Motion Detection Using Simple Image Filtering

EECE 5639: Computer Vision

Michael Ambrozie

Miquel Alberti Binimelis

Introduction

In this project we will explore a simple technique for motion detection in image sequences captured with a stationary camera. In the two datasets of images provided, the intensity values observed at each pixel over time is a constant or slowly varying signal, except when a moving object begins to pass through that pixel, in which case the intensity of the background is replaced by the intensity of the foreground object. Thus, we can detect a moving object by looking at large gradients in the temporal evolution of the pixel values.

General approach

The structure of the code is the following:

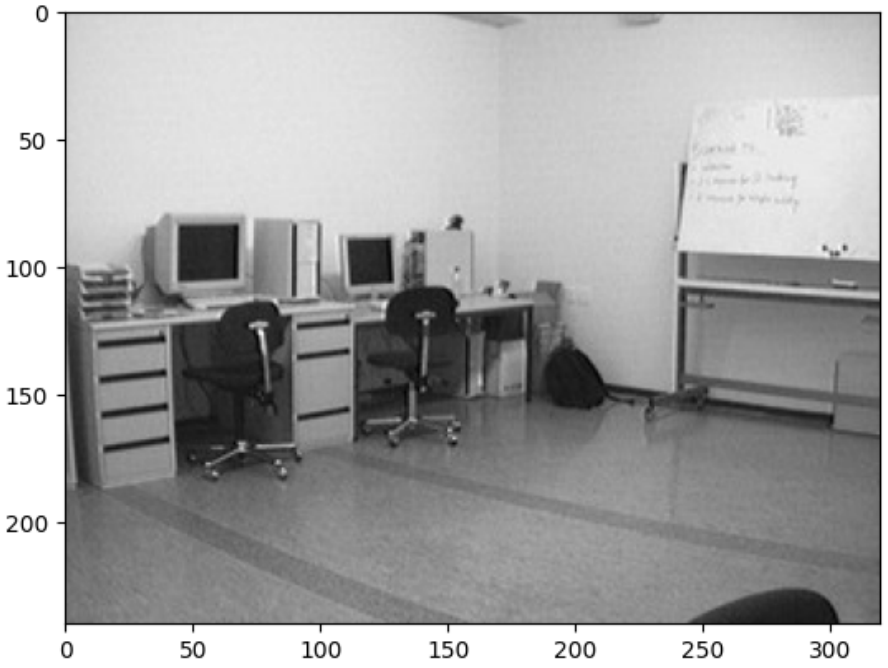
1. Read the sequence of images provided, and convert them to grayscale to detect the variation in a single color channel.
2. The derivative is very sensitive to noise. That's why we have implemented the following smoothing filters to clean the image before the motion detection: 3x3 BOX filter, 5x5 BOX filter, Gaussian filter of the given standard deviation.
3. Now we can compute the temporal evolution of the smoothed images. There are several ways to detect this variation, we have implemented a simple $0.5[-1, 0, 1]$ filter, and a 1D derivative of a Gaussian with a user defined standard deviation.
4. We save the temporal derivative information as a mask with a threshold value: we set to 0 all the values below the threshold, and 1 to the values that remain above.
5. Finally, we apply each mask to its corresponding original frame to appreciate in the original image which objects are moving in the scene.

Smoothing filters

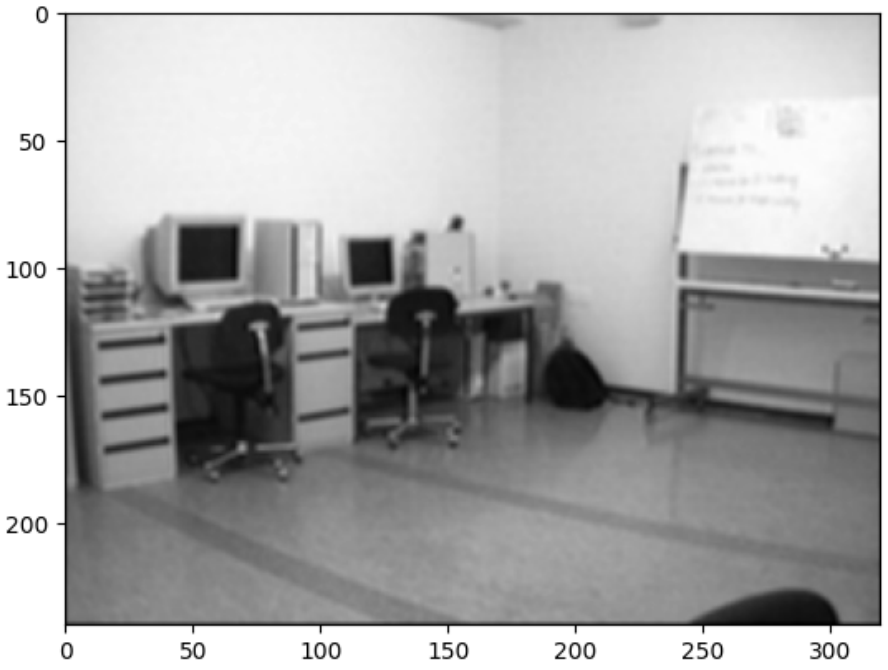
A smoothing filter is just a squared array of weights that we apply to a given image by doing a correlation. In this project we use two types of smoothing filters: BOX and Gaussian filters.

The BOX filter is the simplest one, because all weights have the same value. On the other hand, in a Gaussian filter the weights correspond to a discretized version of a 2D Gaussian.

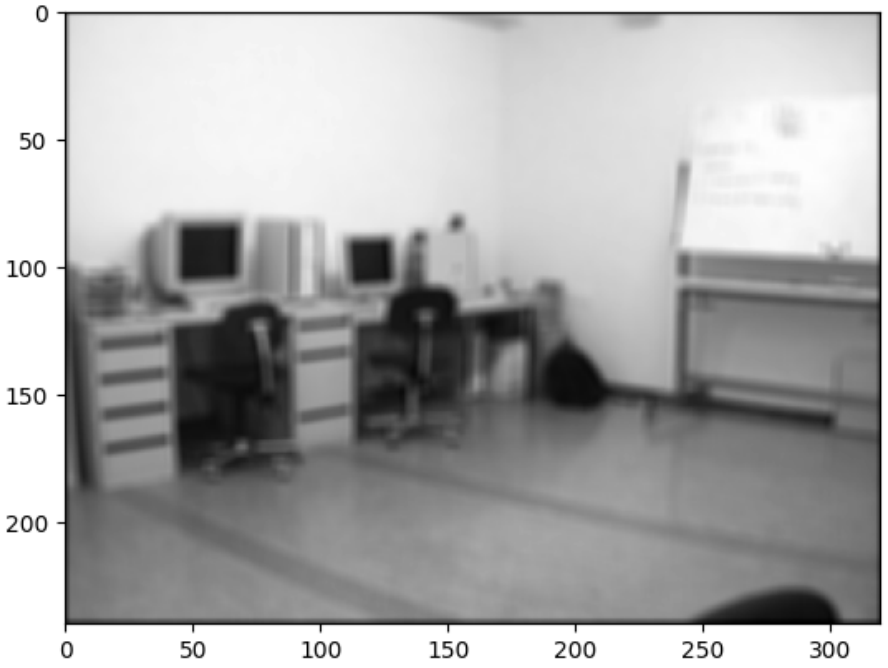
Now we will show examples of how each blurry modifies a particular frame.



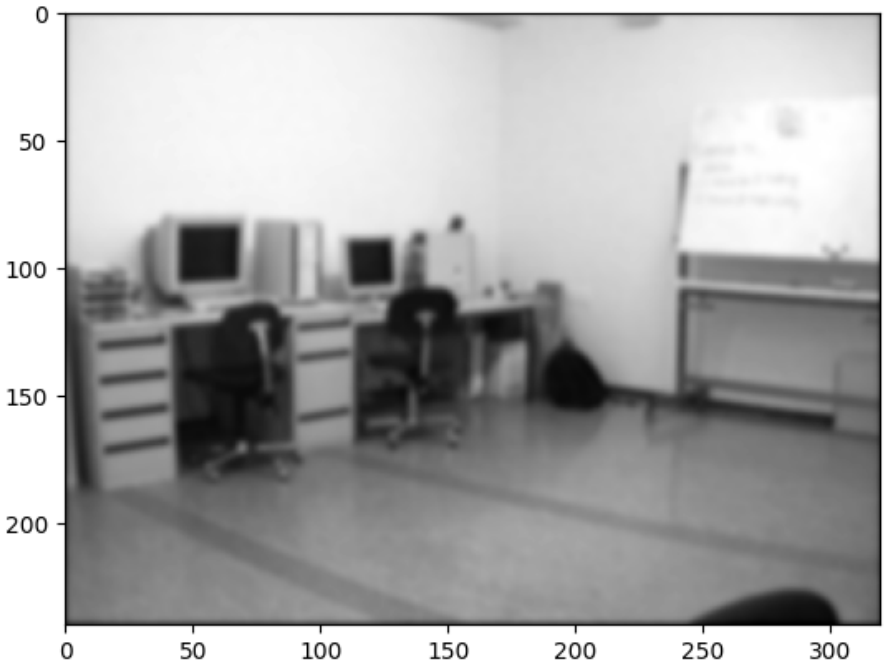
No Smoothing



3x3 Box Filter



5x5 Box Filter



Gaussian Smoothing Filter with standard deviation of 1.4

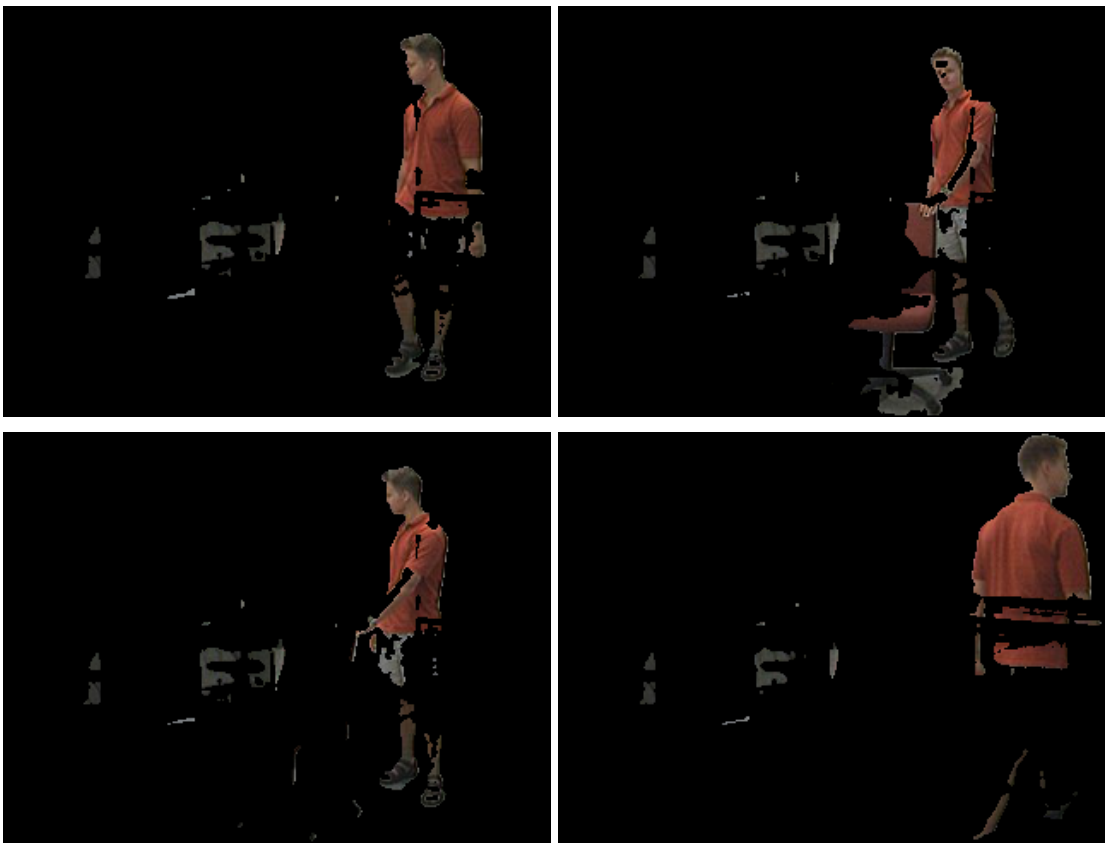
Results with no smoothing filter

The derivative is highly sensitive to random noise. We will explain this afterwards, but for now, let's just observe how noisy the result can be in this case:

As you can see, it is highly corrupted.

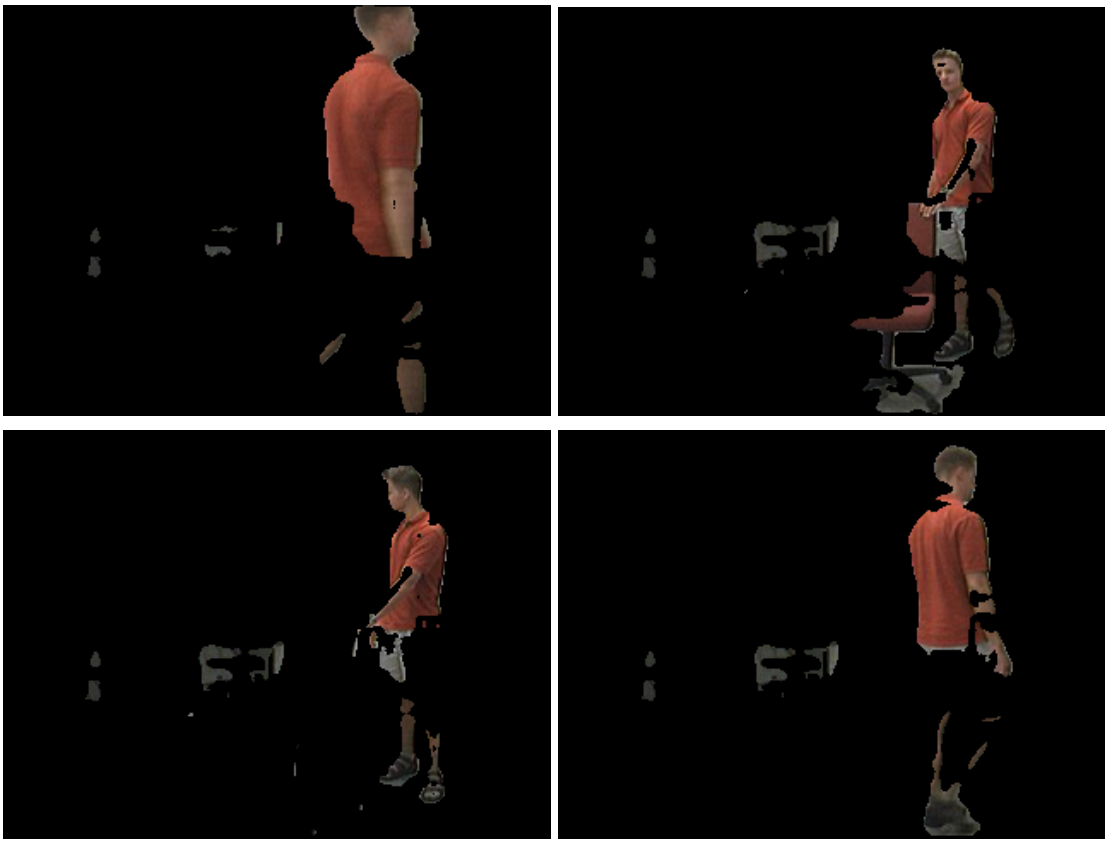
Results with BOX filters

With the BOX smoothing the output using the simple derivative mask improves considerably:



3x3 Box Filter, Simple Time Derivative

We can start to see actual results in the algorithm's outputs. For some reason, the area in the center background is still rising above the threshold (maybe because of a shadow), but it is able to mostly accurately detect the person moving in each image, although noise is still getting through.



5x5 Box Filter, Simple Time Derivative

Once again, the center background section just under the desk seems to rise above the threshold value still despite it not moving. However, there is a noticeable improvement in the smoothness of the edges of the section masked out

Results with Gaussian Filter

As we have said, in a Gaussian filter the weights correspond to a discretized version of a 2D Gaussian. However, we have to take into account that the value of the standard deviation changes the amplitude of the Gaussian: the bigger the standard deviation, the flatter and wider the shape of the Gaussian is. That's why our code asks the user the desired standard deviation in order to compute the optimal size of the filter.

Moreover, this optimal size is not always small, and that's why we have exploited the property that a 2D Gaussian filter is separable into two 1D filters. Here we can see how we can separate this filter:

$$
\begin{aligned}
w(s,t) &= \frac{1}{K} e^{-\frac{s^2+t^2}{2\sigma^2}} \\
&= \frac{1}{K} e^{-(\frac{s^2}{2\sigma^2}+\frac{t^2}{2\sigma^2})} \\
&= \frac{1}{K} e^{-\frac{s^2}{2\sigma^2}} e^{-\frac{t^2}{2\sigma^2}} \\
&= (\frac{1}{\sqrt{K}} e^{-\frac{s^2}{2\sigma^2}})(\frac{1}{\sqrt{K}} e^{-\frac{t^2}{2\sigma^2}})
\end{aligned}
$$

Let's show some results obtained after applying it.

Gaussian Smoothing

Sigma = .07

Sigma = 1.05

Sigma = 1.4


Sigma = 2


Sigma = 3

Increasing values of sigma for the smoothing does seem to have made an improvement in the final product, removing or shrinking some of the smaller blobby errors in the shape.

It is also important to consider the computational cost of increasing sigma, the wider the shape, and the bigger our filter has to be, which obviously implies more computations. There is a noticeable loss in performance when moving from a small box filter to a larger and more complex Gaussian filter. In some tests the box filter took around 2 minutes to execute in our laptops, while the Gaussian was closer to 3.

Derivative filters

Derivative filters are just regular filters that compute a difference. However, in this case we apply them in the temporal domain, in order to appreciate how every single pixel changes over time.

Results with simple derivative filter

The filter we are using is the simplest approximation of a derivative:

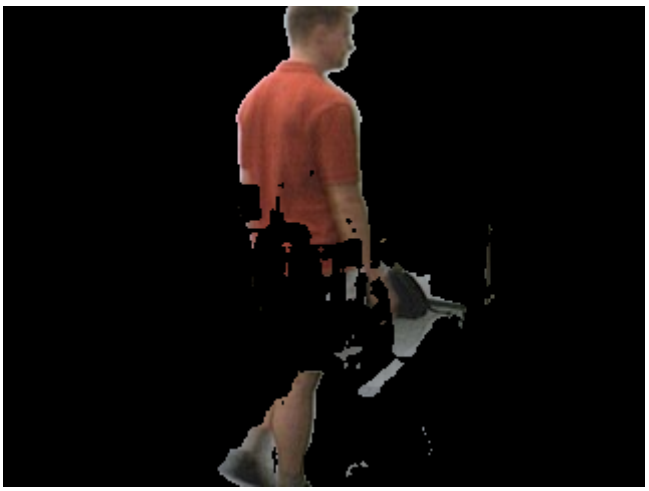
$$0.5[-1 \quad 0 \quad 1]$$

All the images in the previous section were the result after applying this filter, so now we will just focus on a few examples that illustrate the behavior we want to observe. Let's see how it works with no smoothing:

No smoothing, Simple Time Derivative

Same as above with frames with no motion

As you can see, computing the derivative is highly sensitive to noise. That is because we are computing a difference, which implies that, with an additive gaussian noise of zero mean, the errors in the border pixels could end up adding up. The result is much better with a good smoothing.



Gaussian smoothing of σ =0.7, simple Time Derivative

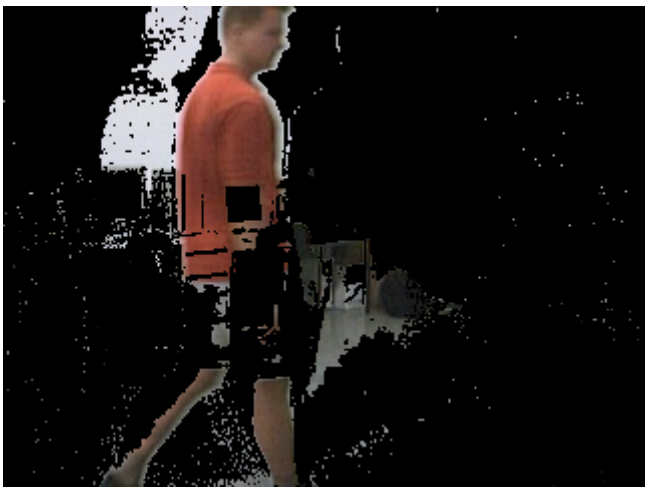
In this last sample we can also observe some of the limitations of our model to detect the trousers in the image. The color of the trousers is almost the same as the color of the background, and that makes our detector fail because the color of the pixel has not changed that much.

Finally, it is also interesting to observe how a change in the illumination of the room can disrupt all the detection. This is an image of the motion detected when the light goes on:

When the light goes on, almost all the pixel values increase considerably because of the brightness of the new light, which results in an increase in the values of the derivatives, and that is wrongly interpreted as movement.

Results using the Derivative of a Gaussian (DoG)

The derivative of Gaussian still computes a difference as the simple filter, but the shape is smoothier. This makes our motion detector more robust to noise.



No smoothing, DoG with $\sigma = 0.7$

BOX 3 smoothing, DoG with $\sigma = 1$

However, as now the mask is considering more frames to compute the derivative (usually 5 or 7, depending on the standard deviation specified) we have to be careful because the mask now reflects movement of longer periods of time. That's why in the result there is the shape of the moving person at 2 different frames. This is not necessarily bad, but we would need a much higher rate of frames per second to see good results.

Thresholding

To illustrate the importance of setting a good threshold, let's see what happens when our threshold is too low:

When computing the derivatives, variations produced because of noise are interpreted as pixels where there is some real movement. These variations are pretty low, but if they are above the threshold they are interpreted as big changes, and that's why the mask is not hiding static pixels in the background.

On the other hand, if the threshold is too high the mask could hide parts of the image where there is real movement but with a low greyscale difference. That is why it is important to figure out a way to compute a good threshold, based on the noise of the image.

We have estimated the noise in the motion masks in order to set the most appropriate threshold for each situation. In order to do so, we apply the EST_NOISE algorithm over the first 18 masks.

Why 18? We noticed that there is no movement in the first 22 frames of both sets of images provided, which implies that every variation on those frames is due to noise. We use 18 instead of 22 because we ignore the firsts frames due to the size of the filter (with the standard deviations that we are testing, the size of a derivative of the gaussian filter is not going to be bigger than 9, which would suppose ignoring the first 4 frames).

The final threshold we use is:

$$threshold = \mu + 3\sigma$$

Where μ and σ are obtained by applying the EST_NOISE algorithm to the first 18 masks. The reason why we multiply the standard deviation by 3 is to have more margin when dealing with outliers. We have chosen 3 and not another scalar like 2 because of the evaluation we have made of the results when testing both values.

Note: μ should be close to 0 as the derivatives should be 0, and the model of the noise we assume has 0 mean.

Conclusion

In this project we have explored the effect of different filters when trying to detect motion in a sequence of images. It has been very satisfying to observe in practice the effect of some of the filters we have studied at a theoretical level. Furthermore, we have appreciated the importance of cleaning an image before processing it, the difficulty of choosing the right threshold, and the computational effort that manipulating images require.

We also want to comment that our code executes the whole algorithm on every run, which includes the smoothing and the application of the temporal filter. However, we could have separated these two tasks into different programs, by saving the intermediate output (the smoothed images) into disk. This other approach is more convenient if we would have to perform more testing using more filters, it would have saved us time.