Simple Motion Detection Using Four Algorithms

EE 454
Section 001

Brian Nguyen
Mark Gundling
Kara Dally
Dan Herlihy

Table of Contents

Summary

The goal of the project is to implement four different methods of motion detection algorithms on short videos. The four algorithms to be implemented are simple background subtraction, adaptive background subtraction, frame differencing, and persistent frame differencing. We perform all four methods at the same time on eight different videos, saving the generated frames and creating videos from them. Each output frame shows the results of the four motion detection algorithms run on the same input sequence. This process is performed on eight datasets and rendered into eight videos displaying the results.

Overall, the purpose of the project is to observe how different motion detection algorithms handle processing the same video. These result allow us to draw conclusions about the strengths and weaknesses of each of the algorithms in processing videos in different conditions and with different mitigating factors. The project demonstrates the importance of selecting an appropriate algorithms for the specific type of video that needs to be processed.

Procedural Outline

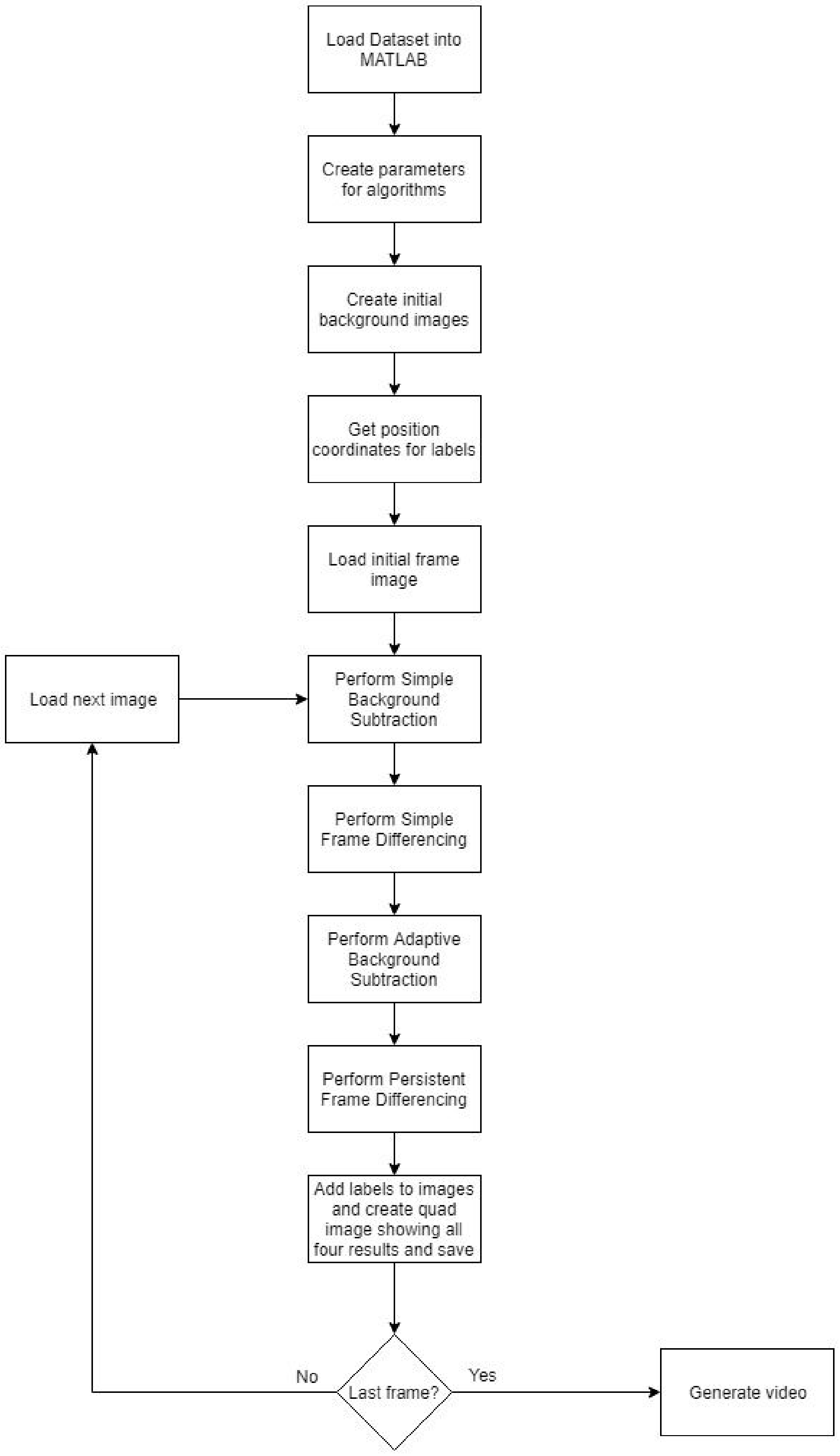

Load Dataset into MATLAB
Create parameters for algorithms
Create initial background images
Get position coordinates for labels
Load initial frame image
Perform Simple Background Subtraction
Perform Simple Frame Differencing
Perform Adaptive Background Subtraction
Perform Persistent Frame Differencing
Add labels to images and create quad image showing all four results and save
Last frame?
No
Yes
Load next image
Generate video

Figure 1. Procedural Flowchart

The main code was panelView. The path to the file was created first and then the properties for the labels on the video were set. VideoWriter was then used to create a video writing object that would save a video to the folder specified in path set above. Variables were declared next such as, the threshold, the alpha parameter, and gamma. The initial background of the frame sequence was then found. The first frame was set as the initial background and the greyscale of it was used in the algorithms because a single pixel value was necessary for simple calculations. If color images were used, each pixel would have three values to deal with making the calculations more complex and require more processing. Each algorithm used a different variable to save the background per frame.

First the simple background subtraction routine is run by setting the loaded image to grayscale and take the absolute value of the difference between the background image and the new image. This generated image is saved for video generation later.

The next algorithm is simple frame differencing. This is accomplished by taking the absolute difference between the previous (background) and current image and running it through the thresholding function in MATLAB with a predefined threshold value. Using this value allows for quick adaptation to changes in lighting or camera motion. The current image is then set as the background for the next iteration of the loop.

The third algorithm is the adaptive background subtraction. This method of motion detection is the same as the simple background subtraction except that the next background image to be used is the the blending of the current and previous using the floor function and parameter alpha. This function rounds each element passed into it to the nearest integer less than or equal to that element. The elements passed into floor are the result of the Equ. # . If alpha is one, this method would achieve the same results as simple frame differencing and if alpha is zero, it achieves the same results as simple background subtraction.

The last algorithm is persistent frame differencing in which motion images are combined with a linear decay term. The absolute difference of the previous image and the current image is put through the same thresholding as simple frame differencing. A temporary image is then created from the maximum value between the previous image minus parameter gamma and zero.

The 4 images from the four different algorithms are concatenated together into one image, labeled, and saved to the disk. After all images have been saved, they are combined into a video.

Experimental Observations

The final code ran as was expected from the project outline and the class lectures. The program takes several minutes to run which is to be expected when doing video analysis and saving every single frame of the video along the way. Below you can find intermediate examples of the four algorithms on frame 68 of New Arena A. These images demonstrate that each algorithm is behaving as expected. The final stitched together versions of the four algorithms can be seen throughout the remainder of the document and demonstrate the correctness of the code.

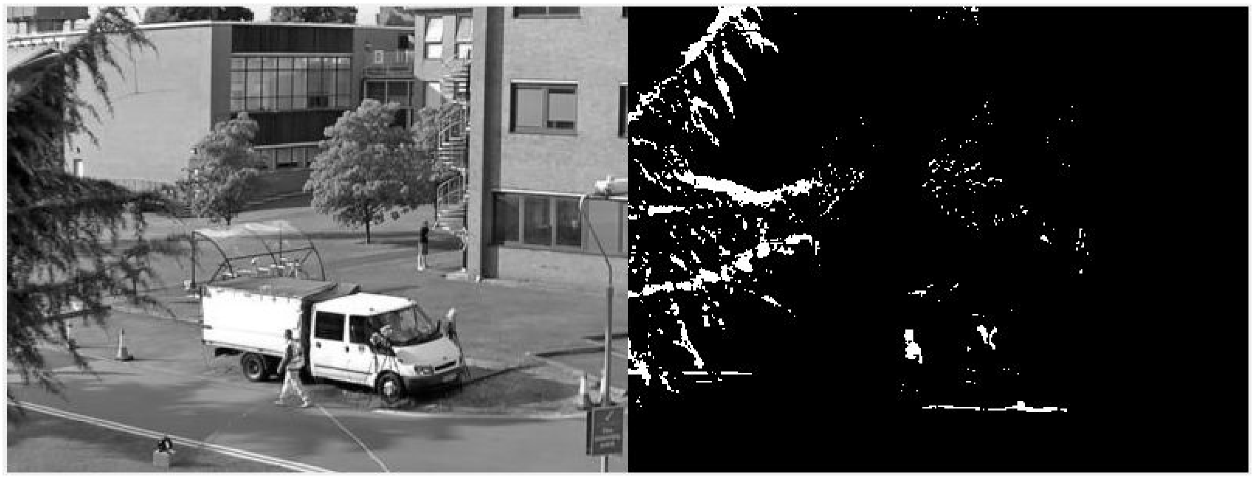


Figure 2. New Arena A frame 68 before and after simple background subtraction

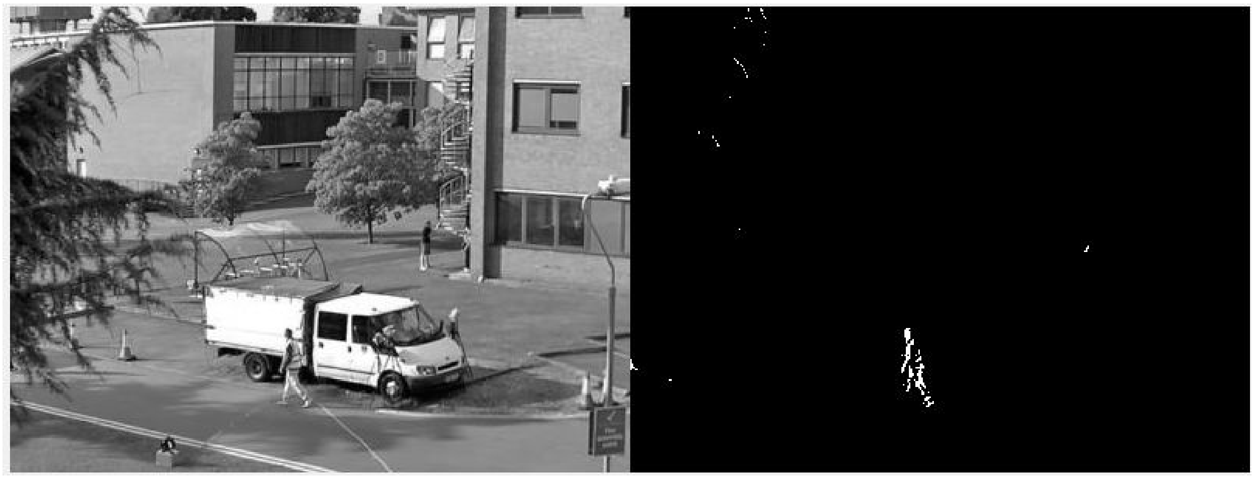


Figure 3. New Arena A frame 68 before and after simple frame differencing

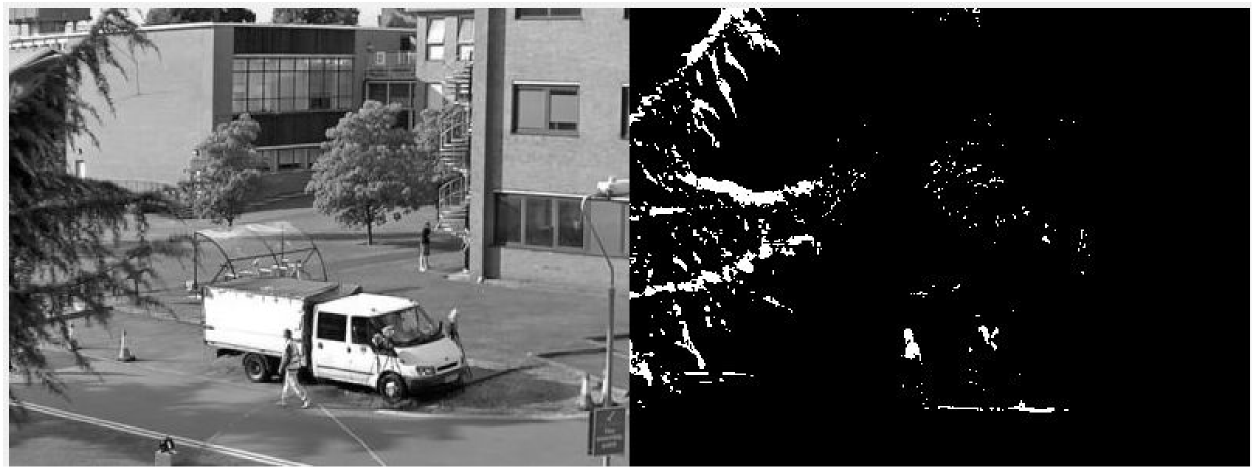


Figure 4. New Arena A frame 68 before and after adaptive background subtraction

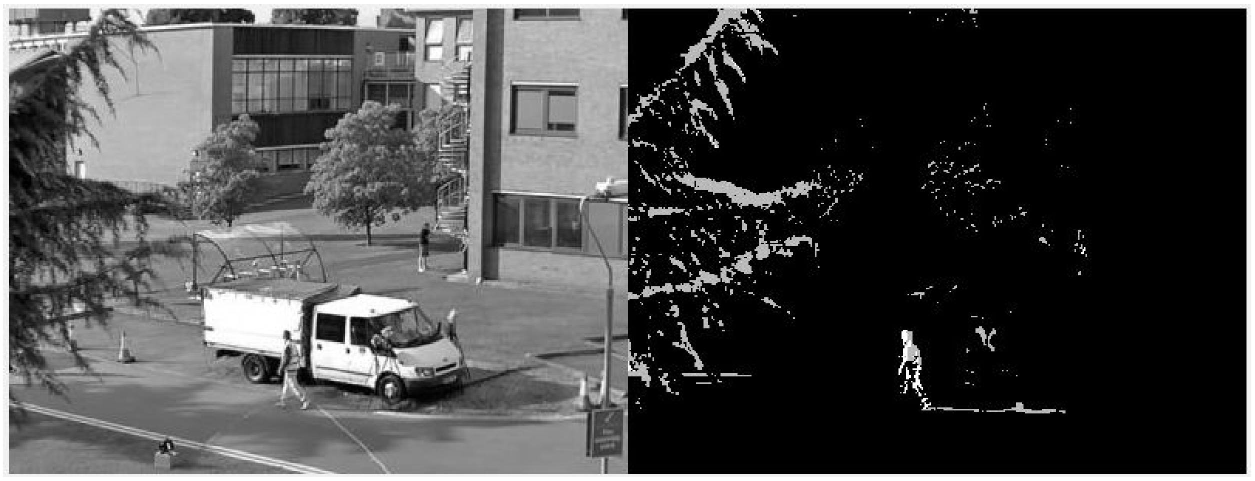


Figure 5. New Arena A frame 68 before and after persistent frame differencing

Quantitative Results

The threshold value is used to determine which pixels are an object (white pixel of value 1) and which pixels are not an object (black pixels of value 0). If the difference between the background and the current frame is greater than the threshold then the pixel value is set to 1, a white pixel. If the difference is less than the threshold value then the pixel value is set to 0, a black pixel. Varying the threshold varies the algorithms sensitivity to change between the background and each frame. The lower the threshold the more sensitive to change. This change could be due to lighting or pixel intensities and not due to an actual change in objects in the image. Figure 6 below shows what happens if the threshold value is set close to zero for frame 68.

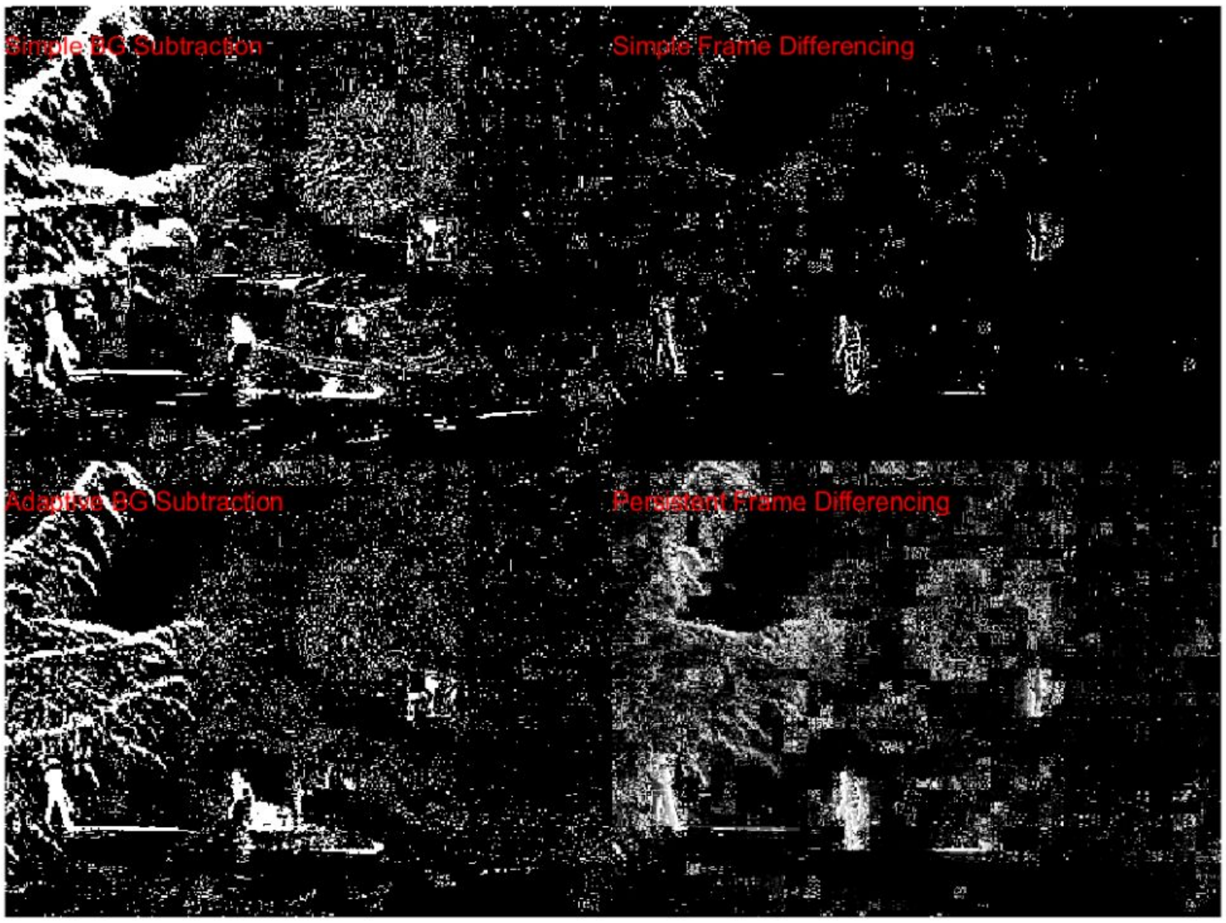


Figure 6. A low threshold value of 5

When the threshold value is close to zero, small variations in light, movement, or camera motion is seen as motion in the frame (also considered noise). Since these small variations are not what we are trying to measure and cause unwanted high pixel values, we want a higher threshold value to filter them out.

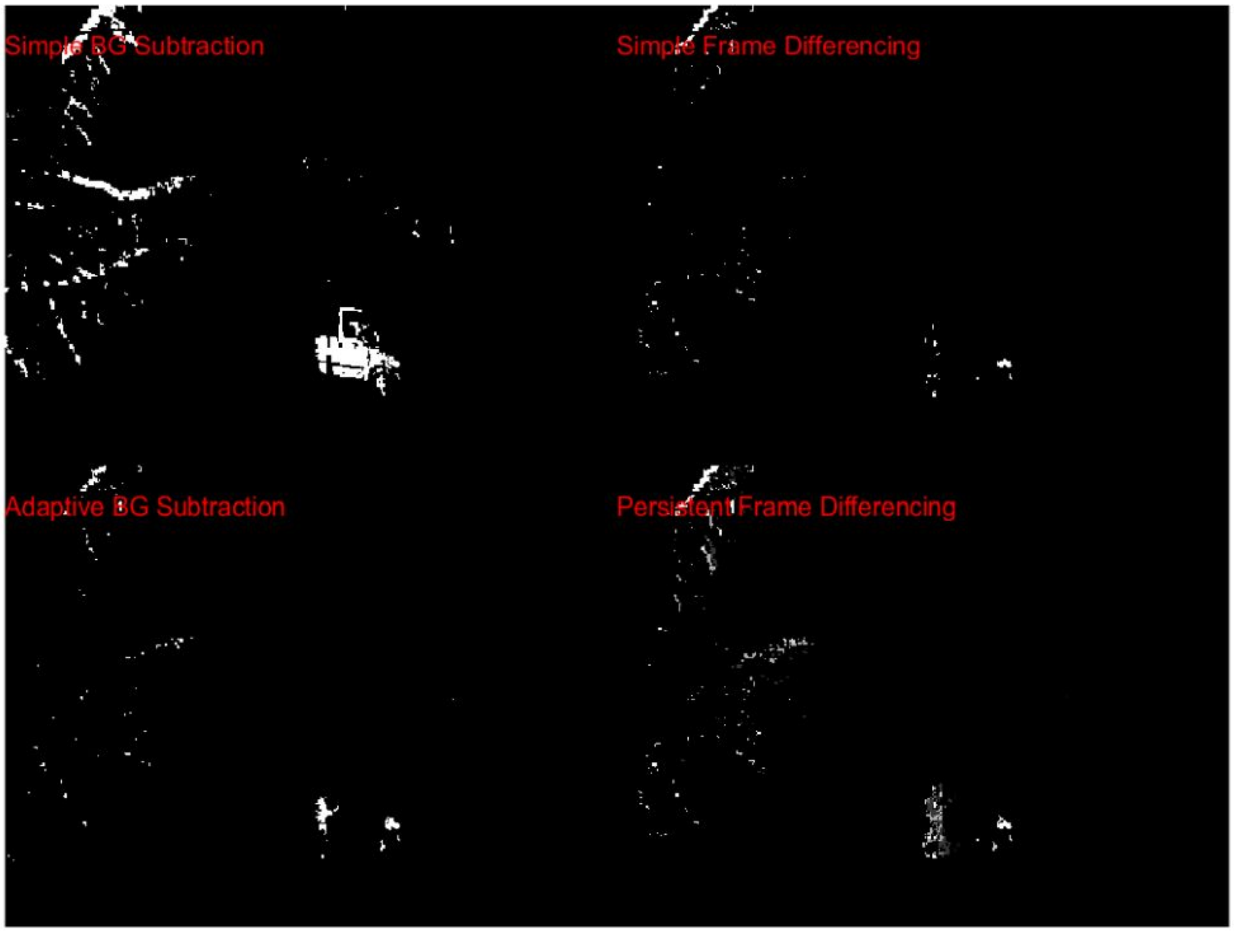


Figure 7. A high threshold value of 60

The image above shows a high threshold value of 60 for frame 68. The high threshold does not see much variation in the background and current frame. This value for the threshold is too high as little change is detected in this frame. A person should be seen in this frame and there is little indication of said person.

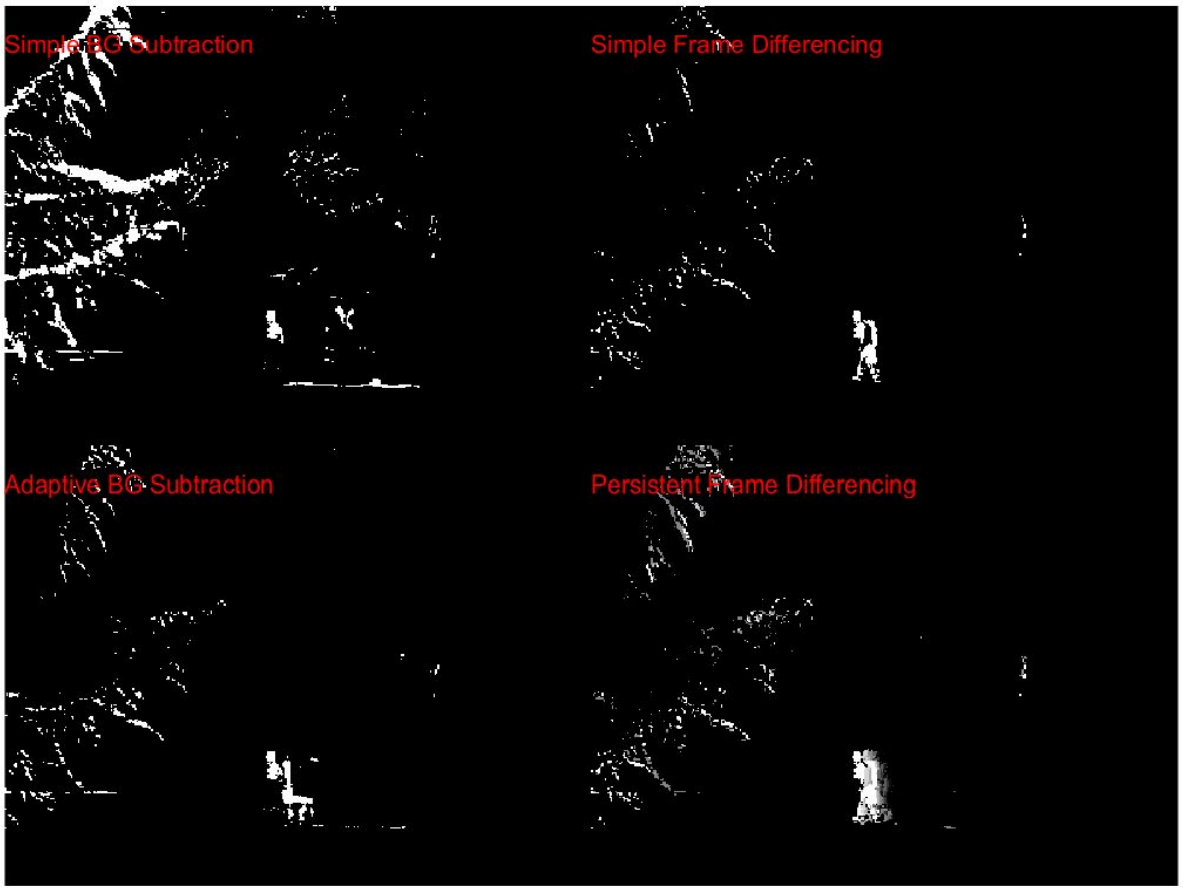


Figure 8. A medium threshold of 30

For a threshold of 30, the person is visible in for each of the background subtraction algorithms. The above image shows this for frame 68. This is a reasonable threshold since we are able to distinguish what is occurring in this frame. It is also in between the high and low value that were tested.

The alpha parameter is used for merging new images into the background model for the adaptive background subtraction algorithm. Decreasing alpha makes it take longer for a stopped object to fade into the background. Increasing alpha will make the object fade into the background quicker. We observed this when we changed the value of alpha in our code.

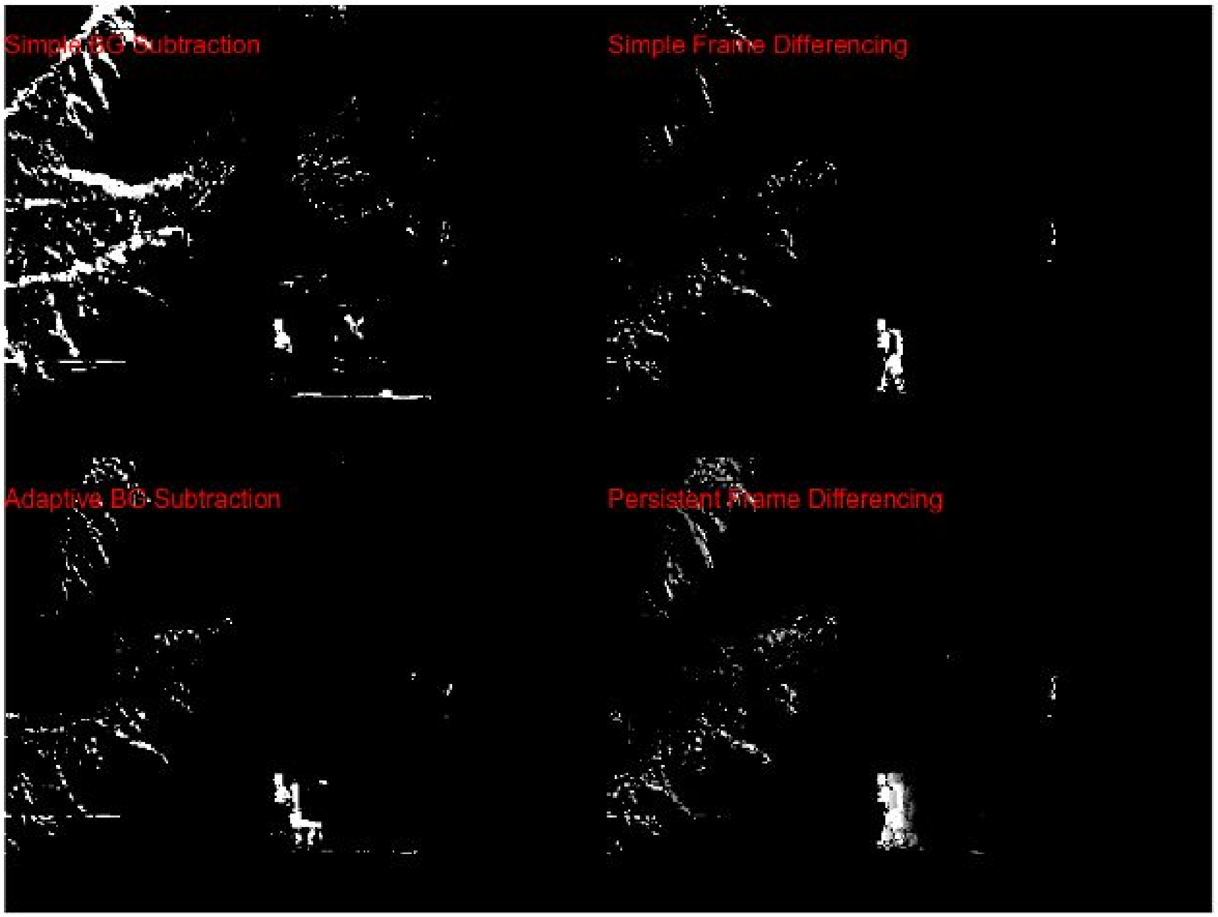


Figure 9. Alpha set at 0.1

Alpha set at 0.1 for above image (frame 68). This is the alpha value that was chosen for the project.

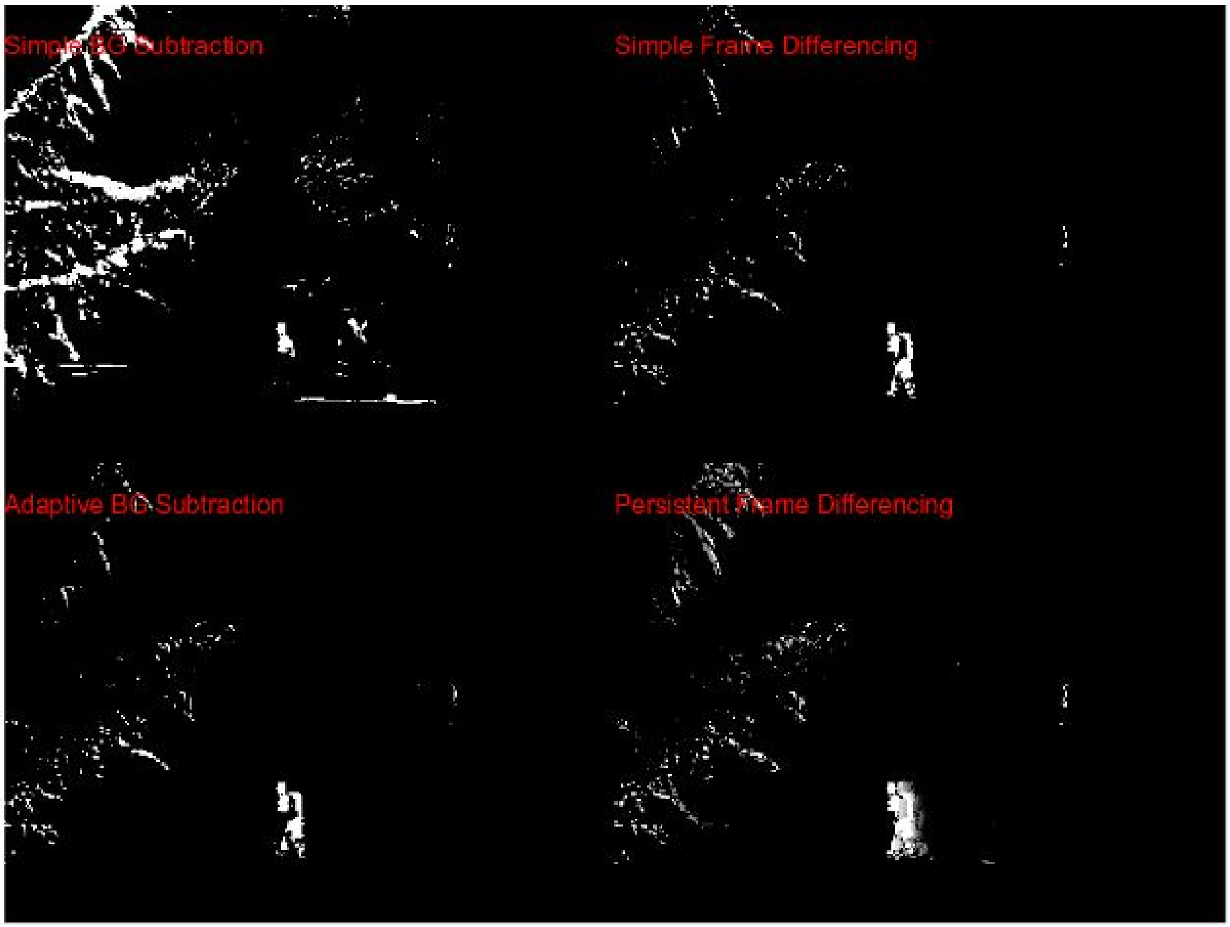


Figure 10. Alpha set at 0.5

Alpha is set to 0.5 at frame 68. Even though it is in between 0 and 1, the image appears to look like the simple frame differencing and is therefore set at too high of a value. Therefore we chose a lower alpha.

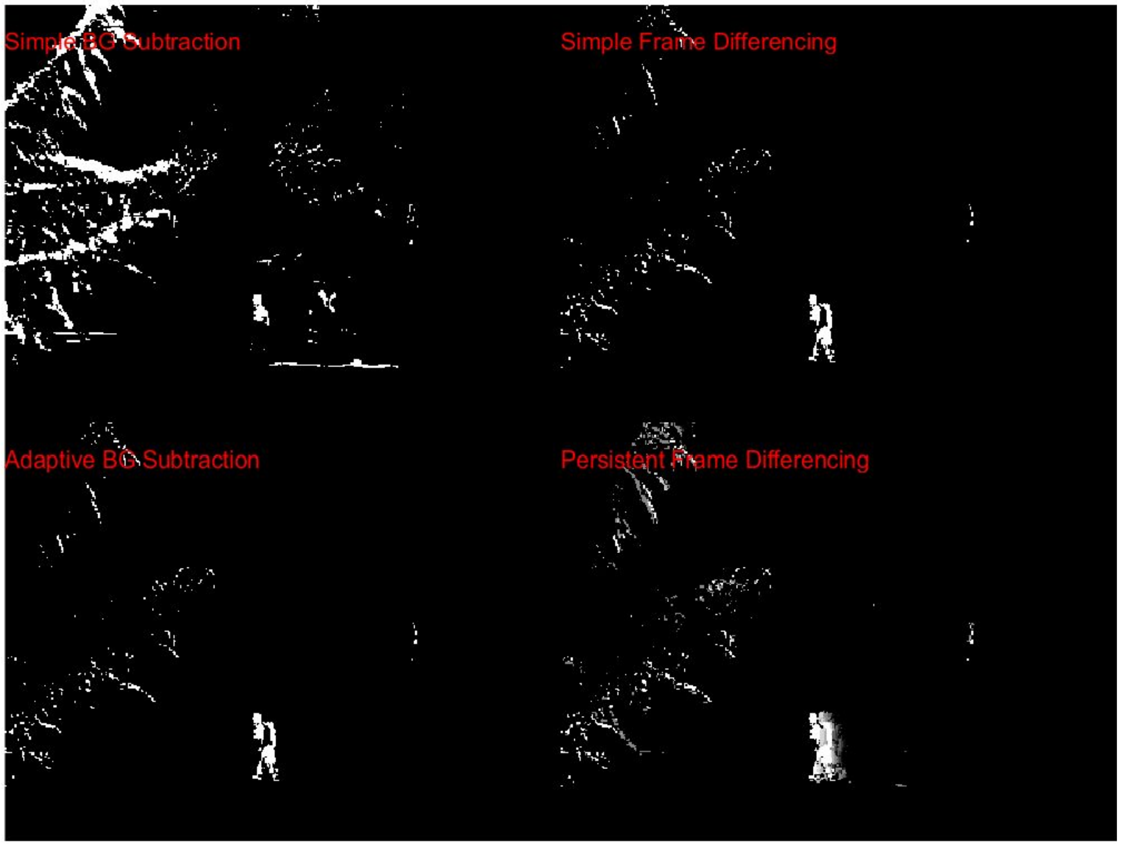


Figure 11. Alpha set at 1

Above figure (Figure 11) proves that when alpha equals one simple frame differencing and adaptive background subtraction are the same because the two images are exactly the same in the photo above (this is also frame 68).

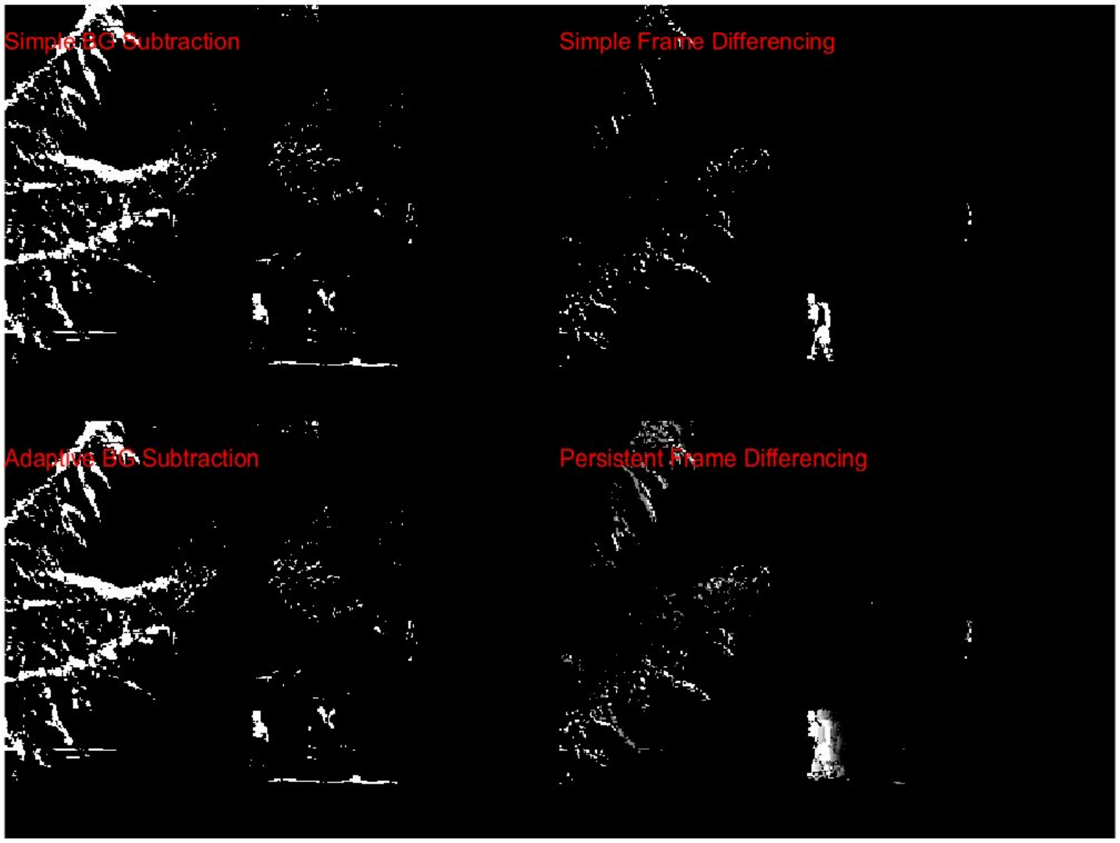


Figure 12. Alpha set at 0

Figure 12 proves that when alpha equals zero, simple background subtraction and adaptive background subtraction produce the same results. Figures 11 and 12 show the same frame of the same video with different alpha values. The linear decay parameter, gamma, affects the trailing edges of the object. As the linear decay term increases, the movement trail behind the object decreases. See figure 13 for a high gamma value and figure 14 for a low gamma.

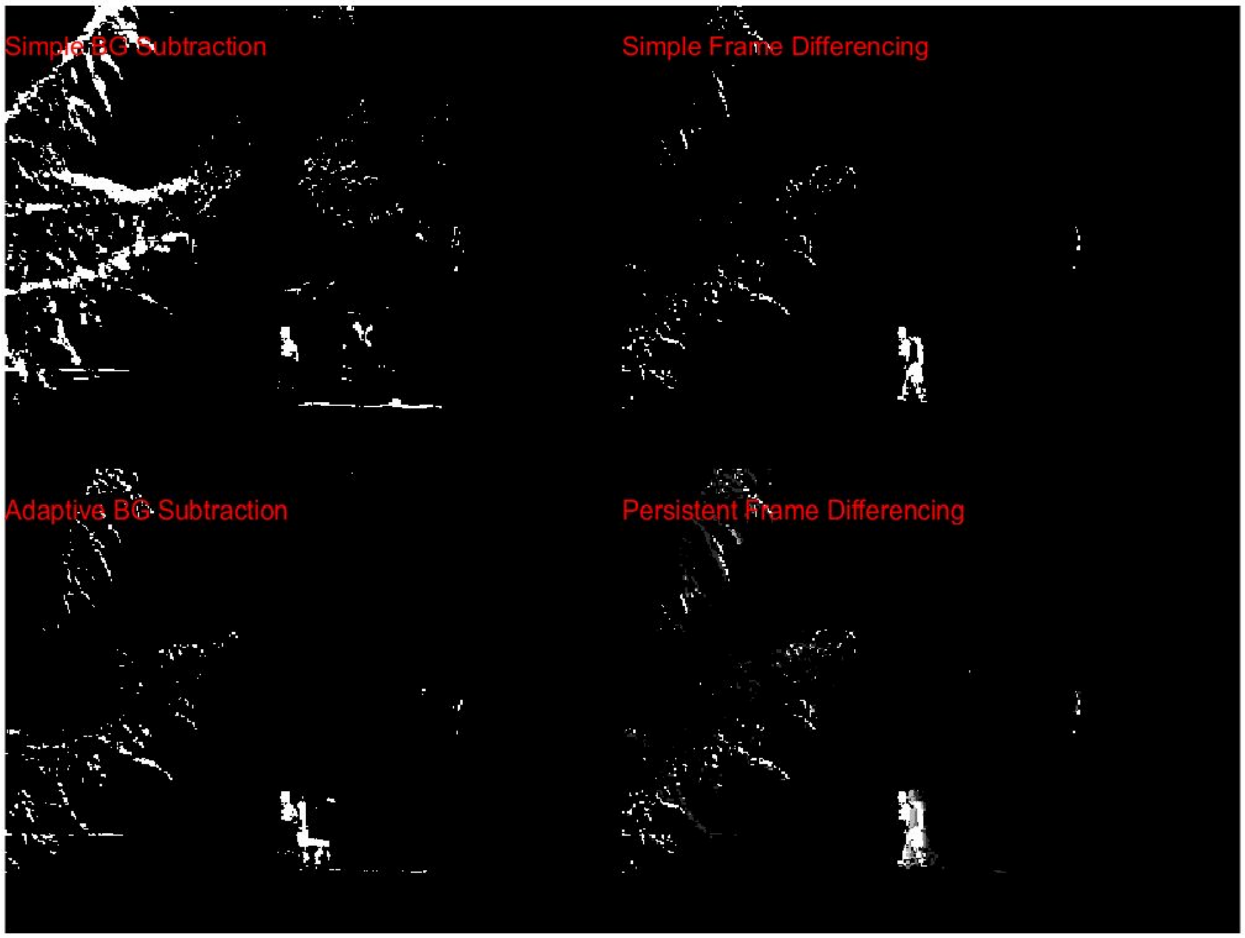


Figure 13. Gamma set to 70

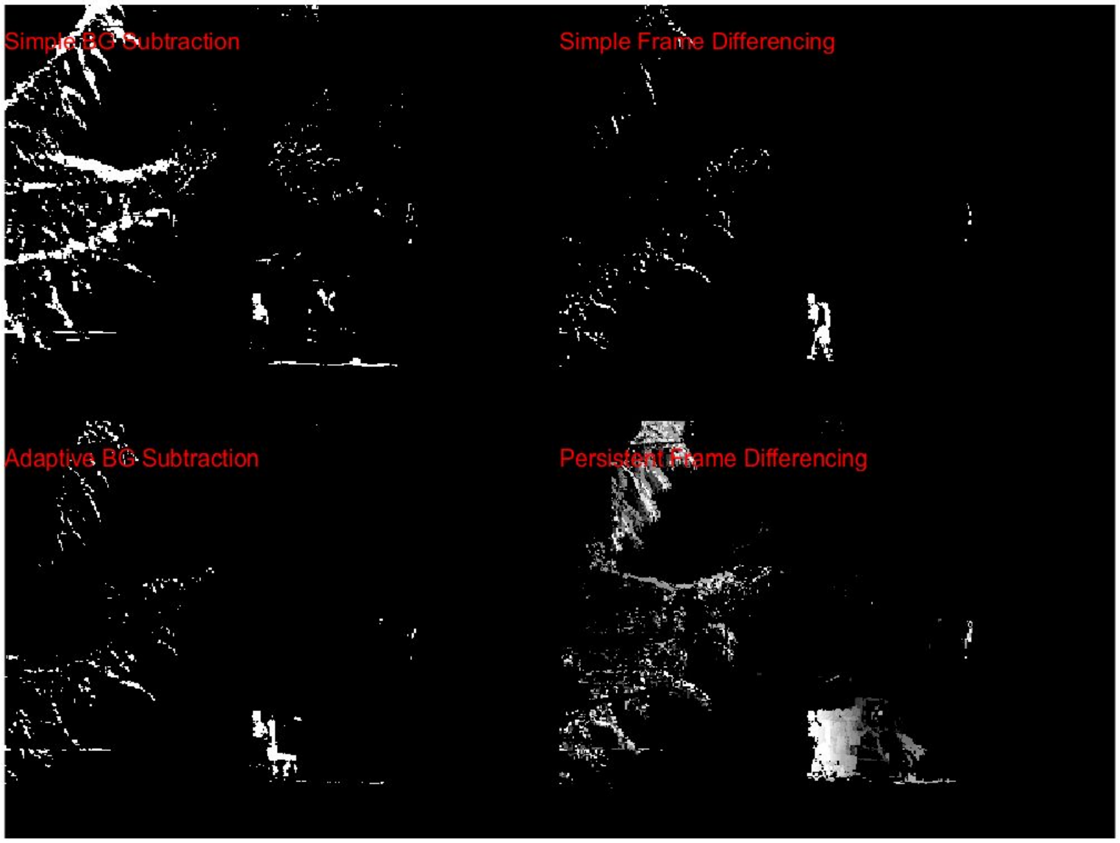


Figure 14. Gamma set to 10

Qualitative Results

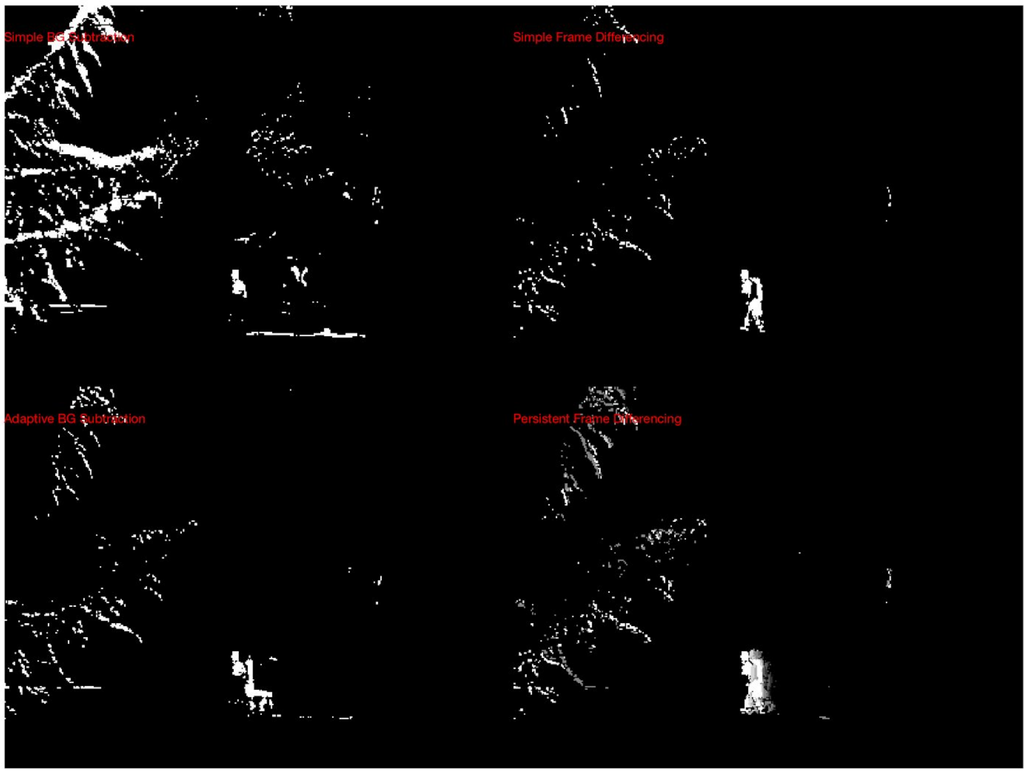


Figure 15. New Arena A Frame 68

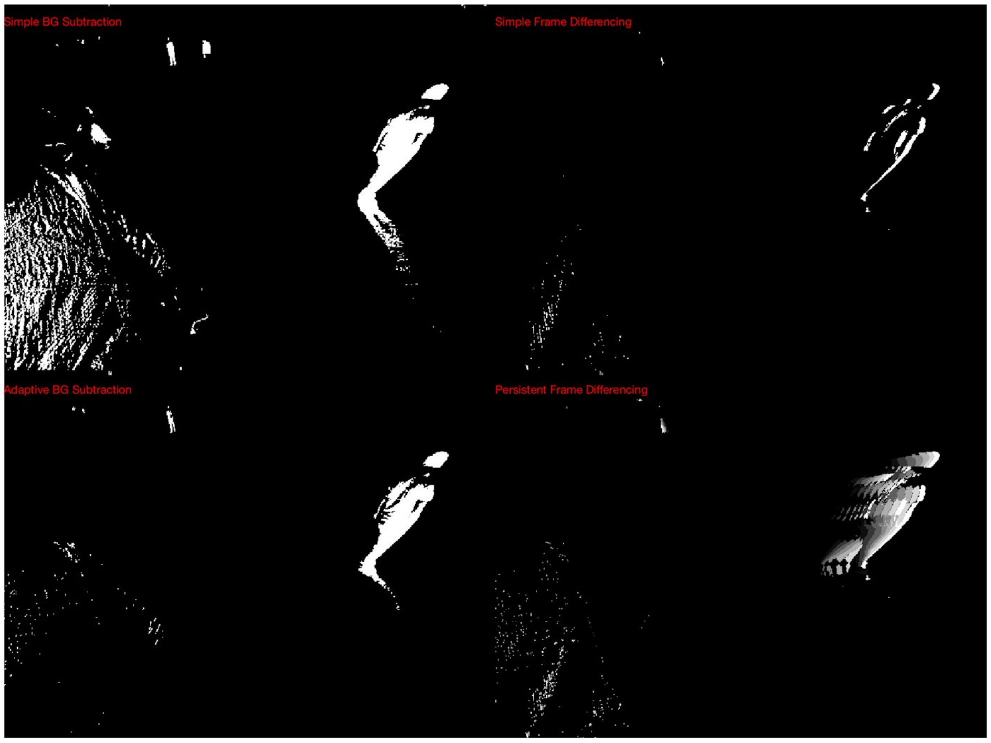


Figure 16. New Arena N Frame 68

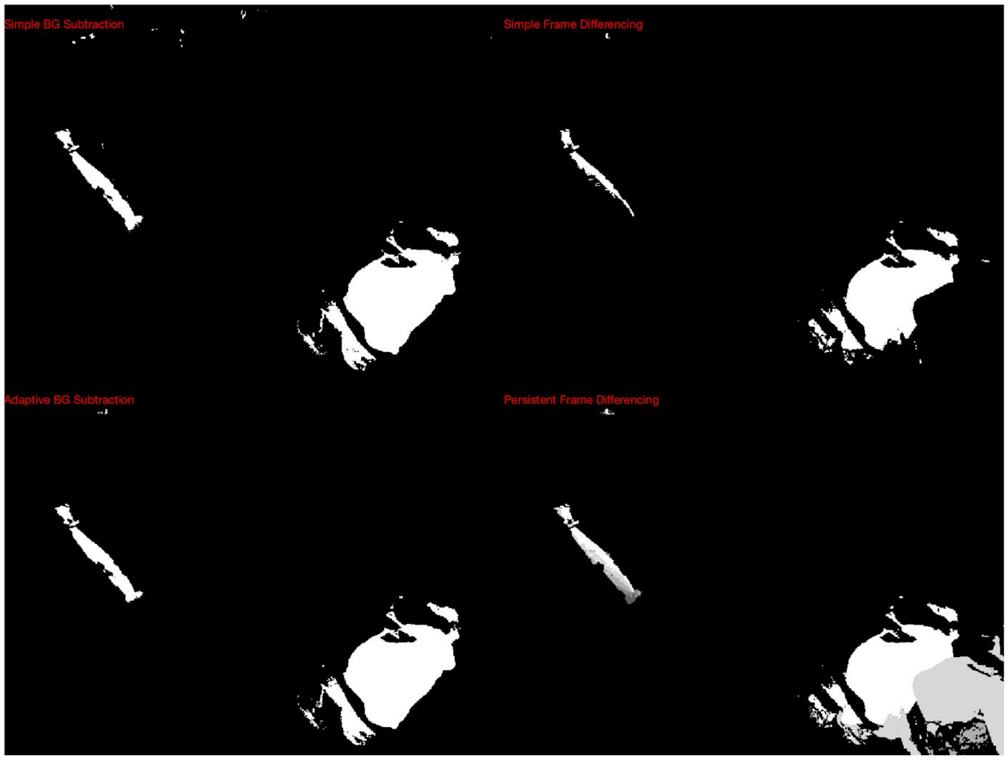


Figure 17. New Arena W Frame 75

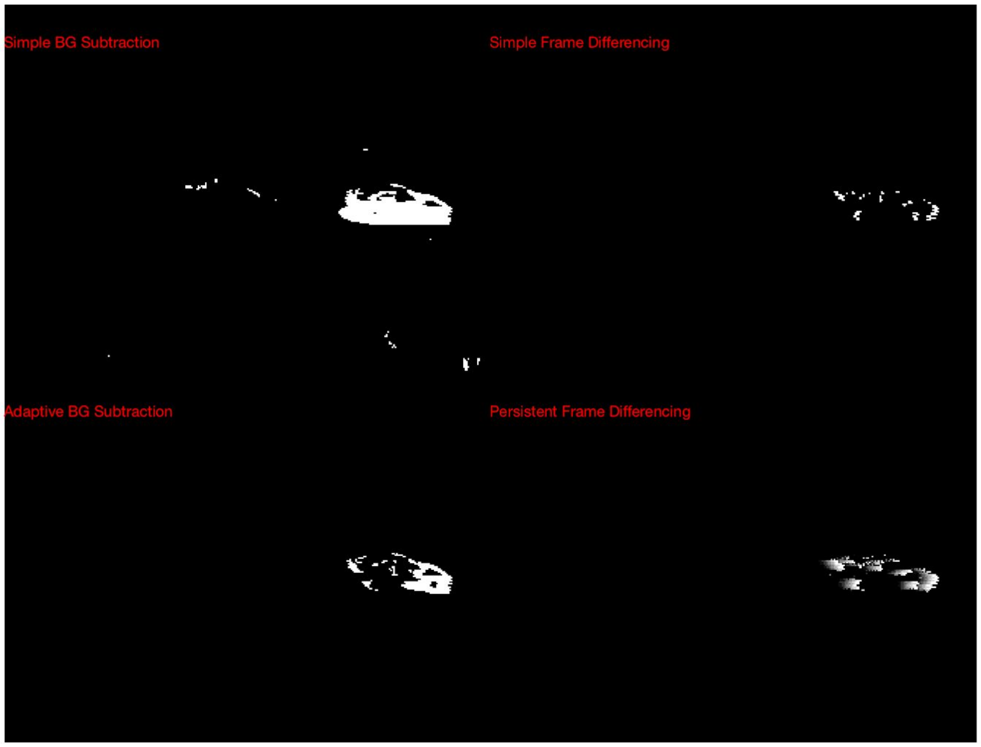


Figure 18. New Get In Frame 534

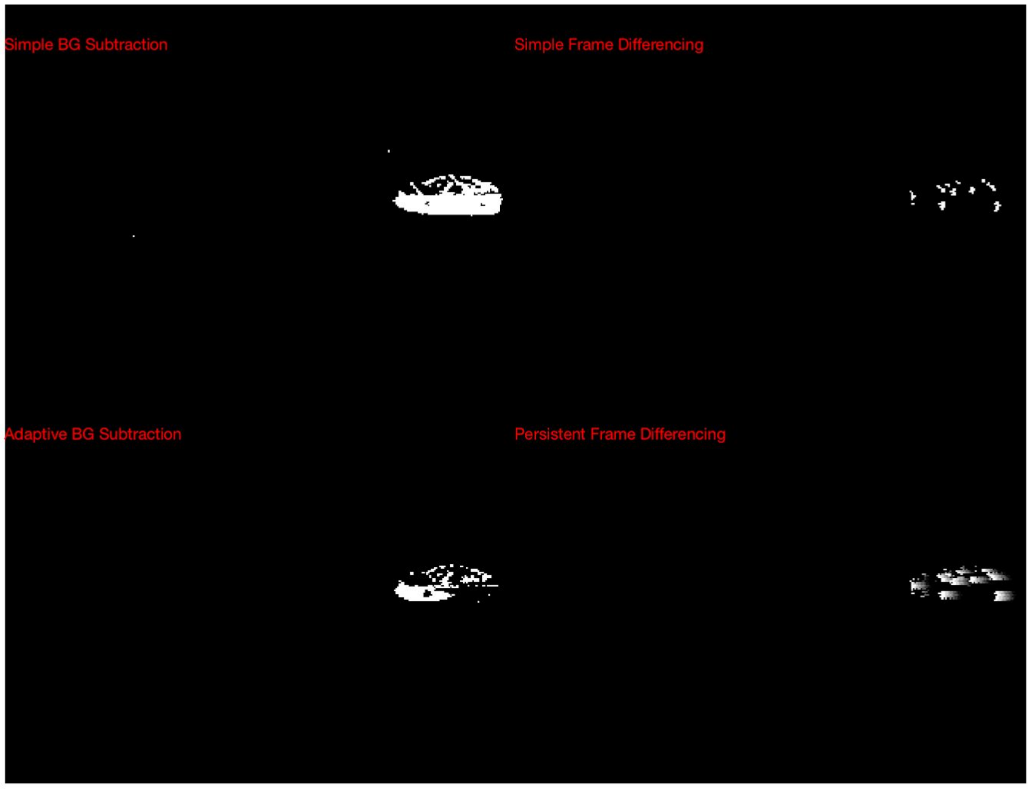


Figure 19. New Get Out Frame 0108

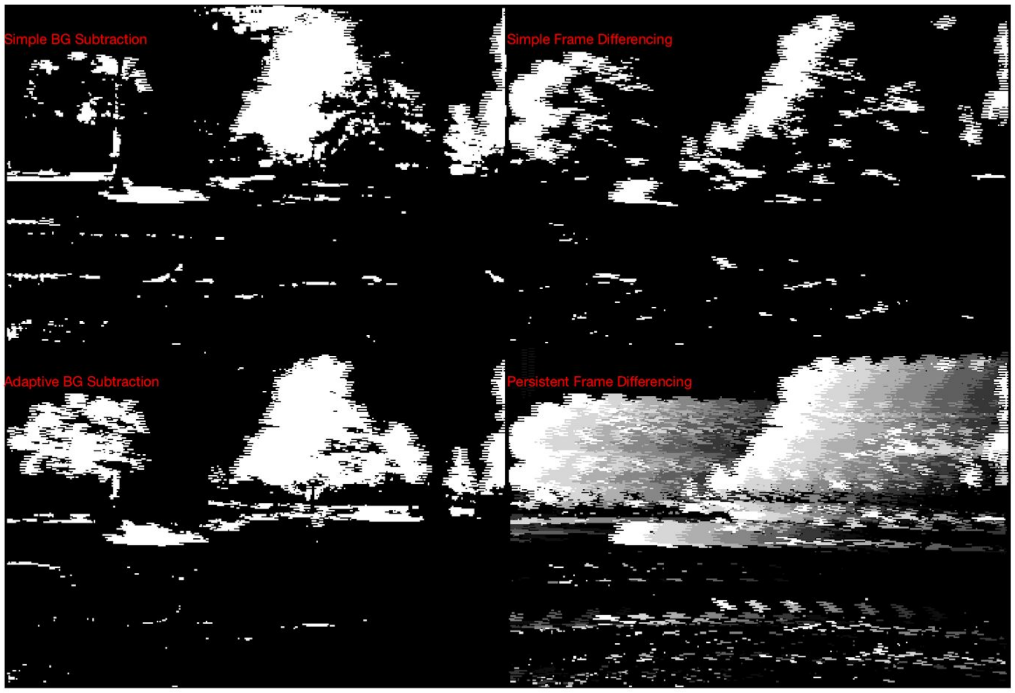


Figure 20. New Move Cam Frame 0163

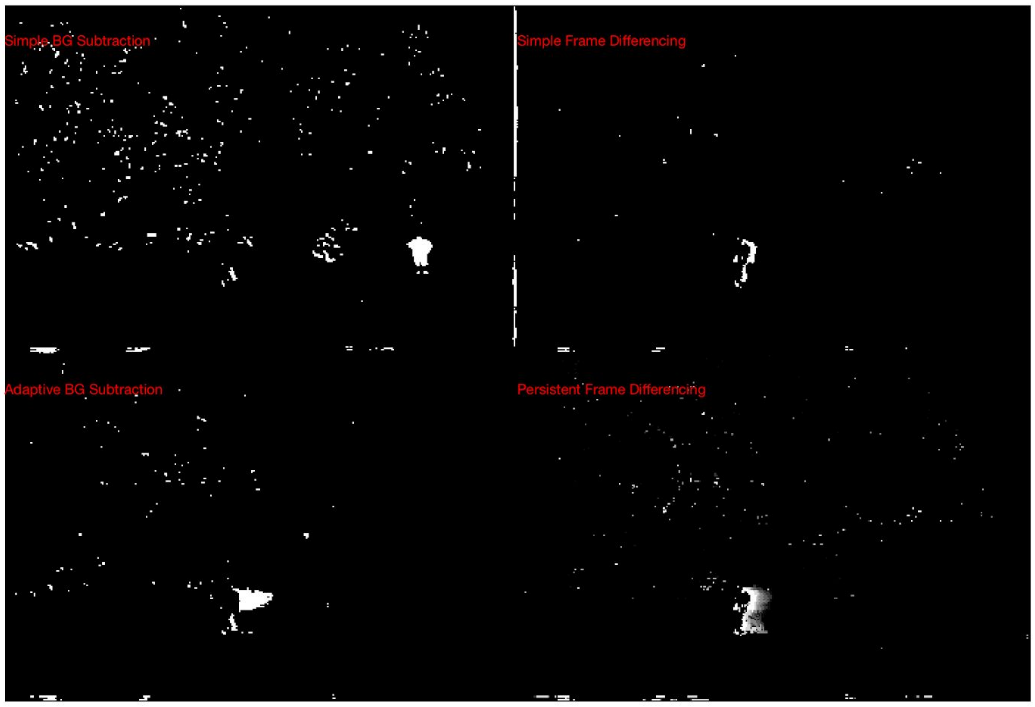


Figure 21. New Trees Frame 102

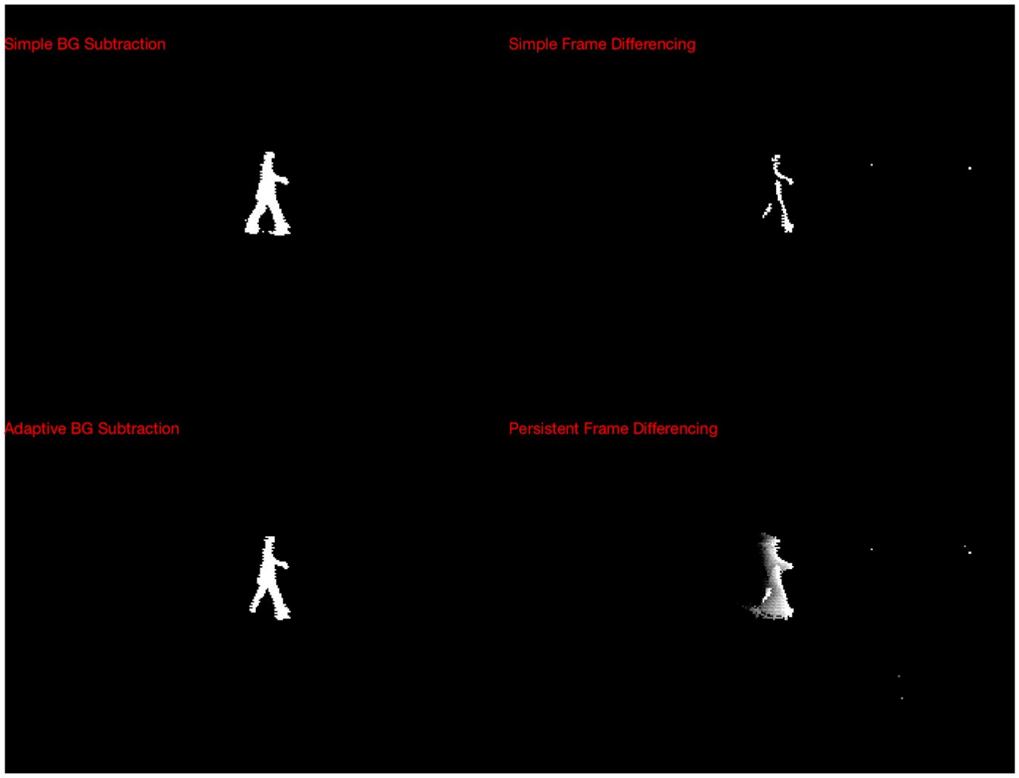


Figure 22. New Walk Frame 0172:

In a comparison of the eight video results, there were some videos that produced better results. Videos that have the least moving objects in them produce the clearest results. This is seen in Figure 22 from video Walk.avi. Since the camera is not moving and nothing in the background in moving besides the main object of interest, the highest pixel values are always the object in interest. Noise is not introduced in this video (partially due to the threshold value) which produces a clearer result as well. Trees.avi produces the worst results of them all. This is due to the wind blowing the leaves in the tree (see Figure 21). If we had set the threshold value high enough to filter out all the noise from the movement of the leaves, we would have lost details in the objects of interest as well. Movecam.avi also shows the effect of noise on the resultant filtered images. When the camera pans to follow the car, the image gets clouded with noise and none of the four methods fully recover.

The single point of failure for all the methods is the background object used for comparison. Since all four algorithms depend on a background image to compare against the current image, the selection of a background image is crucial to a successful motion detection .

Failure due to ghost object occurs in the background subtraction algorithm when an object that is considered part of the background moves out of the image. This will leave behind ghost image because the algorithm sees a change in the background. This is not seen in the other algorithms because the background is not a fixed image. Figure 23 shows the ghost of the person walking away. Comparing the simple background subtraction and simple frame differencing images, you can see that the person moved away from their initial position and left behind a “ghost” image of themselves.

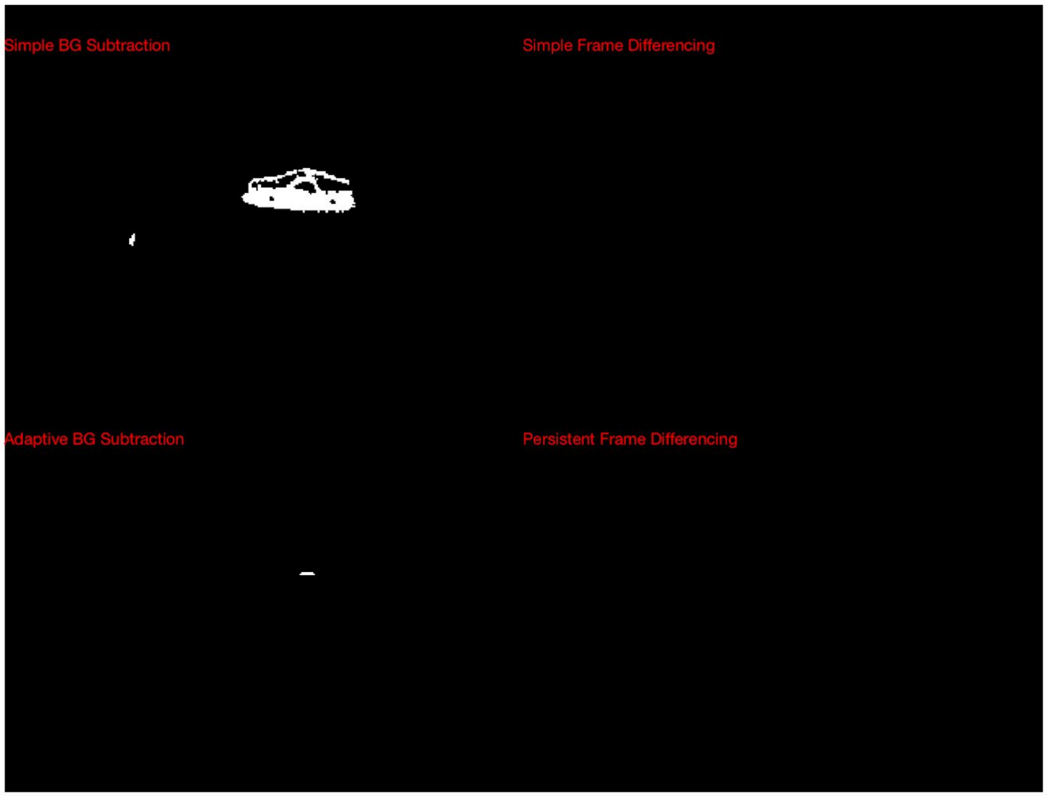


Figure 23. “Ghost” image left in SBS

Failure due to a moving camera occurs mainly in the background subtraction algorithm because the background is set. The other algorithms change the background and are therefore not as affected by the moving camera. Figure 24 shows the last frame of trees.avi where the simple background subtraction image still shows the trees as highlighted, moving objects even though the other methods have removed them because they are the background and not the object of interest.

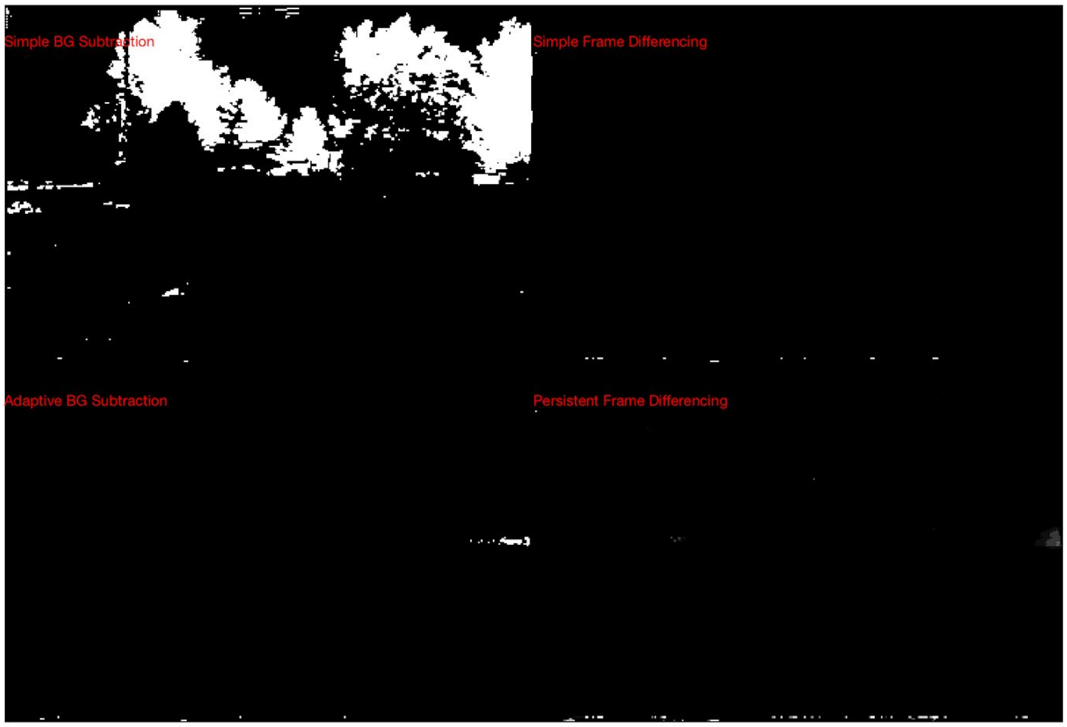


Figure 24. Moving camera leaves unwanted background imagery

The simple background subtraction algorithm fails when there is an object moving in front of a stationary object not classified in the background. The stationary object that is not part of the background is seen as a change already and is therefore seen in white. The object in motion will be displayed in the same manner as the stationary object, causing the object in motion to be lost in the frame.

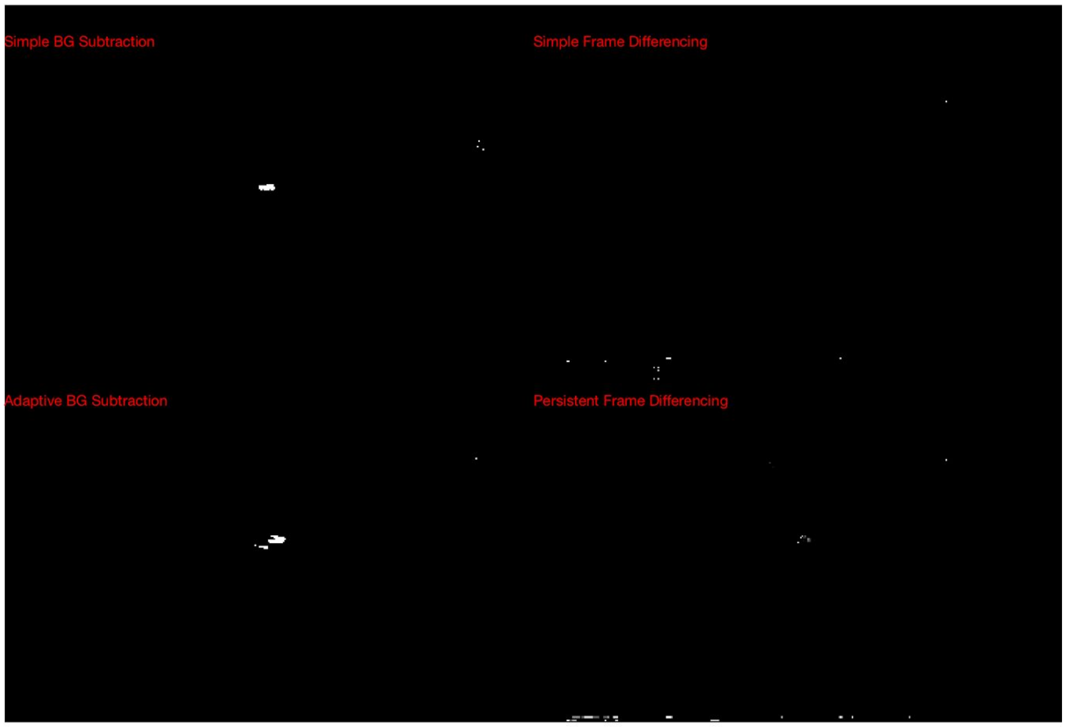


Figure 25. Object moving toward camera

Forward-moving objects can cause the adaptive algorithms to fail. Because the object remains relatively the same size in subsequent frames, it can be erroneously classified as part of the background.

Failure due to illumination changes makes the image appear to have moving objects when in reality there is only a change in intensity of the light and reflection. This failure occurs in all algorithms. This can be seen in Figure 8 and 9 as the tree appears to be a moving object.

Failure due to similar intensities in background and moving object can be seen in Figure 1. The pants of the man are similar to the concrete and are therefore not seen as a moving object. Only the top of the man is seen because there is a greater difference in intensity. This is a problem for all algorithms.

Simple background subtraction can extract the shape of an object only if the intensity or color is sufficiently different from the background. A failure can occur if the object is too similar to the background for background subtraction. Also if an object comes into the image and then stays in the image, any other object that passes in front of it will be missed. If an object was once there and leaves the scene it leaves behind a ghost object (getin.avi). Background subtraction cannot handle a moving camera and is sensitive to changes in illumination (movecam.avi).

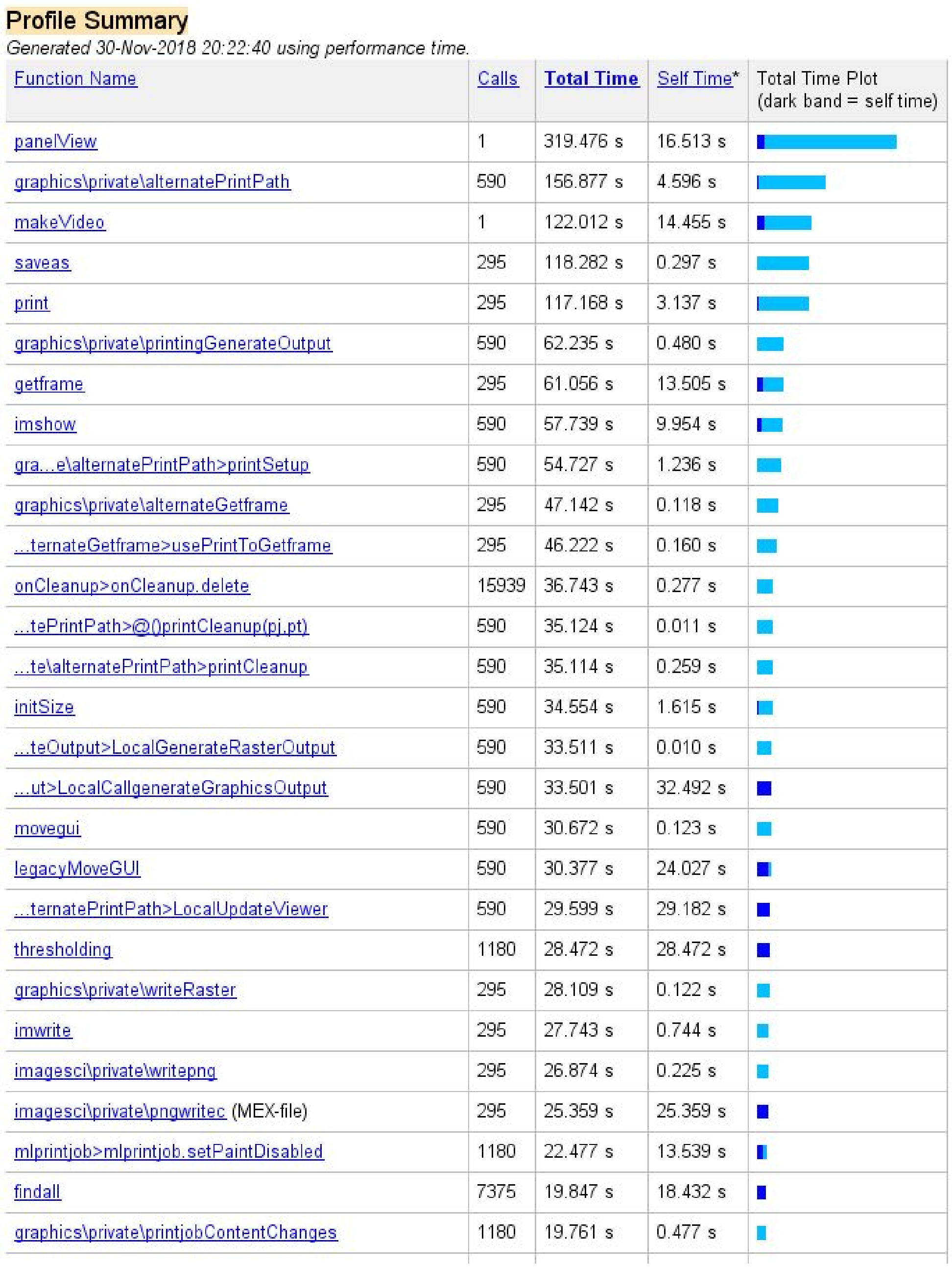
Simple frame differencing does not leave behind ghost objects if a once stationary object that was part of the background leaves the image. It will only detect the leaving and trailing edges of the object though. The frame rate will also affect the result of a moving object.

Adaptive background subtraction is more responsive to changes in illumination and camera motion. Faster moving objects will leave behind a short trail of pixels. The center of the moving object can also begin to disappear if the alpha parameter is not low enough.   But by lowering the alpha parameter it takes longer for stopped objects to disappear.

Persistent frame differencing is responsive to changes in illumination and camera motion and also allows ghost objects to fade away. The moving object leaves behind a trail indicating the direction of the object in motion. The center of the object is not detected. The leading and trailing edges can be made thinner or wider based on the linear decay parameter.

Adaptive background algorithm is shown to be the best algorithm to use for these 8 videos. While both the adaptive background subtraction and the persistent frame differencing algorithms adequately isolate the moving elements of the videos, the adaptive algorithm is superior at only isolating the moving elements without the “trail” behind the moving object. In a video where movement is easy to view from frame to frame, the movement trail has benefit and can hinder isolation efforts. If you are only looking at a single frame, the persistent algorithm would be better because it gives more context of the objects’ movement.

Algorithm Efficiency



Profile Summary

Generated 30-Nov-2018 20:22:40 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
panelView	1	319.476 s	16.513 s	
graphics\private\alternatePrintPath	590	156.877 s	4.596 s	
makeVideo	1	122.012 s	14.455 s	
saveas	295	118.282 s	0.297 s	
print	295	117.168 s	3.137 s	
graphics\private\printingGenerateOutput	590	62.235 s	0.480 s	
getframe	295	61.056 s	13.505 s	
imshow	590	57.739 s	9.954 s	
gra...e\alternatePrintPath>printSetup	590	54.727 s	1.236 s	
graphics\private\alternateGetframe	295	47.142 s	0.118 s	
...ternateGetframe>usePrintToGetframe	295	46.222 s	0.160 s	
onCleanup>onCleanup.delete	15939	36.743 s	0.277 s	
...tePrintPath>@()printCleanup(pj,pt)	590	35.124 s	0.011 s	
...te\alternatePrintPath>printCleanup	590	35.114 s	0.259 s	
initSize	590	34.554 s	1.615 s	
...teOutput>LocalGenerateRasterOutput	590	33.511 s	0.010 s	
...ut>LocalCallgenerateGraphicsOutput	590	33.501 s	32.492 s	
movegui	590	30.672 s	0.123 s	
legacyMoveGUI	590	30.377 s	24.027 s	
...ternatePrintPath>LocalUpdateViewer	590	29.599 s	29.182 s	
thresholding	1180	28.472 s	28.472 s	
graphics\private\writeRaster	295	28.109 s	0.122 s	
imwrite	295	27.743 s	0.744 s	
imagesci\private\writepng	295	26.874 s	0.225 s	
imagesci\private\pngwritec (MEX-file)	295	25.359 s	25.359 s	
mlprintjob>mlprintjob.setPaintDisabled	1180	22.477 s	13.539 s	
findall	7375	19.847 s	18.432 s	
graphics\private\printjobContentChanges	1180	19.761 s	0.477 s	

Figure 26. PanelView.m efficiency

As seen in Figure 26, the total time to properly run through was 319 seconds. The function that took the most time was makeVideo. This is due to the heavy processor load of rendering a video from 295 individual frames. This project requires that each frame be saved after running through the various motion detection algorithms. The act of writing these images to memory takes time (depending on the computer) and cannot be avoided. The number of images to write also plays a factor, since the more images to save, the longer it will take to finish. The makeVideo function is as optimized as we can make it.

Member Contributions

Brian wrote the majority of the code for the team while Dan, Kara, and Mark wrote the report, helped to comment the code, and format the final submission. We all worked together on ensuring the project was completed on time and done to the full extent of the project specifications.