NORTHEASTERN UNIVERSITY

Computer Vision Project #1 Motion Detection via Spatial and Temporal Filtering

Jordan Helderman
Justin Hess

10 February 2017

In order to track motion, we have devised several algorithms involving the use of temporal derivatives to detect moving people using a video recorded on a stationary camera. Using a one dimensional, temporal derivative filtering mask, we detect drastic shifts in the grayscale intensity of pixels on a stationary background when a moving object passes them along the time spectrum. By looking at the large temporal gradients in the pixels where a moving object passed, we can detect and track the motion of the moving objects. To accomplish this, we created a binary mask by thresholding the absolute value of the temporal derivatives and combined the result with the mask to highlight the pixels around the moving objects. Several 2D spatial smoothing algorithms were also used including two box filters and a gaussian filter with varying the sigma value.

1. INTRODUCTION

The problem of motion detection is one of finding regions in a given video where there are moving objects. This problem is of practical importance in surveillance applications, among others. In this report, we will discuss variations of a common algorithm for motion detection.

2. Description of Algorithms

A simple framework for performing motion detection involves taking the derivative of each pixel in a frame with respect to time. In a stationary scene with no sources of noise, this should yield the zero function. If there is a moving object in the video, there should be pixels in the viscinity of the object with non-zero derivatives. Therefore, for such an image, we could simply check for non-zero temporal derivatives to detect the pixels which correspond to moving objects. Since videos contain stochastic noise in virtually any practical application, we set a threshold which the magnitude of the temporal derivative must exceed before we declare a detection. Thermal noise in a sensor can usually be modeled as a spectrally white, Gaussian process. Since the frequency response of an ideal derivative filter amplifies high frequency components, the application of such filters amplifies the high frequency components of white, Gaussian noise, which can result in erroneous detections. For this reason, it is desirable to perform temporal, lowpass filtering to reduce the amplification of the noise process. For similar reasons, it is desirable to perform lowpass filtering in the spatial dimensions. Using these concepts, we can then write a general framework for motion detection with the following pseudocode.

Algorithm 1: Motion Detection Algorithm

input : A video V
output : A binary mask highlighting pixels around the moving objects

```
// initialize filters and thresholds
```
1 $h_t(t) \leftarrow$ temporal derivative filter impulse response;
2 $h_s(x,y) \leftarrow$ spatial filter impulse response;
3 $\tau \leftarrow$ the threshold for the detection function;
```
// apply filters
```
4 $V_s(x,y,t) \leftarrow V(x,y,t) * h_s(x,y)$;
5 $V_t(x,y,t) \leftarrow V_s(x,y,t) * h_t(t)$;
```
// apply threshold
```
6 **return** $V_t(x,y,t) > \tau$;

A more detailed discussion of how we determine the filter impulse responses and the threshold can be found in the following sections.

2.1. Temporal Filtering

For our derivative filter, we will only be considering the central difference derivative, which consists of an impulse response of the form $[0.5, 0, -0.5]$. Intuitively, this filter computes the average grayscale intensity value change for each pixel in the two frames adjacent to the current frame. This will result in an image of temporal gradients. In some variants of this algorith we will additionally apply a temporal, lowpass filter for the reasons mentioned previously. For this lowpass filter, we will use one dimensional Gaussian filters of varying size and standard deviation. Code for generating the temporal filter and one dimensional Gaussian can be found in the code appendix.

2.2. Spatial Filtering

As mentioned previously, the benefit of spatial, lowpass filtering is a reduction in noise. In the filters we considered, this is accomplished by taking a weighted average over a neighborhood of the image. Similarly, for spatial filtering, we will consider two-dimensional box filters and two-dimensional Gaussian filters. Increasing the kernel size and, in the case of the Gaussian filter, standard deviation will increase the degree to which the image is smoothed, which will decrease the noise power but will also decrease our ability to resolve edges. Matlab to implement these two filters can be found in the appendix.

2.3. Thresholding

To decide on a threshold for our detection algorithm, we will model the noise process as a white, Gaussian stochatsic process with a fixed mean and standard deviation which can vary between videos but is constant within the context of a single video. This model yields an obvious detection scheme which involves setting a fixed threshold at some multiple of the standard deviation of the Gaussian process. Although we will not derive the result here, this scheme also has the desirable property of allowing the characterization of the probability of false detection. In terms of the practical application of this scheme, it is difficult to know whether a given frame does not contain a moving object. To work around this, we will assume that the large majority of the pixels in the image are stationary and estimate the mean and standard deviation by the median and median deviation, respectively, to reduce the effect of outliers on the computation. Making this additional assumption eliminates the need to ensure a frame with no moving objects for the calibration of our detection scheme.

3. Experimental Design

To test these algorithms, we will apply these algorithms to the images and visually evaluate the quality of the results. We will vary the parameters of the Gaussian filter and compare to those obtained by using the various sized box filters.

4. Experimental Results and Discussion

In the following section, we will discuss the results obtained by applying our algorithm to the provided videos. First, we will show some examples of where our algorithm performs as expected. Next, we will discuss the effect of varying the parameters of the filters in our algorithm, and, finally, we will discuss some anamolous behavior that we discovered.

4.1. Example of Desired Effect

The image in figure 4.1 shows the expected behavior of the motion detection algorithm under consideration.The image shows an older couple walking towards the bottom edge of the screen, a man beginning to walk from the bottom right of the image to the left side of the image, and two people standing towards the top center of the image. The motion is highlighted in white.

Figure 4.1: A typical motion detection output

We can see that the algorithm correctly highlights the couple and the man walking, while it does not highlight the two people standing in the top of the image.

4.2. Comparison of Threshold Values

Before we implemented our scheme for automatically selecting the detection threshold, we manually tried a number of thresholds to get a sense for which values would work well. For the threshold for the one dimensional temporal derivative filter and two dimensional box filters, the suitable choice was roughly 18. We came to this value by first seeing that a threshold of less than 10 caused too much noise to be picked up that was not part of any moving object, and we didn't want to pick a threshold much higher than 20 or it would eliminate too many of the pixels that had high enough temporal gradient values to indicate motion was taking place at their position. Through a similar method, we found a similar value for the Gaussian filtered temporal derivative is 5.4. It makes intuitive sense that the threshold would be lower for the Gaussian filtered temporal derivative because the Gaussian smoothing reduces the variance of the noise in the derivative measurement. Some images showing thresholds that are too low, too high, and just right are displayed in figures 4.2-4.

In these figures, the motion is highlighted in white. In the image with the high threshold, we detect the motion, but we also detect some noise associated with random fluctuations in the image. In the image with a high threshold, we do not detect motion associated with random noise, but we also miss some of the objects. In the figure with the good threshold, we see a balance between these two goals.

Figure 4.2: Motion Detection with a low threshold

Figure 4.3: Motion Detection with a high threshold

4.3. Comparison of Temporal Filters

We compared the result of applying the temporal derivative filter by itself and convolved with a Gaussian smoothing filter. We found that the time domain behavior of the output using just the temporal derivative filter was more noisy than with the temporal derivative filter convolved with a Gaussian. Specifically, parts of an object moving at a constant velocity detected in one frame would not necessarily be detected in the next frame even though the object is still moving. In contrast, the motion detected using the Gaussian filtered derivative was more stable.

We also varied the parameterization of the temporal, Gaussian smoothing filter to determine the best values to use. We used a length 11 filter for all our experiments and varied the standard deviation of the Gaussian between 0.1 and 5, varying the parameter on an exponential scale. We found that the behavior with lower standard deviations more closely approximates the behavior of the temporal derivative filter without Gaussian smoothing. This makes sense, because, as the standard deviation approaches zero, the Gaussian impulse response approaches a dirac delta function. When convolved with a dirac delta, the output for any filter will be the filter itself. For this reason, we would expect to get the same filter if we convolve with a Gaussian with low standard deviation. At high standard deviations the motion detections become spread in space. This is due to the Gaussian filter impulse response having a wider variance in the time dimension. An example of this behavior can be seen in figure 4.5. In this figure, we can see that the effect is more pronounced by faster moving objects. Specifically, the

Figure 4.4: Motion Detection with a good threshold

detetions are noticably spread for the man on the right. However, the spread is not quite as noticable for the older couple in the bottom center of the image.

4.4. Comparison of Spatial Filters

We will now discuss the techniques we explored for spatial filtering in our motion detection framework. As mentioned previously, the two techniques for spatial filtering that we will explore are box filtering and Gaussian filtering.

On the box filtering, we observed similar results to those we observed when applying the the temporal filter by itself. One of the main differences however was that the image was smoothed out more and reduced some noise. The moving numbers in the upper left did not have high intensity white values as in the video with the temporal derivative alone. There was still one number that was still picked up by the threshold, and highlighted in red in figure 4.6. The moving numbers from the timestamps on the top left were highlighted using the mask and the threshold of 18. Increasing the threshold more than 18 did not help very much, and only decreased the signal and therefore 18 was still sufficient for the two dimensional spatial filtering.

The application of the convolution of the two dimensional 5 x 5 box filter showed even less disparity. The image was smoothed even further and the noise of the moving numbers in the timestamp in the top left were picked up again, just as with the 3 x 3 box filter. However, this time some of the signal was removed further from some of the moving people. As highlighted

Figure 4.5: Spreading due to high standard deviation on temporal Gaussian filter

Figure 4.6: 3x3 Box Filtering

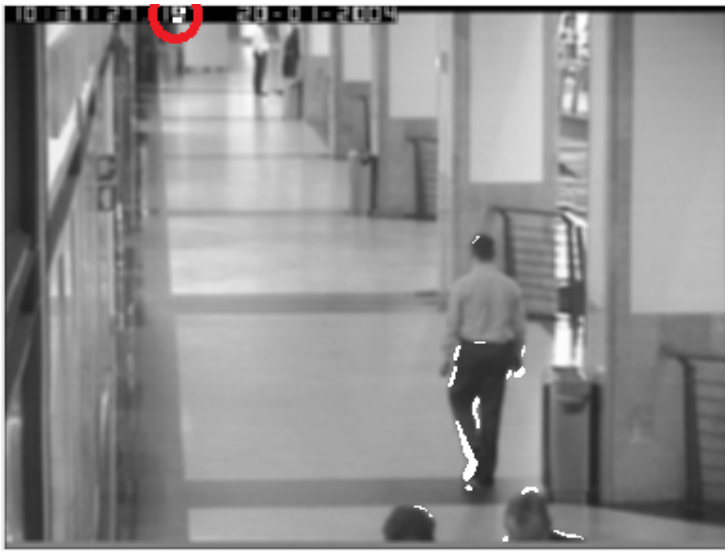


Figure 4.7: 5x5 Box Filtering

in red in figure 4.7, the signal output of the pixels that were originally in the output image applied with just the temporal filter were averaged out further in this video with the 5 x 5 box filter. As a result, some of the signal was lost and these pixels that experienced large shifts in their temporal gradient intensity values were not picked up by our threshold of 18 and the image mask using the threshold. As a result, this proved to be an unnecessary spatial filter for this input video and was 'bad' as it reduced the signal originally picked up by the temporal filter and did not completely remove the unwanted noise in the upper left with the moving numbers that are not classified by us as a moving person of interest.

With regard to the spatial Gaussian smoothing filter, we ultimately found it to be helpful for similar reasons to the temporal Guassian filter. This filter reduced random fluctuations in the detections we obtained. However, we found that the choice of parameters for the filter involved some tradeoffs which we will discuss here. We used a fixed filter size of 50x50, and we varied the standard deviation of the filter between 0.1 and 10 on an exponential scale. Some examples of the behavior of this filter at the extremes and in the 'good' regime can be seen in figures 4.8-10

Similar to the temporal Gaussian smoothing filter, the behavior of the filter approached the unfiltered behavior as the standard deviation of the filter approached zero. This is the case because the impulse response of a two dimensional Gaussian filter approaches a two dimensional dirac delta function as the standard deviation approaches zero. If the standard deviation of the Gaussian is too high, the detections are spread in the image by a proportional amount. This is could be undesirable because it could impede our ability to localize detections in the image. The value for the standard deviation that we eventually chose was 2. We chose this parameter because it provided a balance between the goal of reducing the random variance in the detections and the goal of reducing the spreading of the detections in space and increasing our ability to localize the detections in space.

Figure 4.8: Gaussian Spatial Filter with Low Standard Deviation

Figure 4.9: Gaussian Spatial Filter with High Standard Deviation

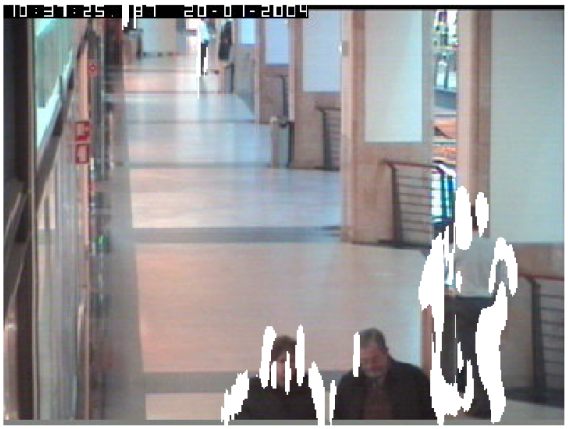
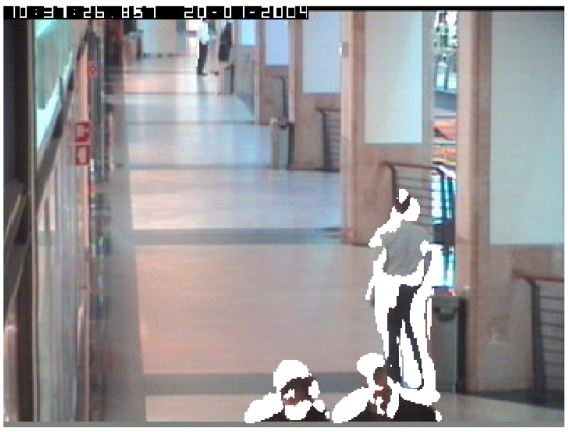


Figure 4.10: Gaussian Spatial Filter with Balanced Standard Deviation



4.5. Anamolous Detections

In testing our algorithms on the provided videos, we also observed a variety of anamolous behavior which could help inform the design of future algorithms and motion detection systems. We document and discuss these behaviors in the following paragraphs

1. Frame Rate

   In videos with lower frame rate, we observed that the region detected around a moving object increases in size. An example of this behavior can be found in figure 4.11. We found that this can be remedied to some degree by lowering the sigma value of the Gaussian smoothing filter on the temporal derivative. This reduces the spreading to some degree, but also increases the noise in the derivative computation. Therefore, we observe that there is a tradeoff between frame rate, the sigma parameter of the Gaussian filter and noise in the derivative computations.

2. Pixels that change regularly that don't correspond to actual motion in the scene

   We also observed detections resulting from movement in an image that was not due to any physical objects moving. Specifically, our algorithms detected the movements of a clock that was in the upper left corner of the video. This is the same behavior that we identified in figure 4.6 when considering the 3x3 box filtering. Although it seems reasonable that the algorithms would detect this movement given their construction, it may not be desirable to report this information given the application. This issue could be remedied by masking the part of the image corresponding to the clock before doing any processing so that no detections are made in that area.

3. Shadows and Reflections

Figure 4.11: Low Frame Rate Spreading

In the office video and the surveillance video, we observed some detections associated with shadows cast by the moving people and reflections of the moving people in the polished tile floor. An example of this can be seen in figure 4.12. It is possible that these phenomena may complicate further processing. The derivatives associated with these shadows and reflections are generally smaller than actual moving objects in the video, so it is possible that this issue could be fixed by increasing the threshold applied to the image. However, increasing the threshold by too much may result in missed detections.

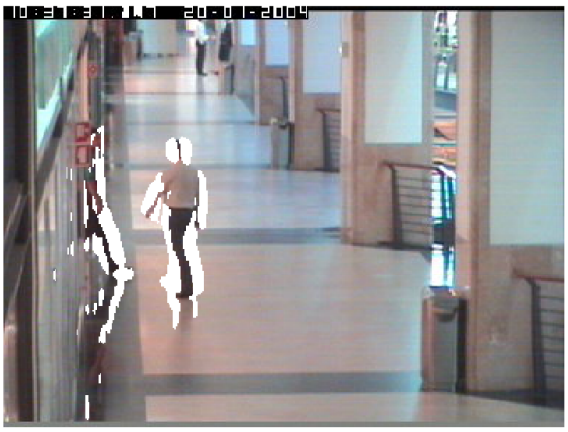
4. Flickering Lighting

   In the red chair video, the lighting was noticably variable in intensity. This results in a greater spread of derivatives corresponding to stationary objects in the image. The effect did not impact our detections in this image to a noticable extent, but this effect could make setting the threshold for detections more difficult in other scenarios. This effect could potentially be mitigated by the filtering on the temporal derivative. Such filtering could potentially reduce the sensitivity of the detector; consequently, we must consider this tradeoff when designing a system using this technique.

5. Conclusion

In this report, we review the design, implementation, and testing of a general algorithm for motion detection. In general, we found that there are benefits to doing spatial and temporal lowpass filtering, and in particular Gaussian spatial and temporal filtering, in terms of reliability, but we also found that these lowpass filtering techniques involve tradeoffs with respect to temporal and spatial resolution. Any system that uses these techniques must consider these tradeoffs in its design. For anyone wanting to experiment with these techniques, the full code we used to test can be found in the appendix.

Figure 4.12: Detections of Reflections and Shadows



APPENDIX A MOTION DETECTION CODE

Temporal Derivative Filter

```
function h = centralDifferenceDerivative()
    filter_matrix = [-1 0 1];
    f_coeff = 0.5;
    h = f_coeff*filter_matrix;
end
```

Temporal Gaussian Smoothing Filter

```
function h = gaussian1d(N, stdev)
    t = (0.5:N-0.5) - N/2;
    h = exp(-0.5 * t.^2 / stdev^2) / sqrt(2 * pi) / stdev;
    h = h / sum(h);
end
```

Spatial Box Filter

```
function h = box2d(M, N)
    h = ones([M, N]);
    h = h / sum(h(:));
end
```

Spatial Gaussian Filter

```
function h = gaussian2d(N, stdev)
    %% a function to generate a 2 dimensional Gaussian filter
    %% get the spatial coordinates
```

```
    x1 = (0.5:M-0.5) - M/2;
    x2 = (0.5:N-0.5) - N/2;
    [X1, X2] = ndgrid(x1, x2);
    X = [reshape(X1, [], 1), reshape(X2, [], 1)];

    %% evaluate the Gaussian
    h = exp(-0.5 * dot(X * cov^-1, X, 2)) / sqrt(det(2 * pi * cov));
    h = reshape(h, M, N);

    %% make sum to one
    h = h / sum(h(:));
end
```

Video Reader

```
classdef CVMovieReader
    %CVMovieReader A class for reading the movies from computer vision
    %    CVMovieReader

    properties
        path_format_spec
    end

    methods
        function obj = CVMovieReader(path_format_spec)
            %CVMovieReader
            %    path_format_spec: a format specification for the movie
            %    files. sprintf(path_format_spec, frame_number) should be a
            %    valid file name for any valid frame.
            obj.path_format_spec = path_format_spec;
        end

        function frames = get_frames(obj,frame_numbers)
            % get_frames a method for reading frames from a movie of this
            % type. Frames are returned as a MxNxK array, where (M,N) are
            % the dimensions of the frames in pixels and K is the number of
            % frames requested
            %
            % frame_numbers: a list of frames to read
            path = sprintf(obj.path_format_spec, frame_numbers(1));
            img = imread(path);
            frames = uint8(zeros([size(img), length(frame_numbers)]));
            for n = 1:length(frame_numbers)
                path = sprintf(obj.path_format_spec, frame_numbers(n));
                frames(:,:,:,n) = imread(path);
```

```
            end
        end
    end
end
```

Main Script

```
clc; clear; close all;
%% parameters
% movie
start_frame = 2;
end_frame = 400;
path_format = ''; % a path format specifier
% temporal filter
temporal_filter_type = 1;
Nt = 11;
% sigma_t = 1.5;
sigma_t = 1;
% spatial filter
apply_spatial_filter = true;
spatial_filter_type = 1;
Ms = 50;
Ns = Ms;
cov = 2^2 * eye(2);
% threshold
% gamma = 25;
gamma = 30;

%% initialize the movie reader
reader = CVMovieReader(path_format);

%% get image size
sz = size(reader.get_frames(start_frame));
sz = sz(1:2);

%% initialize the filters
if temporal_filter_type == 0
    % central difference derivative
    ht = [-0.5, 0, 0.5];
elseif temporal_filter_type == 1
    % central difference derivative convolved with a Gaussian filter
    ht = conv([-0.5, 0, 0.5], gaussian1d(Nt, sigma_t));
else
    error('Invalid Filter Type');
end
```

```
figure;
stem(ht);
% manipulate to make filtering easier
ht = repmat(permute(ht, [1, 3, 2]), sz);
if spatial_filter_type == 0
    % box filter
    hs = ones([Ms, Ns]) / Ms / Ns;
elseif spatial_filter_type == 1
    % gaussian filter
    hs = gaussian2d(Ms, Ns, cov);
else
    error('Invalid Filter Type');
end

%% iterate over the frames
figure;
disp('Press Enter to Advance')
for n = start_frame:end_frame - size(ht, 4)
    %% load the frames
    % load enough to produce a single frame of the temporal derivative
    frames = reader.get_frames(n:n+size(ht,3)-1);
    orig_frame = ifftshift(frames, 4);
    orig_frame = orig_frame(:,:,:,1);
    frames = double(frames);
    % make grayscale
    frames = squeeze(mean(frames, 3));

    %% spatial filtering
    if apply_spatial_filter
        frames = imfilter(frames, hs);
        spatial_filtered_frame = ifftshift(frames, 3);
        spatial_filtered_frame = spatial_filtered_frame(:,:,1);
    end

    %% temporal derivative
    filtered_frame = sum(frames .* ht, 3);

    %% thresholding
    if n == start_frame
        % set the treshold
        mu = median(filtered_frame(:));
        dev = (filtered_frame - mu).^2;
        sigma = sqrt(median(dev(:)));
        threshold = mu + gamma * sigma;
```

```
            disp(threshold);
            disp(mu);
            disp(sigma);
            disp(gamma);
            disp(max(abs(filtered_frame(:))));
            disp(min(abs(filtered_frame(:))));
        end
        threshold_frame = abs(filtered_frame) >= threshold;

        %% mark the original image
        idx = repmat(threshold_frame, [1, 1, 3]);
        orig_frame(idx) = 255;

        %% display the result
        imshow(orig_frame);
        pause(0.05);
    end
```