CS6643 Computer Vision

Assignment 4 - Optical Flow

Name: Yunnan Li NetId: yl3651

Pratical Problem Report

Problem 1: Normal Flow

1.1 Experiment process

In this problem I will implement a simple algorithm to calculate and show the normal flow from a pair of toy video sequence images.

Firstly, I use 'imread' function to input all of the toy images into matlab. So that I could choose two consequence images to do the experiment. And I can show the original image at the end to compare with the normal flow image. (see Figure 1)

```
%read images
image1 = imread('toy_formatted2.png');
image2 = imread('toy_formatted3.png');
image3 = imread('toy_formatted4.png');
image4 = imread('toy_formatted5.png');
image5 = imread('toy_formatted6.png');
image6 = imread('toy_formatted7.png');
image7 = imread('toy_formatted8.png');
image8 = imread('toy_formatted9.png');
```

Figure 1, image input code

Next step is smoothing images. But before that, because the format of toy image is unit8, which is a kind of unsigned integer and used to represent the graphic, I have to change the format of each image matrix form unit8 to double, so that in the following step, the temporal gradient matrix and spatial derivation values can be positive or negative.

The most important part of this step is smoothing process. I used gaussian_filter.m, which is given on NYU classes. The reason why I need apply smoothing to images is that, optical flow assumes smooth object boundaries. At the end of this part I

output filtered result to check effect. The filtered image will be show in next part. (see Figure 2)

```
%applying smoothing to the images
sigma = 4.5;
filtered_image1 = gaussian_filter(double(image1), sigma);
filtered_image2 = gaussian_filter(double(image2), sigma);
filtered_image3 = gaussian_filter(double(image3), sigma);
filtered_image4 = gaussian_filter(double(image4), sigma);
filtered_image5 = gaussian_filter(double(image5), sigma);
filtered_image6 = gaussian_filter(double(image6), sigma);
filtered_image7 = gaussian_filter(double(image7), sigma);
filtered_image8 = gaussian_filter(double(image8), sigma);

figure(1);
imshow(filtered_image1);
title('Filtered image1');
```

Figure 2, Image smoothing code

After that I have to calculate the temporal gradient image $\frac{\partial I}{\partial t}$. In order to do that, I choose toy image 1 and image 2 to calculate difference from the blur version of those two images. (see figure 3)

```
%calculate temporal gradient
offset = 90; %use to move double pixel value to integer range

temporal_gradient = double(filtered_image2)-double(filtered_image1);
figure(2);
imshow(temporal_gradient+offset,[0,255]);
title('Temporal gradient map');
```

Figure 3, temporal gradient code

Next step, I estimate the spatial derivatives, $I_x = \frac{\partial I}{\partial x}$ and $I_y = \frac{\partial I}{\partial y}$. But in real code, for simplicity, we just have to calculate $I_x = I(x+1, y, t) - I(x, y, t)$. It is the same for $I_y = I(x, y+1, t) - I(x, y, t)$. What's more, because the rightmost column of and lowest row can not be calculated, so I set them to 0. After all the calculation, I also output them as the result figure, but there is something special, because the value of those two spatial derivatives may be negative or positive, but only positive value has meaning for showing image, so I move the value range into

a positive range, so that all the values are positive. The result figures will be shown in result discussion part. (see Figure 4)

```
%estimate spatial derivatives
e_x = zeros(266,534);
for i=2:266
    for j=1:534
        e_x(i-1,j) = double(filtered_image1(i,j))-double(filtered_image1(i-1,j));
    end
end
figure(3);
imshow(e_x+offset,[0,255]);
title('Spatial derivatives of x');

e_y = zeros(266,534);
for i=1:266
    for j=2:534
        e_y(i,j-1) = double(filtered_image1(i,j))-double(filtered_image1(i,j-1));
    end
end
figure(4);
imshow(e_y+offset,[0,255]);
title('Spatial derivatives of y');
```

Figure 4, spatial derivatives code

Then, this part is the most significant section of this experiment, I have to get normal flow for each pixel. The functions to calculate normal flow are:

$$I_x u + I_y v + I_t = 0$$

$$\Downarrow$$

$$\nabla I^t \boldsymbol{u} = -I_t, \text{ where: } \mathbf{u}=\begin{bmatrix} u \\ v \end{bmatrix}, \nabla I^t = \begin{bmatrix} I_x \\ I_y \end{bmatrix}$$

After I got the normal flow vectors, I overlay them on the original gray level toy image, which will be shown in result discussion section. (see figure 5)

```
%calculate normal flow
u=zeros(265,533);
v=zeros(265,533);

for i=1:265
    for j=1:533
        temp = e_x(i,j)*e_x(i,j)+e_y(i,j)*e_y(i,j);
        u(i,j) = -1.0*double(temporal_gradient(i,j))*e_x(i,j)/temp;
        v(i,j) = -1.0*double(temporal_gradient(i,j))*e_y(i,j)/temp;
    end
end

figure(5);
imshow(image1);
hold on;
quiver(u,v);
title('Flow vector map');
hold off;
```

Figure 5, normal flow code

Finally, I also run my program to image 4 and image 5 in normal_flow_compare.m, which has the same procedure and the result from those two images will be shown in result discussion section.

1.2 Experiment result and discussion

1.Original image (Figure 6) and Gaussian filter image (Figure 7)

Figure 6, original image

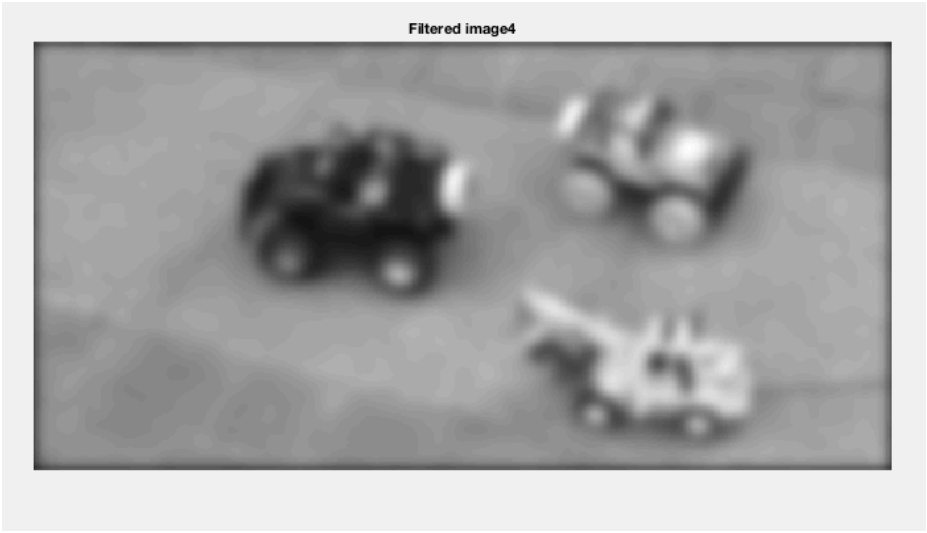

Figure 7, Gaussian filtered image

From the comparison we can see that, Gaussian filtered image becomes blurrier and smoother. In this experiment, the sigma I choose is 4.5, so that the image become blurrier and more suitable for optical flow smooth assumption.

2. Temporal Gradient (Figure 8)

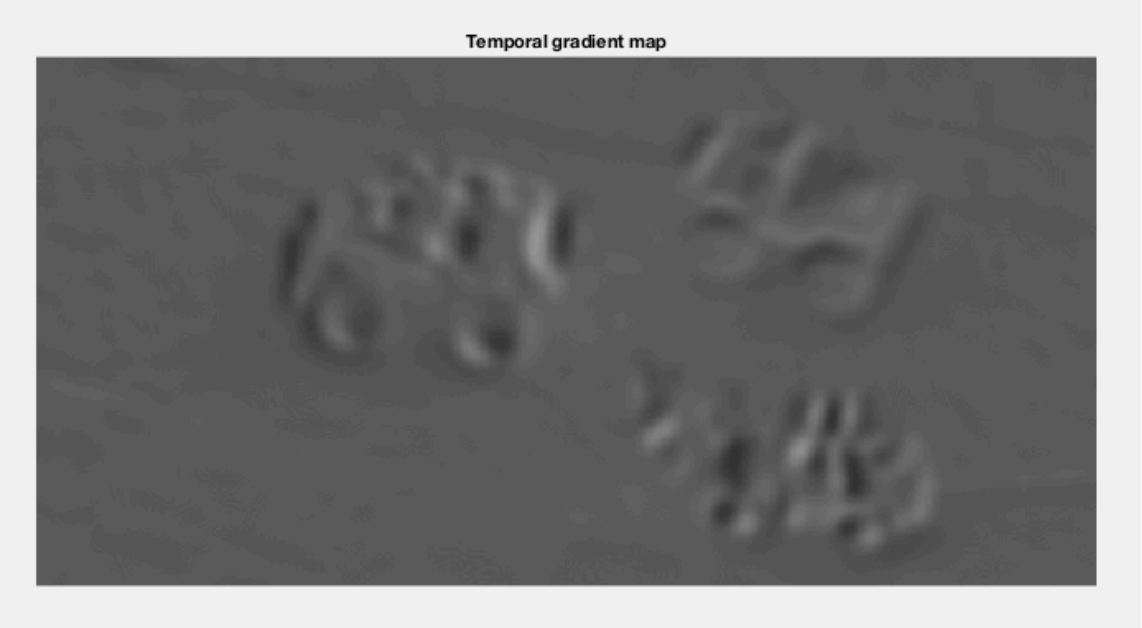

Figure 8, temporal gradient

From this temporal gradient figure, we can easily find out the outline of three toy cars, which means that they move more obviously. This figure have been process by moving to positive value range.

3. Spatial derivatives (see Figure 9 and Figure 10)

Figure 9, spatial derivatives of x

Figure 10, spatial derivatives of y

In those two spatial derivatives figures, the outline of toy cars are not distinct very well. As the temporal gradient, spatial derivatives are all move to positive range too.

4. Normal flow (unfiltered: Figure 11, filtered: Figure 12)

Figure 11, unfiltered normal flow result image

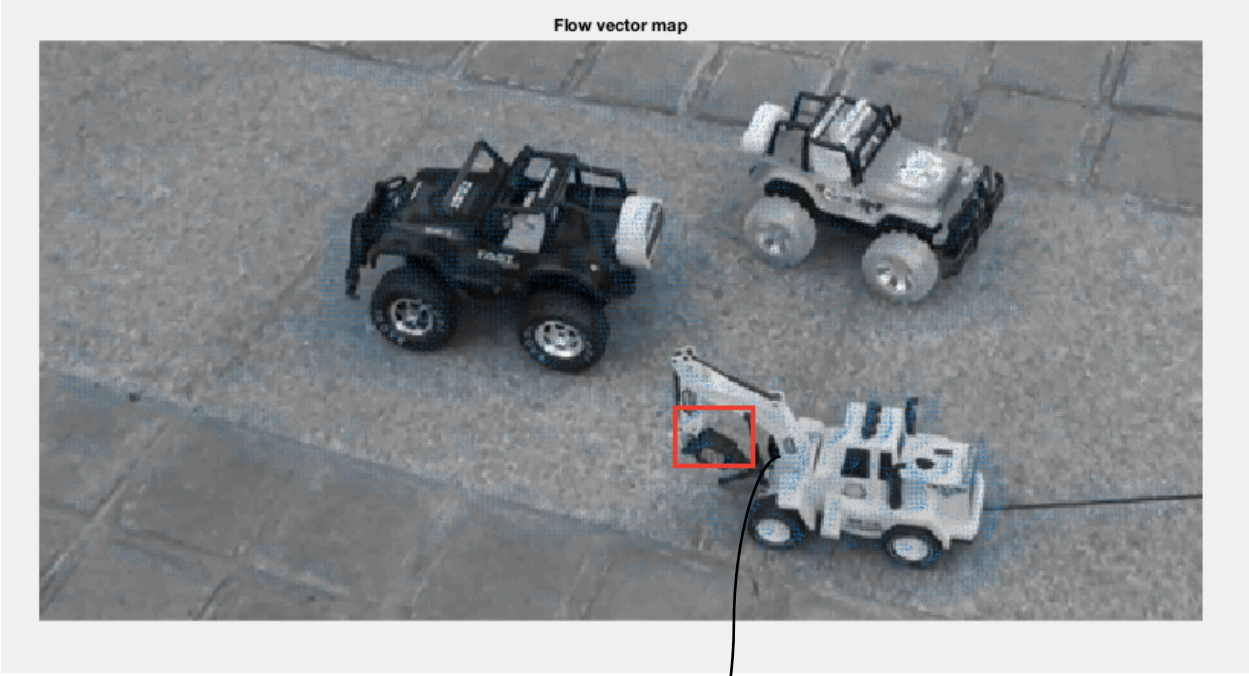

Figure 12, filtered normal flow image

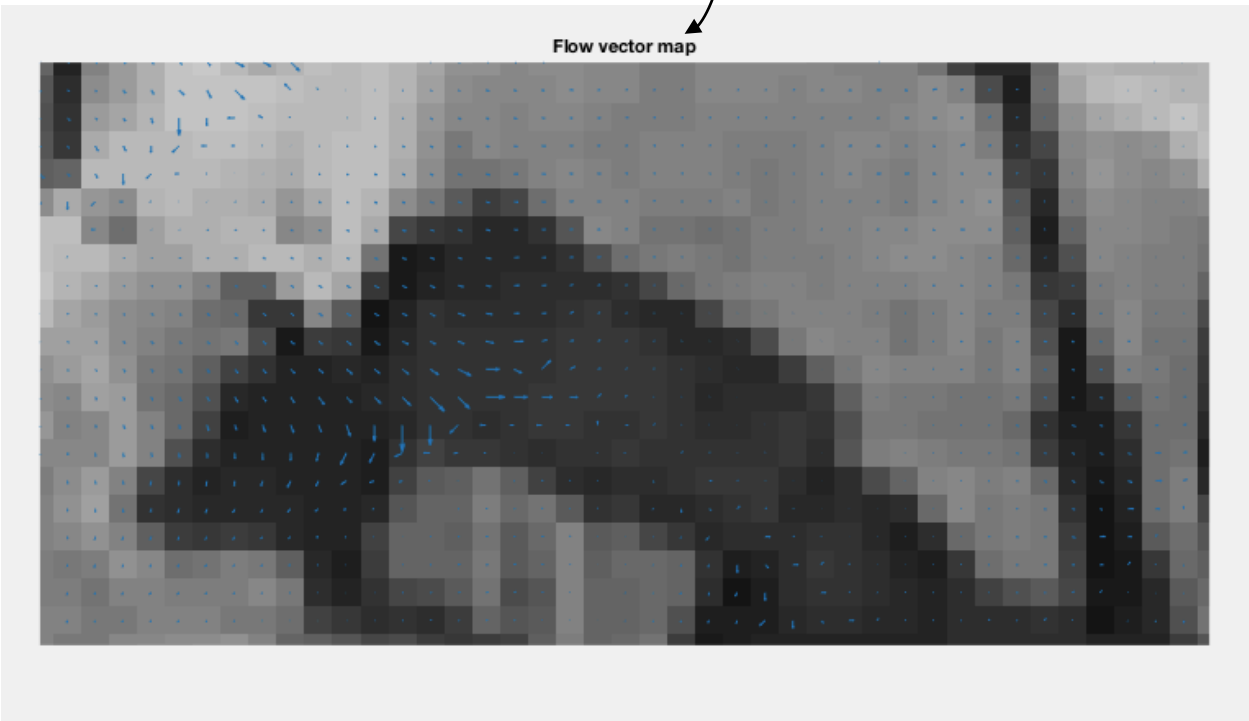

Figure 13, red rectangular area zoom in

From comparing figure 11 and figure 12, we can get that, the image filtered has larger area of normal flow than unfiltered image, which means that Gaussian filter

makes image more smooth, so that optical flow algorithm can be applied on more pixel and get more accurate result.
Figure 13 shows the normal flow of lower right toy car. So we can see that most of the vector point to lower right corner, which is the movement direction of this toy car, so I think this algorithm works well.

5. Result of image 4 and image 5

Figure 14, normal flow of image 4 and image 5

Figure 15, normal flow of image 1 and image 2

By applying the program on image 4 and image 5, the result is almost the same as the result of image 1 and image 2. I think because the speed and direction of those two situation is almost the same, so it is reasonable to get similar results.

Problem 2: Flow calculated over pixel neighborhood

2.1 Experiment process

The process of this experiment is almost the same as problem 1. The difference is in the normal flow vector part. In this experiment, I have to choose 2×2 pixel neighborhood to estimate the **u** matrix. The function to calculate normal flow over pixel neighborhood are the same:

$$\nabla I^t \boldsymbol{u} = -I_t$$

But in this experiment, we have 2×2 measurements to solve **u** matrix, so:

$$\begin{gathered} Au + b = 0 \\ \Downarrow \\ A^T Au = -A^T b \\ \Downarrow \\ \text{So: } u = -(A^T A)^{-1} A^T b \end{gathered}$$

If I assume that, the four pixel I choose is $P_1(x_1, y_1)$, $P_2(x_2, y_2)$, $P_3(x_3, y_3)$ and $P_4(x_4, y_4)$. The position of them like this:

P_1	P_2
P_3	P_4

Then for the three functions above, the columns of A are x and y components of the spatial derivatives of x and y for each of four pixel neighborhood, and the columns of b are temporal gradients of those four pixels. Then I write a matlab program to implement this algorithm. (see Figure 16)

```
%calculate normal flow
v=zeros(266,534,2);

for i=1:265
    for j=1:533
        a=[e_x(i,j),e_y(i,j);
           e_x(i,j+1),e_y(i,j+1);
           e_x(i+1,j),e_y(i+1,j);
           e_x(i+1,j+1),e_y(i+1,j+1);];
        b=[temporal_gradient(i,j);
           temporal_gradient(i,j+1);
           temporal_gradient(i+1,j);
           temporal_gradient(i+1,j+1);];
        temp=-pinv(a'*a)*a'*b;
        v(i,j,1)=temp(1);
        v(i,j,2)=temp(2);
    end
end

figure(1);
imshow(image1);
hold on;
quiver(v(:,:,1),v(:,:,2));
title('Flow on raw image');
hold off;
```

Figure 16, code of normal flow over neighborhood pixels

2.2 Experiment result show and discussion

Situation 1 ------ sigma = 1.0

Figure 17, Normal flow on raw image when sigma = 1.0

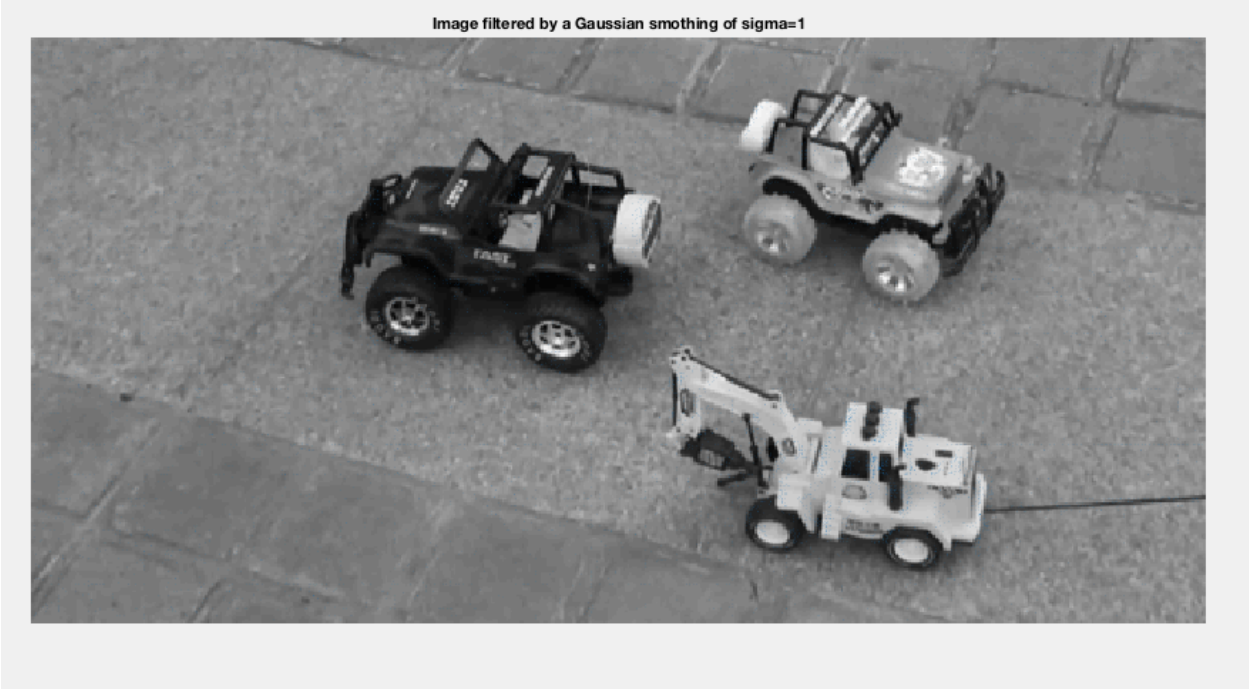

Figure 18, Normal flow on filtered image when sigma = 1.0

Situation 2: sigma = 2.0

Figure 19, Normal flow on raw image when sigma = 2.0

Figure 20, Normal flow on filtered image when sigma = 2.0

From those two images with different sigma values, I can get that: when sigma value unchanged, filtered image can get better results than raw image, for example the flow area is obviously larger than raw image when sigma is 2.0; when we all use filtered image, lager sigma value results in better result, for example, filtered image with sigma 2.0 has larger area of normal flow than filtered image with sigma 1.0. So we may be better choosing some larger sigma value to make optical flow obvious.

Bonus Problem: Apply optical flow to my video sequence pair of images

In this experiment, the process is just like the experiment 1 and I took a video of rolling a round can, and choose two consecutive frame to calculate its optical flow. It is the first time I use matlab to process video file, the format of my video file is mov, matlab gives us a VideoReader function to input this video into matlab. This file can be read as a 4 dimensional matrix, the first three dimension represent the RGB value of each frame, the fourth dimension is used to distinct different frame. The rate of my film is 30 frames per second, so I took a 7 seconds video and choose two consecutive frames which can show obviously but not big movement.

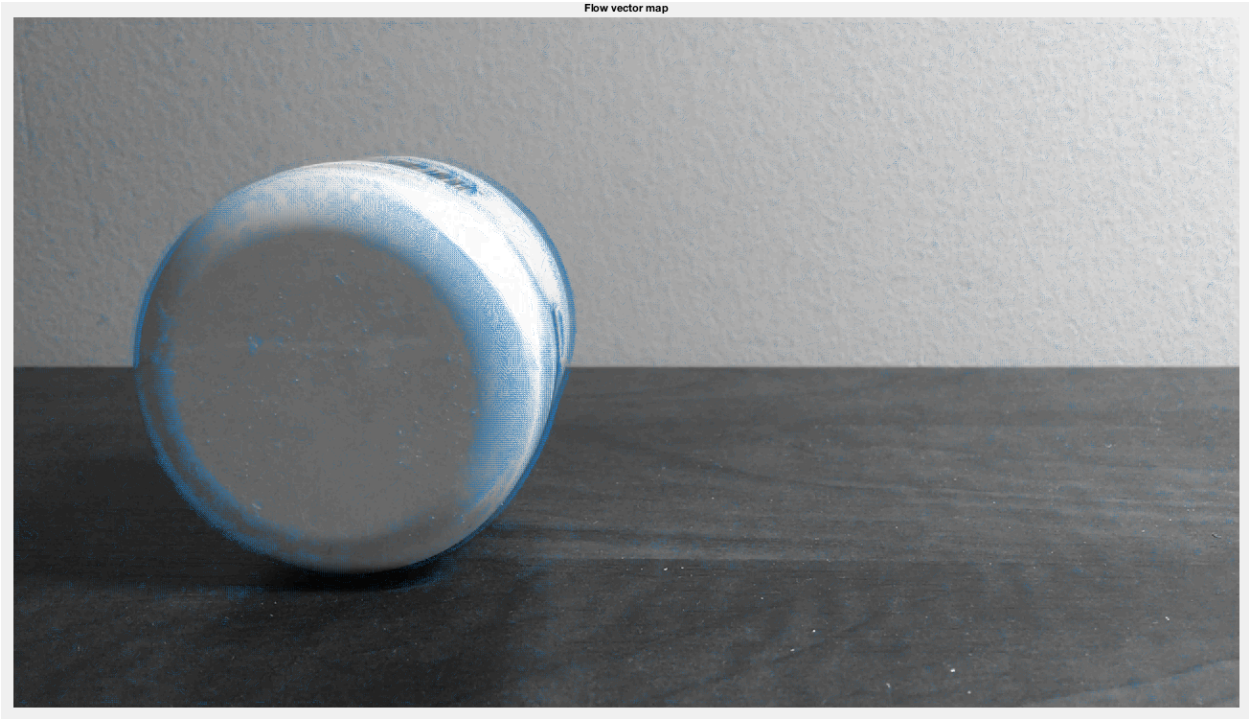

Figure 21, the normal flow of my roll can image

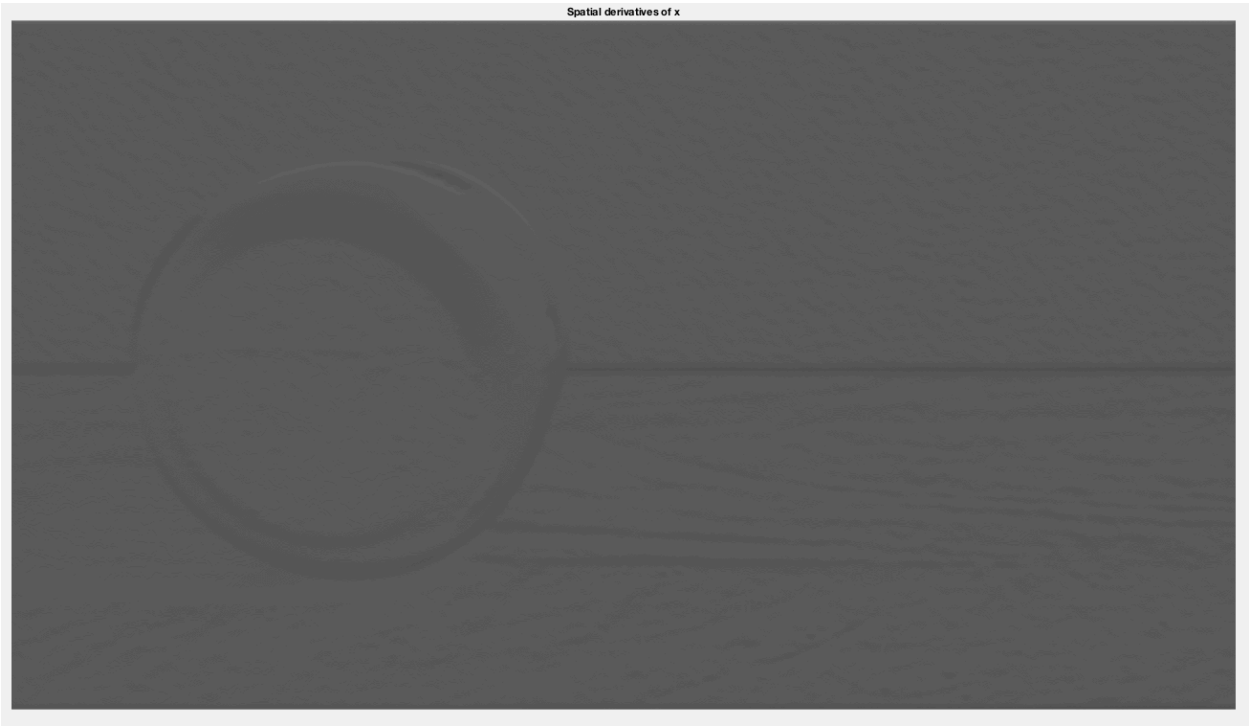

Figure 22, Spatial derivative of x

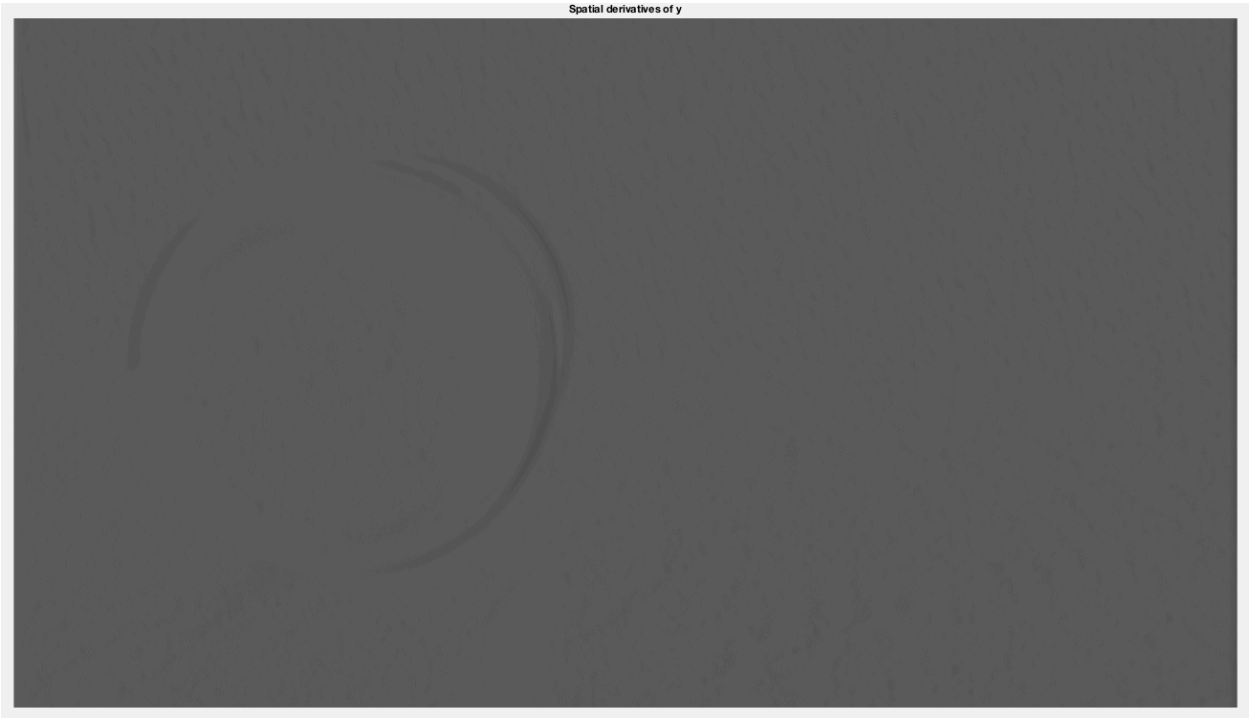

Figure 23, Spatial derivative of y

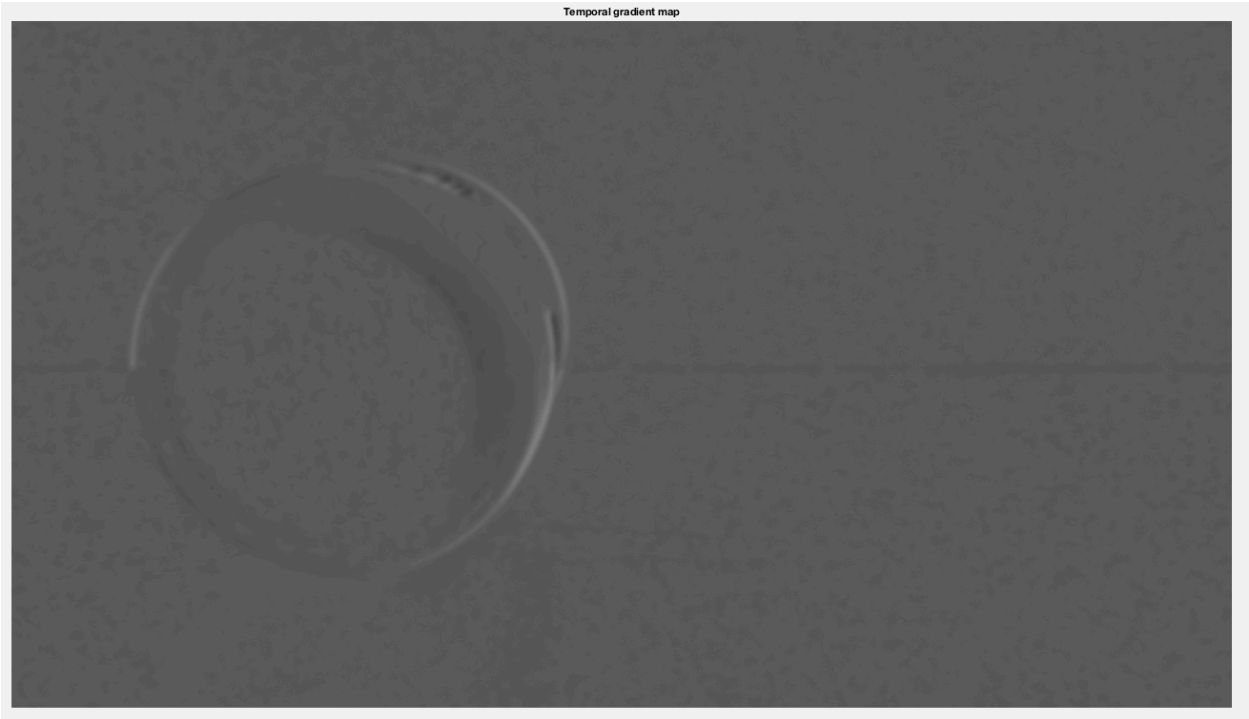

Figure 24, temporal gradient

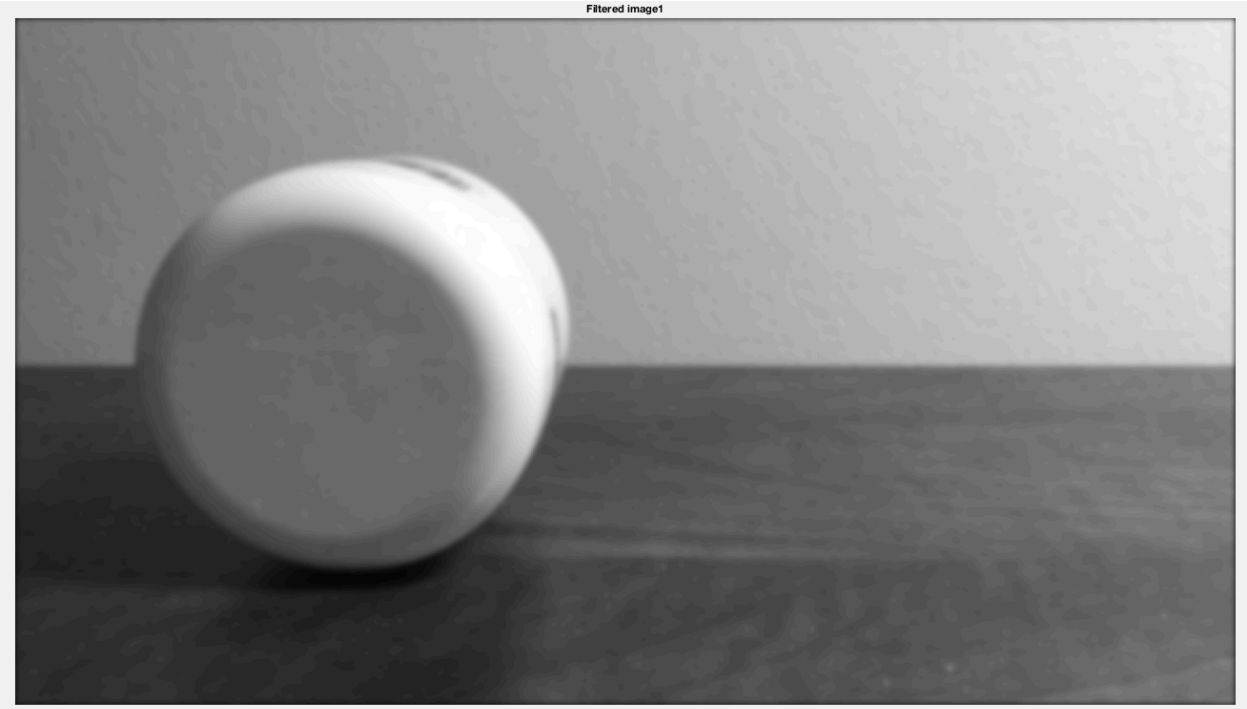

Figure 25, image filtered with sigma = 4.5