Excercise: Optical Flow from 2 images

Stefan Karlsson

November 2012

1 Introduction

First, read and understand sections 1, 2 and 3 of the document 'opticalFlowByMoments.pdf' (skip section 4 for now) in addition to chapter 12 of the book[1]! OBSERVE: unfortunately, this excercise requires a windows version of matlab.

Start by simply typing 'runMe' at the prompt. This will display a synthetic image sequence generated on the fly. You will be working in the script 'runMe.m', which only calls the function 'vidProcessing'. 'vidProcessing' is our main entry point for all video processing in this course[1]. Once its called, the figure will open and display the video together with whatever else information we desire. The code also supports different sources of video. Instead of a synthetic sequence, you can load a video by changing 'movieType' as indicated in 'runMe'. Try the provided 'LipVid.avi' file. If you have a camera you can also use it for input.

Once you have finished viewing the results of the video processing, simply close the window to return back from the function. At that point, the datastructures dx, dy and dt are returned.

Central to this excercise is the 3D gradient of the video sequence. This is calculated automatically inside 'vidProcessing'. In order to view what the 3D gradient looks like, we can show its 3 component images. This can be done by setting the argument 'method' inside of 'runMe' to method = 'gradient'. Do this, and re-run 'runMe.m'.

Are the gradient components as you expect them to be?

1.1 Brightness constancy

A very important assumption to most optical flow algorithms, is the brightness constancy constraint(BCC). In plain words, this means simply that the brightness of a point remains constant from one frame to the next, even though its position will not. A first order description of the BCC is sometimes called the

[1] You are not required to understand the 'vidProcessing' function, but you are encouraged to look through it if you have time over after the completion of the excercise

optical flow constraint equation, and is given as Eq. 12.97 in the book[1]. It can be written as:

$$I_t + v_x I_x + v_y I_y = 0$$

where $\vec{v} = \{v_x, v_y\}$ is the motion we are trying to estimate with optical flow algorithms. We can also write $\vec{v} = -|\vec{v}|\{\cos(\phi), \sin(\phi)\}$, where ϕ is the angular direction of the motion, and write the optical flow constraint equation as:

$$I_t = -|\vec{v}|\left(\cos(\phi)I_x + \sin(\phi)I_y\right)$$

The quantity $(\cos(\phi)I_x + \sin(\phi)I_y)$ is found in the r.h.s, and is sometimes called a 'directional derivative'. It is the rate of change in a particular direction[2], namely ϕ.

Now, run again the script 'runMe', with the same settings as before, observe the gradient component images. Does the optical flow constraint equation seem to hold? When you close the figure, try to do so as the pattern is moving at an angle $\phi = 45^o$ (this is when the pattern is moving **towards the lower right corner**[3] as indicated in figure 1. After you have closed the figure, enter the code:

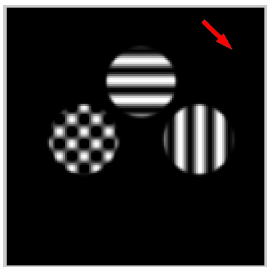

Figure 1: the position of the pattern as it is moving with direction $\phi = 45^o$. This is what you should see when shutting down the figure(time it well). Red arrow shows the motion vector.

```
figure;subplot(1,2,1);imagesc(dt);
colormap gray; axis image; title('dt');

subplot(1,2,2);imagesc(-(dx+dy));
colormap gray; axis image; title('-(dx+dy)');
```

The images should be very similar, explain why.

[2] In fact, I_x and I_y are both directional derivatives, with orthogonal directions (one along the x-axis, the other the y-axis)

[3] remember that matlab has its origin at the top left corner of the figure, and that the y axis is pointing downwards

Now, go to the script 'runMe.m', and change the argument spdFactor = 4. This will make the motion in the sequence 4 times faster. Shutting down the figure at the right time may now prove difficult (depending on how fast your computer is). Go to the function 'vidProcessing.m'. On line 192, there is a 'pause' statement. Temporarily increase the value that is paused (to 0.1 from 0.001, say).

```
% a brief pause lets matlab update figures, and is good for computer
% stability. Change this value to introduce lag-time.
pause(0.001);
```

Now re-run 'runMe.m'. As before, shut down the figure just as the motion is $\phi = 45^o$. After that, run once more the code

```
figure;subplot(1,2,1);imagesc(dt);
colormap gray; axis image; title('dt');

subplot(1,2,2);imagesc(-(dx+dy));
colormap gray; axis image; title('-(dx+dy)');
```

Are the two images still similar under higher motion?

Redo the same experiment once more, but this time change the scale at which the gradient is calculated. The code supports 2 different scales (coarse and fine). Change the argument 'bFineScale' to bFineScale=0, in order to use coarse scale derivatives.

Are the two images 'dt' and '-(dx+dy)' similar now?

1.1.1 take-home message

You should understand that higher motions makes the optical flow constraint equation fail. You should understand that this can be compensated for by using a coarser scale of derivatives. you should also understand that the coarser scale destroys some of the important detail in the image[4].

2 Task: Edge filtering

Your first task is to implement edge detection in the image sequence. This is a common task in video processing, both for motion tasks as well as a range of other computer vision challenges. The function 'DoEdgeStrength.m' is there for this task. The way we will do this edge detection is very simple. Let the 2D gradient of the image be $\nabla_2 I(\vec{x}) = \{I_x(\vec{x}), I_y(\vec{x})\}$, then edges are found by high values of $|\nabla_2 I(\vec{x})|$.

[4]There are many ways of estimating gradients. For optical flow, the optical flow constraint equation is the most important constraint to hold. Testing how well it holds in the fashion described above, tests how well suited your derivative estimation is for optical flow purposes

Your task is to make sure that the function 'DoEdgeStrength.m' correctly returns such a 2D image, of the current frame in the video.

Once you are done with the task, make sure you have some valid data in the variables dx, dy and dt (these are set as you execute runMe), and type

```
edgeIm = DoEdgeStrength(dx,dy);
figure;
imagesc(edgeIm);
colormap gray;axis off;axis image;
```

Is this image what you expected?
In order to view the edge detection in realtime[5] on the video feed, change the argument in the runMe script:

```
Method = 'edge';
```

3 Task: Optical flow by Lucas and Kanade

Optical flow is to be calculated by the function 'DoFlow(dx,dy,dt,method,gradInd)'. Open this function in the editor. All flow methods in this function will be estimated using spectral moments. Make sure you understand the lines regarding moment calculations well.

Using the moments, form the correct structures in the code for implementing the Lukas and Kanade method. If you are unsure about the theory, review the document 'opticalFlowByMoments.pdf', up until section 3.
Once you are done, the configuration for running the LK algorithm is with arguments:

```
method = 'LK';
```

run the LK algorithm on the synthesized sequence. Does the algorithm perform as you would expect? Are there any points in the sequence where performance is better than other places?

Explain in detail the output.

References

[1] J. Bigun. *Vision with Direction.* Springer, Heidelberg, 2006.

[5] this is especially fun if you can get a camera working, and viewing yourself