Motion Detection

Song Jiang and Mengyun Xia

Abstract

In this project we will explore 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. In this case, the intensity values observed at a 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.

1 Introduction

Motion detection is an interesting and classical topic in the computer vision community, which is a widely-used tool in video surveillance, object tracking, and *etc*. In this paper, we propose a simple implementation for it, based on several image filtering techniques and we will discuss which one is better.

2 Algorithm

***Algorithm 1.** Motion Detection based on Temporal Filtering*

Input: a frame sequence I_i, $1 \leq i \leq n$, a 1D temporal filter f_T of length m, a pre-defined threshold θ

Output: a temporal derivative sequence D_i, and the segmented motion objects F_i, $1 \leq i \leq n$

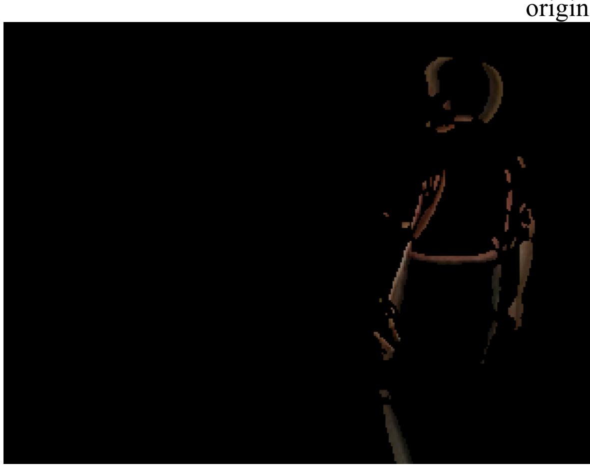
Initial: for $\forall i$, $D_i = \mathbf{0}$

Set s as $s = (m-1)/2$;
for $i = s+1 : n-s$ **do**
  for $j = 1 : s$ **do**
    $D_i = D_i + f_T(s-j+1)I_{i-j} + f_T(s+j+1)I_{i+j}$;
  end for
end for
for $i = 1 : n$ **do**
  Get a mask M_i for I_i by using Eq. 1
  Obtain F_i by $F_i = M_i \odot I_i$;
end for

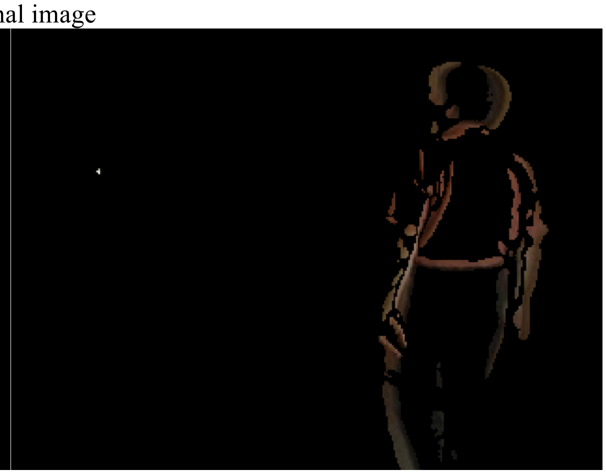
3 Experiments

1) Read in a sequence of image frames and make them grayscale and apply a 1-D differential operator at each pixel to compute a temporal derivative.
   (a) Derivative of Gaussian window size: 5*tsigma

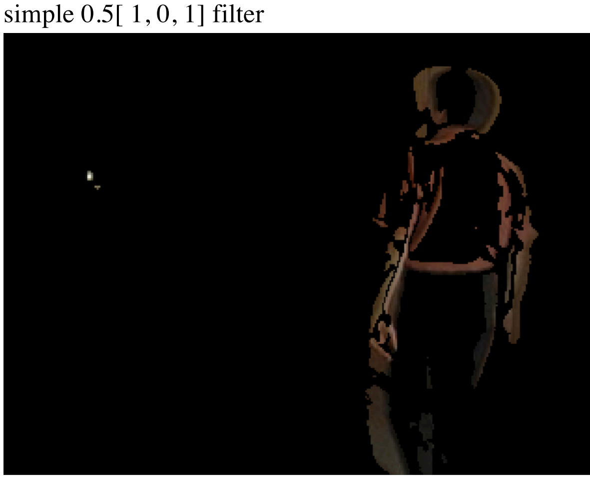
original image



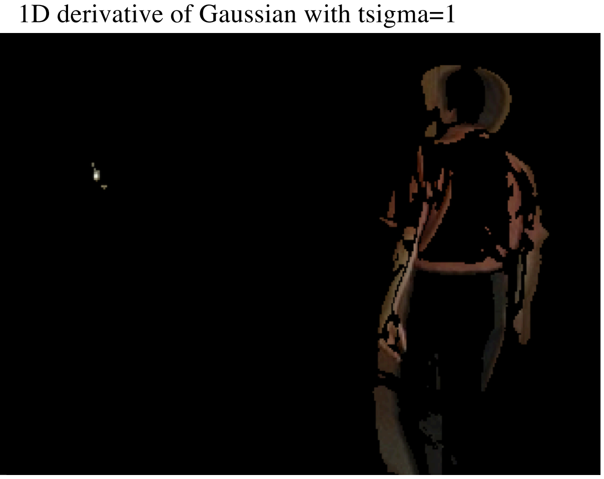
simple 0.5[1, 0, 1] filter



1D derivative of Gaussian with tsigma=1



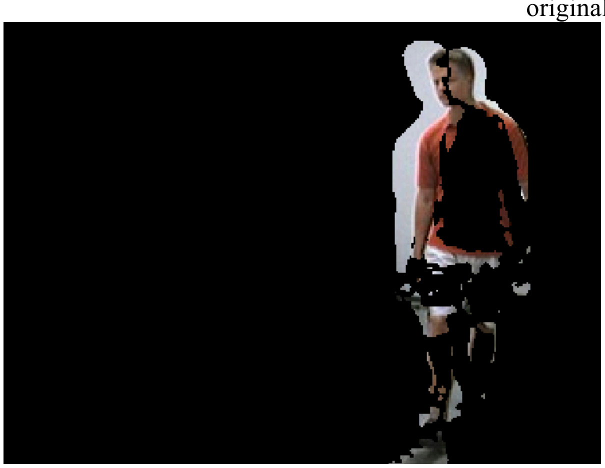
1D derivative of Gaussian with tsigma=1.8



1D derivative of Gaussian with tsigma=2.6

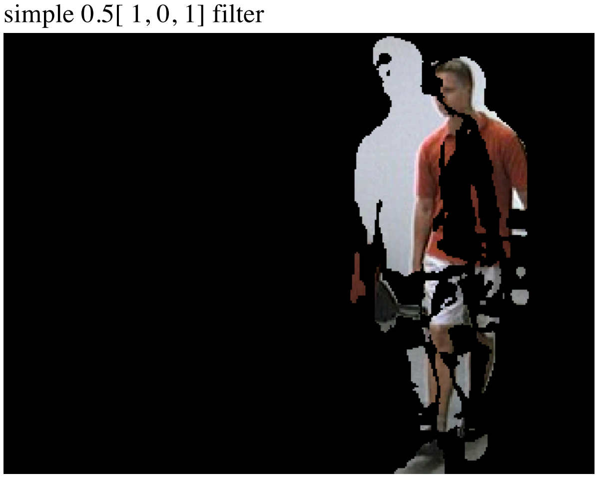
(b) Gaussian filter window size: 5*ssigma

original image

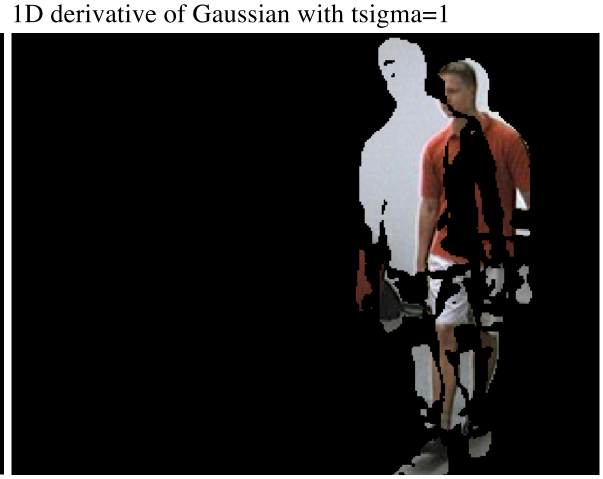


simple 0.5[1, 0, 1] filter

1D derivative of Gaussian with tsigma=1

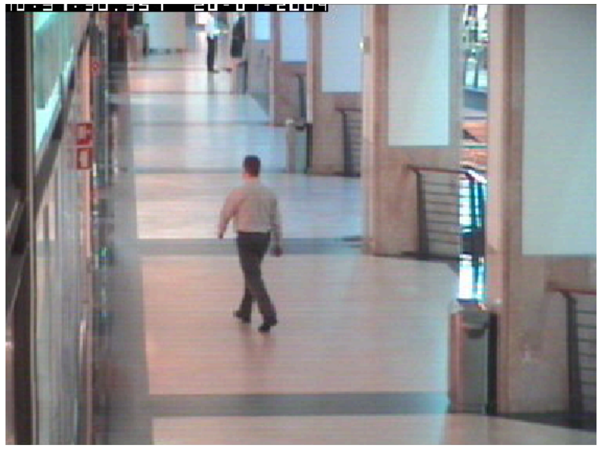


1D derivative of Gaussian with tsigma=1.8

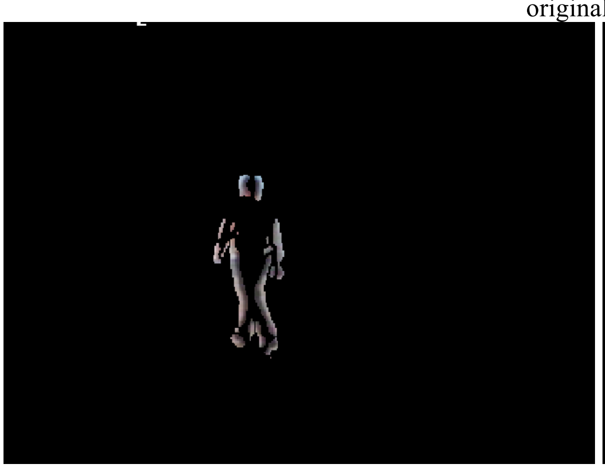


1D derivative of Gaussian with tsigma=2.6

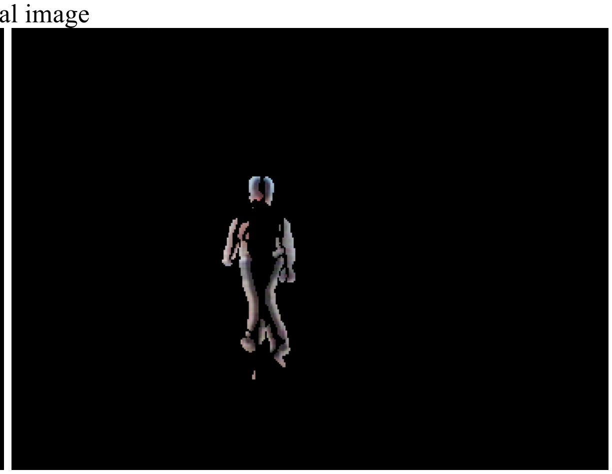
(c) Threshold at every image, take the standard deviation of the Gaussian noise in every image to be the threshold.



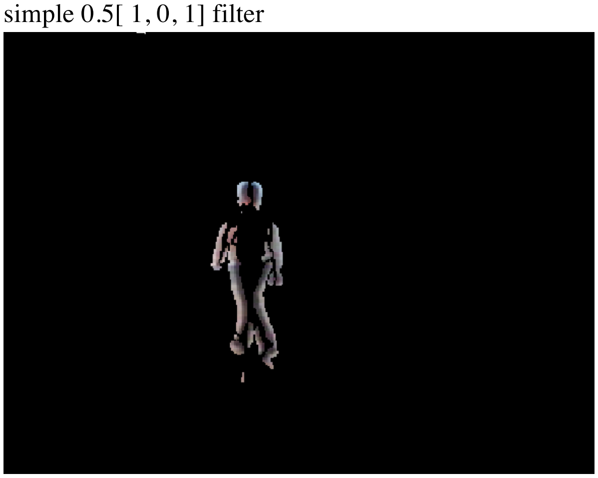
original image



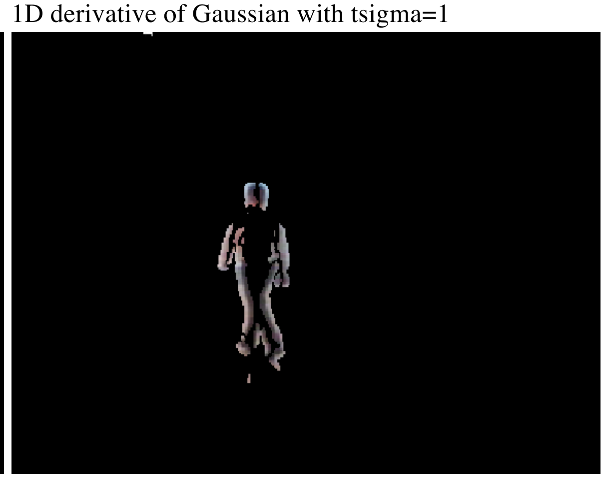
simple 0.5[1, 0, 1] filter



1D derivative of Gaussian with tsigma=1



1D derivative of Gaussian with tsigma=1.8



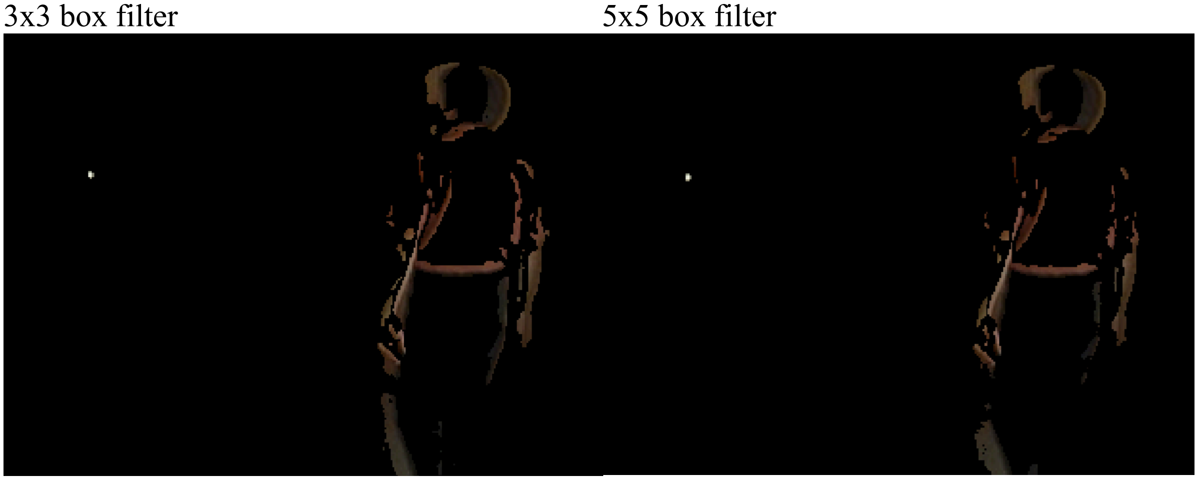
1D derivative of Gaussian with tsigma=2.6

We can see that under simple 0.5[1, 0, 1] filter, the boundaries are little and not complete. In Gaussian filter, the detection is better. And when tsigma equals to 1, the noise is less than tsigma equals to 1.8 or 2.6. So 1D derivative Gaussian with tsigma=1 is the best choice in this situation.

2) Applying a 2D spatial smoothing filter to the frames to reduce the noise before applying the temporal derivative filter. For the spatial smoothing filter, we try and compare 3x3, 5x5 box filters and 2D Gaussian filters with a defined standard deviation. In the report, we choose 1D derivative of Gaussian with tsigma=1 above in step (1) as reference.

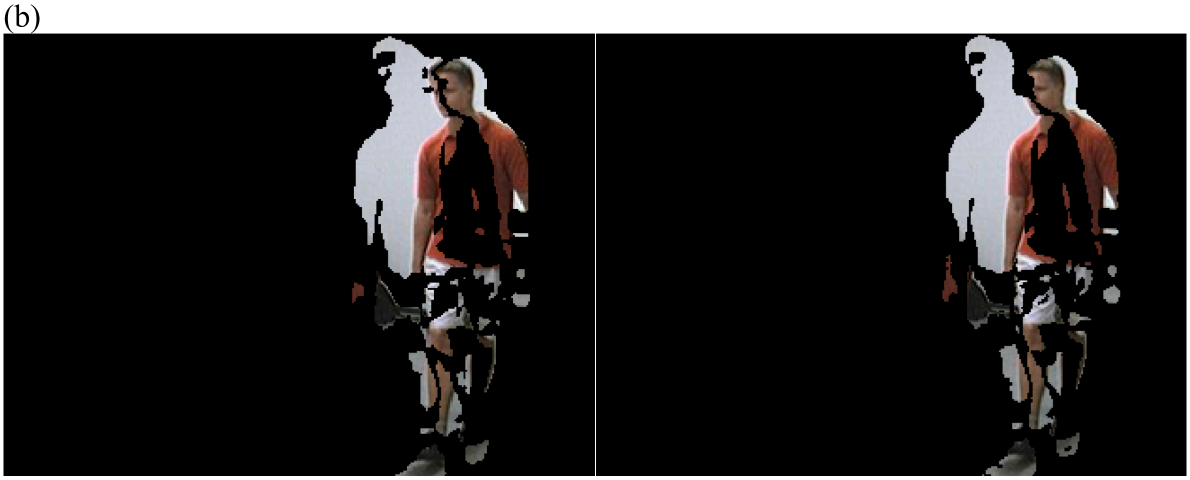
(a)

3x3 box filter 5x5 box filter



2D Gaussian filter with ssigma=1 2D Gaussian filter with ssigma=1.8

(b)

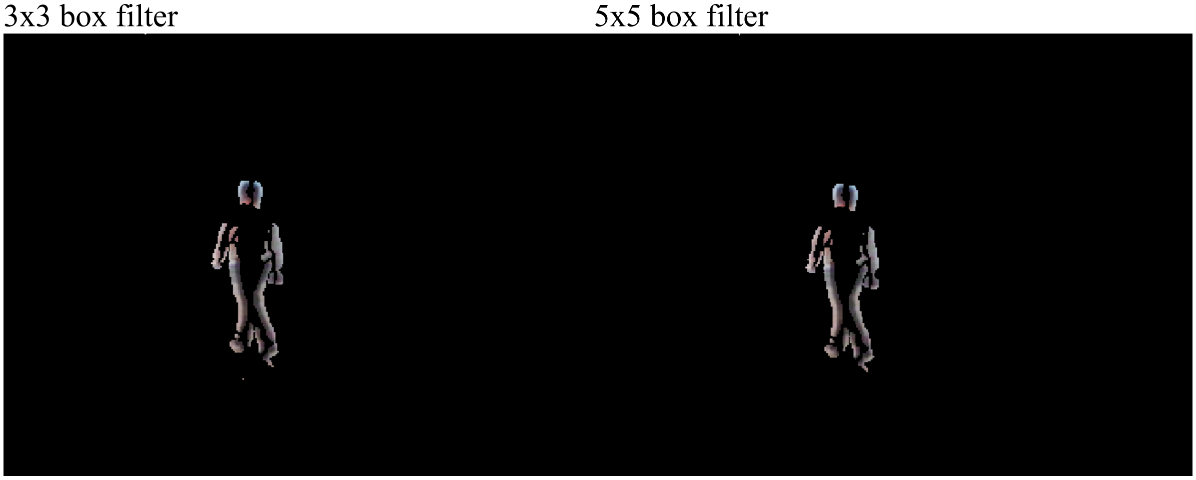


3x3 box filter 5x5 box filter

2D Gaussian filter with ssigma=1 2D Gaussian filter with ssigma=1.8

(c)

3x3 box filter 5x5 box filter

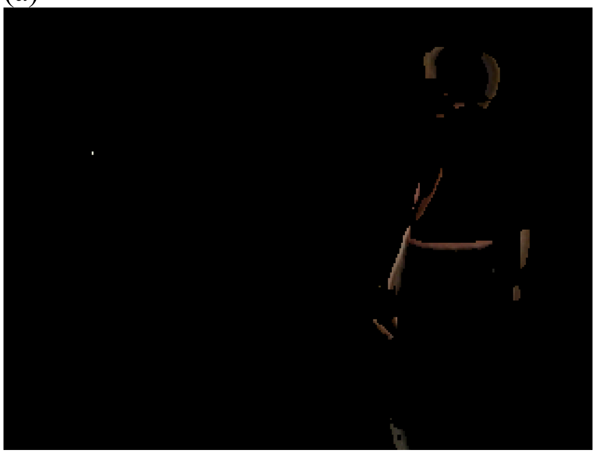


2D Gaussian filter with ssigma=1 2D Gaussian filter with ssigma=1.8

We can see that under 5x5 box filter or 2D Gaussian filter with ssigma=1.8, it filters too much important information and make the detection not complete. 3x3 box filter and 1D Gaussian filter with ssigma=1 is both effective to make the output better.

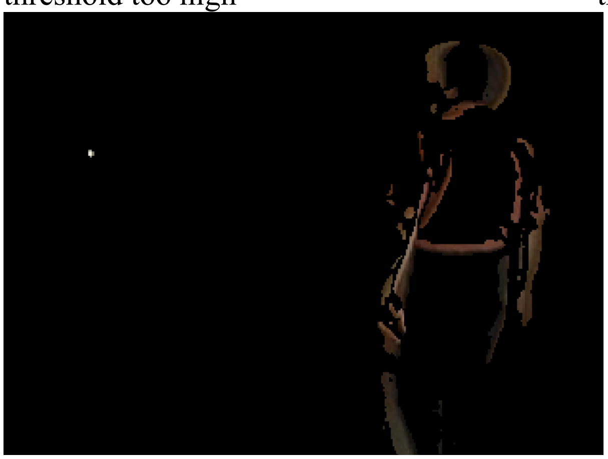
3) Vary the threshold to get the mask and see what works best. Design a strategy to select a good threshold for each image. Then Combine the mask with the original frame. In this report, we choose 1D derivative of Gaussian with tsigma=1 above in step (1) and 2D Gaussian filter with ssigma=1 as reference.

(a)



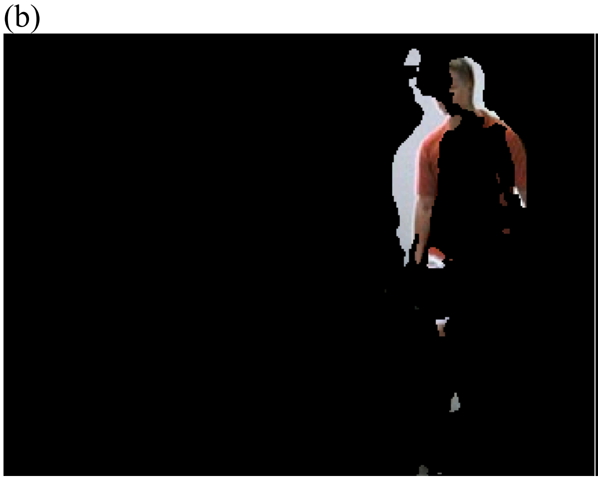
threshold too high

threshold too low



best threshold

(b)

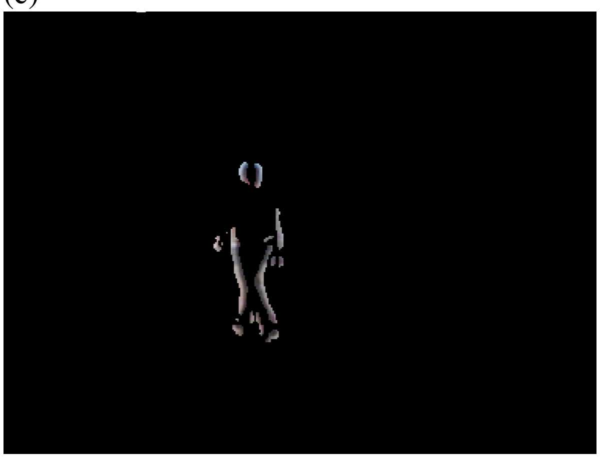


threshold too high

threshold too low

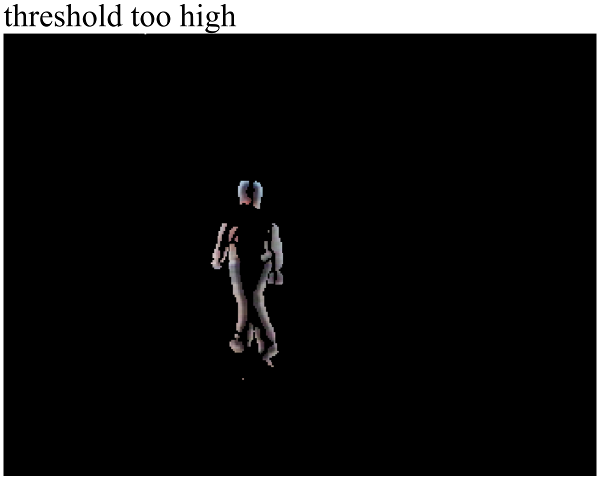
best threshold

(c)



threshold too high

threshold too low



best threshold

We can see that if the threshold is too high, then there will be more gaps. If the threshold is too low, there will be more noise. If the threshold is suitable, the detection is wonderful.

4 Conclusion

When doing the motion detection, we can first use a 2D Gaussian filter to erase noise, then apply a 1D derivative of Gaussian filter with the standard deviation of 1.0 or 1.4 at each pixel of a series of continuous frames to compute a temporal derivative. Choose a reasonable threshold and threshold the absolute values of the derivatives to create a 0 and 1 mask of the moving objects. Finally combine the mask with the original frame to display the results.

4 Appendix

```
clear;
close all;
clc;

%%read and store multiple image frames
file_path = './EnterExitCrossingPaths2cor/';
img_path_list = dir(strcat(file_path,'*.jpg'));
%% the number of image frames
img_num = length(img_path_list);
      %% used to store original image frames
      img = cell(1,img_num);
       %%used to store gray values of image frames
      img_gray = cell(1,img_num);
      finalImageSF=cell(1,img_num);
      finalImageG1=cell(1,img_num);
      finalImageG2=cell(1,img_num);
      finalImageG3=cell(1,img_num);
      for i=1:img_num
          image_name=img_path_list(i).name;
          image_name=strcat(file_path,image_name);
          img{i}=imread(image_name);
          finalImageSF{i}=double(img{i});
          finalImageG1{i}=double(img{i});
          finalImageG2{i}=double(img{i});
          finalImageG3{i}=double(img{i});
          %% make image frames grayscale
          img_gray{i}=rgb2gray(img{i});
      end

      h3=fspecial('average',[3,3]);
      h5=fspecial('average',[5,5]);
      G1=fspecial('Gaussian',[5,5],1.0);
      G2=fspecial('Gaussian',[9,9],1.8);
      I_G1=cell(1,img_num);
      for i=1:img_num
          I_G1{i}=imfilter(img_gray{i},G1,'replicate');
      end

      %%the length value of the matrix correspond to one image frame
      len=size(img{1},2);
      %%the width value of the matrix correspond to one image frame
      wid=size(img{1},1);
```

```
    %%sf_G1 is a?wid x len?matrix,
    %%each entry is also a (1 x img_num) matrix
    gx1_G1=cell(wid,len);

    for i=1:wid
        for j=1:len
            for k=1:img_num
                gx1_G1{i,j}(k)=I_G1{k}(i,j);
            end
        end
    end
    sf_G1=gx1_G1;
    gx2_G1=gx1_G1;
    gx3_G1=gx1_G1;
    %%create a simple 0.5[-1, 0, 1] filter
    sim_f=[-0.5,0,0.5];
    t1=1.0; %%value of sigma t1
    t2=1.8; %%value of sigma t2
    t3=2.6; %%value of sigma t3
    gx1_mask=zeros(1,5);%range 5*t1
    gx2_mask=zeros(1,9);%range 5*t2
    gx3_mask=zeros(1,13);%range 5*t3

    %%a 1D derivative of Gaussian filter with standard devition of 1.0
    for x=-2:2
        gx1_mask(x+3)=-x/(t1*t1)*exp(-(x*x)/(2*t1*t1));
    end
    %%a 1D derivative of Gaussian filter with standard devition of 1.8
    for x=-4:4
        gx2_mask(x+5)=-x/(t2*t2)*exp(-(x*x)/(2*t2*t2));
    end
    %%a 1D derivative of Gaussian filter with standard devition of 2.6
    for x=-6:6
        gx3_mask(x+7)=-x/(t3*t3)*exp(-(x*x)/(2*t3*t3));
    end

    for i=1:wid
        for j=1:len
            %sf_G1{i,j}=imfilter(double(sf_G1{i,j}),sim_f,'replicate');
gx1_G1{i,j}=imfilter(double(gx1_G1{i,j}),gx1_mask,'replicate');
            %gx2_G1{i,j}=imfilter(double(gx2_G1{i,j}),gx2_mask,'replicate');
            %gx3_G1{i,j}=imfilter(double(gx3_G1{i,j}),gx3_mask,'replicate');
            %gx1_G1{i,j}=imfilter(gx1_G1{i,j},h3,'replicate');
            %gx1_G1{i,j}=imfilter(gx1_G1{i,j},h5,'replicate');
            gx1_G1{i,j}=imfilter(gx1_G1{i,j},G1,'replicate');
            %gx1_G1{i,j}=imfilter(gx1_G1{i,j},G2,'replicate');
        end
    end


```

```
    thr_sf_G1=cell(1,img_num);
    thr_gx1_G1=cell(1,img_num);
    thr_gx2_G1=cell(1,img_num);
    thr_gx3_G1=cell(1,img_num);

    for k=1:img_num
        for i=1:wid
            for j=1:len
                %thr_sf_G1{k}(i,j)=sf_G1{i,j}(k);
                thr_gx1_G1{k}(i,j)=gx1_G1{i,j}(k);
                %thr_gx2_G1{k}(i,j)=gx2_G1{i,j}(k);
                %thr_gx3_G1{k}(i,j)=gx3_G1{i,j}(k);
            end
        end
    end



    %calculate the threshold
    thres1=cell(1,img_num);
    thres2=cell(1,img_num);
    thres3=cell(1,img_num);
    thres4=cell(1,img_num);
    for i=1:img_num
        %thres1{i}=(std(double(thr_sf_G1{i}(:)))+7)*ones(wid,len);
        thres2{i}=(std(double(thr_gx1_G1{i}(:)))+7)*ones(wid,len);
        %thres3{i}=(std(double(thr_gx2_G1{i}(:)))+7)*ones(wid,len);
        %thres4{i}=(std(double(thr_gx3_G1{i}(:)))+7)*ones(wid,len);
    end

    %Convert image to binary image by thresholding
    for i=1:img_num
        %thr_sf_G1{i}=im2bw(abs(thr_sf_G1{i})-thres1{i},1/255);
        thr_gx1_G1{i}=im2bw(abs(thr_gx1_G1{i})-thres2{i},1/255);
        %thr_gx2_G1{i}=im2bw(abs(thr_gx2_G1{i})-thres3{i},1/255);
        %thr_gx3_G1{i}=im2bw(abs(thr_gx3_G1{i})-thres4{i},1/255);
        %finalImageSF{i}(:,:,1) = finalImageSF{i}(:,:,1).*double(thr_sf_G1{i});
        %finalImageSF{i}(:,:,2) = finalImageSF{i}(:,:,2).*double(thr_sf_G1{i});
        %finalImageSF{i}(:,:,3) = finalImageSF{i}(:,:,3).*double(thr_sf_G1{i});
        finalImageG1{i}(:,:,1) = finalImageG1{i}(:,:,1).*double(thr_gx1_G1{i});
        finalImageG1{i}(:,:,2) = finalImageG1{i}(:,:,2).*double(thr_gx1_G1{i});
        finalImageG1{i}(:,:,3) = finalImageG1{i}(:,:,3).*double(thr_gx1_G1{i});
        %finalImageG2{i}(:,:,1) = finalImageG2{i}(:,:,1).*double(thr_gx2_G1{i});
        %finalImageG2{i}(:,:,2) = finalImageG2{i}(:,:,2).*double(thr_gx2_G1{i});
        %finalImageG2{i}(:,:,3) = finalImageG2{i}(:,:,3).*double(thr_gx2_G1{i});
        %finalImageG3{i}(:,:,1) = finalImageG3{i}(:,:,1).*double(thr_gx3_G1{i});
        %finalImageG3{i}(:,:,2) =
```

```
finalImageG3{i}(:,:,2).*double(thr_gx3_G1{i});
        %finalImageG3{i}(:,:,3) = finalImageG3{i}(:,:,3).*double(thr_gx3_G1{i});
    end

    %{
    %display the result of images filtered by a simple[-0.5 0 0.5] filter
    for i=1:img_num
        figure(),imshow(uint8(finalImageSF{i}));
        pause(0.05);
    end
    %}

    %display the result of images filtered by a 1D derivative
    %of Gaussian filter with standard devition of 1.0
    for i=1:img_num
        figure(),imshow(uint8(finalImageG1{i}));
        pause(0.05);
    end

    %{
    %display the result of images filtered by a 1D derivative
    %of Gaussian filter with standard devition of 1.4
    for i=1:img_num
        figure(),imshow(uint8(finalImageG2{i}));
        pause(0.05);
    end

    %display the result of images filtered by a 1D derivative
    %of Gaussian filter with standard devition of 1.8
    for i=1:img_num
        figure(),imshow(uint8(finalImageG3{i}));
        pause(0.05);
    end
    %}
```