# Motion Detection Using Simple Image Filtering

EECE5639 Computer Vision — Project 1
Fall Semester 2021

Cheung, Spenser & Liming, Tim
*Dept. of Electrical and Computer Engineering*
*Northeastern University*
*Boston, MA, USA*

***Abstract* — In this project we explored simple techniques for motion detection in image sequences captured with a stationary camera, where most of the pixels in the image belong to a stationary background and relatively small moving objects pass in front of the camera.**

## I. Introduction

To detect motion in a sequence of images taken by a stationary camera, we can look to the background of the image to help us. As long as the background of the image remains constant or slowly varying, we can detect motion by looking at the intensity of the background versus the intensity of a moving object in the foreground. The moving object will replace the background intensity with its own intensity and thus we can look at the large gradients in the pixels to detect that motion.

In this project we will create, explore, and compare several different algorithms and filters to find the best performance and values to use for threshold cutoffs.

Code for our project can be found in our Github repository[1], linked in the Appendix.

## II. Description of Algorithms

The first task of the project involved reading in a sequence of image frames and making them grayscale. We made use of the OpenCV python package `cv2` and used its `imread` function to read in an image and then used `cvtColor` to transform it to a grayscale image. Reading all the images in the folder and storing them in an array ahead of time allowed us to speed up the processing part of the program.

Once we had our images in an easy to use format, we used a 1D differential operator to compute a temporal derivative.

After taking the absolute value of the output of our derivative function, we then applied the `threshold` function that mapped the pixels in our images to either 0 or 1 depending on whether they were above a certain threshold value or not.

With our mask created, we then combined our original image with our mask to highlight only the pixels (or objects) that are moving in the image. The `cv2` function `bitwise_and` does this well, although we originally used the `multiply` function which should essentially do the same thing since our mask is a binary one.

## III. Experiments

Using our algorithm, we proceeded to alter our original temporal filter from [-1, 0,1] to $\frac{1}{2}$[-1,0,1]; we also implemented Gaussian filters of varying kernel sizes and sigma values, as well as varying our threshold values. Our primary interest focused on having a threshold that could adapt to changes in different

lighting condition scenarios; in both provided image sets, an event occurs in which the remaining frames become much brighter than the previous frames. To tackle this, we choose a reference pixel from the first frame and then observe that pixel's value as the frames go on. When the difference between our reference frame's pixel value and the current frame's pixel value exceeds a certain value, we then update our image thresholding value.

## IV. Observations

One of the first observations we noticed was that the convolution of our 1D operator and the original image produced a lot of noise (see Fig 1).

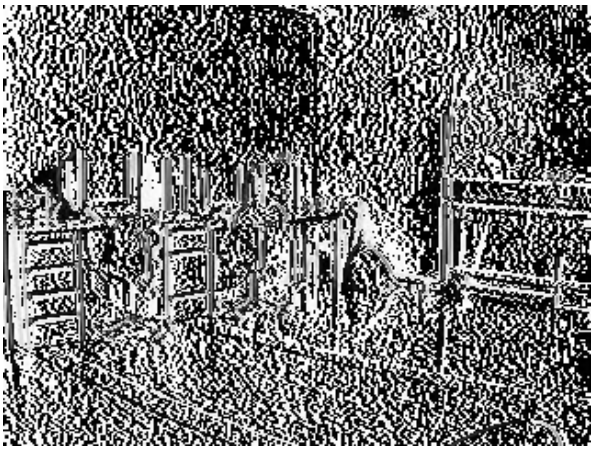

Figure 1: `convolve` Result

We used the `convolve` function that is a part of the `scipy.ndimage` package to produce this result. Taking the absolute value of this image produced no altered image.

We found a function from the OpenCV package that seemed to produce a more desirable result with less background noise called `absdiff` that takes the absolute value of the difference of two images. By feeding it two consecutive images, we can calculate the difference between the two images and then take the absolute value of that result (see Fig 2).

Figure 2: `absdiff` Result

The difference between Figure 1 and Figure 2 is startling and made us question the results. According to the Object Recognition slides [2], the temporal derivative is $It = \frac{\partial I}{\partial t}$ and can be found using frame differencing (ie. (frame t) - (frame t+1) = temporal derivative). We ultimately decided to move forward with the `absdiff` function after running the other experiments.

We added a 0.5 multiplier to our original 1D operator and convolved that with our original image to compare it to our first result in figure 1. The resulting image (see Fig 3) ended up being slightly less noisy, which made the edges seem much more apparent.

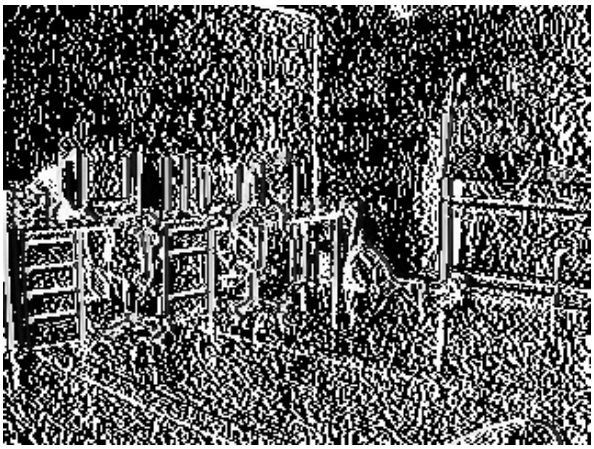

Figure 3: Simple 0.5 * [-1, 0, 1] Filter Result

This result shows that as the multiplier decreases, so does the background noise and the vertical edges become more apparent.

Convolving our original image with a 1D Gaussian operator introduces some slight blurring (see Fig 4).

Figure 4: 1D Gaussian Filter Result

This result cleans up some of the background noise further along in the image processing pipeline, but the introduction of 2D spatial smoothing is what really starts to get better results.

When running our algorithm with the updating threshold value, we observe a lot of frames that detect motion as we expected. However, there are some frames where our detection does not turn out as planned.

In these reference frames and motion detection frames, we observe the Office image set. We have chosen to show frames in which the chair is just being moved, and where the chair is currently in motion (see Figures 5 and 6).

Figure 5: Office - Man grabs chair

Figure 6: Office - Chair in motion

The black and white frame in Figure 5 displays our motion detectors output. This frame displays how since the chair is just being moved, the stationary part at the bottom does not get detected; yet the top of the chair where the man's hand touches, does move and is detected. This result carries over to Figure 6, where the chair is being moved by the man. The chair and man are both outlined in white, verifying that they are both in motion.

In the same image set, the man comes into frame and turns on a lightswitch. This action causes the following frames to have an increase in brightness, which will cause all the pixel values to go up. Using the updating threshold value, the detection algorithm displays false positives caused by the sudden

pixel value increase, as well as the thresholding adjustment afterwards (see Figures 7 and 8).

Figure 7: Office - Man flips lightswitch

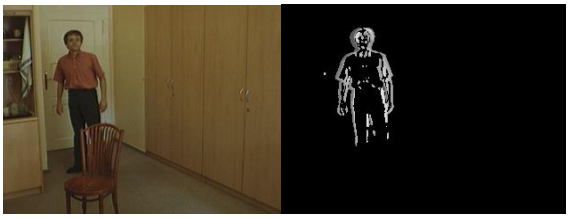

Figure 8: Office - Man walks away from lightswitch

In Figure 7, the motion detection result shows a cluster of false positive detections at the top of the frame. Then a little later in the sequence, we see the motion detection result in Figure 8, showing that the false positive detections go away.

Not all of our motion detection results are good though. During our experiments, we noticed that any horizontal motion caused issues with results (see Figure 9).

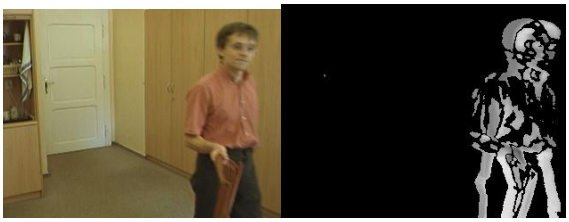

Figure 9: Office - Poor detection

This result shows a bad detection case where the mask does not overlay properly on the moving object.

The following figures (Fig 10 and Fig 11) show that our detection algorithm can successfully adjust to changes in the ambient light. Notice how in Figure 10 the scene is stationary (see left image) yet there is motion detected (see right image). Then in Figure 11, as the ambient light appears brighter (see left image), we no longer have any motion detected (see right image).

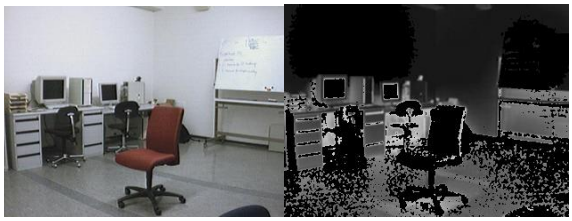

Figure 10: Red Chair - Illumination change detected

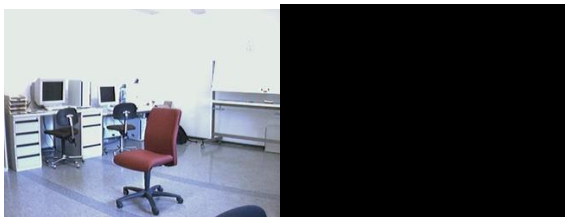

Figure 11: Red Chair - No false detection from brighter image

The following figures (Fig 12 and Fig 13) show that our detection algorithm can successfully detect movement even when that object was previously stationary and considered to be a part of the background. Notice how in Figure 12 the man has moved the chair in the mask (right image) and then in Figure 13, after the man has walked off, the chair is no longer moving and thus returns to the background and is no longer detected.

Figure 12: Red Chair - Chair being moved

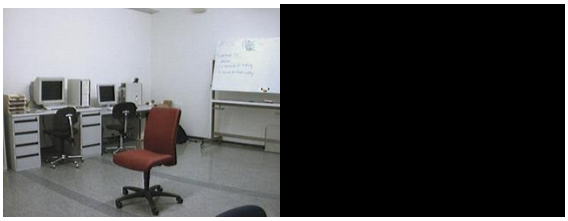

Figure 13: Red Chair - Chair stationary

In the following figure (Fig 14), the result of the temporal filtering can be seen in the final mask. Because we are finding the difference between two frames, if the frame rate is low and the speed of an object across the consecutive frames is high, we get the result that looks like heavy artifacting, or the appearance of two men. This result might not be as apparent if the frame rate of the supplied images was higher, or if the man wasn't walking as fast through the office.

Figure 14: Red Chair - Two men detected

In the following figure (Fig 15), we can see motion detection displayed along the desk in the right image. Because the ambient lighting has changed, it now appears that the man is casting a shadow on the desk as he walks along. Our motion detector is picking this up because the pixels are changing due to his shadow moving even though it is not considered to be a moving object. We could filter this undesirable effect by using sharper edge detection or more likely, by increasing our threshold. Additionally, if the threshold was able to be dynamically altered for each image, we could see that this shadow could be filtered out as simple gaussian noise.

Figure 15: Red Chair - Shadow of man causing false detections

## V. Conclusions

One of the most apparent limitations of the motion detector that we have developed in this project lies in the temporal derivative section. By looking at the difference of two consecutive frames we lose some data in the process and the final result when the mask is recombined with the original image we can see the artifacts of two superimposed images.

A higher frame rate of the given images might have better results, as some of the motion is very quick and the difference between frames can be striking. Further filtering or preprocessing of the images might also have a better effect on bounding the motion detecting mask as well.

The chosen threshold was able to handle the changes in lighting decently well, but because the threshold is left as a fixed value and not something that is dynamically calculated, we may be unintentionally hindering the performance of our detector in settings with lower lighting.

## VI. Appendix

[1] Project 1 Repository
https://github.com/scheung97/EECE5639_ComputerVision/tree/main/Project1

[2] Object Recognition, Pg. 43 - 44
http://www.cs.cmu.edu/~16385/s19/lectures/lecture21.pdf