Motion Detection Using Simple Image Filtering

Song Jiang and Mengyun Xia
EECE 5639, Computer Vision
Feburary 14, 2016

Abstract

In this paper we present a computational approach 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. This approach utilizes the spatial smoothing filters, temporal derivative filters and thresholding technique. Depending on the requirements of a particular task, appropriate parameters such as the size of the mask of Box filter, standard deviation of 2-D Gaussian filter and magnitude of thresholding, are chosen to control the resolution. We compare some different ways and choose the best one to complete the motion detection.

1. Introduction

In this project you 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.

2. Description of Algorithms

1. Read in a sequence of image frames and make them grayscale.
2. As enough frames are available, apply a 1-D differential operator at each pixel to compute a temporal derivative.
3. Threshold the absolute values of the derivatives to create a 0 and 1 mask of the moving objects.
4. Combine the mask with the original frame to display the results.

3. Experiments

1. Read in a sequence of image frames and make them grayscale.

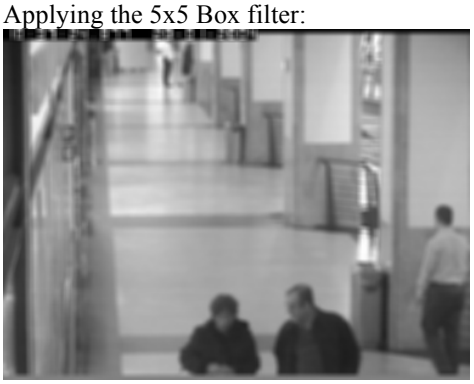
```
%%read and store multiple image frames
file_path = './EnterExitCrossingPaths2cor/';
img_path_list = dir(strcat(file_path,'*.jpg'));
img_num = length(img_path_list); %% the number of image frames
img = cell(1,img_num); %% used to store original image frames
img_gray = cell(1,img_num); %%used to store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

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.

```
h3=fspecial('average',[3,3]);
h5=fspecial('average',[5,5]);
G1=fspecial('Gaussian',[5,5],1.0);
G2=fspecial('Gaussian',[7,7],1.4);
G3=fspecial('Gaussian',[9,9],1.8);
I_h3=cell(1,img_num);
I_h5=cell(1,img_num);
I_G1=cell(1,img_num);
I_G2=cell(1,img_num);
I_G3=cell(1,img_num);
for i=1:img_num
    %%spatial smoothing filter with 3X3 box filter
    I_h3{i}=imfilter(img_gray{i},h3,'replicate');
    %%spatial smoothing filter with 5X5 box filter
    I_h5{i}=imfilter(img_gray{i},h5,'replicate');
    %%spatial smoothing filter with 2D Gaussian filters with standard deviation ssigma=1.0
    I_G1{i}=imfilter(img_gray{i},G1,'replicate');
    %%spatial smoothing filter with 2D Gaussian filters with standard deviation ssigma=1.4
    I_G2{i}=imfilter(img_gray{i},G2,'replicate');
    %%spatial smoothing filter with 2D Gaussian filters with standard deviation ssigma=1.8
    I_G3{i}=imfilter(img_gray{i},G3,'replicate');
end
```

Applying the 3x3 Box filter:

Applying the 5x5 Box filter:



Applying the 2D Gaussian filter with std.=1.0:

Applying the 2D Gaussian filter with std.=1.4:

Applying the 2D Gaussian filter with std.=1.8:

When the mask is bigger, although it will make smaller noise variance of the output, it will also make the image more blurring and it is expensive to compute. Gaussian filter is weighted average filter and it causes less blurring of edges. So, in this project, we choose the 2D Gaussian filter with std.=1.0 to filter the original images.

3. For the temporal derivative filter, we try a simple 0.5[-1, 0, 1] filter and a 1D derivative of a Gaussian with a defined standard deviation. For the Gaussian filter, we try increasing values of standard deviation.

The formula of the 1D derivative of a Gaussian:

$$g'(x) = -\frac{1}{2\sigma^2} 2xe^{-\frac{x^2}{2\sigma^2}}$$

```
len=size(img{1},2); %%the length value of the matrix correspond to one image frame
wid=size(img{1},1); %%the width value of the matrix correspond to one image frame
sf_G1=cell(wid,len);%%sf_G1 is a (wid x len) matrix, each entry is also a (1 x img_num) matrix
%%store each corresponding gray values of multiple image frames filtered by a 2D Gaussian filter
for i=1:wid
    for j=1:len
        for k=1:img_num
            sf_G1{i,j}(k)=I_G1{k}(i,j);
        end
    end
end
gx1_G1=sf_G1;
gx2_G1=sf_G1;
gx3_G1=sf_G1;
sim_f=[-0.5,0,0.5]; %%create a simple 0.5[-1, 0, 1] filter
t1=1.0; %%value of sigma t1
t2=1.4; %%value of sigma t2
t3=1.8; %%value of sigma t3
gx1_mask=zeros(1,5);%range 5*t1
gx2_mask=zeros(1,7);%range 5*t2
gx3_mask=zeros(1,9);%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 1.4
for x=-3:3
    gx2_mask(x+4)=-x/(t2*t2)*exp(-(x*x)/(2*t2*t2));
end
%%a 1D derivative of Gaussian filter with standard devition of 1.8
for x=-4:4
    gx3_mask(x+5)=-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');
    end
end
```

4. 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.

Most pixels belonging to the background change little and thus their temporal gradients are close to zero. We

can model these values as Gaussian zero mean noise and estimate the standard deviation of this noise. If we choose the threshold bigger than the mean standard deviation of that noise, then we can extract the moving objects with the background in a good performance.

```
%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}(:)))*ones(wid,len);
    thres2{i}=std(double(thr_gx1_G1{i}(:)))*ones(wid,len);
    thres3{i}=std(double(thr_gx2_G1{i}(:)))*ones(wid,len);
    thres4{i}=std(double(thr_gx3_G1{i}(:)))*ones(wid,len);
end
```

Set the value of threshold and threshold the absolute values of the derivatives to create a 0 and 1 mask of the moving objects.

```
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);
%store gray values of each image frames filtered by different temporal derivative filter
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
```

```
final1_img=cell(1,img_num);%%store the final mask after thresholding
final2_img=cell(1,img_num);
final3_img=cell(1,img_num);
final4_img=cell(1,img_num);
%Convert image to binary image by thresholding
for i=1:img_num
    thr_sf_G1{i}=im2bw(abs(thr_sf_G1{i})-thres{i},1/255);
    thr_gx1_G1{i}=im2bw(abs(thr_gx1_G1{i})-thres{i},1/255);
    thr_gx2_G1{i}=im2bw(abs(thr_gx2_G1{i})-thres{i},1/255);
    thr_gx3_G1{i}=im2bw(abs(thr_gx3_G1{i})-thres{i},1/255);
    %Combine the mask with the original frame
    final1_img{i}=double(img_gray{i}).*double(thr_sf_G1{i});
    final2_img{i}=double(img_gray{i}).*double(thr_gx1_G1{i});
    final3_img{i}=double(img_gray{i}).*double(thr_gx2_G1{i});
    final4_img{i}=double(img_gray{i}).*double(thr_gx3_G1{i});
end
```

5. Display the results.

```
%display the result of images filtered by a simple[-0.5 0 0.5] filter
for i=1:img_num
    imshow(final1_img{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
    imshow(final2_img{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
    imshow(final3_img{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
    imshow(final4_img{i});
    pause(0.05);
end
```

4. observations

1. Set the threshold not changed, and compare different derivative filters.

If images are filtered by a simple [-0.5 0 0.5] filter, noise is not increased much, a few edges are missing.

If images are filtered by a 1D derivative of Gaussian filter with standard deviation of 1.8, due to the bigger standard deviation, the images are smoothed more and the result has fewer edges.

If images are filtered by a 1D derivative of Gaussian filter with standard deviation 1.4, the effect is good, the noise is decreased and the edges are clearer.

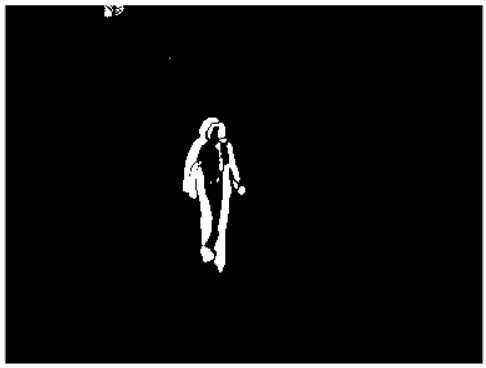
If images are filtered by a 1D derivative of Gaussian filter with standard deviation 1.0, the effect is also good, the noise is decreased and the edges are clearer.

2. Use a 1D derivative of Gaussian filter with standard deviation of 1.0, change threshold to compare the results.

If the threshold is too high, then there are too many gaps.

If the threshold is too low, then there are too many edges.

If the threshold is suitable, then the edge is clear.



5. 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.

6. Appendix

```
clear;
close all;
clc;

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

    h3=fspecial('average',[3,3]);
    h5=fspecial('average',[5,5]);
    G1=fspecial('Gaussian',[5,5],1.0);
    G2=fspecial('Gaussian',[7,7],1.4);
    G3=fspecial('Gaussian',[9,9],1.8);
    I_h3=cell(1,img_num);
    I_h5=cell(1,img_num);
    I_G1=cell(1,img_num);
    I_G2=cell(1,img_num);
    I_G3=cell(1,img_num);
    for i=1:img_num
        %%spatial smoothing filter with 3X3 box filter
        I_h3{i}=imfilter(img_gray{i},h3,'replicate');
        %%spatial smoothing filter with 5X5 box filter
        I_h5{i}=imfilter(img_gray{i},h5,'replicate');
        %%spatial smoothing filter with 2D Gaussian filters with
         standard deviation ssigma=1.0
        I_G1{i}=imfilter(img_gray{i},G1,'replicate');
        %%spatial smoothing filter with 2D Gaussian filters with
         standard deviation ssigma=1.4
        I_G2{i}=imfilter(img_gray{i},G2,'replicate');
        %%spatial smoothing filter with 2D Gaussian filters with
         standard deviation ssigma=1.8
        I_G3{i}=imfilter(img_gray{i},G3,'replicate');
    end

    len=size(img{1},2); %%the length value of the matrix correspond
```

```
                                to one image frame
42.      wid=size(img{1},1); %%the width value of the matrix correspond to
                                one image frame
43.      sf_G1=cell(wid,len);%%sf_G1 is a?wid x len?matrix, each entry is
                                also a (1 x img_num) matrix
44.      %%store each corresponding gray values of multiple image frames
         filtered by a 2D Gaussian filter
45.      for i=1:wid
46.          for j=1:len
47.              for k=1:img_num
48.                  sf_G1{i,j}(k)=I_G1{k}(i,j);
49.              end
50.          end
51.      end
52.      gx1_G1=sf_G1;
53.      gx2_G1=sf_G1;
54.      gx3_G1=sf_G1;
55.      sim_f=[-0.5,0,0.5]; %%create a simple 0.5[-1, 0, 1] filter
56.      t1=1.0; %%value of sigma t1
57.      t2=1.4; %%value of sigma t2
58.      t3=2.6; %%value of sigma t3
59.      gx1_mask=zeros(1,5);%range 5*t1
60.      gx2_mask=zeros(1,7);%range 5*t2
61.      gx3_mask=zeros(1,13);%range 5*t3
62.
63.      %%a 1D derivative of Gaussian filter with standard devition of
     1.0
64.      for x=-2:2
65.          gx1_mask(x+3)=-x/(t1*t1)*exp(-(x*x)/(2*t1*t1));
66.      end
67.      %%a 1D derivative of Gaussian filter with standard devition of
     1.4
68.      for x=-3:3
69.          gx2_mask(x+4)=-x/(t2*t2)*exp(-(x*x)/(2*t2*t2));
70.      end
71.      %%a 1D derivative of Gaussian filter with standard devition of
     1.8
72.      for x=-6:6
73.          gx3_mask(x+7)=-x/(t3*t3)*exp(-(x*x)/(2*t3*t3));
74.      end
75.
76.      for i=1:wid
77.          for j=1:len
78.
     sf_G1{i,j}=imfilter(double(sf_G1{i,j}),sim_f,'replicate');
79.
     gx1_G1{i,j}=imfilter(double(gx1_G1{i,j}),gx1_mask,'replicate');
80.
     gx2_G1{i,j}=imfilter(double(gx2_G1{i,j}),gx2_mask,'replicate');
81.
          gx3_G1{i,j}=imfilter(double(gx3_G1{i,j}),gx3_mask,'replicate');
82.          end
83.      end
84.
85.      thr_sf_G1=cell(1,img_num);
86.      thr_gx1_G1=cell(1,img_num);
87.      thr_gx2_G1=cell(1,img_num);
```

```
    thr_gx3_G1=cell(1,img_num);
    %store gray values of each image frames filtered by different
     temporal derivative filter
    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

    final1_img=cell(1,img_num);%%store the final mask after
     thresholding
    final2_img=cell(1,img_num);
    final3_img=cell(1,img_num);
    final4_img=cell(1,img_num);
    %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);
        %Combine the mask with the original frame
        final1_img{i}=double(img_gray{i}).*double(thr_sf_G1{i});
        final2_img{i}=double(img_gray{i}).*double(thr_gx1_G1{i});
        final3_img{i}=double(img_gray{i}).*double(thr_gx2_G1{i});
        final4_img{i}=double(img_gray{i}).*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
        imshow(final1_img{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
        imshow(final2_img{i});
        pause(0.1);
    end
    %display the result of images filtered by a 1D derivative
    %of Gaussian filter with standard devition of 1.4
```

```
143.     for i=1:img_num
144.         imshow(final3_img{i});
145.         pause(0.05);
146.     end
147.     %display the result of images filtered by a 1D derivative
148.     %of Gaussian filter with standard devition of 1.8
149.     for i=1:img_num
150.         imshow(final4_img{i});
151.         pause(0.05);
152.     end
```