EECE5639

COMPUTER VISION

PROJECT 1

ROHIT RAO NAINENI

NU ID- 001738178

ABSTRACT:

This project deals with a simple technique for motion detection in image sequences captured with a stationary camera where most of the pixels belong to a stationary background and relatively small moving objects pass in front of the camera. The intensity values observed at a pixel over time is an approximately constant 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.

DESCRIPTION OF ALGORITHMS

The basic concept of motion detection algorithm is based on background change detection. In this project, the intensity values are observed at a pixel over time, the temporal change detection discerns between moving and stationary objects by comparing set of images. The temporal change detection is carried out in three steps.

a. Read in a sequence of image frames and make them grayscale.
b. The difference in frames (derivative) is computed as enough frames are available. Either a two frame difference or a 1-D differential operator is applied at each pixel to compute a temporal derivative.
c. Motion is detected by thresholding regions of large temporal change. The thresholding is to calculate the absolute values of the derivatives to create a 0 and 1 mask of the moving objects.

EXPERIMENTS

1. A 1-D differential operator is applied to compute a temporal derivative. For the temporal derivative filter a simple 0.5[-1, 0, 1] filter and a 1D derivative of a Gaussian with a user defined standard deviation sigma is used. The results are compared by applying Gaussian filters with different values of sigma.
2. Secondly, 2D spatial smoothing filter is applied to the frames before applying the temporal derivative filter. For the spatial smoothing 3x3, 5x5 box filters and 2D Gaussian filters with a user defined standard deviation sigma are used.
3. A strategy to select a good threshold for each image is designed. The threshold is varied to get different masks and then compared.

Sample videos EnterExitCrossingPaths2cor, RedChair and Office are used.

VALUES OF PARAMETERS USED

For the temporal derivative filter a simple 0.5[-1, 0, 1] is used. A 1D derivative of Gaussian Kernel is calculated by first calculating the Gaussian kernel with filter size as ceil(5*(sigma)) and then calculating its gradient. The different values of sigma used are 1, 1.4 and 4.2.

Threshold calculation:

The thresholds for the change detection are proportional to the noise present in the difference images.

Most pixels belonging to the background change little and thus their temporal gradients consist solely of noise. The noise is modelled by a normal distribution N(0, σ^2) with zero mean and described by the standard deviation σ. We assume that the noise is independent for each pixel both spatially and temporally but follows the same distribution. We estimate the variance of the noise for each pixel at (x,y) according to the formula

$$\sigma_{x,y}^2 = \frac{1}{N-1}\sum_{t=1}^{N}\left(I_{x,y}^t - \mu_{x,y}\right)^2$$

where μ is the mean pixel value in the time domain for each pixel. Each estimate of the variance should converge to the true value given our assumptions. However, some values will be confounded by the motion in the image, and will therefore have a higher variance. In order to choose a more robust threshold value, we assume that the motion confounds fewer than half of the pixels, and select the median variance as an estimate of the noise variance for the camera in the temporal domain. We then choose a threshold such that $thresh = 5 * \sqrt{med\left(\sigma_{x,y}^2\right)}$ so that ~0.999999 of the noise values will not exceed the threshold.

In practice, this proved to be not the case for our example, since we saw values that exceeded the noise in areas where the truth was constant e.g. walls or floors. This indicated that our noise model may be invalid due to flickering light or other causes.

OBSERVATIONS AND CONCLUSIONS:

i. Temporal Difference filter = 0.5[-1 0 1] vs 1D Gaussian derivative filter:

Shown below in order of appearance is the original image, the difference filter (0.5[-1 0 1]) image, the thresholded difference filter mask, the Gaussian derivate filter image, and the thresholded Gaussian derivative filter mask. The frame was centered at frame 60. The original value of sigma was 1, so we used 5 Gaussian coefficients. The second value of sigma was 1.4, where we used 7 Gaussian coefficients.

Figure 1: Original Image

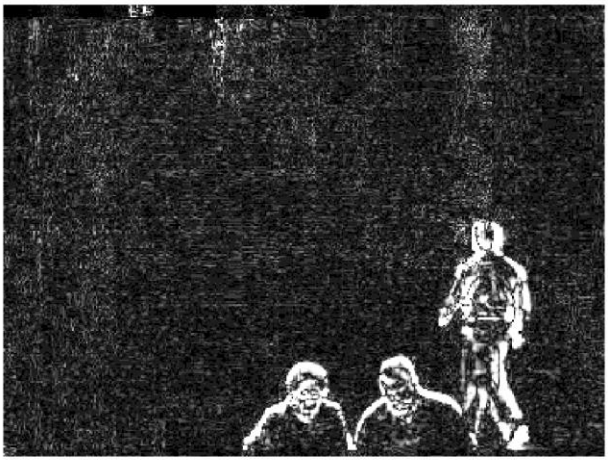

Figure 2: Difference Filtered Image (0.5[-1 0 1])

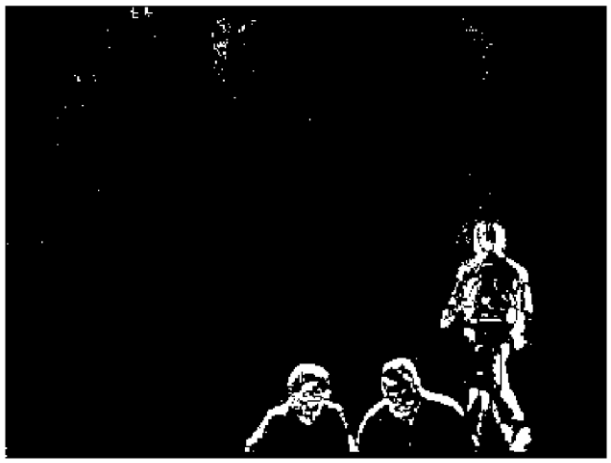

Figure 3: Difference Filtered Image (0.5[-1 0 1])

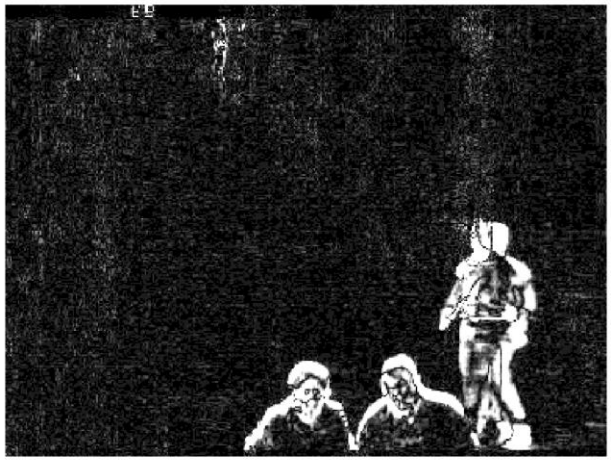

Figure 4: Gaussian Filtered Image Sigma = 1.0

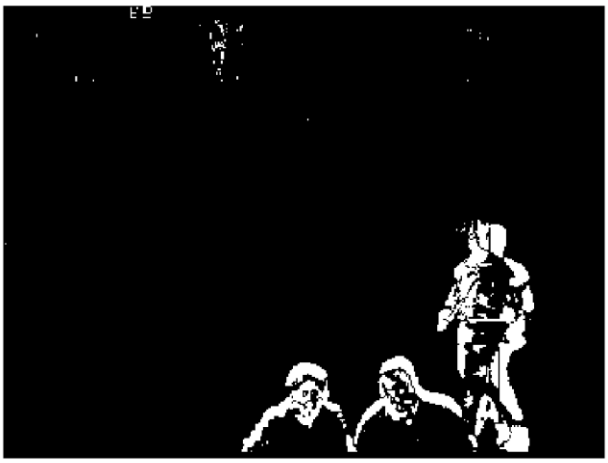

Figure 5: Gaussian Filtered Image Thresholded

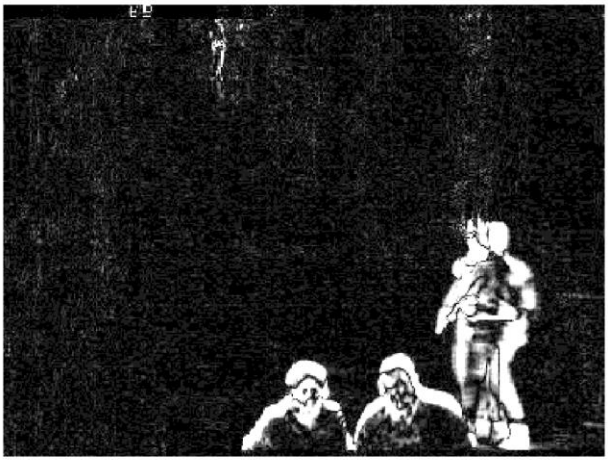

Figure 6: Gaussian Derivative Sigma = 1.4

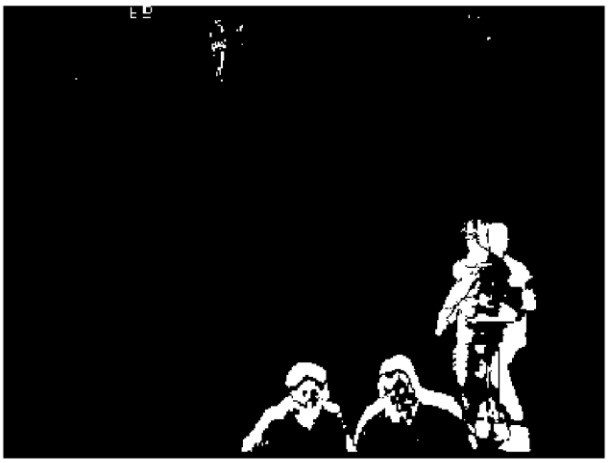

Figure 7: Gaussian Derivative Sigma = 1.4 Thresholded

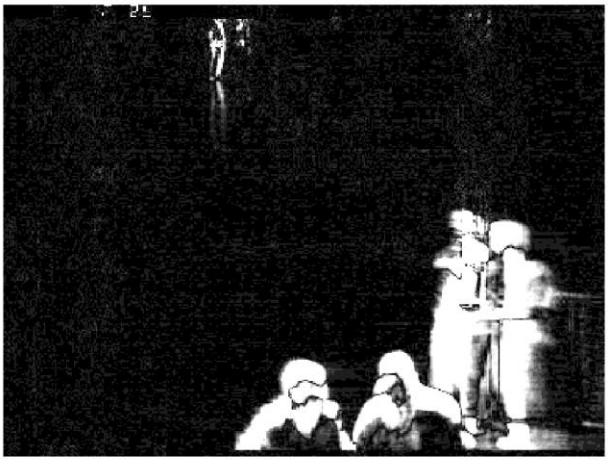

Figure 8: Gaussian Derivative Sigma = 4.2

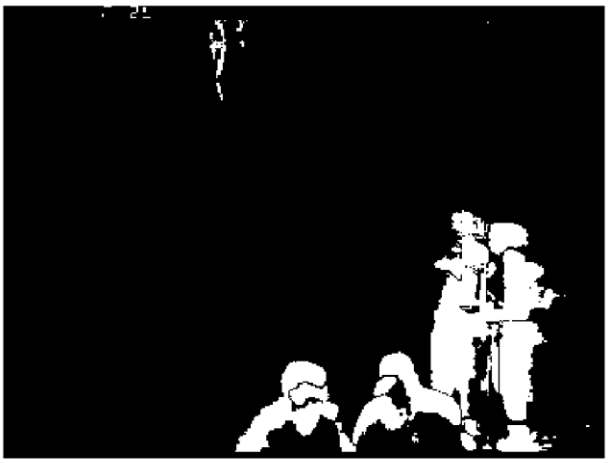

Figure 9: Gaussian Derivative Sigma = 4.2 Thresholded

When comparing the filtered images, the Gaussian filter performs better. The width of the detected motion is larger in the spatial domain, and the noise appears reduced. Both these features make it easier to distinguish the motion from noise. While the thresholded images appear similar, the Gaussian filter produces a more continuous contour. However, when taken to the extreme as in the last case, the spatial blurring makes the image not cohesive.

ii. 2D spatial smoothing

Below are images showing the difference that spatial smoothing has on the images. First box filters are shown, then spatial Gaussian filters of different values of sigma. The length of the filter in both dimensions was equal to $5 \times sigma$.

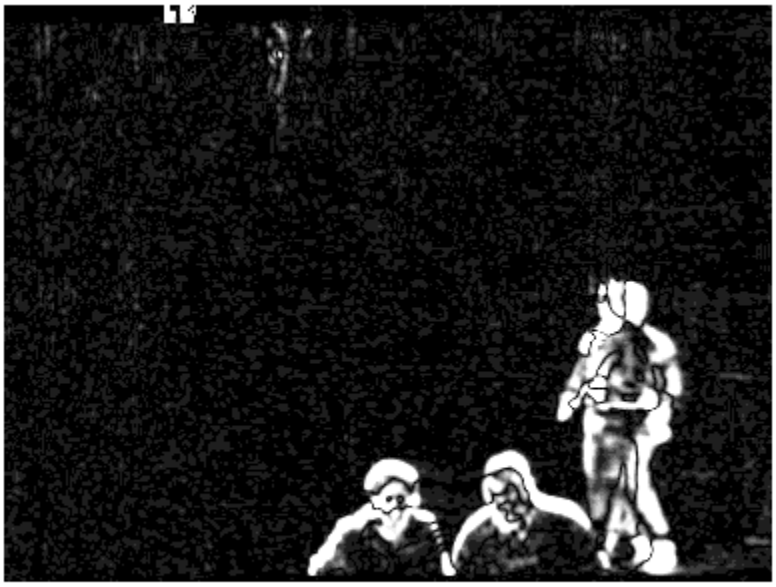

Figure 10: 3x3 Box Filter

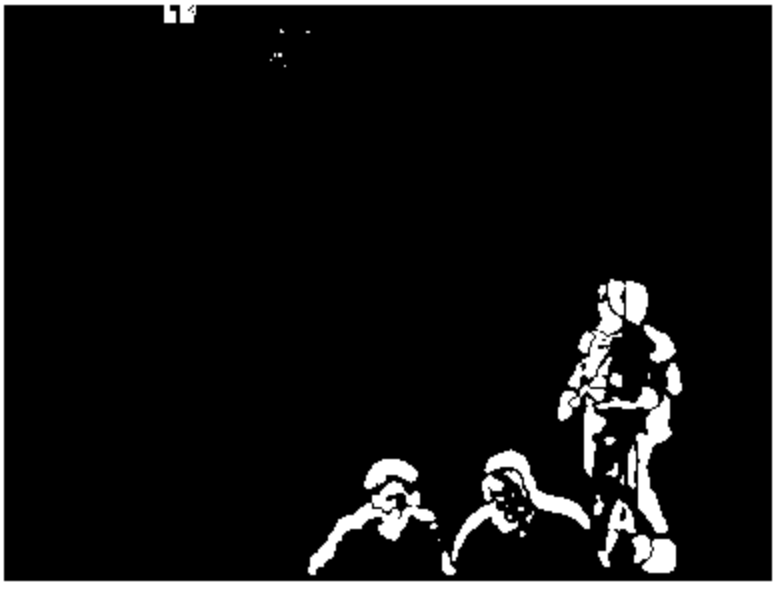

Figure 11: 3x3 Box Filter Thresholded

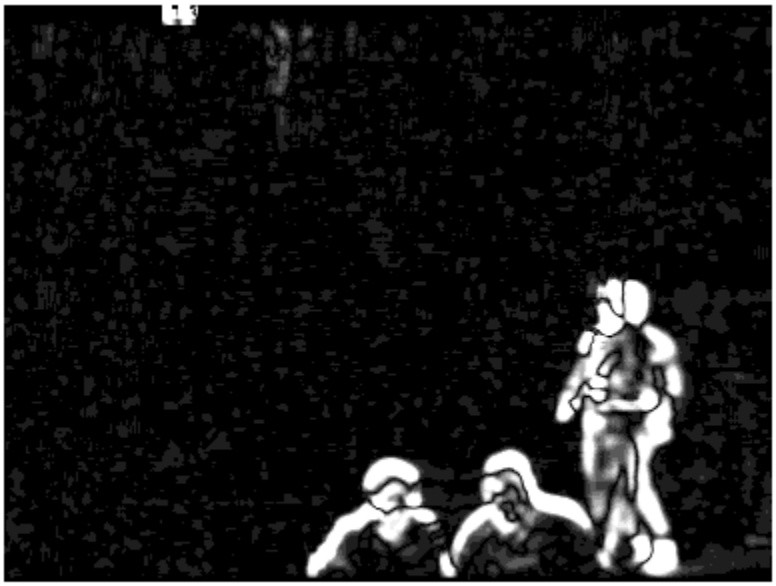

Figure 12: 5x5 Box Filter

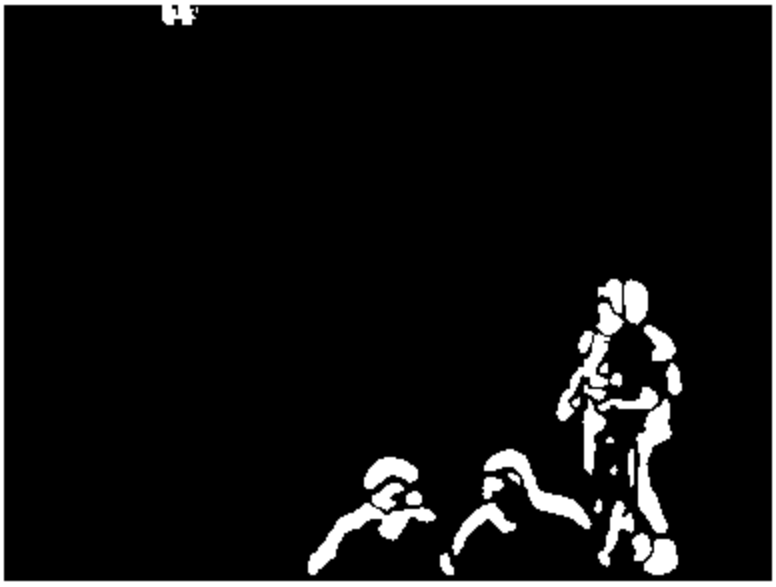

Figure 13: 5x5 Box Filter Thresholded

Figure 14: Gaussian Filter Sigma = 1

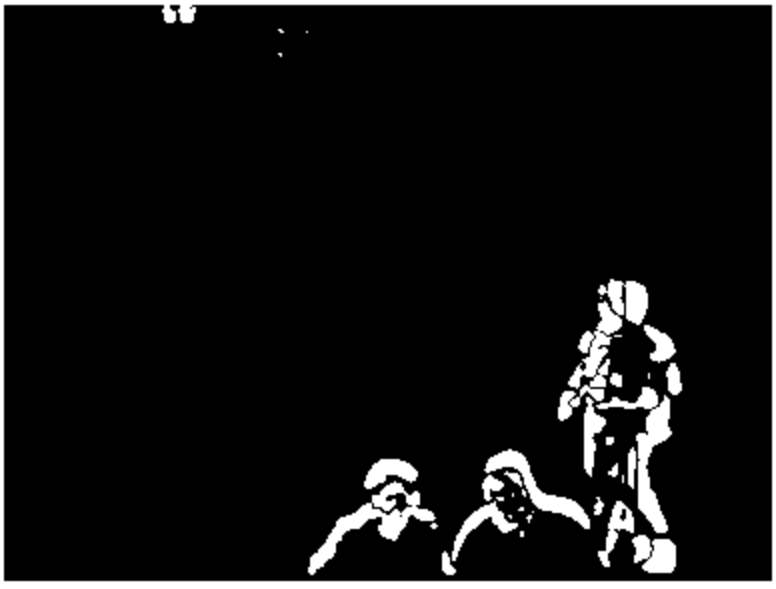

Figure 15: Gaussian Filter Sigma = 1 Thresholded

Figure 16: Gaussian Filter Sigma = 1.4

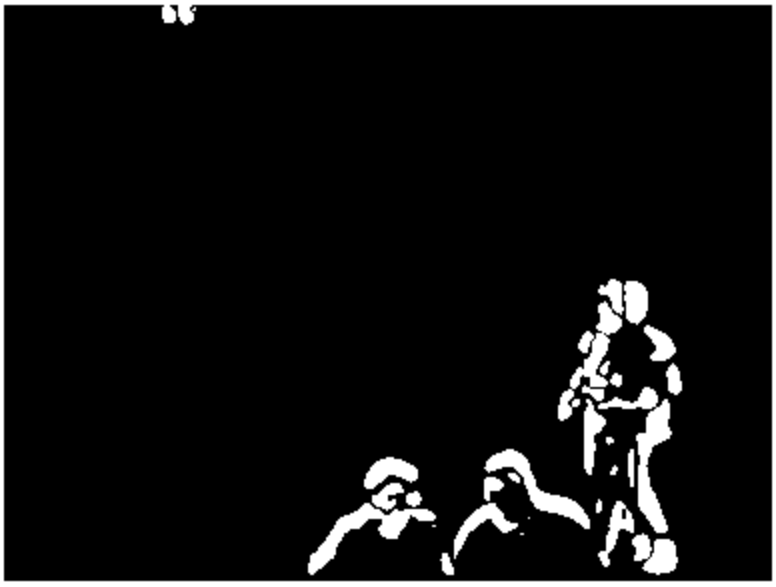

Figure 17: Gaussian Filter Sigma = 1.4 Thresholded

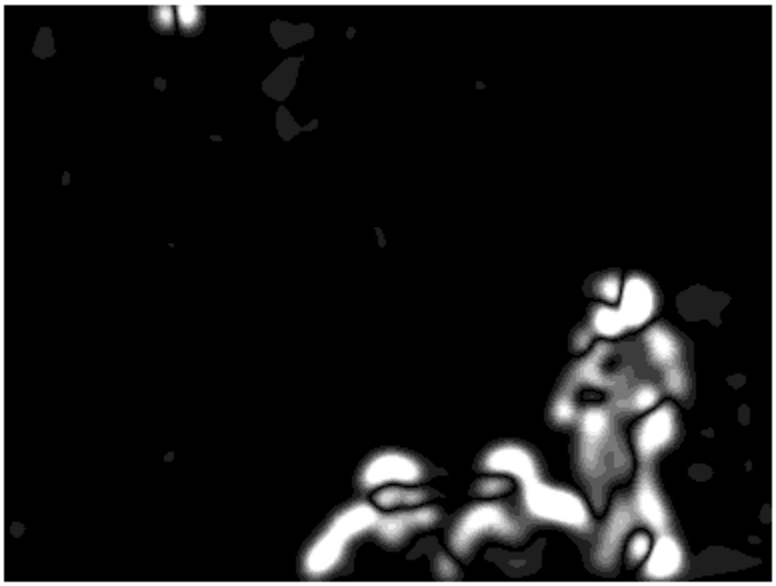

Figure 18: Gaussian Filter Sigma = 4.2

Figure 19: Gaussian Filter Sigma = 4.2 Thresholded

As we increase the size of the spatial filter, the image becomes more blurred, while the noise is reduced. This is true for both box filters as well as for the Gaussian filter. This can be helpful because it reduces the noise for each pixel and can increase the area where motion is detected. However, for a filter that is too large, such as the final filter with sigma = 4.2, the impact can be so great that the objects of interest are no longer able to be distinguished as they could be using the other filters. We found that the first 3x3 box filter is very similar to a Gaussian filter with sigma = 1 and a 5x5 box filter is very similar to a Gaussian filter with sigma = 1.4. The value of sigma = 1.4 works the best of our three cases because it reduces the noise more while still allowing the targets of interest (i.e. the three persons) to still be distinguished from one another by not excessively blurring boundaries.

iii. Thresholding

Below are shown the effects of applying different thresholds to the filtered image to create a mask.

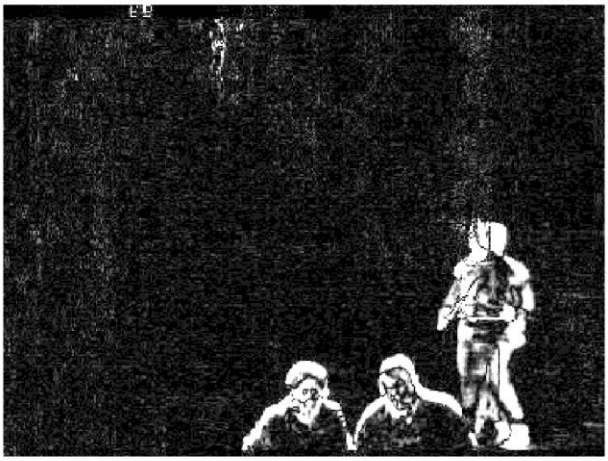

Figure 20: Filtered Image

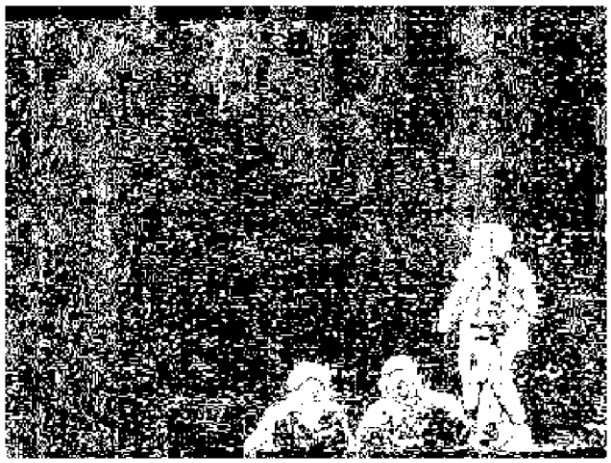

Figure 21: Threshold = 1

Figure 22: Threshold = 3

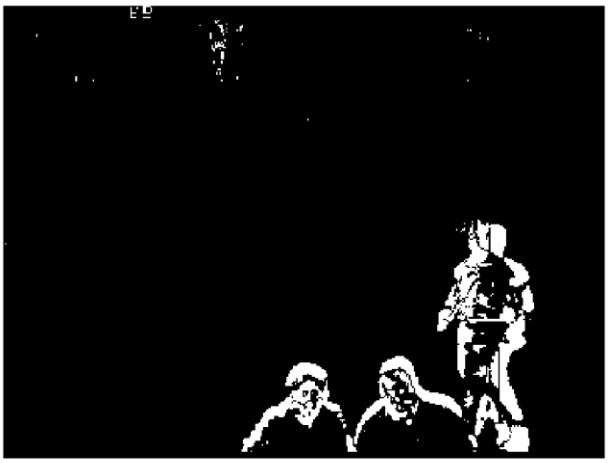

Figure 23: Threshold = 6

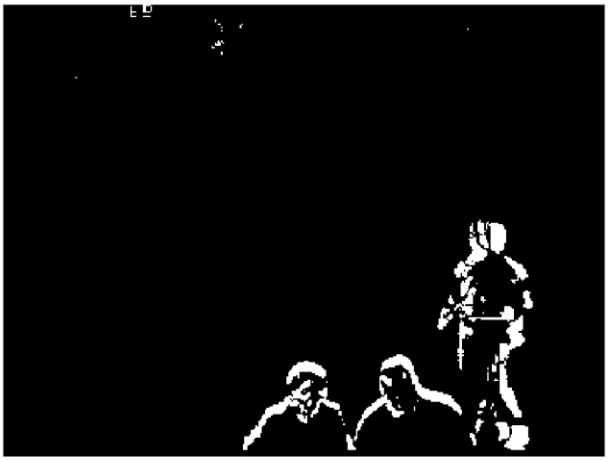

Figure 24: Threshold = 10

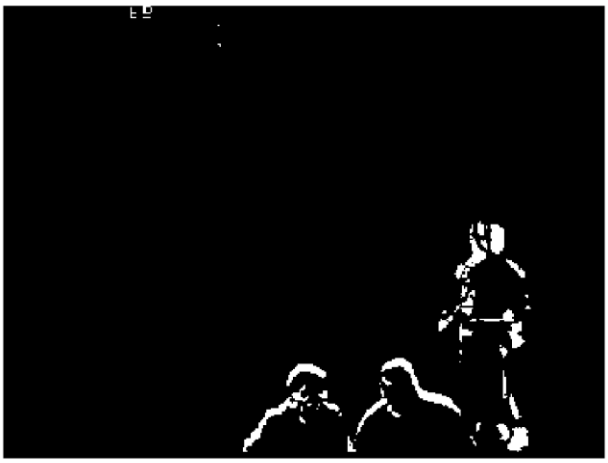

Figure 25: Threshold = 15

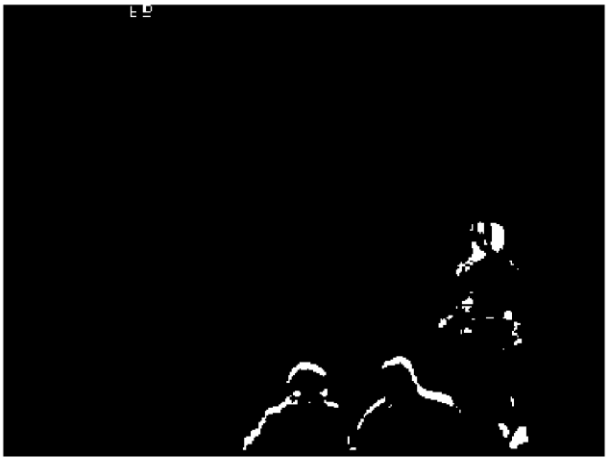

Figure 26: Threshold = 21

As the threshold increases, the amount of noise decreases significantly. However, the pixels that are detected above the noise also decrease, and the number of detections gets reduced. The optimal point occurs when there is the lowest threshold that remains above the maximum expected noise. Using our technique of noise estimation and assumptions, we estimated the noise to be 0.729. By setting the threshold to 5 times this value, we expect 99.9999% of the noise to fall beneath 3.645. The image we obtain is shown below.

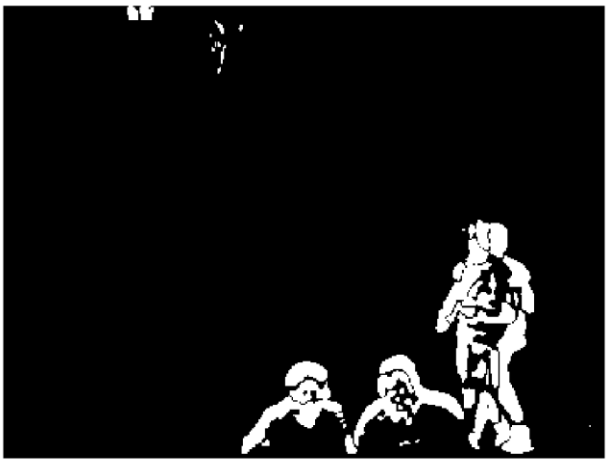

Figure 27: Threshold = 3.645

Appendix

Main.m

```
clc;

clear all;

close all;


srcFiles = dir([pwd '\EnterExitCrossingPaths2cor\*.jpg']);

% the folder in which ur images exists


%% prep display

figure(1), placeplot(1,1); fax(1) = gca;

figure(2), placeplot(2,2); fax(2) = gca;

figure(3), placeplot(3,3); fax(3) = gca;

figure(4), placeplot(4,4); fax(4) = gca;


%% gaussian kernel

tsigma = 1;


% Determine filter length

filterLength = ceil(5*(tsigma)) + mod(ceil(5*(tsigma))-1,2);

n = (filterLength - 1)/2;

x = -n:n;


% Create 1-D Gaussian Kernel

c = 1/(sqrt(2*pi)*tsigma);

gaussKernel = c * exp(-(x.^2)/(2*tsigma^2));


% Normalize to ensure kernel sums to one
```

```
gaussKernel = gaussKernel/sum(gaussKernel);

%% Create 1-D Derivative of Gaussian Kernel
derivGaussKernel = gradient(gaussKernel);
derivGaussKernel = derivGaussKernel/sum(abs(derivGaussKernel));



%% choose time domain kernel
kerneltype = 2;
switch(kerneltype)
   case 1,
      timekernel = 0.5 .* [-1 0 1];
   case 2,
      timekernel = derivGaussKernel;
end

numFrames = length(timekernel);

%% declare spatial Gaussain kernel
ssigma = 1;

% Determine filter length
filterLength = ceil(5*(ssigma)) + mod(ceil(5*(ssigma))-1,2);
n = (filterLength - 1)/2;
x = -n:n;

% Create 1-D Gaussian Kernel
c = 1/(sqrt(2*pi)*ssigma);
gaussKernel = c * exp(-(x.^2)/(2*ssigma^2));
```

```
% Normalize to ensure kernel sums to one
spaceKernel = gaussKernel/sum(gaussKernel);



%% process
for i = 60 % 1 : 400

   for f = (0-floor(numFrames/2)):(0+floor(numFrames/2))
      filename = strcat([pwd '\EnterExitCrossingPaths2cor\'],srcFiles(i+f-1).name);
      I = rgb2gray(imread(filename));
      bg(:,:,f+1+floor(numFrames/2)) = double(I); % image-time matrix: (row,column,frame)
   end

   fg = zeros(size(bg));

   %% 2D smoothing filter
   % choose filter
   smooth_type = 4;
   switch(smooth_type)
      case 1, % no smoothing
         filt = 1;
      case 2, % 3x3 box filter
         filt = ones([3 3]);
      case 3, % 5x5 box filter
         filt = ones([5 5]);
      case 4, % 2D Gaussian
         filt = bsxfun(@times,spaceKernel,spaceKernel.');
   end
```

```
%normalize
filt = filt ./ sum(sum(abs(filt)));

% frame by frame convolution
bg_smooth = zeros(size(bg));
for f = 1:numFrames
   bg_smooth(:,:,f) = conv2(bg(:,:,f),filt,'same');
end


%% Correlate with 1D gaussian in the temporal domain
frameFactor = bsxfun(@times,double(bg_smooth),shiftdim(timekernel,-1));
fr_diff = abs(sum(frameFactor,3));


%% Get noise estimate for thresholding
% get variance per pixel
bg_var = sum( (bsxfun(@minus,bg_smooth,mean(bg_smooth,3)).^2), 3)...
   ./ (numFrames - 1);
% choose median variance across the camera : assumes that less than half
% of the pixels are confounded with motion
bg_thresh = 5 * sqrt( median( reshape(bg_var,1,[]) ))

%   bg_thresh = 6;

%% Threshold
Mask = image_threshold ( fr_diff, bg_thresh );
```

```
  %% Display
  imshow(uint8(bg(:,:,ceil(numFrames/2))),'Parent',fax(1));
  imshow(uint8(bg_smooth(:,:,ceil(numFrames/2))),'Parent',fax(2));
  imshow(uint8(fr_diff)*30,'Parent',fax(3));
  imshow(Mask,'Parent',fax(4))
  pause(0.0000001);
end
```

image_threshold.m

```
function gray = image_threshold ( gray, a )

%*******************************************************************************80
%
%% IMAGE_THRESHOLD resets gray pixels to black or white based on a threshold.
%  Parameters:
%
%    Input, uint8 GRAY(:,:), the original data.
%
%    Input, uint8 A, the threshold value.  Pixels of value A or less
%    are reset to black.  0 <= A <= 255
%
%    Output, uint8 GRAY(:,:), the image data, whose pixels are now black
%    or white.
%
  i = find ( gray <= a );
  j = find ( a < gray );
```

```
  gray(i) = 0;
  gray(j) = 255;

  return
end
```