Image Processing
Motion Detection

- Problem Description

Problem description

- One of the greatest challenges in the digital world is detecting and tracking moving objects in live videos. Although human is good in this field, automating this procedure is challenging. For this purpose, there are some methods such as frame differencing and Background Subtraction; however, we still need to accompany these methods with some Image Processing techniques to improve the result. In this project I have used these advanced approaches along with some techniques like contour, threshold, noise reduction, and dilation.

Input:

input file could be any video or webcam that we want to detect moving objects in it. This is one sample frame of our input video.

Output:

Our desire output file is a video that constrict moving objects with rectangles, synchronously. This is one sample frame of our output.

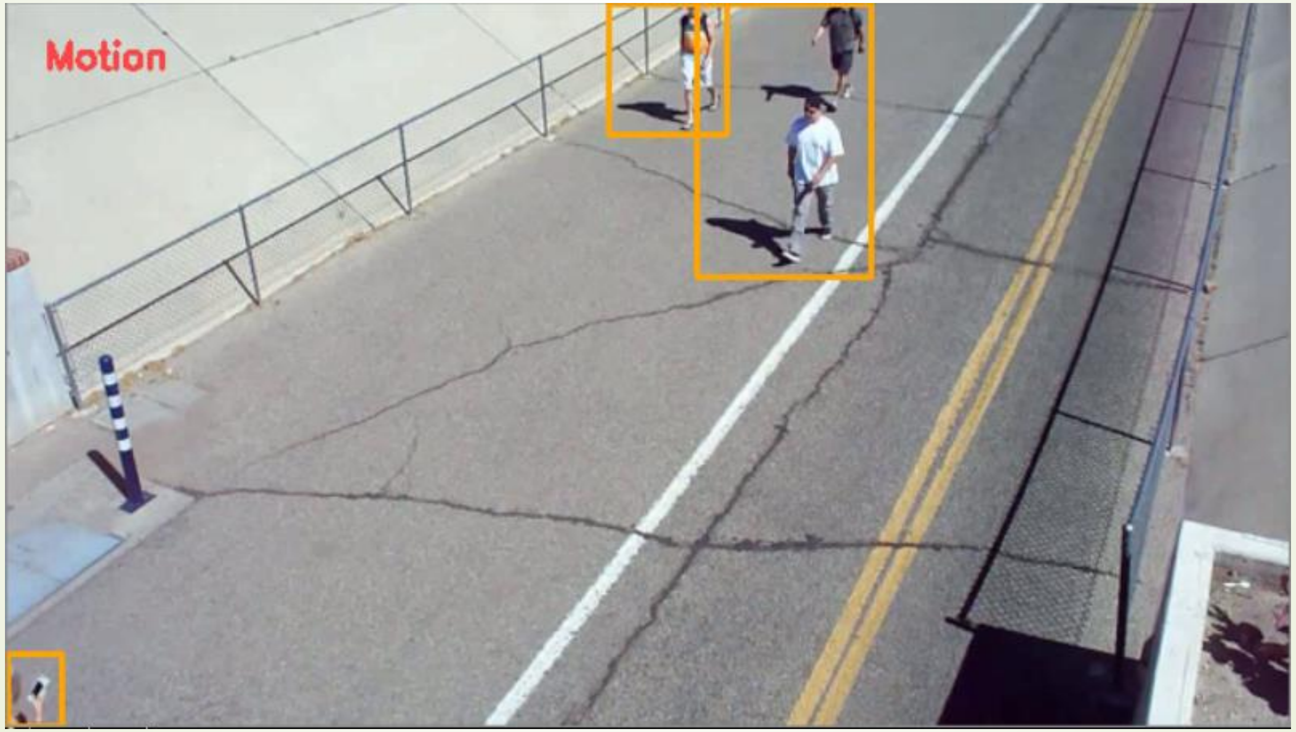


Image Processing
Motion Detection

- Work Process

A. Fame differencing method

A. Step 01

preparation

- Fetching Two frames from video or webcam
- Converting these frames to Grayscale mode to reduce complexity.
- Subtracting intensity of each corresponding pixels in these two frames in order to find pixels that are more probable to be part of moving objects.

Original frames

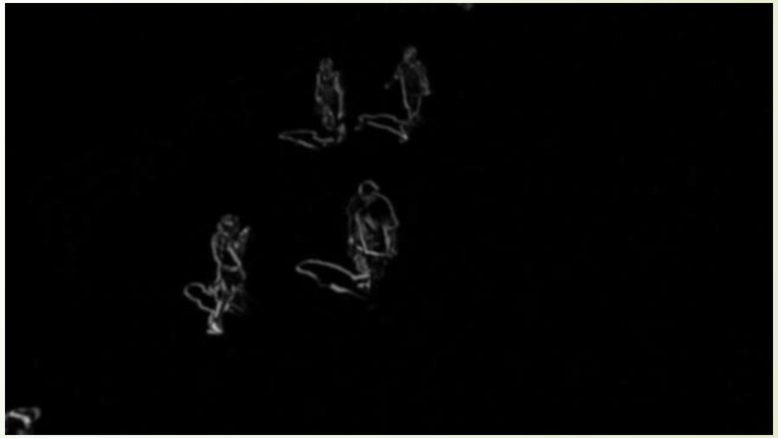
Grayscale mode

Frame difference

A. Step 02
Noise Reduction

- Noise is an inevitable part of digital images. We need to deal with the noise first to make next steps more efficient. For instance, noise could have a detrimental effect on edge detection. For this purpose, we used the GaussianBlur filter with different kernel sizes, and the results are as below.

GaussianBlur (3,3)

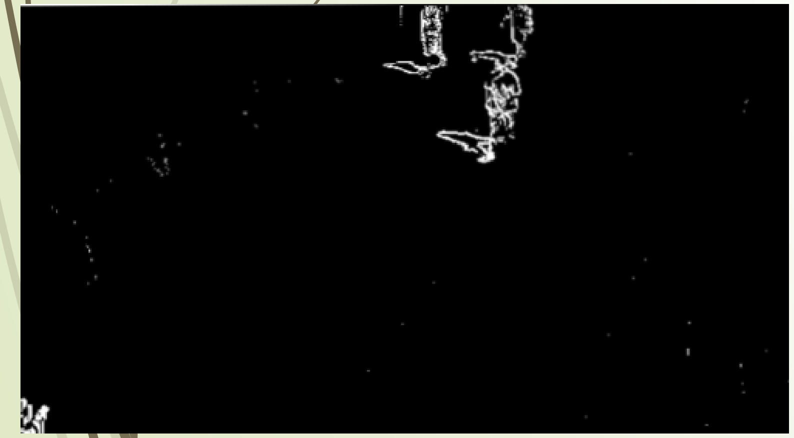


GaussianBlur (5,5)

A. Step 02

Noise Reduction

- For comparing blurred images with different kernel sizes, It is better that we compare their corresponding outputs after the threshold step. In the blurred image with a kernel size of (3,3), we have filtered noise significantly. In the kernel (5,5), although we have been able to remove the noise, we have lost some valuable details.



Threshold of Original

Threshold of GaussianBlur (3,3)

Thresholf of GaussianBlur (5,5)

A. Step 03

Threshold

- *Problem:* In some frames like the one in the next slide, we can find some pixels that although are not a part of our moving object, have been detected as moving pixels. And this is because that frame differencing technique is sensitive to intensity change, and any illumination change can cause false detection ,which categorizes those pixels as moving ones.
- *Solution:* In this stage, we use threshold technique to eliminate pixels with a very small intensity level. These pixels are considered noisy pixels and are less likely to be part of our moving object. In this technique, we specify an intensity value as a threshold. Any pixel value lower than this cut off number is reinitialized to zero, and others are set to zero.
- *cv.threshold():* Input variables of this function are: a frame in grayscale, a threshold value, the max number for pixels that exceed the threshold, and the thresholding type

A. Step 03
Threshold

- As we can see in the image with threshold=10, we have a significant number of false-positive pixels, and in the image with threshold=30, we have lost valuable pixels belonging to our moving objects. But in the image with threshold=20, we have been able to preserve moving objects pixels as much as possible and filter false-positive pixels simultaneously.

Threshold 10

Threshold 20

Threshold 30

A. Step 04

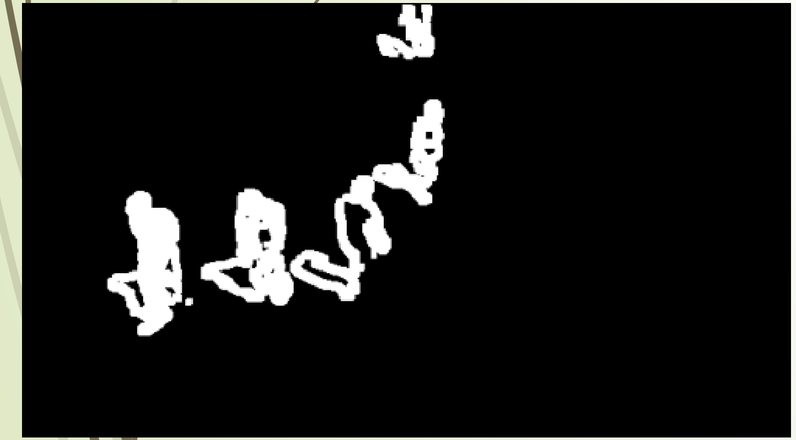
Dilation

- *Problem:* After the threshold stage, we end up with some pixels belonging to our moving objects but unfortunately have not recognized by previous techniques. These missed pixels can be seen as holes in moving objects.
- *Solution:* For solving this issue, we can fill these holes with some approaches like Closing and Dilation. In this stage, we benefit from the dilation technique.
- *cv.dilate():* Dilation function has a few arguments like src, dst, element, anchor, iteration, borderType, and borderValue. These elements refer to the source image, output image, the shape of the kernel, location of kernel center, number of times the dilation is repeated, and bordering values in cases that kernel is working on the border pixels of an image, respectively.

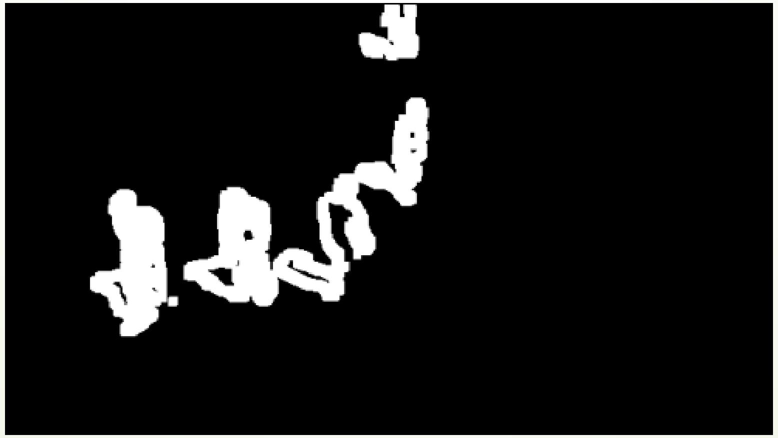
A. Step 04

Dilation

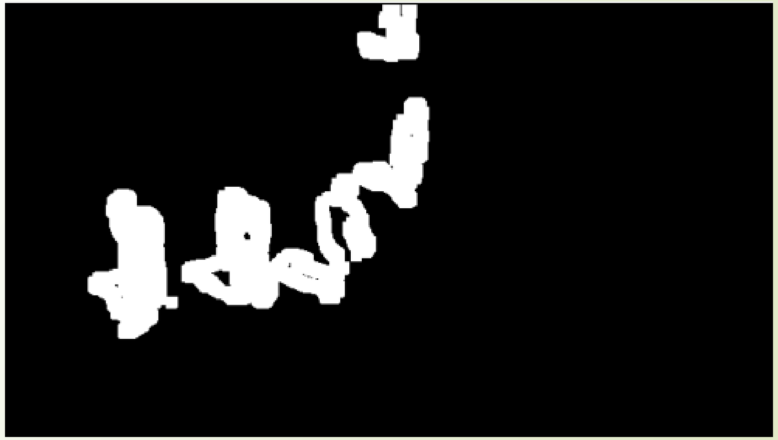
- In the image with iteration 3, we can see better result than 2 or 4 as it could efficiently fill the holes while reasonably maintains the objects separated.



Dilation, Iteration=2



Dilation, Iteration=3



Dilation, Iteration=4

A. Step 05

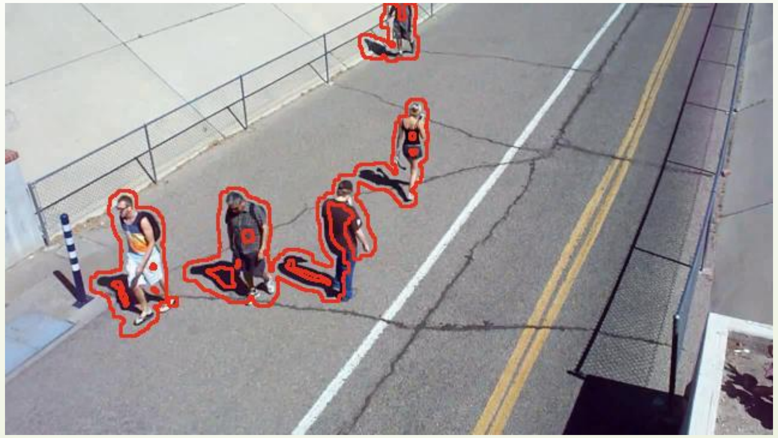
Contour

MEMORIAL UNIVERSITY

- *Problem:* Till now we could effectively find pixels that show intensity change between each two frames and filter some noise. But our goal is finding moving objects not moving pixels.
- *Solution:* For tackling this problem we can use contour techniques. In this technique, we can connect continuous points with the same color or intensity in a boundary with a curve. This approach is used for detecting different objects in an image.
- cv.findContours(): This function takes several arguments. One of these is the mode. In this argument we can choose RETR_LIST, RETR_EXTERNAL, RETR_CCOMP, and RETR_TREE. The first mode is the simplest one, which just creates contour but doesn't represent any information about parent and child relationships between the objects. RETR_EXTERNAL only retrieves parent object and do not consider children. RETR_CCOMP returns all contours which the holes in objects and external contours are labeled in order as a level-2 and level-1 hierarchy. RETR_TREE is the most complete one retrieving contours with a full hierarchy. The last mode is very prevalent, and we use this too. The second argument is method. For this, we can select CHAIN_APPROX_NONE or CHAIN_APPROX_SIMPLE. In the first one, all the points in contours are preserved while in representing some boundaries like a line, two points suffice. So, for saving up memory, we can use the latter one, CHAIN_APPROX_SIMPLE.

A. Step 05
Contour

- After finding contour, for better understanding, we visualize Contoures with the following function.
- cv.drawContours(): The first argument is the source image that we want to draw contour on it. The second is the contours, which should be a Python list. The third argument is an index of contours and used when we want to draw an individual contour, but as we want to draw all contours, we pass -1. The next two arguments are curves color and their thickness, respectively.



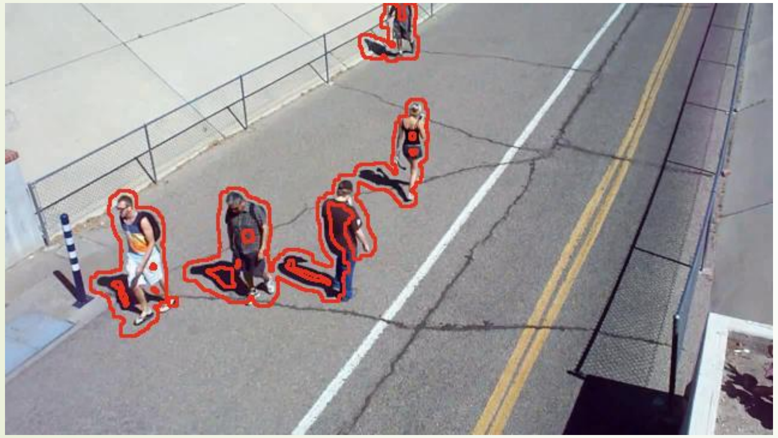
A. Step 06

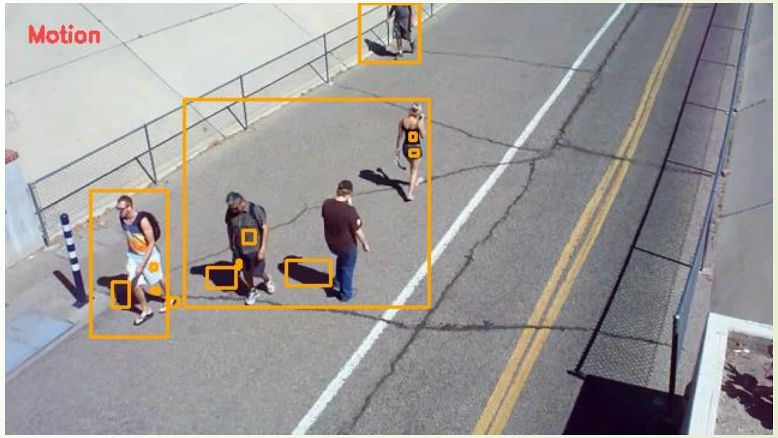
Bounding Rectangle

- *Problem:* Till now, we have successfully detected the moving object and represented them with contours, but the result seems a little messy. Also it could be hard for users to see some details of objects clearly, as contours may cover them.
- *Solution:* To make the output more applicable and user-friendly, we draw simple rectangles around those objects. Therefore, I constrict the detected moving objects with Rectangles. To do this, we have two options "Straight Bounding Rectangle" and "Rotated Rectangle". The latter one, cv2.minAreaRect(), considers the rotation of objects for retrieving a constricting rectangle with minimum area. This function returns the top left corner, width, and height and angle of each rectangle. But in this section, we use the Straight Bounding Rectangle method.
- *cv2.boundingRect():* This function takes a list of contours and returns coordinates of top left corner, width, and height of a rectangle for each contour

A. Step 06
Bounding Rectangle

- In the picture below we can see how cleaner it is when we substitute contours with bounding rectangles.


Contours



Bounding Rectangles

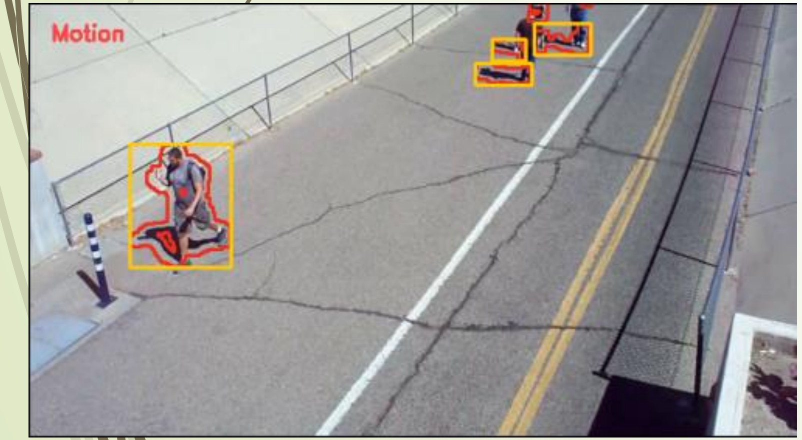
A. Step 07

Threshold

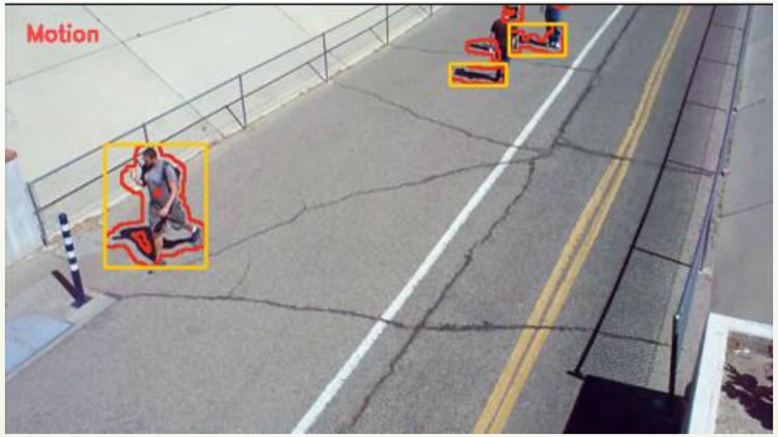
- *Problem:* In the outcome of previous step there are some small rectangels that are part of objects but have been detected as an individual moving object.
- *Solution:* To address this problem I use thresholding technique and filter contours with small areas.
- *cv.contourArea(contour):* This function gets contours list and return the area of each one helping me to filter out very small one that are likely to be part of a bigger object or even could be noise.

A. Step 07
Threshold

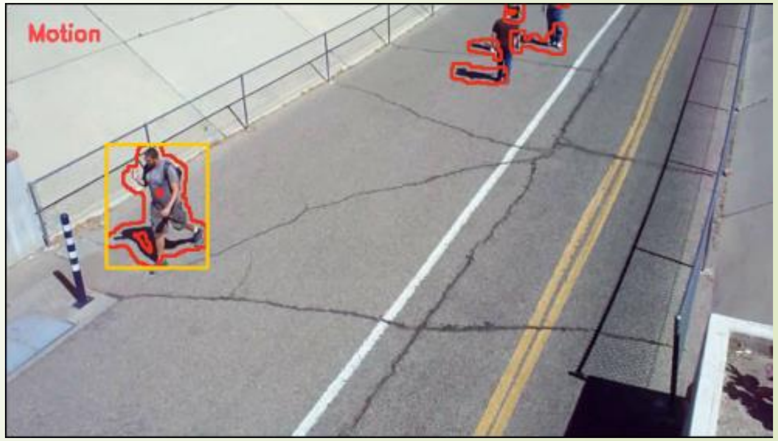
- In this frame, we can represent and compare the results of different thresholds better. In the Threshold=300, we have two moving objects on top of the frame, but it has detected three of them. In threshold=900, it was utterly insensitive to small objects; however, in threshold=600 along with a high sensitivity to small moving objects, it could effectively filter very small contours that belong to main moving objects.



Threshold 300



Threshold 600



Threshold 900

B. MOG2 method

B. Step 01

Foregound Output

- For implementing this method, I used the following function and applied this to each frame, and the output mask is way better than the Frame Differencing method. In the next two slides, I have compared six frames from various positions between “Frame Differencing” and Background Subtraction” methods.

cv.createBackgroundSubtractorMOG2()

Frame Differencing: *Raw mask*

Frames: 200, 300, 400, 500, 600, 700

Frame 200

Frame 300

Frame 400

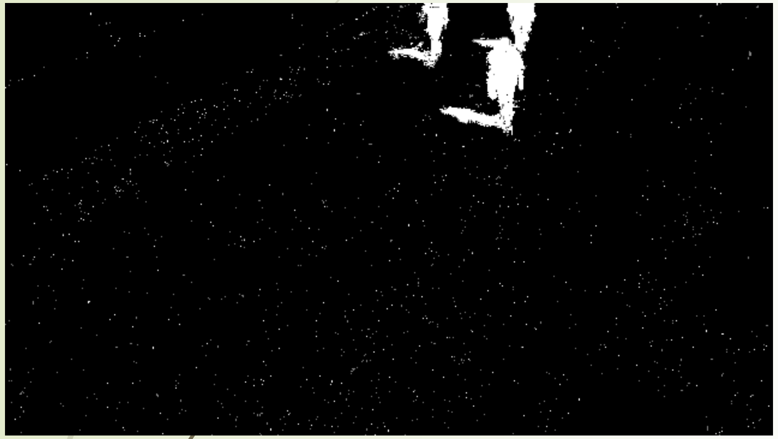
Frame 500

Frame 600

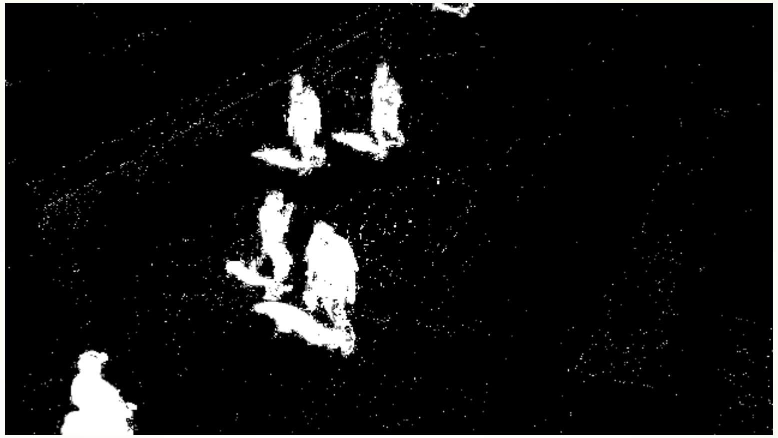
Frame 700

MOG2: *Raw mask*

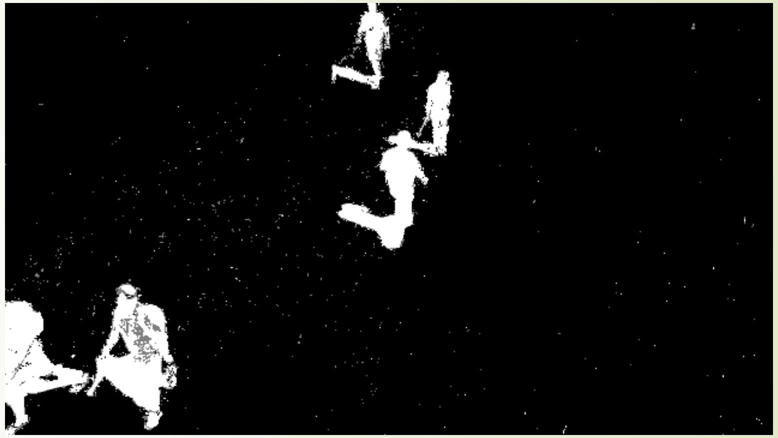
Frames: 200, 300, 400, 500, 600, 700



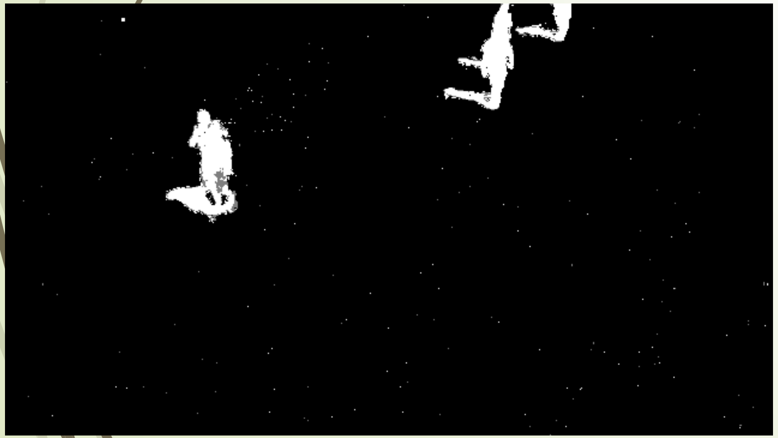
Frame 200



Frame 300



Frame 400



Frame 500

Frame 600

Frame 700

MOG2: *Raw mask*

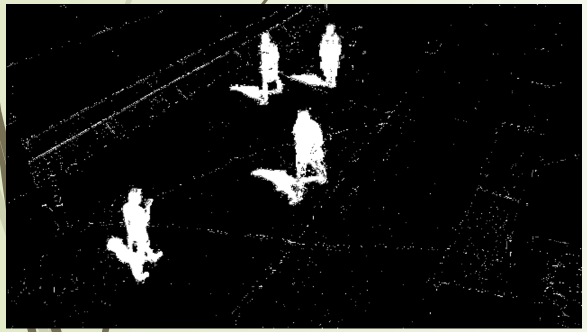
MEMORIAL UNIVERSITY

- *Problem:* Although we can get a great foreground mask out of MOG2 function, still there are some noises, and we can apply some Image processing techniques to improve it.
- *Solution: In the following, we use some Image processing techniques like Blurring, Thresholding, and Dilation/Opening to improve the result. But since these techniques have already been discussed in previous pages, I just use these without repeating those detail.*

B. Step 02

Noise Reduction

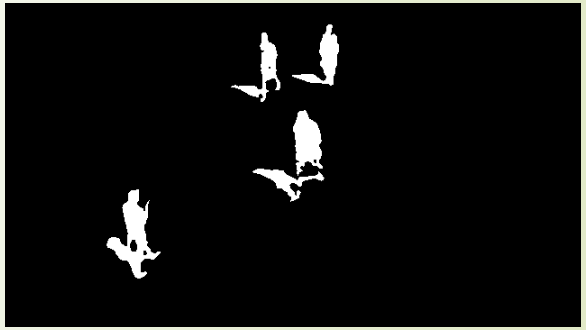
- I have used three kernel sizes in GaussianBlur function, but for better understanding, I compered their threshold=200 output. We can see all blurred images with different kernel sizes bring significant improvement, but I chose the filter size of five as it preserves object details very well.



Original

Gaussian (3,3)

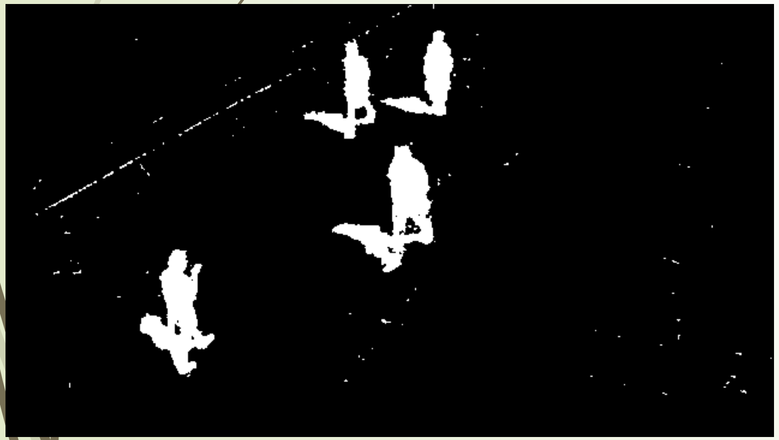
Gaussian (5,5)



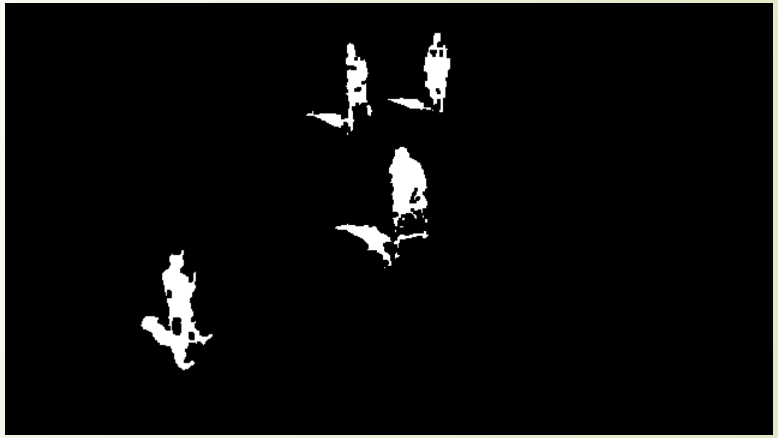
Gaussian (7,7)

B. Step 03
Threshold

- Three different thresholds have been set, 100, 200, 250. I have chosen th=200 as it shows better output over the other two.

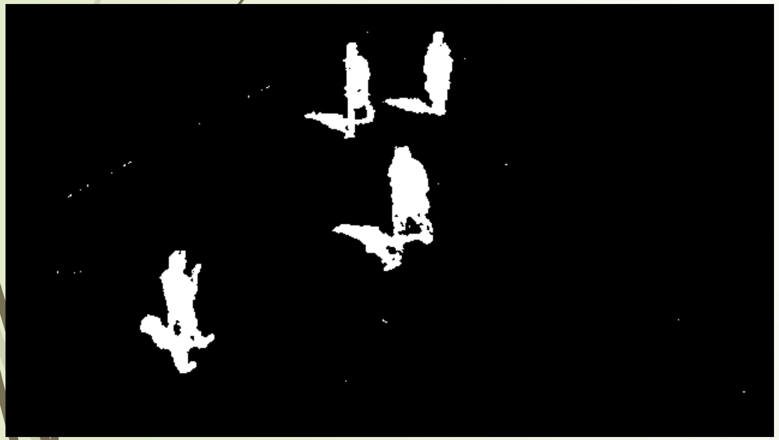

Threshold = 100


Threshold = 200


Threshold = 250

B. Step 04
Opening

- To remove the noise better, I used the opening technique with kernels size of three and five and compared them with the image without opening. And as we can see the kernel three shows a better result.

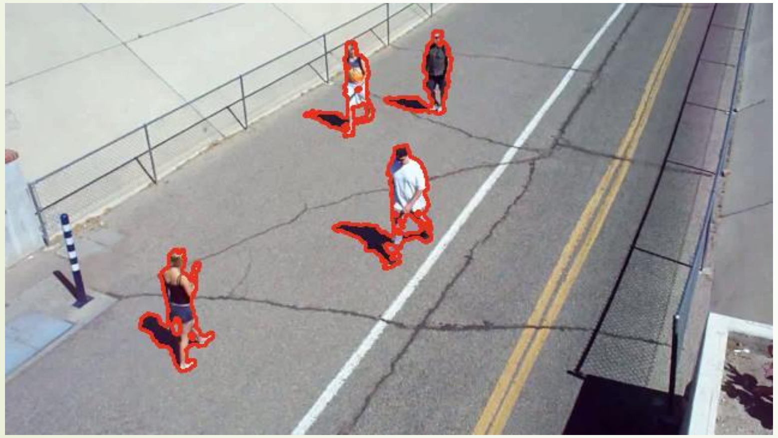

Without Opening


Opening (3,3)

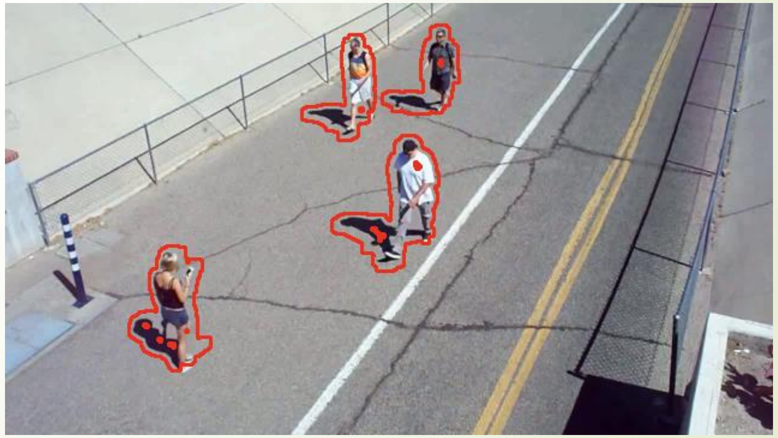

Opening (5,5)

B. Step 05
Contour

- In the following image, we can see the result of contours in two different methods, Frame Differencing and MOG2. As it is clear in these two images, both methods have been able to detect moving objects very efficiently, but edge detection in MOG2 is more precise and has less lost pixels(holes) in the moving objects.



MOG2

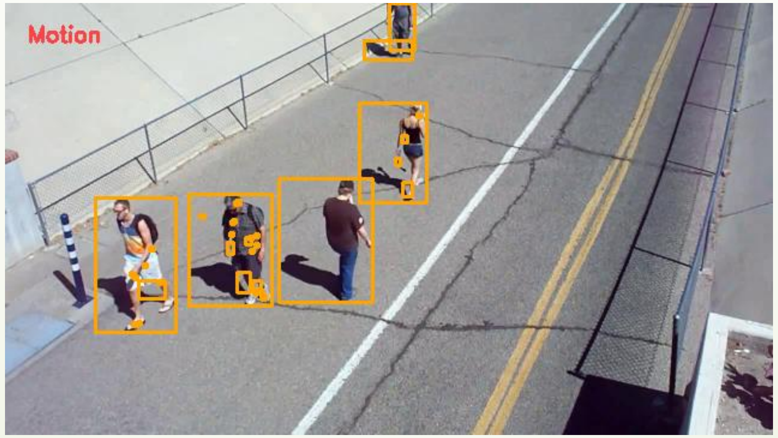


Frame Differencing

B. Step 06

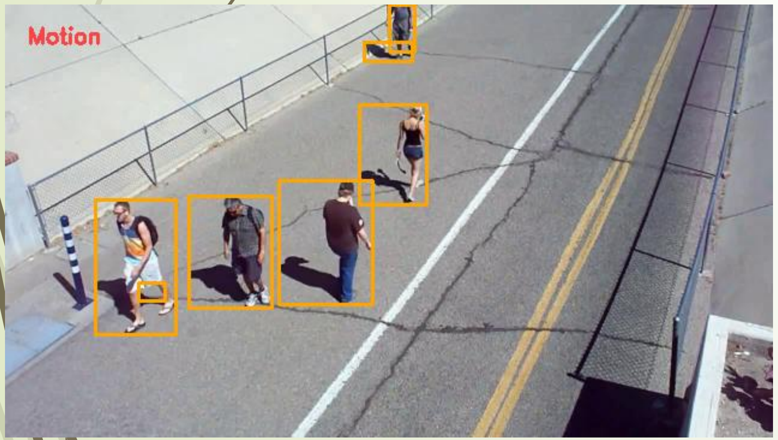
Bounding Rectangle

- In this step we use contour to draw rectangle around our detected objects.

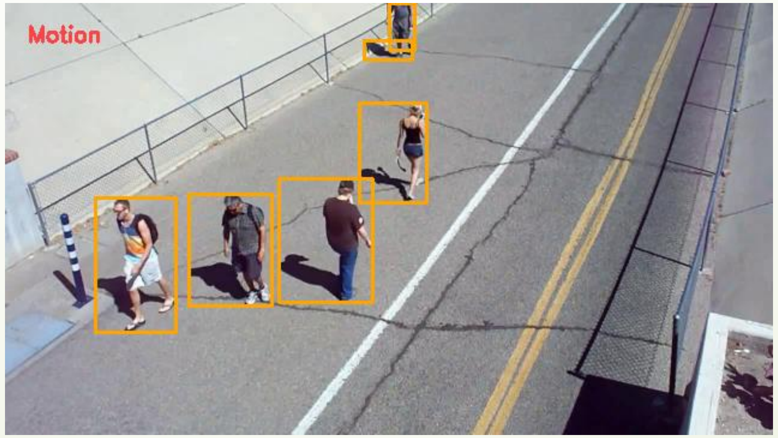


B. Step 07
Threshold

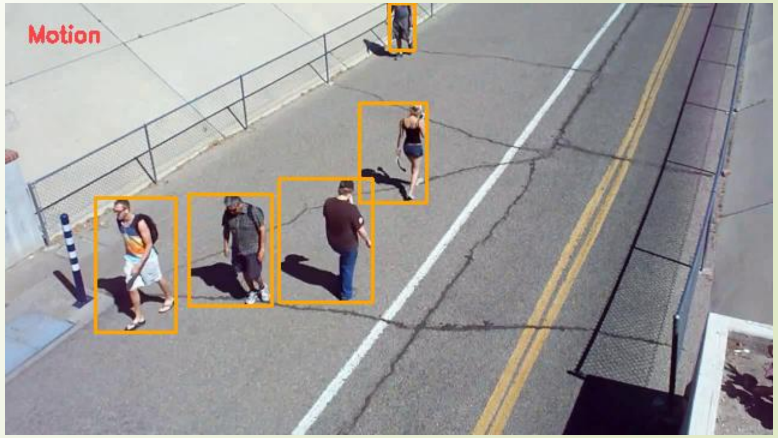
- As it was evident in the previous image, we had a lot of undesirable rectangles. Therefore, in this step, we filter contours with small areas. I have picked 300 for threshold as we can see it has effectively filtered out all small contours that are part of our main objects or are noise.



Threshold = 100



Threshold = 200



Threshold = 300

Comparing result of two methods

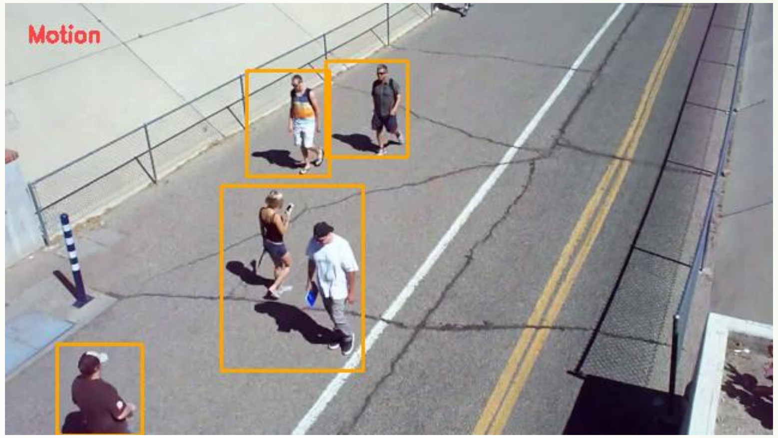
Frame Differencing & MOG2

Output: Frame Differencing

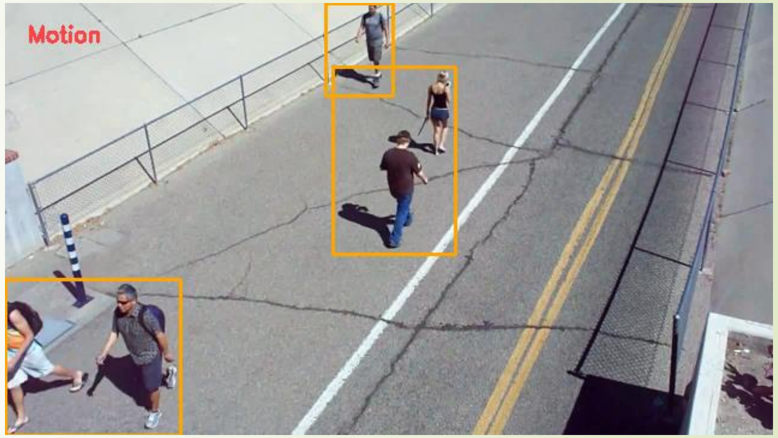
Frames: 200, 300, 400, 500, 600, 700

MEMORIAL UNIVERSITY

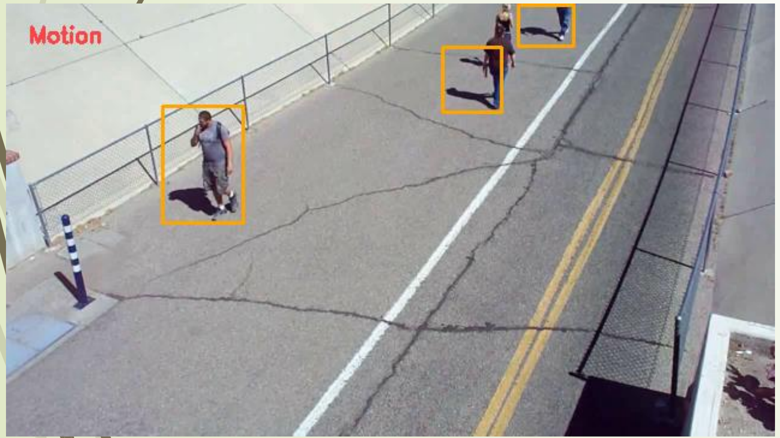
Frame 200



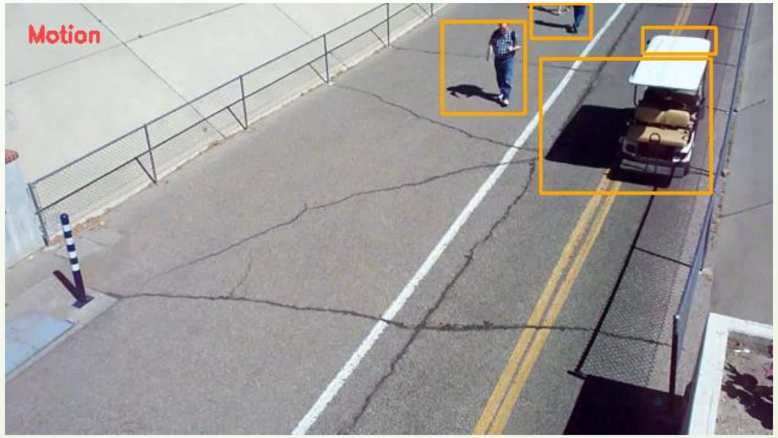
Frame 300



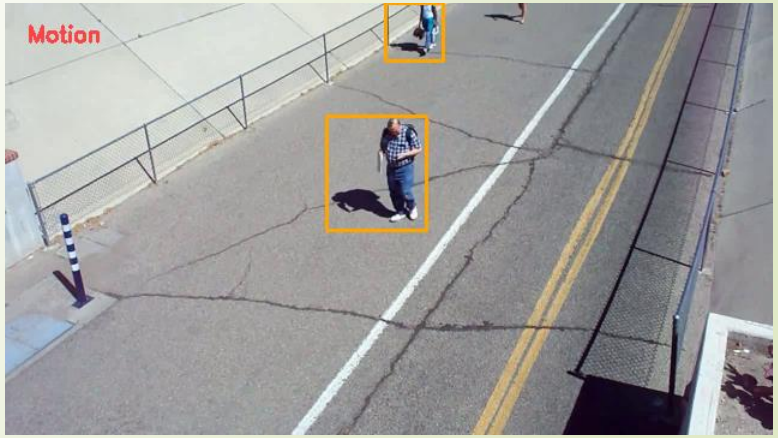
Frame 400



Frame 500



Frame 600

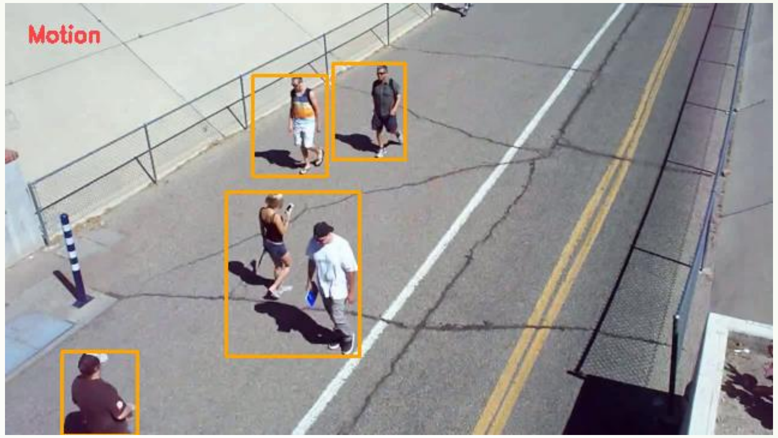


Frame 700

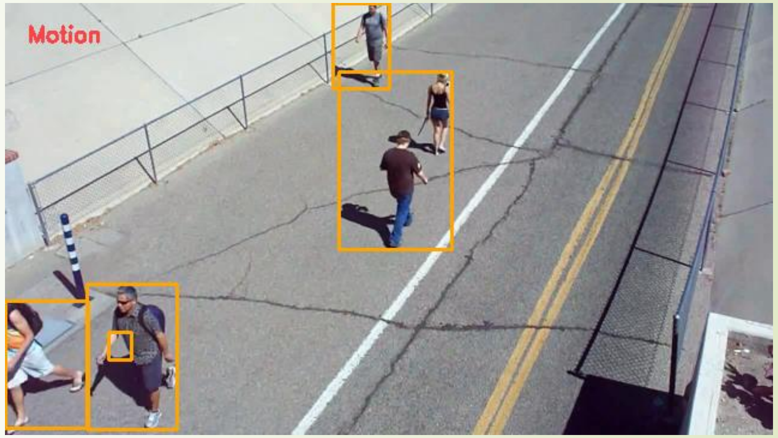
Output: MOG2

Frames: 200, 300, 400, 500, 600, 700

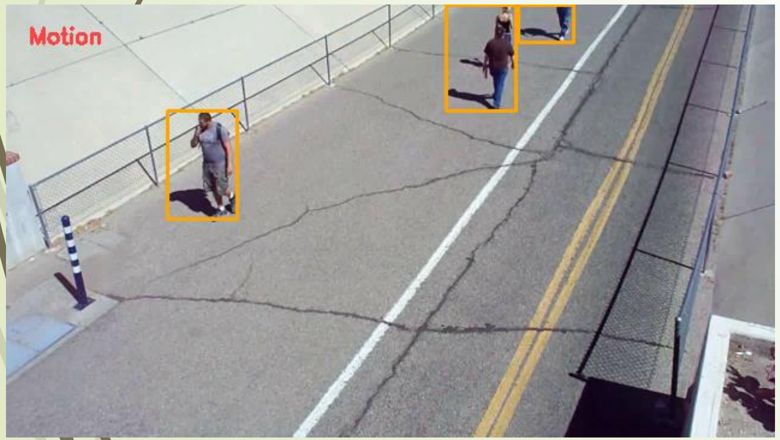
Frame 200



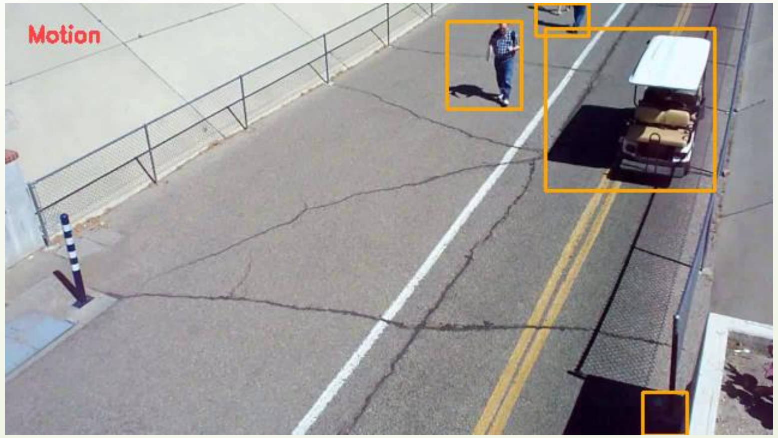
Frame 300



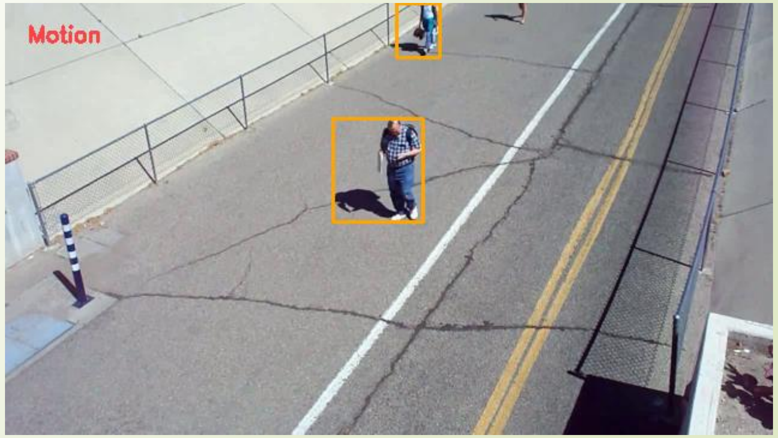
Frame 400



Frame 500



Frame 600



Frame 700

Conclusion:

- Over this PowerPoint, we have seen that implementing Image processing techniques can significantly improve the output images of both motion-detection methods.
- In the last two pages, we can see that MOG2 is a little more capable of precisely detecting moving objects. I expected that MOG2 performs by far better, but thanks to Image Processing techniques, both methods did great in their task in this sample video.