

**CMPEN 454, Project 3, Fall 2018**

Xi Li  
Qi Chang  
Eryk Maciej Heyssler  
Brian Mata

## ***Introduction***

The goal of this project was to implement four separate methods of tracking, namely simple background subtraction, simple frame differencing, adaptive background subtraction, and persistent frame differencing, and apply them onto a number of given videos, to ultimately compare and contrast the differences in their functionality. The first focus was on the specifics of the algorithms themselves; how each procedure works within a mathematical sense, and how it is able to detect changes within the video. The second focus was on the differences between the different procedures. It was very interesting to see how much the methods differed from video to video, specifically how each reacted to factors such as scale and noise, and other, more subtle variables such as the effect of a moving ‘background’ object. Each algorithm had its own specific nuances on a case to case basis. In other words, one algorithm wasn’t strictly “better” than the other, it simply depends on the situation and how one wishes to detect change in video.

## ***Method***

### **Simple Background Subtraction**

Simple background subtraction is arguably the most straightforward and uncomplicated of the 4 algorithms. It assumes the background as a static image with no objects present, therefore anything that moves is considered to be an object. Intuitively, this would work best when presented with an initial frame in which there truly are no objects, meaning that nothing within the frame is subject to move. Any object that moves must originate from outside of the frame for this method to be effective. Therefore, this method is prone to vulnerabilities such as a parked car that suddenly starts moving, trees that move subtly in the wind, and changes in the camera.

Implementation:

```
Background = Image_matrix(:,:,1);  
Image_matrix_SBS = zeros(m,n,num-2);  
parfor i = 1:(num-2)  
    image_temp = abs(Image_matrix(:,:,i)-Background);  
    Image_matrix_SBS(:,:,i) = threshold(image_temp, lamda);  
end
```

### Simple Frame Differencing

Frame differencing is much similar to background subtraction, only differs in that the background is not static any more and is replaced with the previous frame. Therefore, frame differencing quickly adapts to changes in lighting or camera motion and solves some problems in background subtraction. It eliminates the ghosts behind the objects that start up, as well as the stopped objects. However, the moving objects detected by frame differencing are not solid, only edges. Besides, this method fails to detect an object moving towards or away from the camera.

#### Implementation

```
Background = zeros(m,n);
Image_matrix_SFD = zeros(m,n,num-2);
num_jump = 0;
for i = (1+num_jump):(num-2-num_jump)
    image_temp_add = abs(Image_matrix(:,:,i+num_jump)-Background);
    image_temp_min = abs(Image_matrix(:,:,i-num_jump)-Background);
    image_temp = MatrixAnd(image_temp_add,image_temp_min,lamda);
    Background = Image_matrix(:,:,i);
    Image_matrix_SFD(:,:,i) = threshold(image_temp,lamda);
end
```

### Adaptive Background Subtraction

Adaptive background subtraction combines background subtraction and frame differencing therefore solves most of the problems listed above. When building the background model, it gets the weighted sum of BS and FD:  $\alpha$  to the previous frame and  $1-\alpha$  to the cumulated background model. Thus,  $\alpha = 0$  yields background subtraction and  $\alpha = 1$  yields frame differencing. However, this method introduces trails behind fast moving objects and causes fade in the centers of large slow moving objects.

#### Implementation

```
Background = zeros(m,n);
Image_matrix_ABS = zeros(m,n,num-2);
alpha = 0.3;
for i = 1:(num-2)
    image_temp = abs(Image_matrix(:,:,i)-Background);
    Background = alpha*Image_matrix(:,:,i) + (1-alpha)*Background;
    Image_matrix_ABS(:,:,i) = threshold(image_temp,lamda);
end
```

### Persistent Frame Differencing

Persistent frame differencing will calculate the difference between the current frame and previous frame first. And then it will combine the previous difference information in a proper way to show the trajectory of the moving target. It is useful especially to show the trail where the crowds are following. More interesting, it will enhance the video by a feeling of drifting.

#### Implementation

```
Background = zeros(m,n);
H = zeros(m,n);
gamma = 0.01;
Image_matrix_PFD = zeros(m,n,num-2);
for i = 1:(num-2)
    image_temp = abs(Image_matrix(:, :, i) - Background);
    M = threshold(image_temp, lamda);
    Temp = (H - gamma) .* threshold(H - gamma, 0);
    H = M .* threshold(M - Temp, 0) + Temp .* threshold(Temp - M, 0);
    Image_matrix_PFD(:, :, i) = H;
    Background = Image_matrix(:, :, i);
end
```

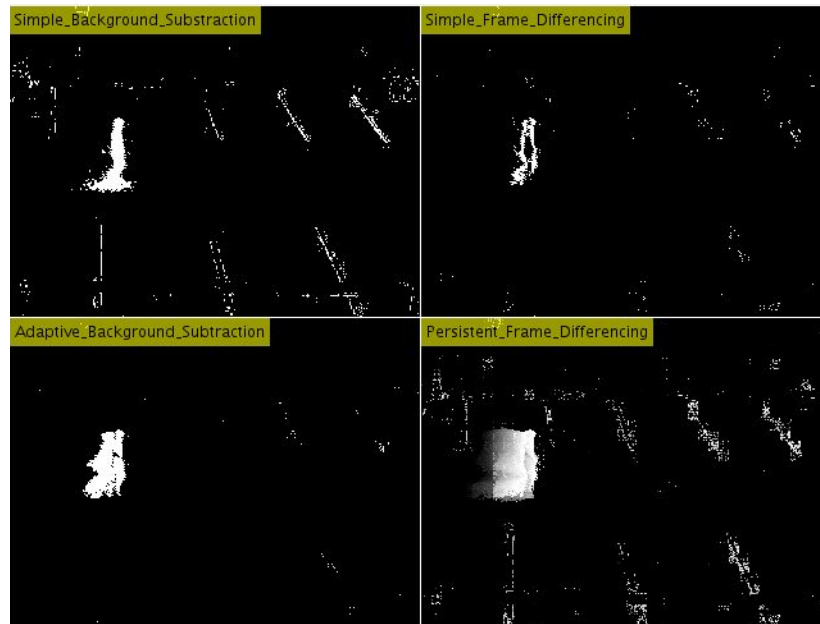
### ***Results***

#### Quantitative results

For **pixel difference threshold**, we choose 0.15 after several attempts of numerical substitution. When we set it higher, like 0.30, the objects will lose shape. In the figure below, the heads are almost gone.



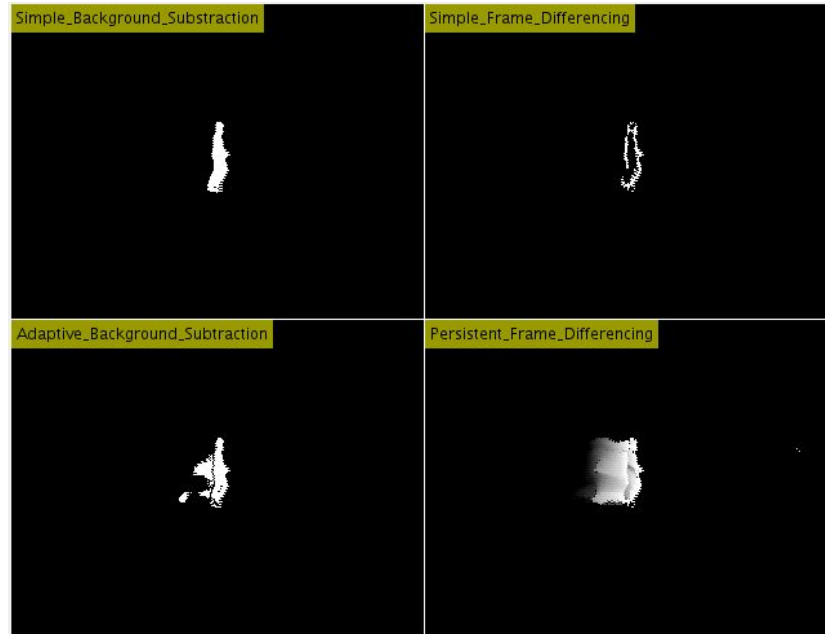
When we set it lower, like 0.05, there are noises in the output images. In the figure below, there are many white dots which are not the desired objects.



With regard to parameter “**alpha**” in adaptive background subtraction, it is set to 0.30 which is also the result of numerical substitutions. In the following figure, when we tune alpha to 0.8, the edge of object in adaptive background subtraction is almost as thin as that in frame differencing.

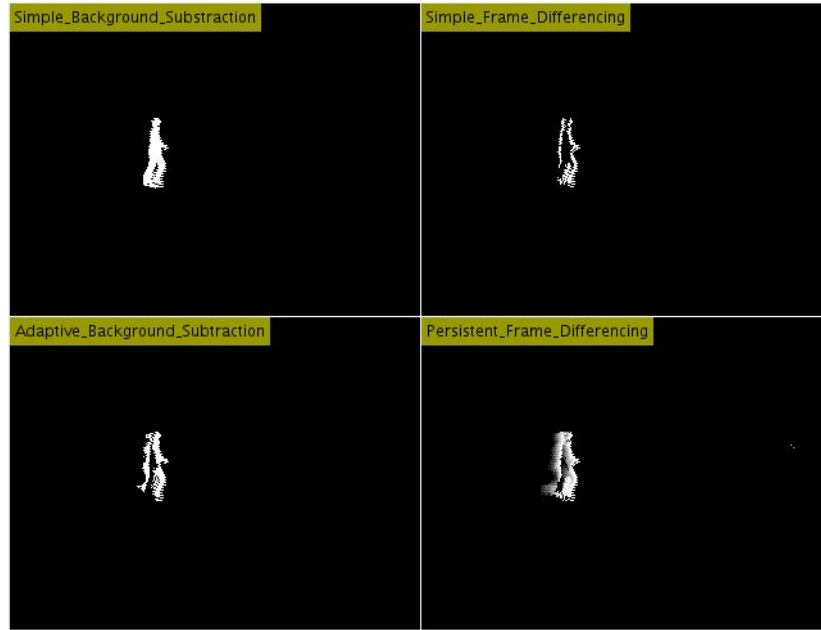


And when we set it to 0.1, the object edge in adaptive background subtraction is much thick than that in frame differencing, as the following figure shows.

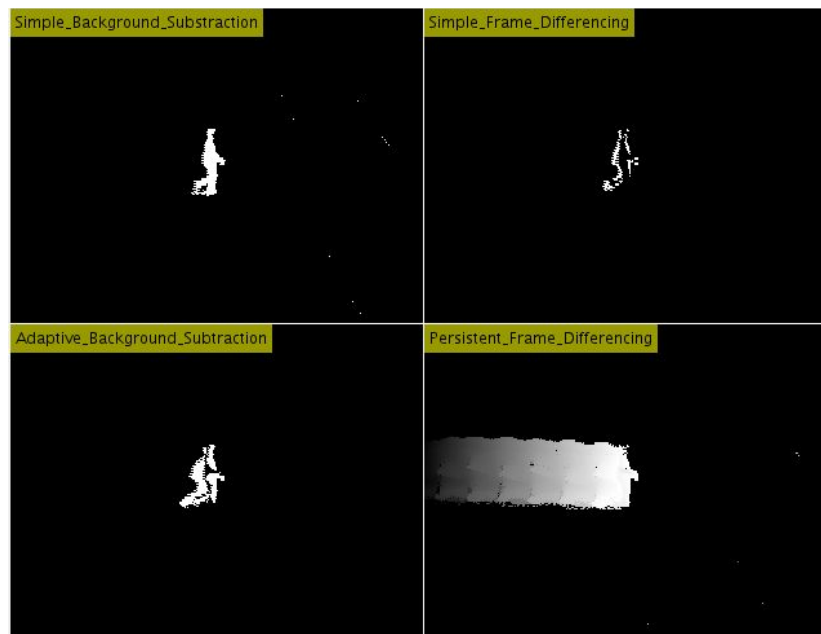


Adaptive background subtraction is actually frame differencing when alpha is 1 and is background subtraction when alpha is 0. Thus, when we set alpha a higher value, the weight of frame differencing is increasing, and the edge of object gets thinner. When alpha gets smaller, the weight of background subtraction is increasing, so the object has thicker edge and even becomes solid.

As for **linear decay parameter**, we set it to 0.05 which, still, is derived from numerical substitutions. When we set it higher, like 0.2, we observe that the trail is almost gone but the object becomes hollow.

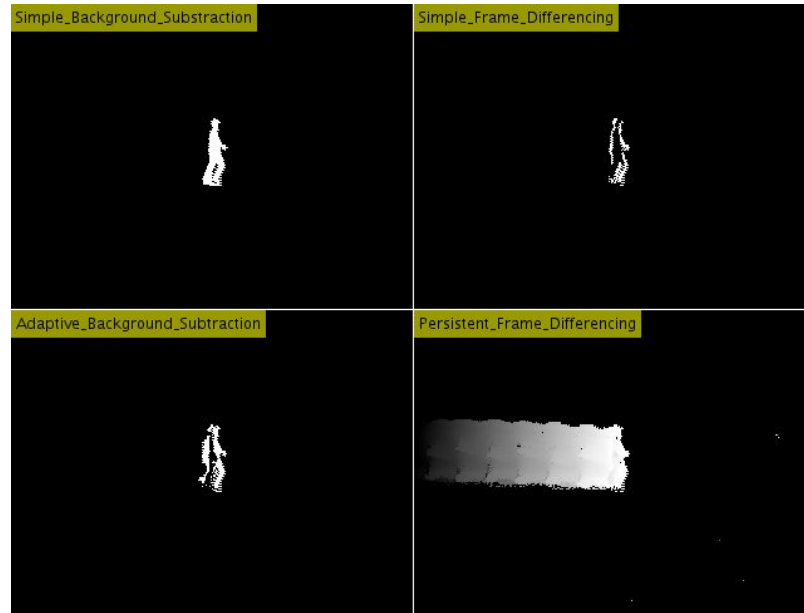


Then we set it to 0.01 and find that although the object is solid again, the trail is too long. Based on these observations, we choose 0.05 at last, which gives a solid object with toleratable trail.



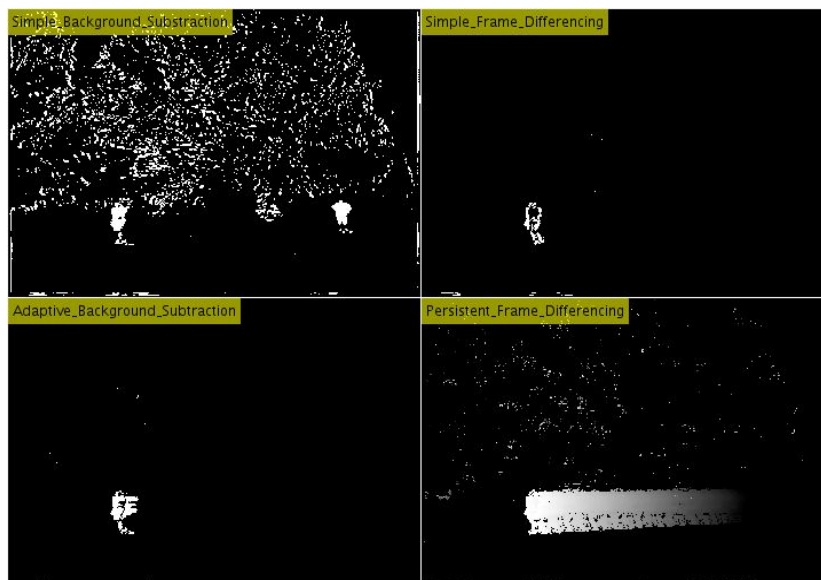
## Qualitative results

### *1. walk*



This video was the simplest case from the dataset. It involved a static background with a single object passing through it. There aren't any changes to lighting or in the camera, nor does this video have a particularly large amount of noise, so each method is able to detect the walking figure easily.

### *2. Trees*





The biggest caveat in this video is the presence of the rustling trees which results from the movement of the leaves, as well as their small changes in lighting as they move. As one can plainly see, simple background subtraction treats each tiny change in position as its own motion, and makes these results very apparent, due to the fact that it treats every moving object as important. On the other hand, adaptive background subtraction does not detect as much of this same movement. This is due to the fact that the individual movement of the leaves is relatively slow, so much so that the leaves blend in to the background at a rapid pace. Simple frame differencing nearly mimics this idea, as it is very responsive to these changes, and displays them for an incredibly short period of time as they are occurring. Persistent frame differencing is similar, though due to its trails, makes it much more obvious that there is motion in this area which cannot be otherwise misinterpreted for noise.

It is also important to note how the methods handle the movement of the person. Because the person is initially thought of as the background, simple background subtraction leaves a 'hole' in the initial location of the person. The person is also moving at a moderately fast pace, which is made apparent in the trails left behind by adaptive background subtraction and persistent frame differencing.

### 3. *Movecam*



Simple\_Background\_Subtraction



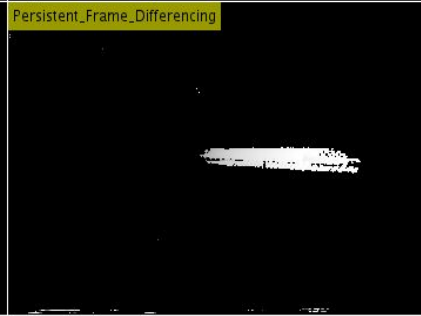
Simple\_Frame\_Differencing



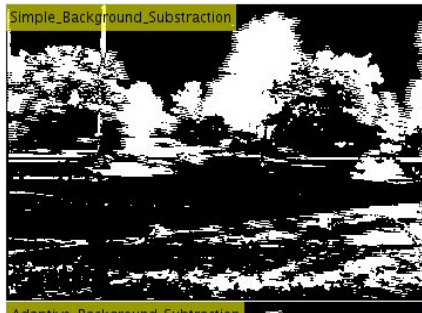
Adaptive\_Background\_Subtraction



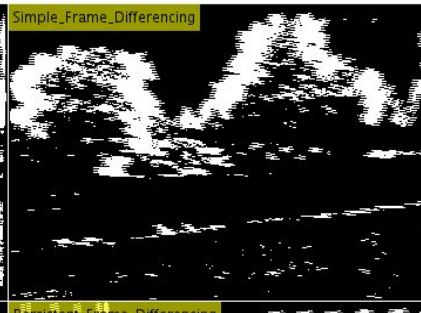
Persistent\_Frame\_Differencing



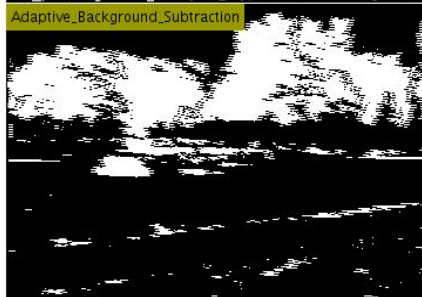
Simple\_Background\_Subtraction



Simple\_Frame\_Differencing

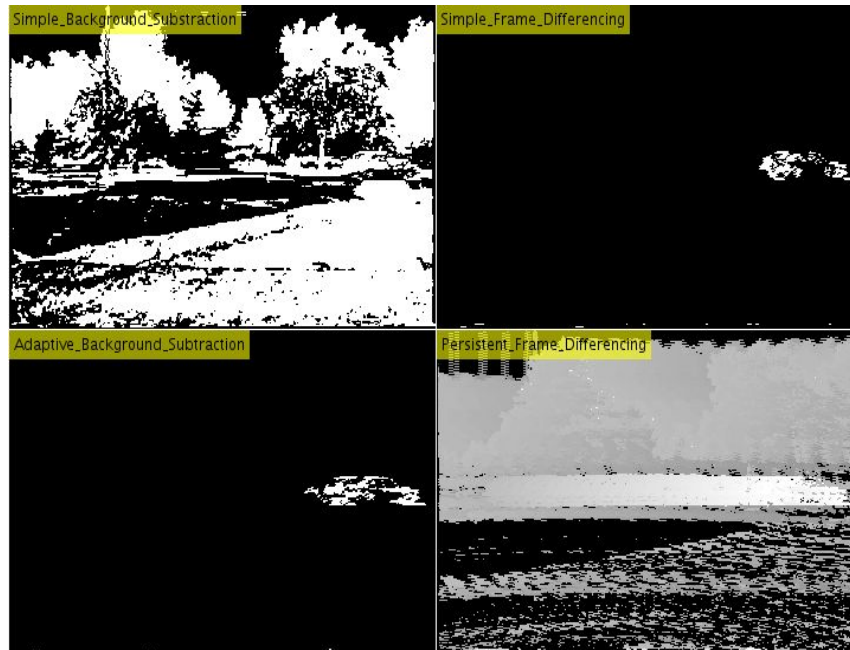


Adaptive\_Background\_Subtraction



Persistent\_Frame\_Differencing



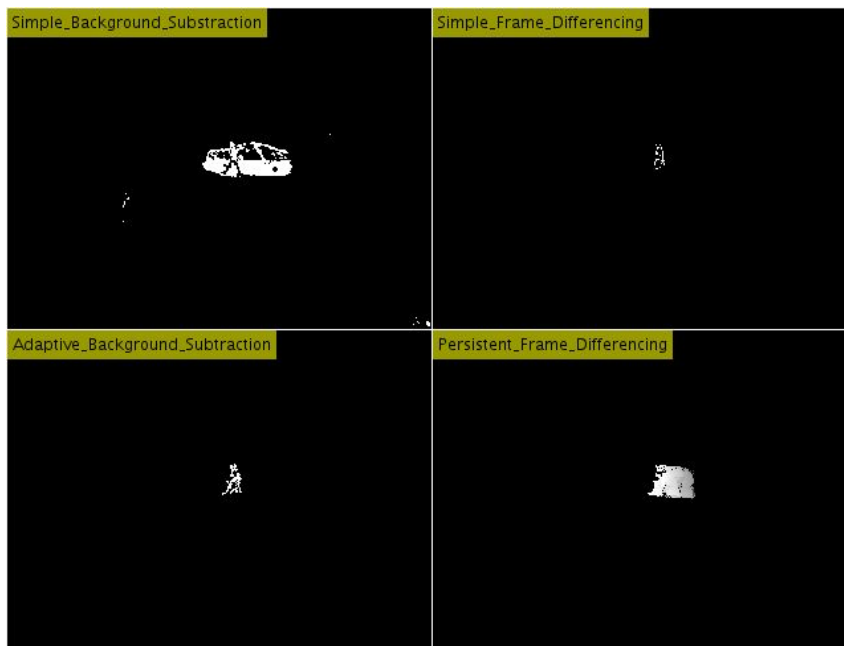


This video introduces a moving camera, which yields some interesting results. The first thing of note is how simple background subtraction handles the motion of the camera. Since it assumes a static background, any slight deviation from it will be detected as an object. Hence, the actual shift of the camera is detected as an object, and there is now what can be described as a 'hole' left over from the background. The other methods detect this change as well, though briefly, and do not leave it lingering behind as if it were an entirely separate object.

Another thing to note is how the methods handle movement towards the camera. While simple background subtraction is severely hindered by its inability to handle camera motion, it does detect the moving car when it is approaching directly towards the camera effectively. On the other hand, simple frame differencing does an especially poor job of this due to the fact that it only detects the leading and trailing edge of the car.

In the moment where the car slows down drastically to turn, every method other than simple background subtraction has a hard time, due to the fact that the differencing operations are based on previous frames, meanwhile adaptive background subtraction tends to fade very slow moving objects into the background (as if they were stopped).

#### 4. *getout*

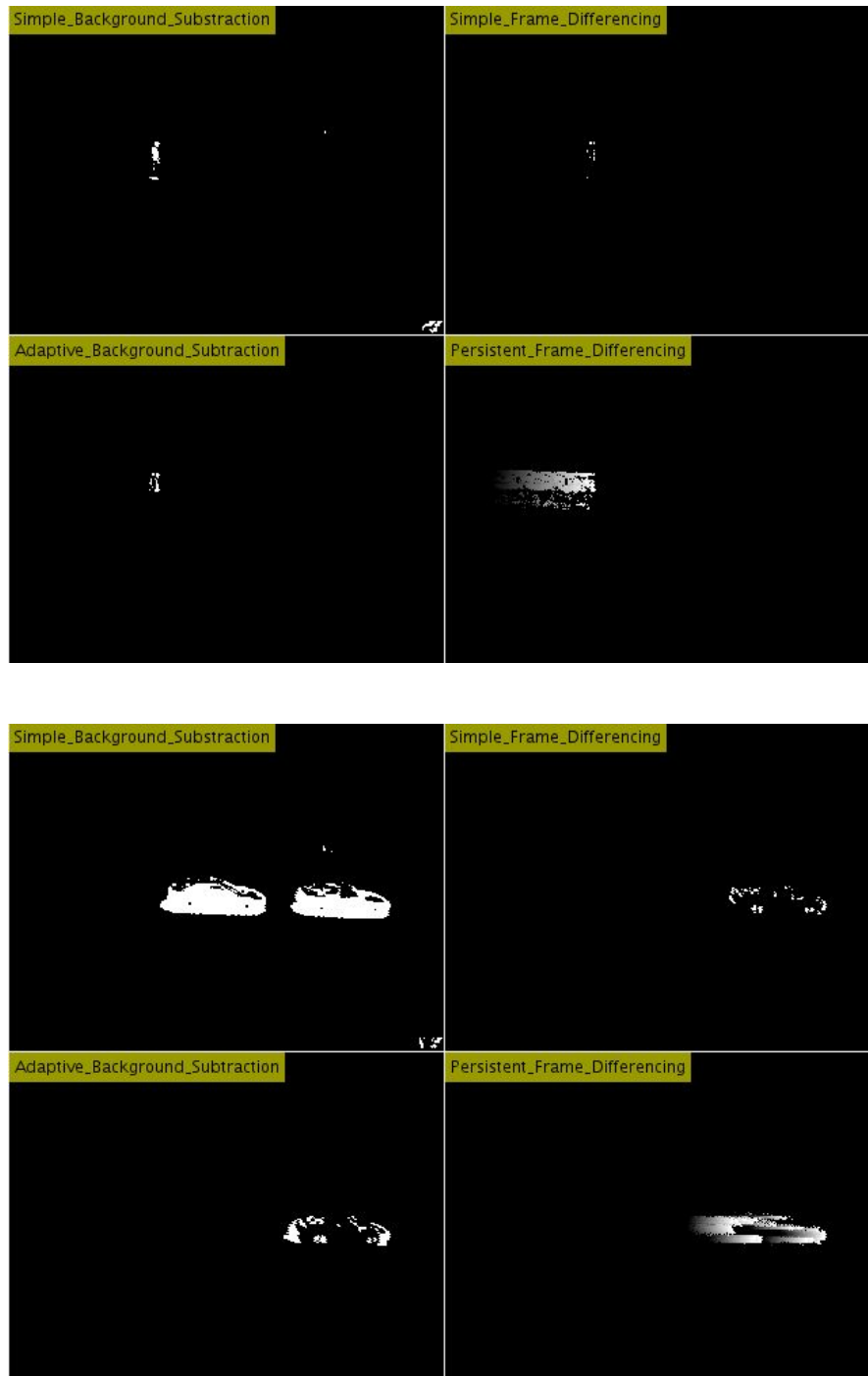




This video involved a car coming into frame, stopping, and then a person getting out of the car. Unsurprisingly, once the car stopped moving, simple background subtraction did not identify the car as part of the background- it instead kept it as a static object. Simple frame differencing had the most trouble detecting the person as he was getting out of the car, since it is a slow movement, most of which is towards the camera.

The methods were able to detect the man as he walked away, to some degree. Since his pants were of a similar color to the background, only his torso was being detected. This is made obvious in the frame above with respect to simple background subtraction-- the man's pants are colored black signifying that they are 'background' while the rest of him is white. The trail left behind from persistent frame differencing actually proved beneficial with regards to this color similarity, as it made the presence of the person much more obvious.

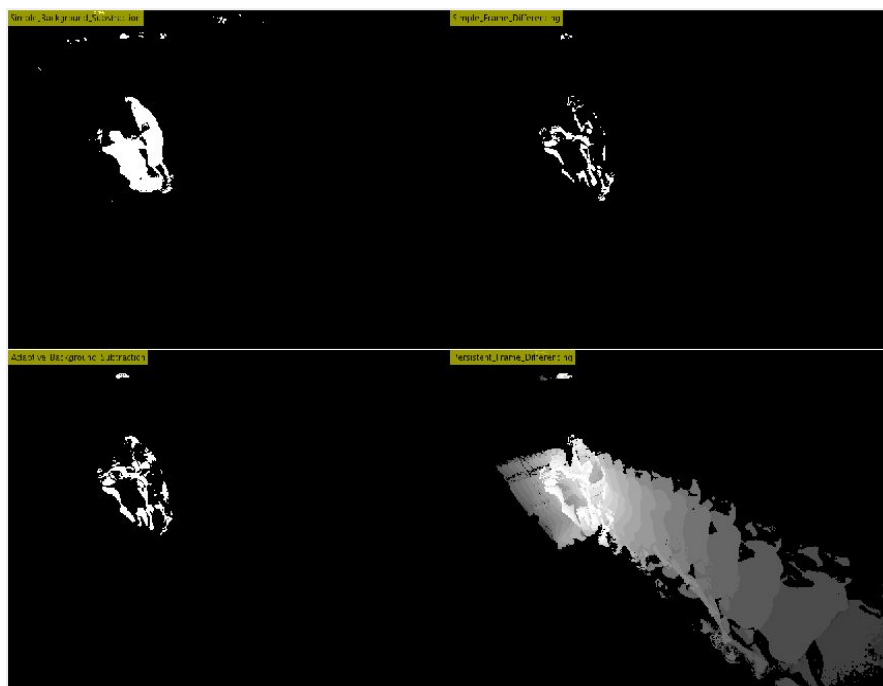
## 5. *Getin*

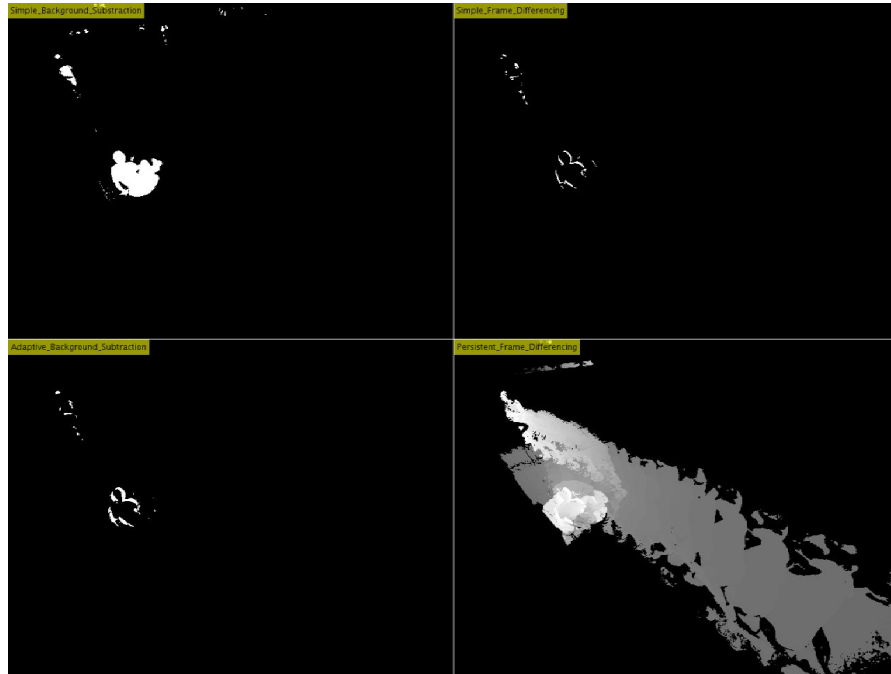


As in the trees example, we are presented with an object which is initially part of the background. Unsurprisingly, simple background subtraction leaves a ‘hole’ at the initial location of the car once it starts moving away. An important thing to note is that there is an occlusion that occurs once the person is about to enter the car. Each algorithm is able to detect the top of the person's head once they get behind the car, though they disappear once they enter.

Because of the relatively slow speed of the car, simple frame differencing has a somewhat difficult time detecting the car or the person when it is moving. This is because of the low delay that was set between frames. Naturally, adaptive background subtraction and persistent frame differencing handle this slow movement better due to the presence of their trails.

## 6. *ArenaW*



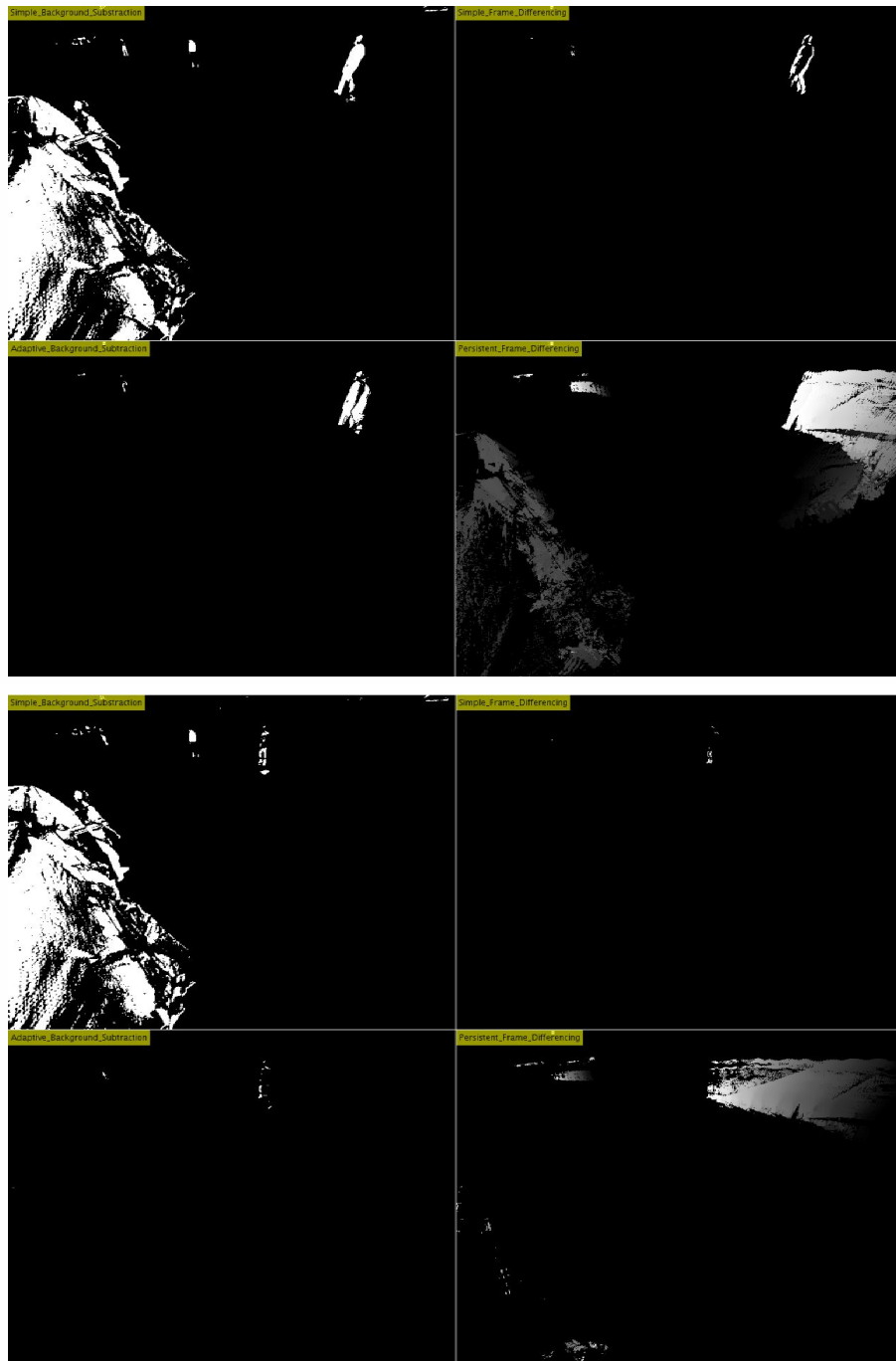


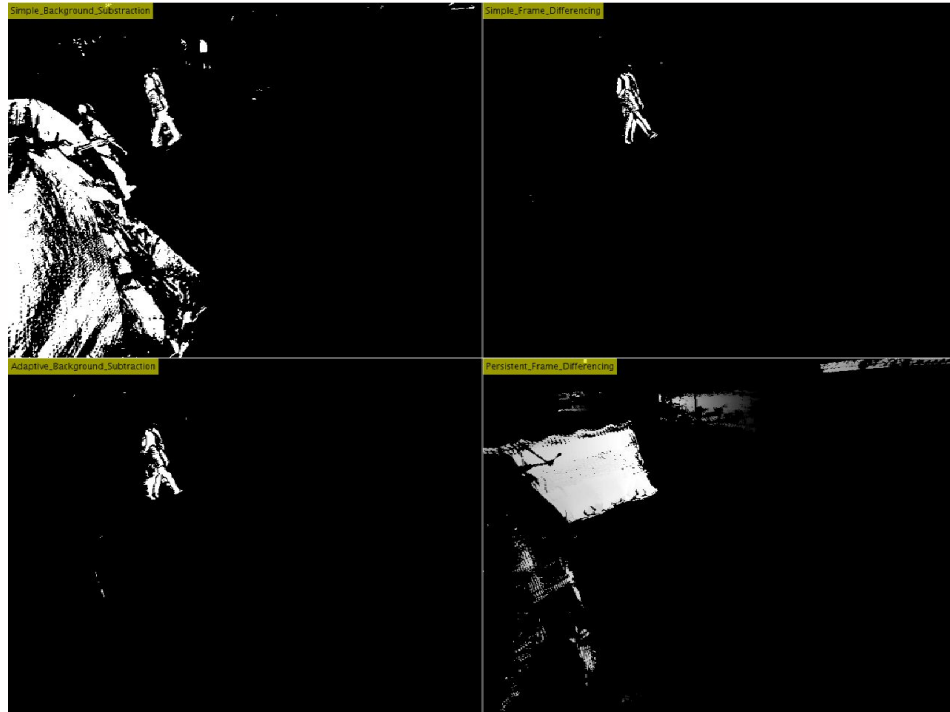
The first event of interest is when the second man falls. In simple frame referencing and adaptive background subtraction, the second man nearly disappears, even though he is arguably an object of importance. This is because while he is on the ground, he has come to a near standstill, and so he simply fades into the background.

After the fall of the second man, it is a little difficult to interpret through simple background subtraction, or either of the frame differencing methods, where exactly the first man has gone. The only method that makes his location somewhat apparent is persistent frame differencing, where he is seen to run off to the far background and slightly to the left. Once again, because of the increase in width of the edges due to linear decay, the man is easier to see. The worst visibility with respect to this instance is in simple frame differencing. This is because the man at this point is moving directly away from the camera, and, because simple frame differencing only detects the leading and trailing edges, the man is difficult to spot.



## 7. *ArenaN*





In this database, we can see some people are walking in a warehouse of some kind. Or at the very least, it looks like there is some railing obstructing our view. If we were to perform some kind of tracking, we should first try to join disconnected parts using some kind of dilation or other technique. Also, background subtraction had trouble with transient objects such as someone walking in the corner of the screen for a brief moment before leaving.

However, when we are going to show and save the image. The codes feedback us an error, which is the system out of memory. The error happens in because we use 12 worker at the same time and each image frame is larger than the other image sequences. To solve this problem, we give up the methods of parallax computing and then use the normal way. The code is attached on matlab code on the final section.

## 8. *ArenaA*



This video is the one in which the differences between the algorithms were most apparent. For starters, only simple background subtraction was able to identify the subtle movement of the people in the background, and the front door of the truck that was opened. As a consequence to its sensitivity however, it was the one that most explicitly displayed the large tree in the foreground. Another important point to note about simple background subtraction is that it

detects the people in the front once they all meet and are at a standstill. This is contrary to adaptive background subtraction and simple frame differencing, which are very vulnerable to slow or stopped objects. Persistent frame differencing was able to mitigate this somewhat through its trails; it made it evident that something was happening even though it was hard to tell exactly *what* was happening. Simple frame differencing was the worst one for this video, as it never showed an accurate representation of exactly how many people were present at any given moment, due to the short delay and relatively little amount of movement once the people were in a cluster.

### Interpretation

From running the methods on various scenarios, it became clear that no one method was strictly better than the other. Each method brought a different advantage based on the situation, while simultaneously having a detrimental effect in others. For example, simple background subtraction was great at detecting very subtle movement, and movement towards and away from the camera. However, it became extremely ineffective upon changes in camera and moving objects that were a part of the background. Simple frame differencing was able to handle these cases, though it had possibly the worst visibility, especially when it came to very slow movement, or movement towards and away from the camera. Adaptive background subtraction was actually quite similar to simple frame differencing, though it was able to pick out slow movements more effectively due to the trails on the objects. Unlike simple background subtraction, it did not leave ‘holes’ behind in objects that were once part of the static background.

If we had to choose one method for motion detection however, it would have to be persistent frame differencing. Of course, there are a few downsides to this method. Subtle movement that may be deemed unimportant, like the rustling of the leaves in “*trees*” and the large branch swaying on the left hand side of “*ArenaA*” become more pronounced than one would probably like. In addition, the ‘ghost trail’ might obscure some activity especially if multiple objects are moving together within a close proximity. Nevertheless, we chose this method because we think that it accomplishes the task of motion tracking the most effectively, despite its drawbacks. For starters, slow moving objects are easily tracked due to the trails that this algorithm produces. This is especially important because we cannot assume all objects in nature are going to be moving at a constant speed-- some are bound to slow down or even stop completely. A great example of this was seen in “*ArenaA*”, where the second man fell down, and became nearly invisible in simple frame differencing and adaptive background subtraction. This ease of visibility is also important in objects that are moving towards the camera, or away from it, which persistent frame differencing is able to handle well, again, as seen in “*ArenaA*”. We mentioned earlier that the trail can obscure activity when objects are in close proximity. While

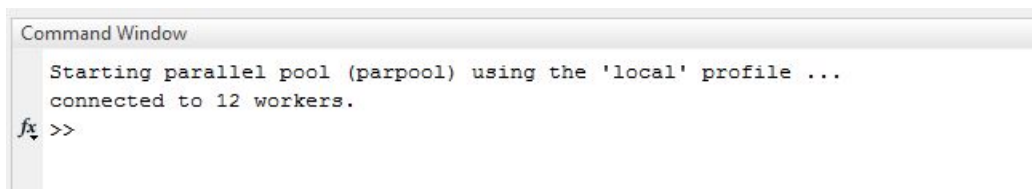
this is true, there is also a slight advantage to this; the trail might make the activity itself a bit unclear, but also makes it obvious that there *is* activity. Objects in close proximity often overlap themselves, which would lead to occlusion of activity (of the objects) anyway, and thus persistent frame differencing would be most effective, especially if the objects were moving slowly.

### Algorithm Efficiency

The MatLab® profiler is a built in tool that measures where code takes up the most time upon running it-- it is essentially a tool that can be used to improve efficiency. The following image shows the data from the function *main*, which is the bulk of where our code takes spends its time. Unsurprisingly, the function call to *imshow* is the largest portion of the time spent, taking up 62.1% of the total time. Our next most significant strain on our resource is line 73, which consists of `parfor i = 1:num-2`. This line is a for loop responsible for showing the result of our program, and takes up 26.4% of the total time. The next two lines are line 9 and 23, which also represent for loops. Line 9 essentially loads the data from the dataset, while line 23 is accountable for simple background subtraction. What was surprising about the result of the profiler was the fact that the other loops which were involved with the other methods were not included in this top 5 list of most resource-dependent lines. We expected them to hold much higher places in the ranking.







### Parallax Computing

In this project, we were inspired to use the idea of the parallax computing to process the images in video reading section, simple background subtraction section and the final combination section. The reason why we can use this is that the each loop is independent from each other. In another words, we can parallax process the image frame in the same time. Its efficiency will depend on the worker on CPU. It will reduce the time to process a huge sequence of images.



```
Command Window
Starting parallel pool (parpool) using the 'local' profile ...
connected to 12 workers.
fx >>
```

### Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
<a href="#">81</a>	<code>imshow&gt;Show_Matrix(:,:,i));</code>	211	17.052 s	62.1%	
<a href="#">73</a>	<code>parfor i = 1:num-2</code>	1	7.256 s	26.4%	
<a href="#">9</a>	<code>parfor i = 3:num</code>	1	1.123 s	4.1%	
<a href="#">23</a>	<code>parfor i = 1:(num-2)</code>	1	0.387 s	1.4%	
<a href="#">36</a>	<code>image_temp = MatrixAnd(image_t...</code>	211	0.240 s	0.9%	
All other lines			1.407 s	5.1%	
Totals			27.466 s	100%	

## Conclusion

In this project, we learned about 4 methods of motion tracking, from the most basic, intuitive algorithm, to ones that were more complex. We observed that each algorithm was able to motion track to some extent, and that their functionality was in part based on a case by case basis. There were also some interesting caveats to the threshold values, which greatly affected the method's effectiveness in identifying objects. This also led to an intriguing insight on the relationship between the methods. As an example, when the value of alpha is changed to 1, adaptive background subtraction becomes frame differencing. Perhaps the most fascinating part of this project is how much of the results were qualitative; one can only imagine the various applications to which any of these individual methods could be applied to, for whatever effect one could desire.

We had a total of two meetings to discuss the project, during which everyone was present. We first met briefly just to look over the project and see what was to be done. By the time thanksgiving break was over, we were finished with the coding part of the project, which was assembled in a group effort. During the second meeting, we all started the report. Xi and Qi primarily worked on the quantitative results, while Eryk and Brian worked on the qualitative results, though everyone had a contribution to each part of the entire report.