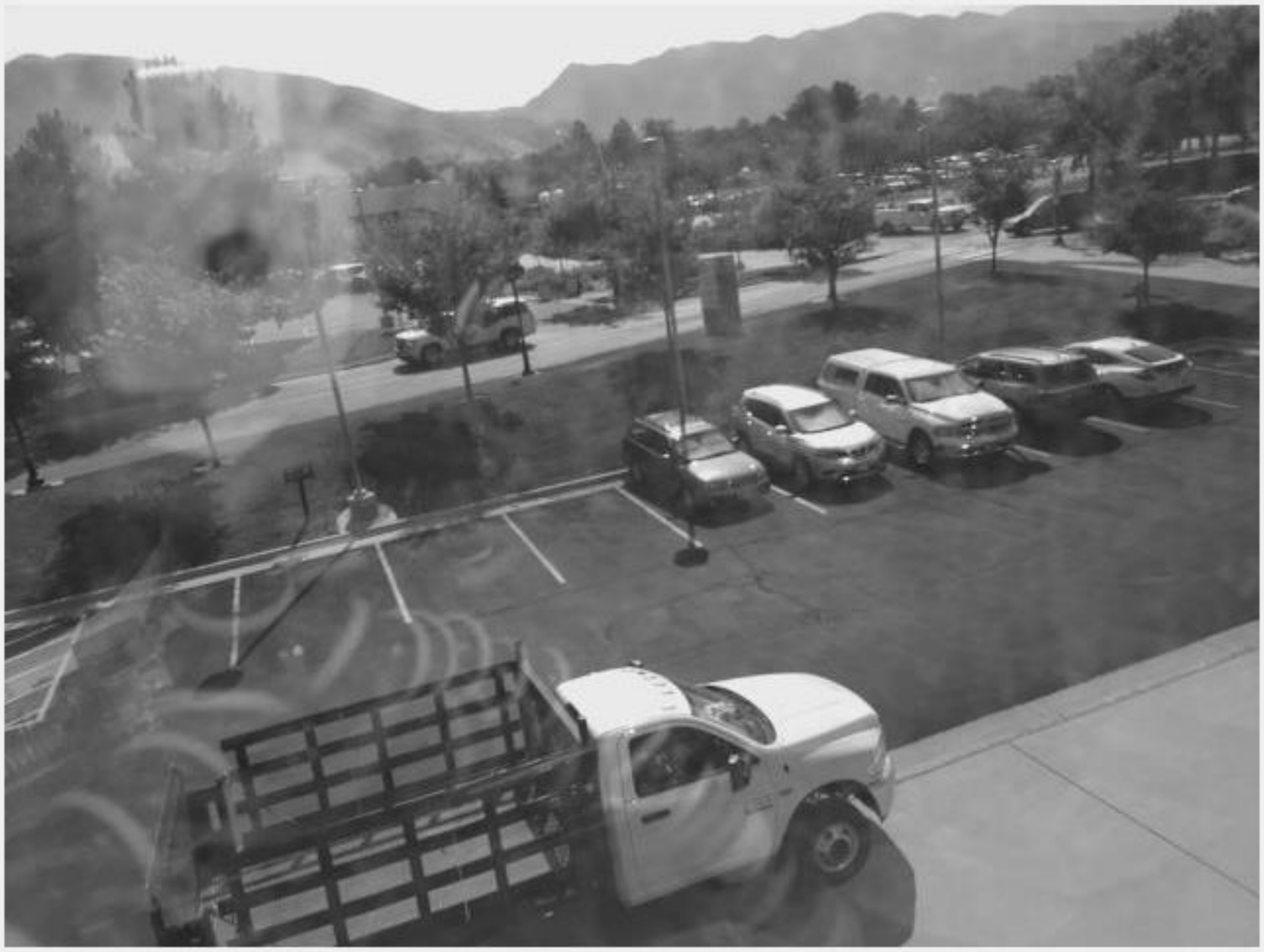


Figure 3: Gray Difference Image

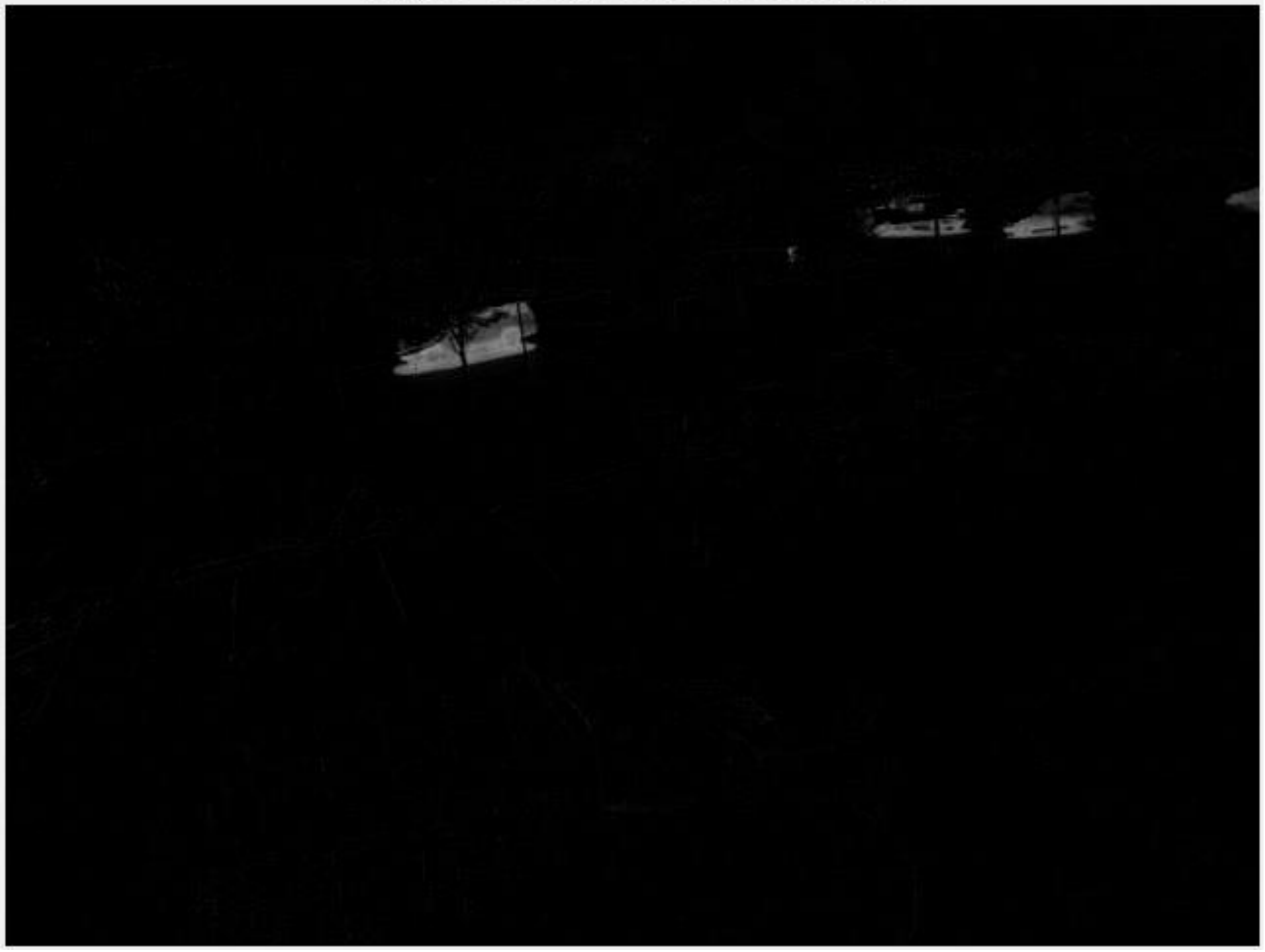


Figure 4: Outlined moving things frame 11

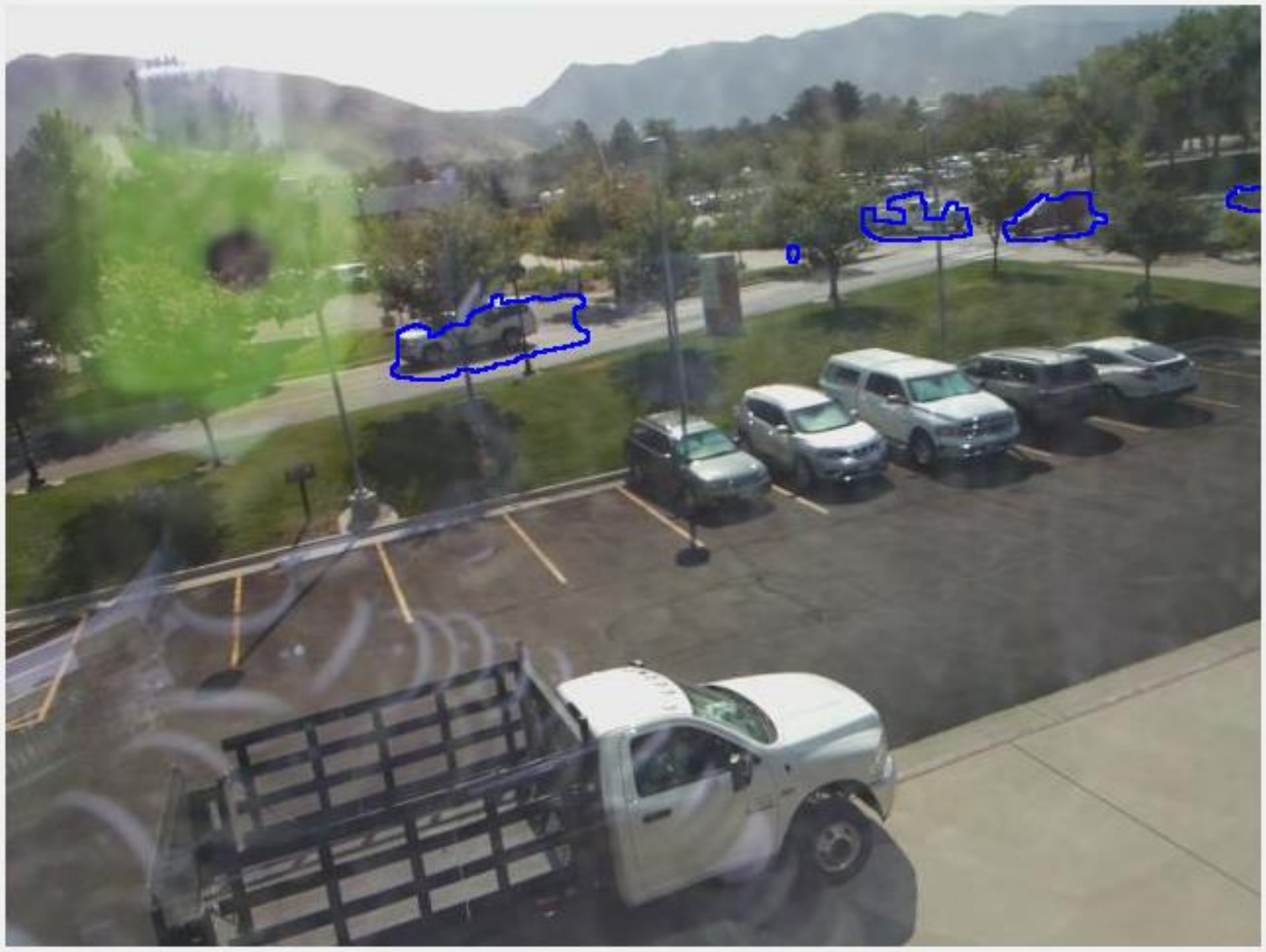


Figure 5: Binary classification of the moving things frame 11

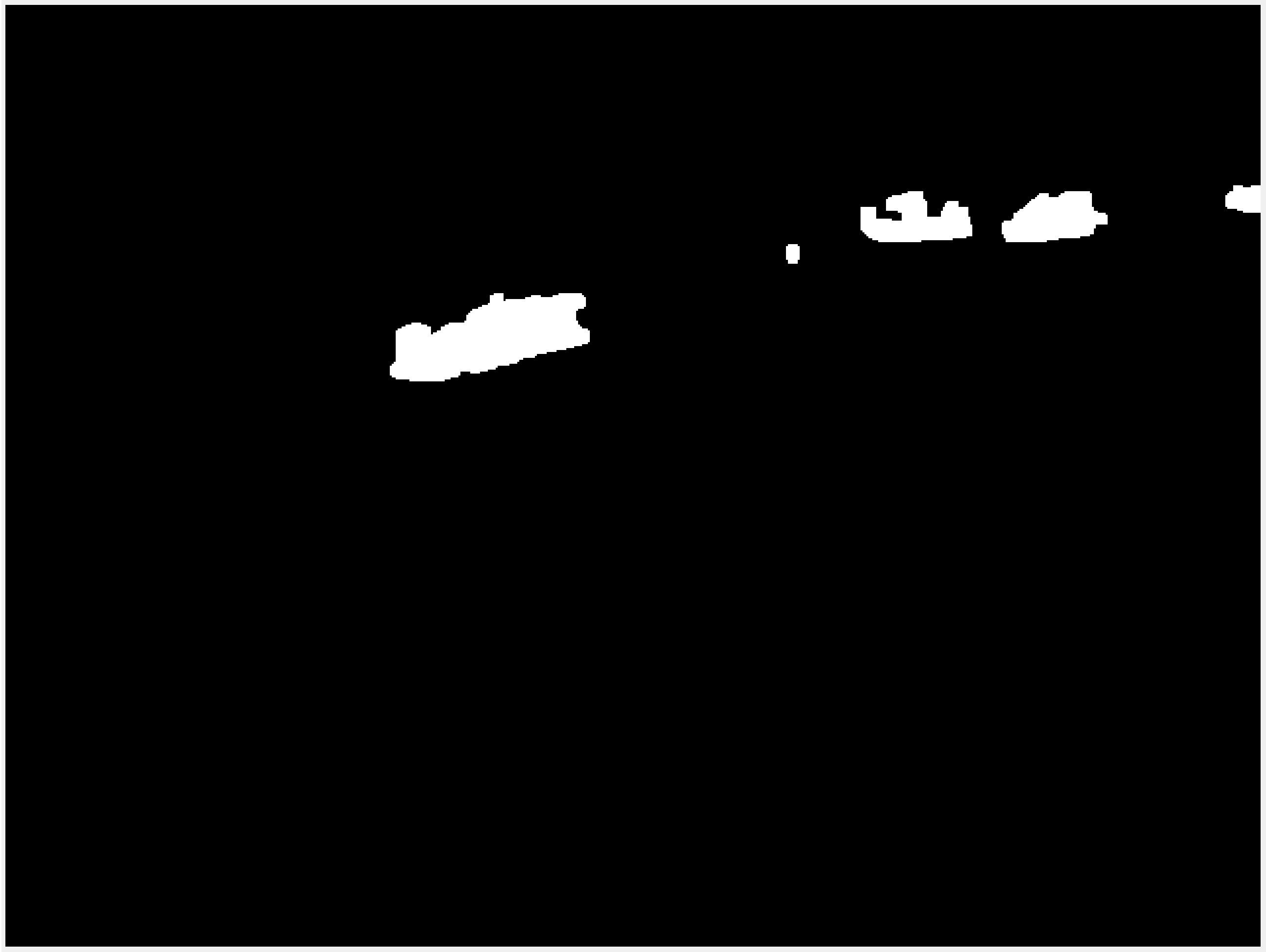


Figure 5: Binary classification of the moving things frame 11

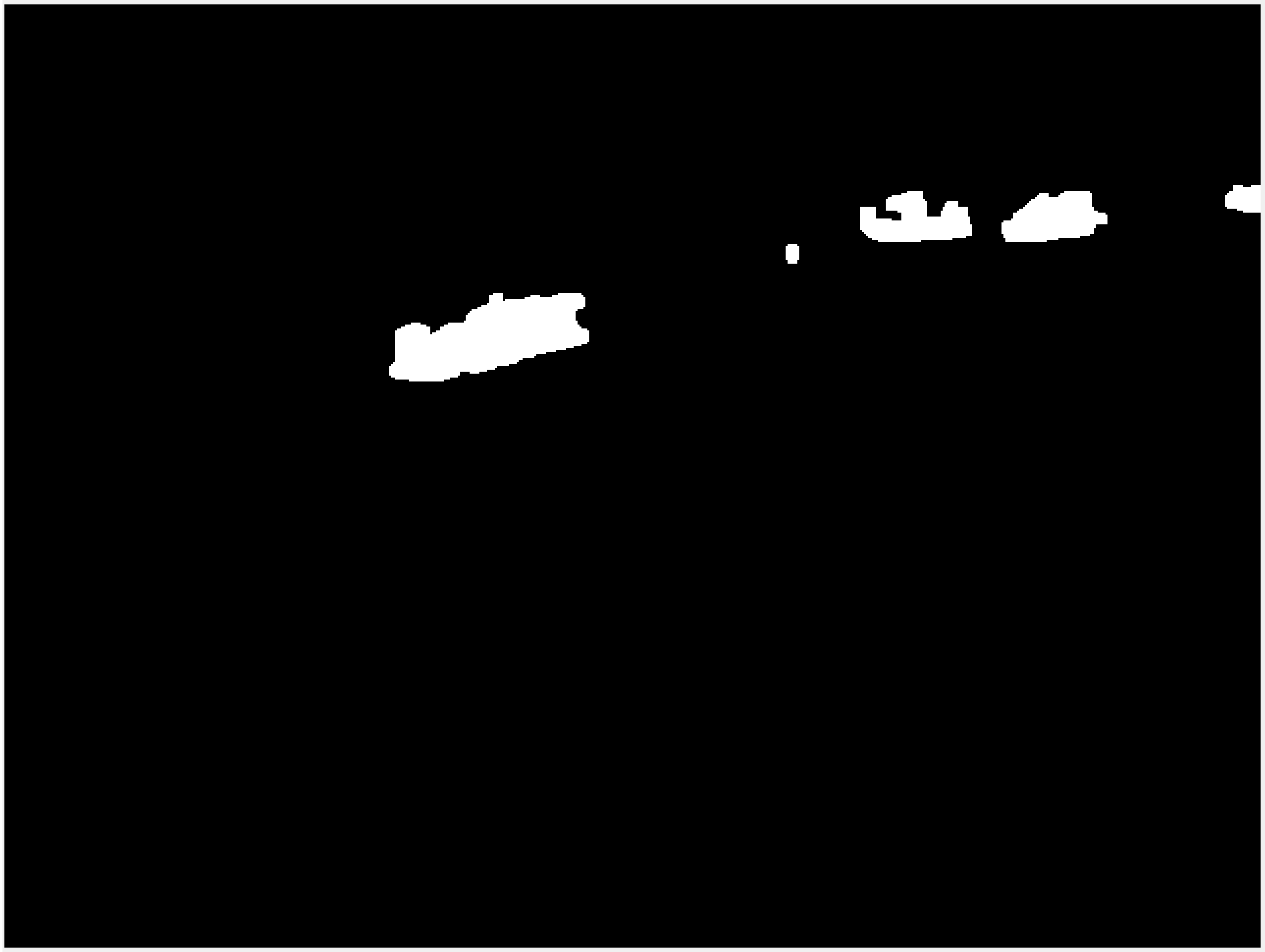


Figure 6: Focusing on the SUV on the road

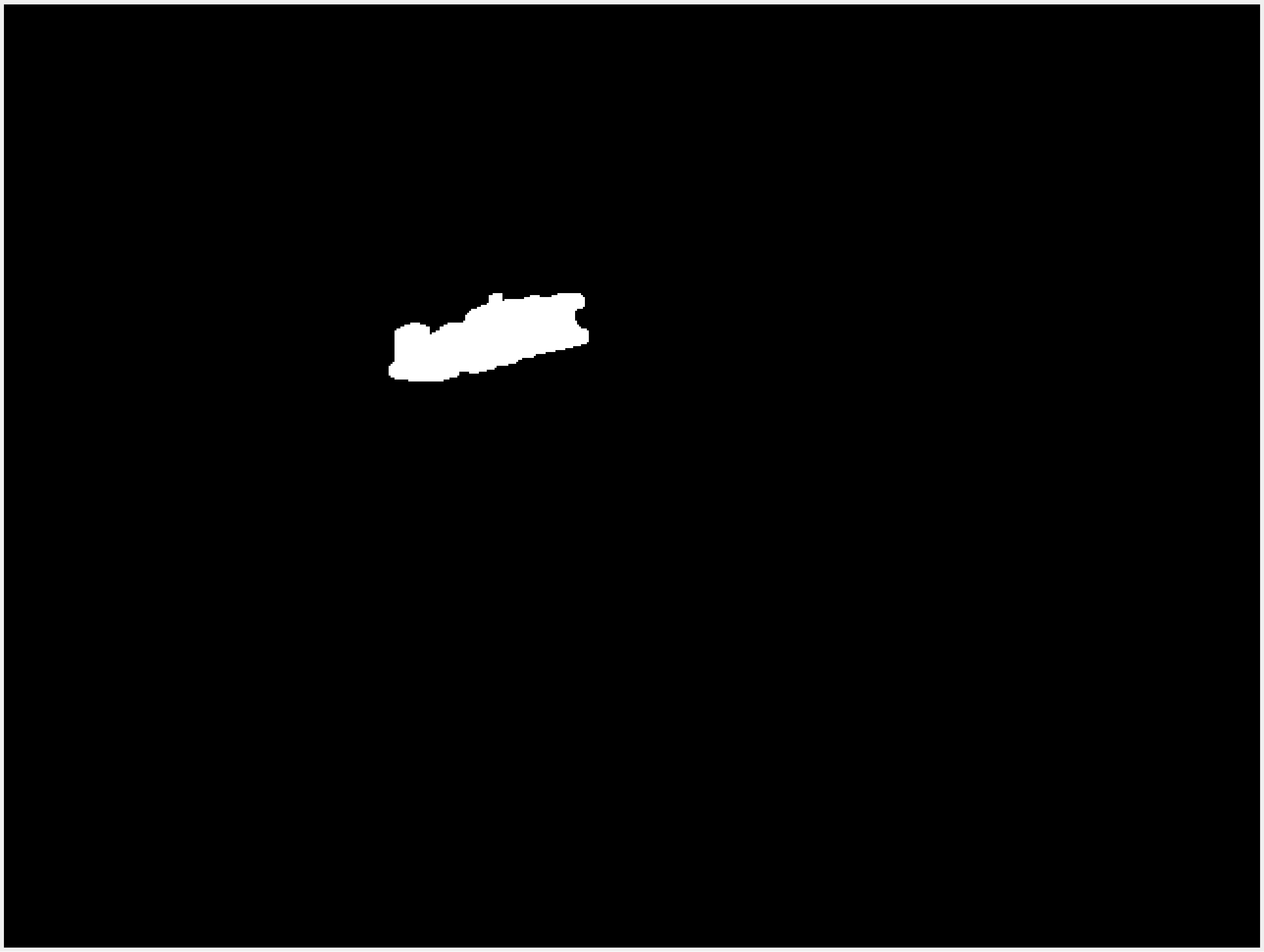


Figure 7: Focusing on the SUV on the road

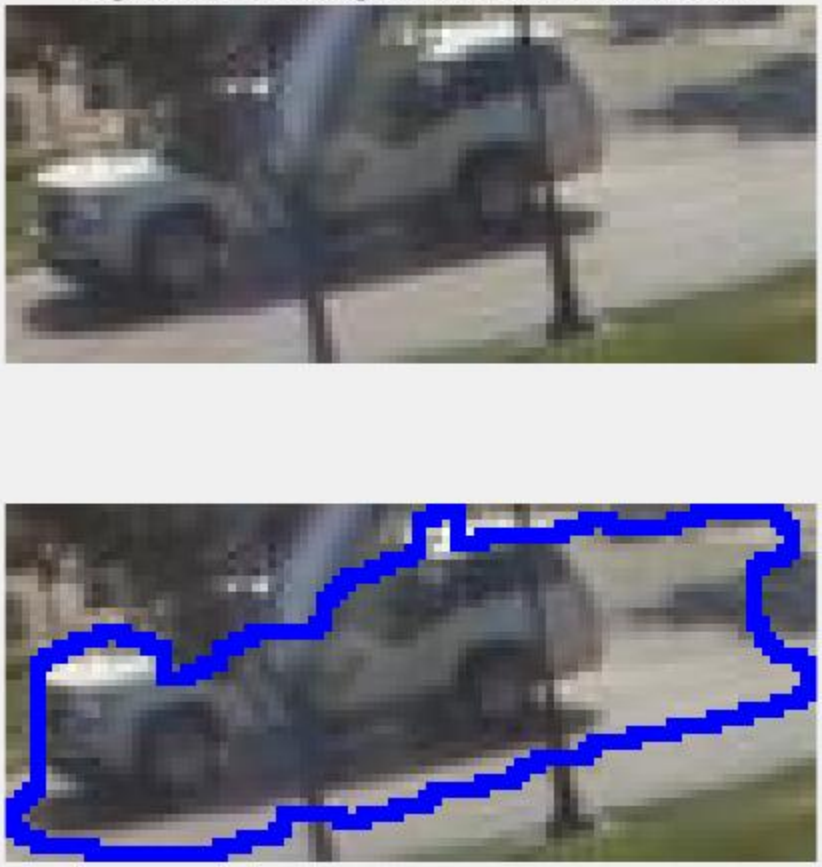


Figure 8: Lines of the SUV

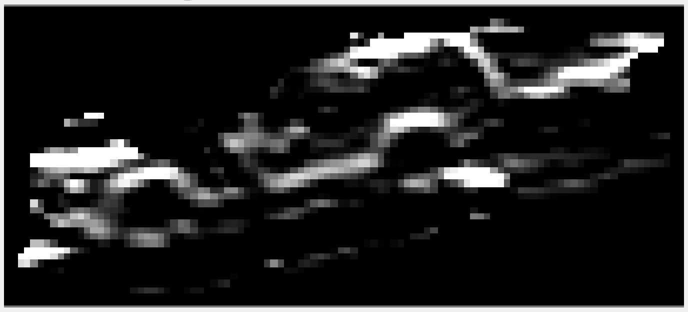


Figure 9: Trimmed truck segment

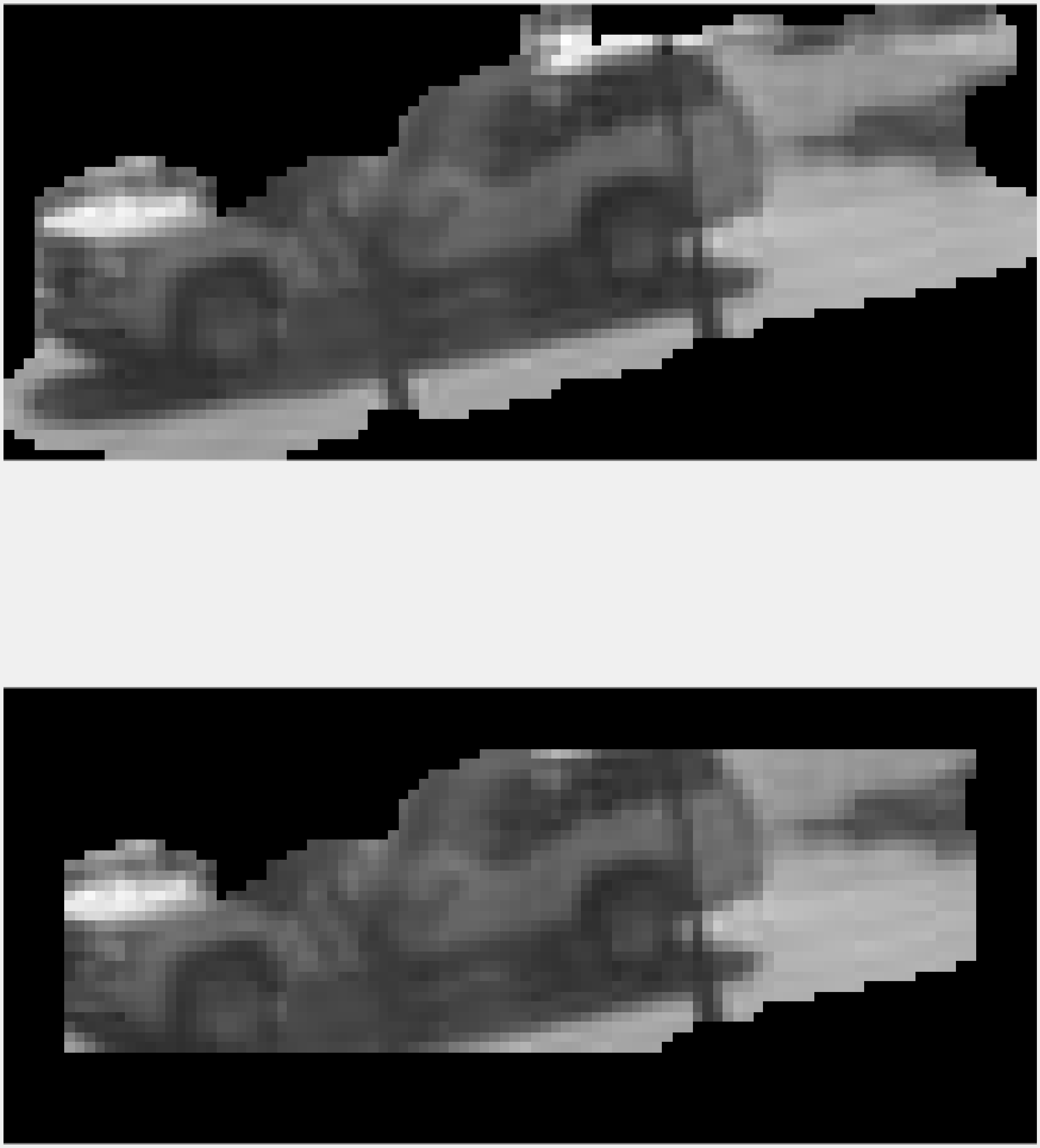
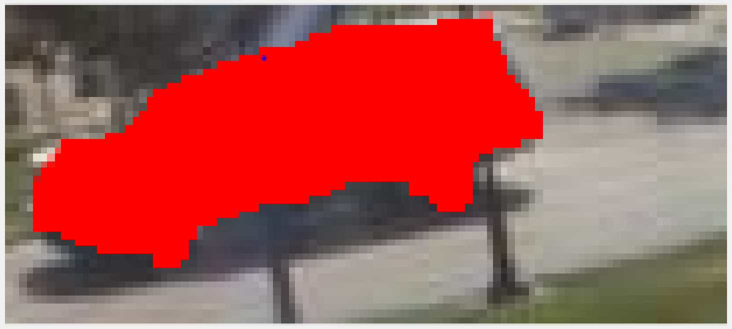


Figure 10: Hand Segmented



2) For the Hough transform of my video I looked at the first 15 frames of our video, where a car passes down the length of the road. For frame I used the segmentations of the moving things, labeled them with bwlabel, selected the one that matched, and recorded its centroid. These centoids were the input for the hough transform. I then used these to cacluate the hough transform. I then looked in the accumulator H from the H transform for the maximum, and used that to get the proper Rho and Theta. I then used those to calculate the line to plot across a background frame of the video.

Figure 11 shows the background with the center of mass of the one moving car labeled with red markers. **Figure 12** shows the resultant hough line drawn from these points. **Figure 12.1** shows a section of the accumulator array where the intersection of the sinusoidal function converge on the proper rho and theta used do draw the line in **Figure 12**.

Figure 11: Centroids of 1 moving car

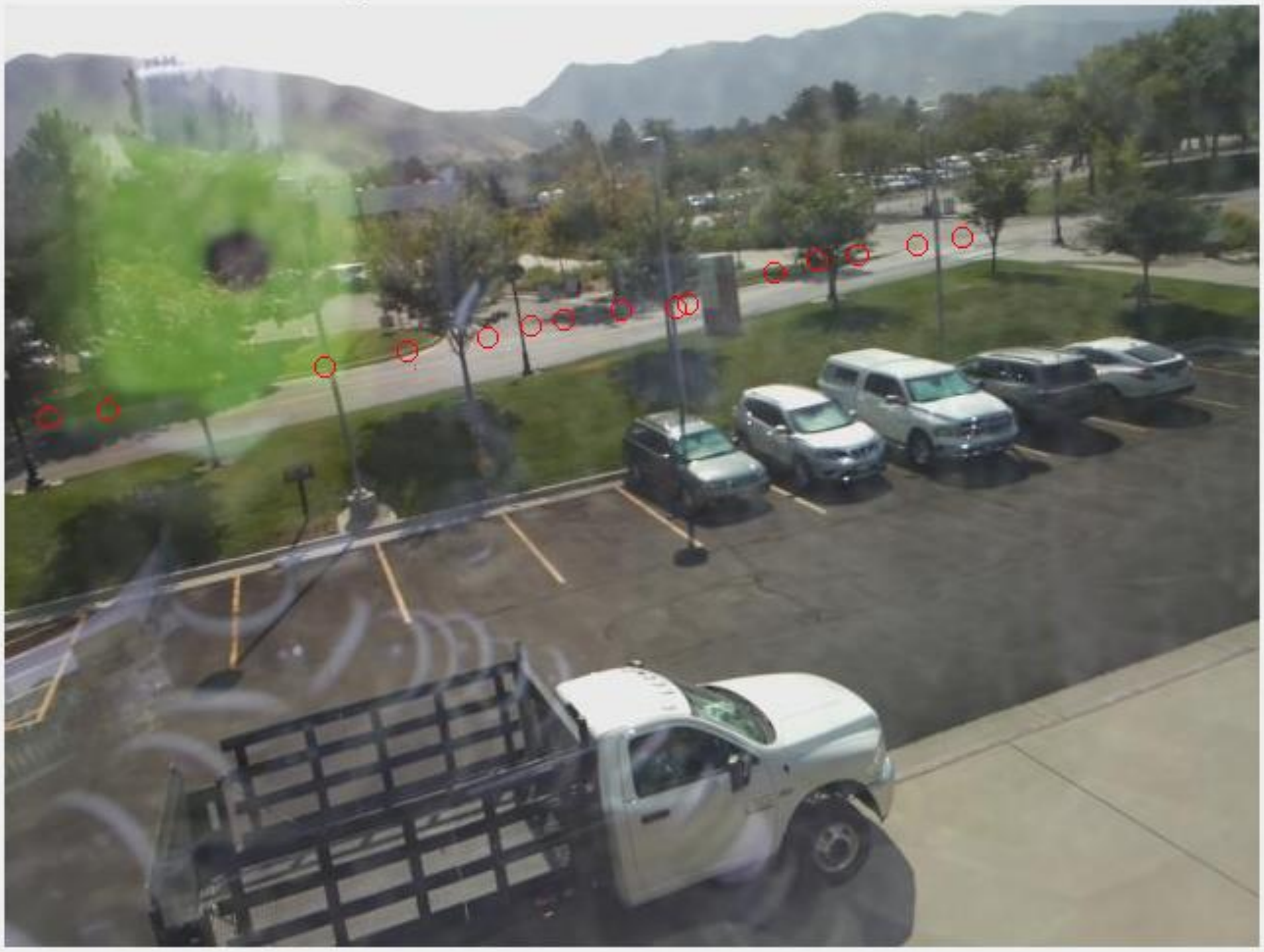


Figure 12: Hough Line of moving Car

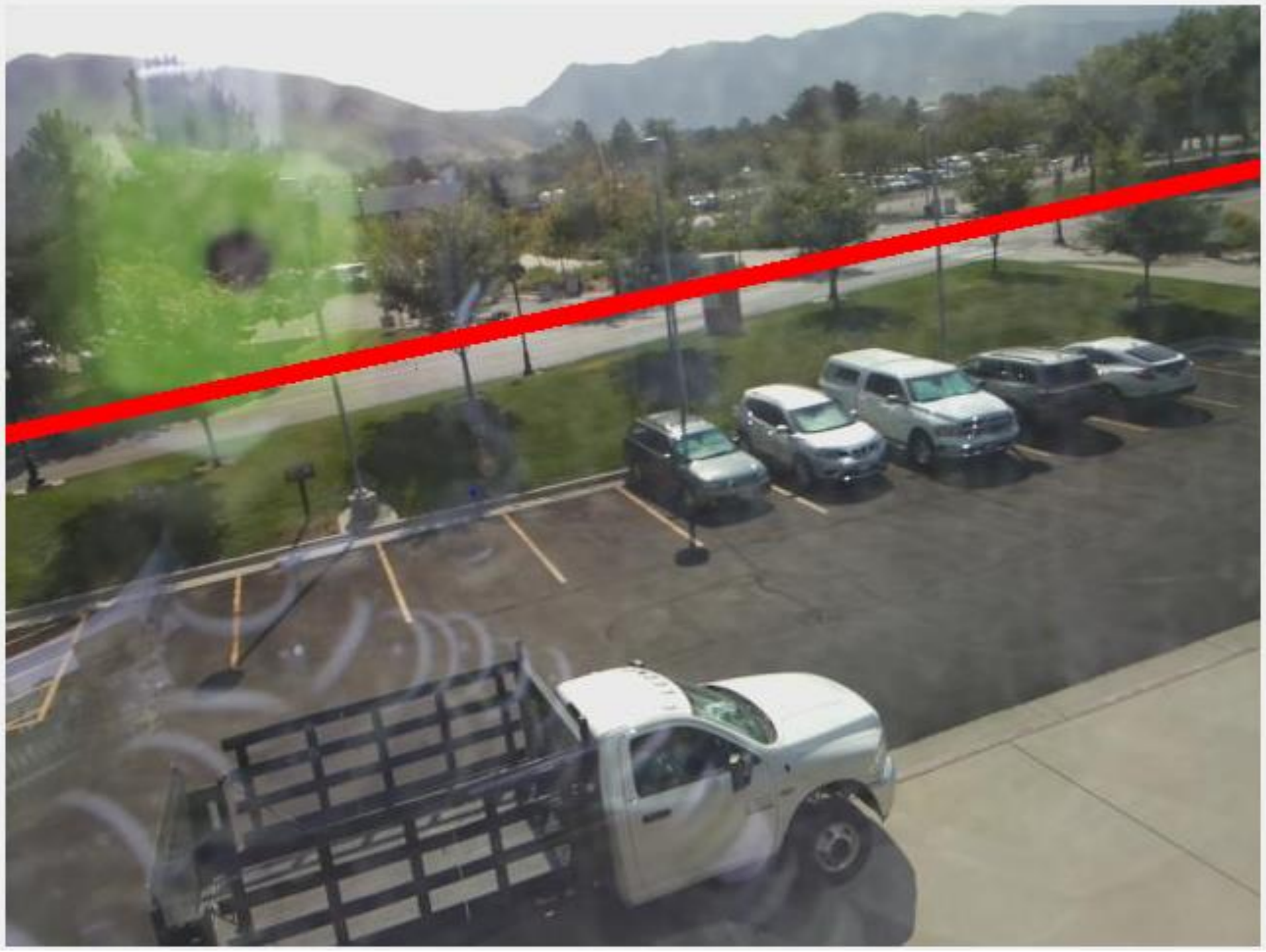
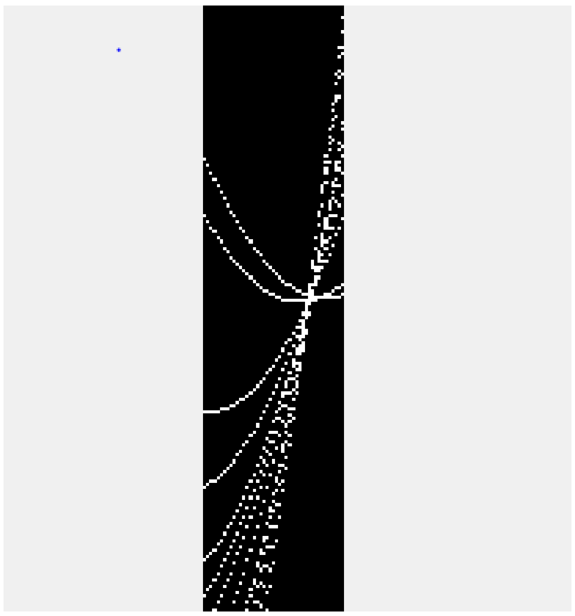
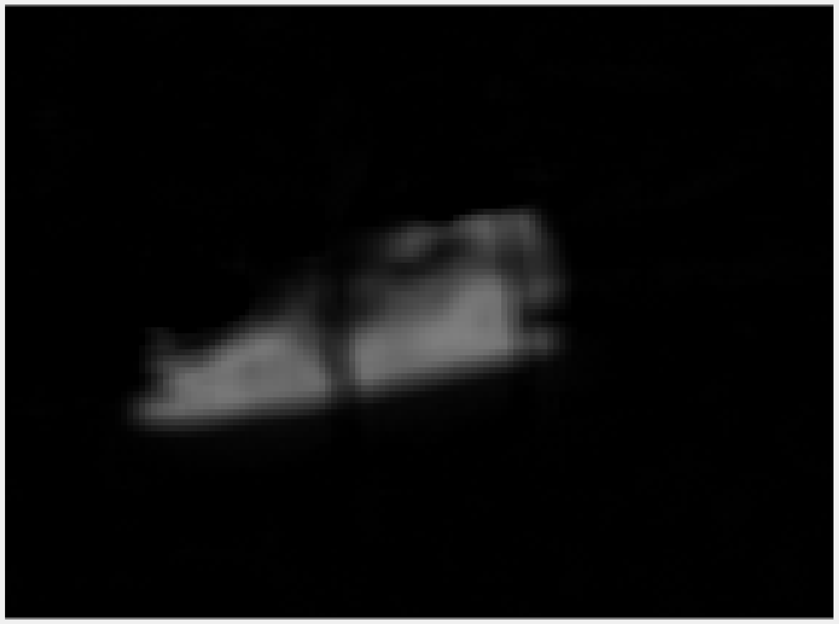


Figure 12.1: Hough Accumulator Array



3) The graph cuts implemented assembles a pixel by pixel similarity and uses EIGS to extract the 6 member eigen vectors for each pixel compared to each other. Then by using K means I can group these pixels based on this eigen vector as the feature vector. This can be useful for binary classification rather than whole image segmentation. Because the process involves making a M*N X M*N matrix and working with it, limiting M x N is necessary to make the process tractable. Additionally, grayscaling the image and filtering helps to make binary classification easier and more reasonable. The best case input could be a difference image, where the result of the graph cut would be a binaraization of what is part of the moving thing (the difference) and what is background. To achieve this I first took the background and frame 11 gray scale images and average filtered them with a filter size of 5. Frame 11 was then subtracted from the background and this was subsectioned to a tractable size around the moving object of interest (**Figure 13**). This was input to the graph cuts where the first eigenvector was used for clustering. The resulting grouping can be seen in **Figure 14**. By eroding and dilating this GC image I was able to eliminate the noise outside of the desired segmentation (**Figure 15**) resulting in **Figure 16**. When I tried to use all of the eigned values for each vector to cluster the entire section typically was mostly clustered together. Additionally images any bigger than this start to result in

Figure 13: Sub section diff im



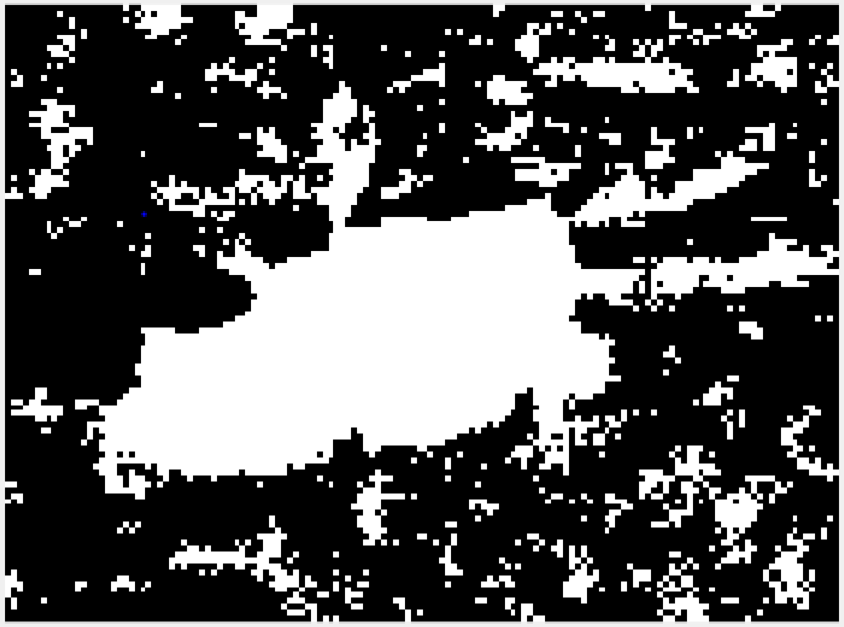


Figure 14: Graph cuts region 1

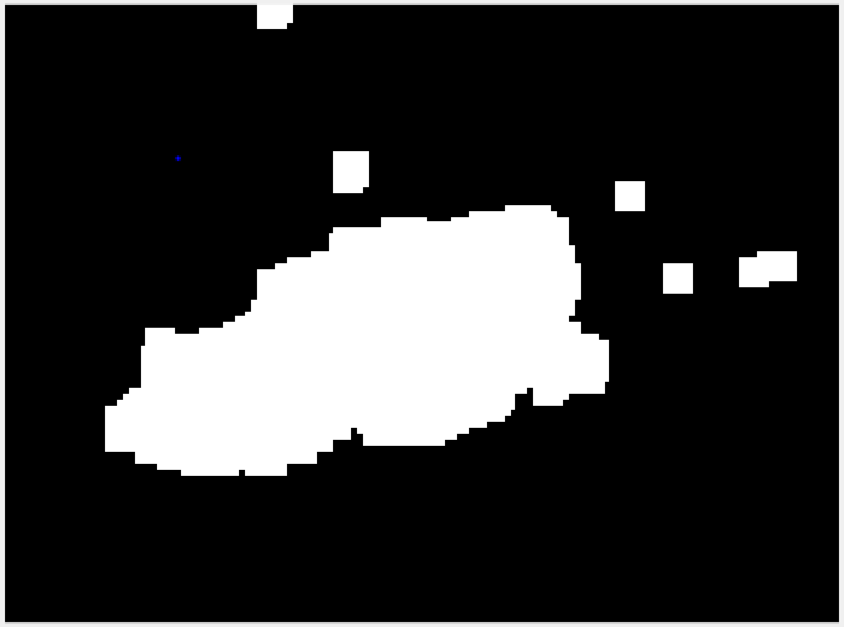
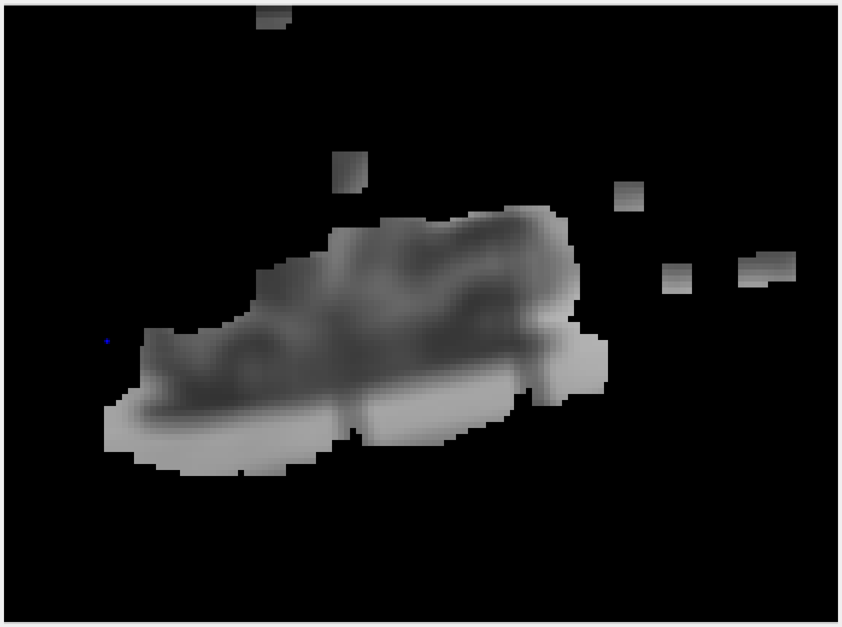


Figure 15: Cleaned up GC region 1

Figure 16: Segment from GC



4) For the watershed the algorithm first ensures that the values have been put into the negative domain. We then go to each pixel and check all of its neighbors. By padding with 0s we ensure that the edges will never be a basin in a pad image, thus we can easier handle edges. For each pixel we see what the minimum of its 8 connected neighborhood is. If it is tied for the basin or it is the basin then it is assigned 5. Otherwise it is assigned 1:9 according to what pixel is the lowest in its neighborhood. Then all contigous 5 groupings (basins) are labeled. All pixels are then checked to see if they are a 5. If they are not a 5 then we look down the path of pixels that they point to until we reach one that is a 5. We then set all pixels in that path to the group number of that last pixel that points to itself (5). In this way the watershed regions grow based on what pixels point to which basins. They grow in, contrary to a typical watershed which grows out from the basin. To threshold we could erode this full region down to some level. We can see in **Figure 17** the input to the WS which is a filtered (blurr and median) difference image from frame 11. **Figure 18** shows the surface of **Figure 17** where we can see roughly where the deepest WS region should be. There are however several peaks (basins when inverted) in this area however and we will only be finding the deepest of them. **Figur 19** shows the resultant deepest watershed region with a spread of 132 grayscale value. This area is overlayed on the frame 11 and on the difference image (**Figure 20**, and **Figure 21** respectively.) As can be seen the deepest watershed region seems to be in the dark shadow of the car passing over the white road. This makes sense as this would result in a very large difference thus a deep basin in the watershed input.

Figure 17: Watershed Input

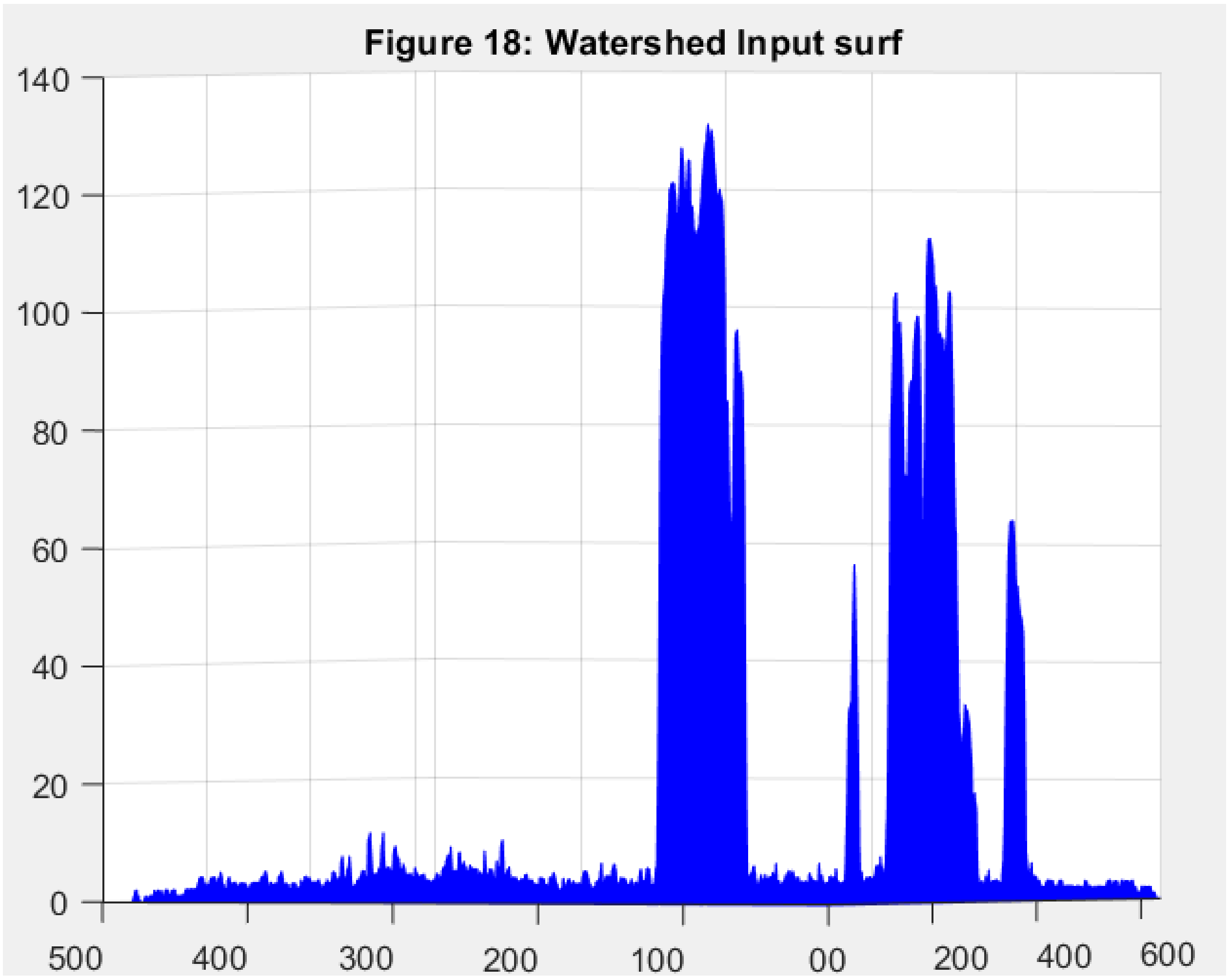


Figure 18: Watershed Input surf
140
120
100
80
60
40
20
0
500
400
300
200
100
00
200
400
600

Figure 19: Deepest water shed

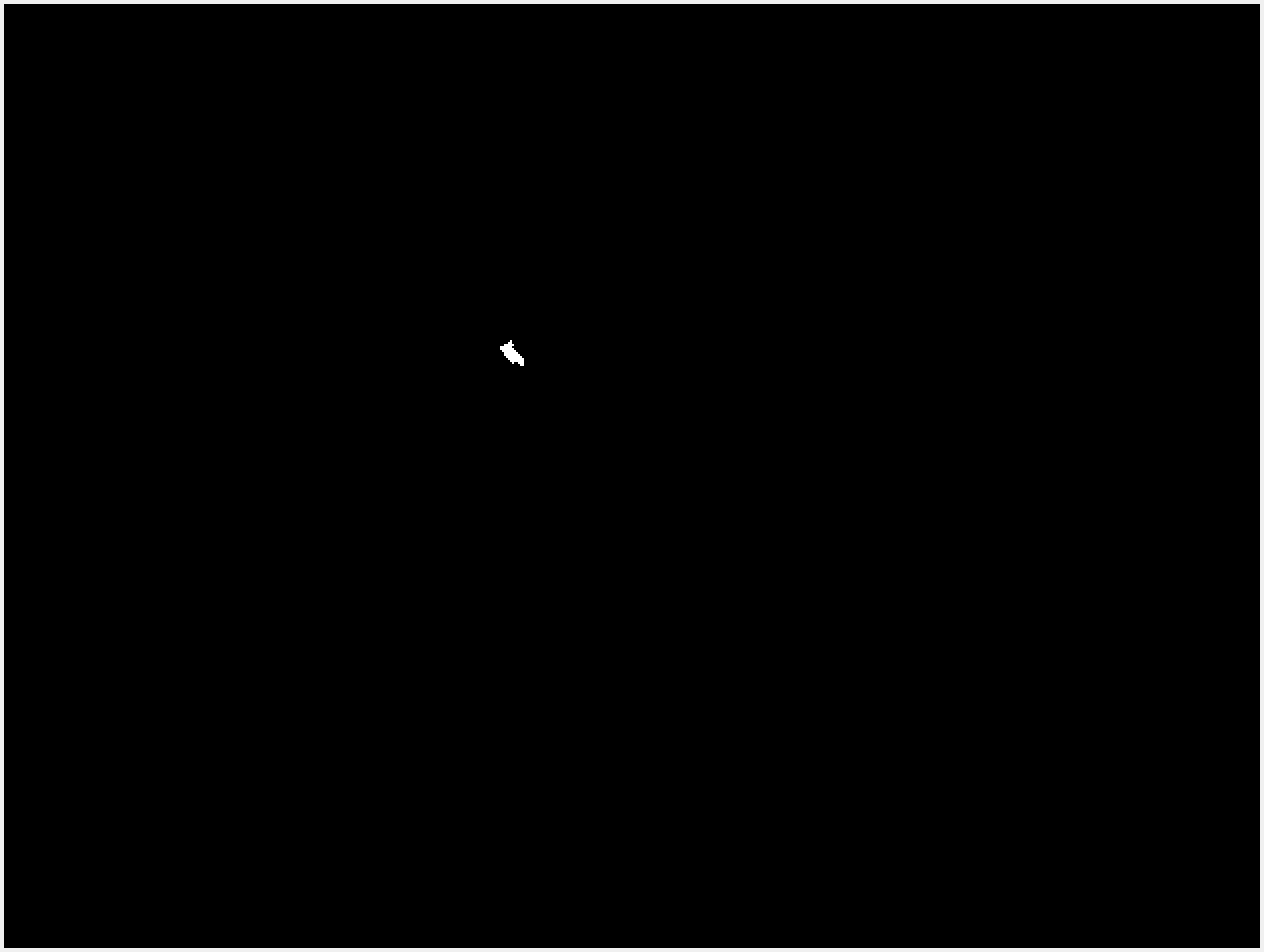


Figure 20: Deepest water shed on bkcground

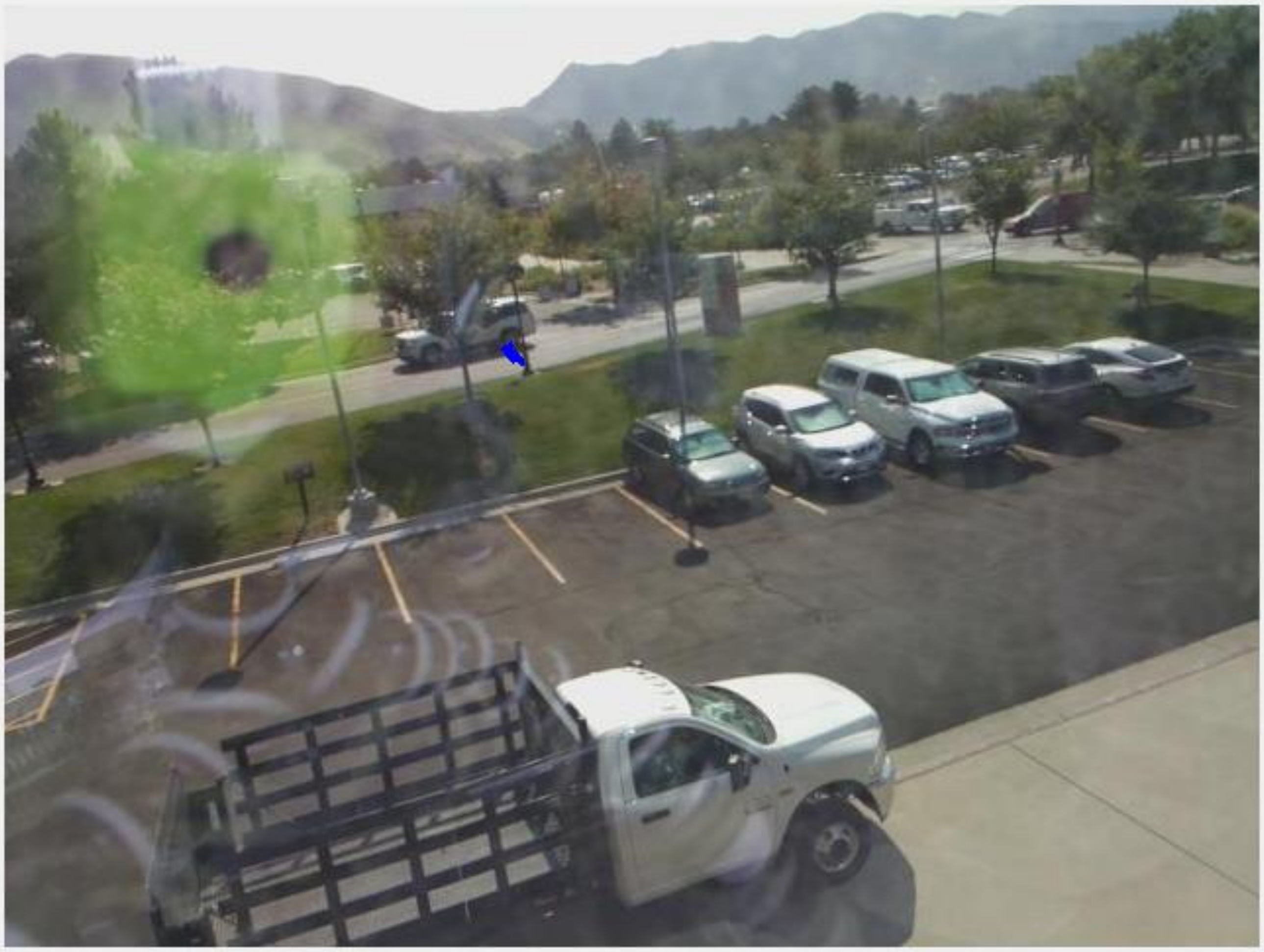


Figure 21: Deepest water shed on diff im