Excercise: Optical Flow by the 3D structure tensor

Stefan Karlsson

November 2012

1 Introduction

First, read and understand 'opticalFlowByMoments.pdf' in addition to chapter 12 of the book[1]. Make sure you have completed the previous excercise 'Optical Flow from 2 images', before starting this one.

As in the previous excercise, typing `runMe` at the prompt will display a synthetic image sequence. The script `runMe.m` only calls the function `vidProcessing` which 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. When the figure is closed, the datastructures `dx`, `dy`, `dt` **and** the `gradInd` value will be returned. A difference in this excercise, compared to previous, is that we send one additional argument to vidProcessing: `nofTimeSlices`.

`nofTimeSlices` will determine how many frames of the video to use, in order to estimate optical flow. This concerns the summation[2] of the 3D gradient backwards in time for calculating the moments. It does NOT concern how many frames are used for estimating the 3D gradient - we always use two consecutive frames to estimate a 3D gradient, regardless of `nofTimeSlices`.

As before, `dx`, `dy` and `dt` make up the 3D gradient of the video. However, they will not be 2D matrices as they are returned from `vidProcessing`, but 3D. That is to say, `dx` is going to be of some width and height, roughly equal to the width and height of the video(as before), but `dx` will also have a depth. This depth will be equal to `nofTimeSlices`. At any single time in the video, we will have a `dx`, `dy` and `dt`, and the depth will correspond to how far back in time we wish to incorporate information, in order to estimate something at the current time. The most current time frame in `dx`, is indexed by `gradInd`, so that the current x-derivative component image is found as `dx(:,:,gradInd)`

[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

[2]or integration if you prefer

Now, in runMe.m set:

```
nofTimeSlices = 15;
```

Re-run in runMe with synthetic video, and close the figure. You now have gathered dx, dy, and dt that are 126x126x15 in size[3].

Now, investigate how much variation is in the dx components over time by typing:

```
%sums along x and y only:
energyOfDxComps = squeeze(sum(sum(dx.^2)));
%displays the result
figure; stem(energyOfDxComps); xlabel('depth'); ylabel('energy');
```

Can you see that all component images, except for the most current one, has been scaled down in a smooth fashion. The shape is that of half a Gaussian function.

2 Task 1: 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 $|\nabla_2 I(\vec{x})|$.

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

The remarks inside the 'DoEdgeStrength.m' function tell you all that you need to know of the format of dx, and dy. Once you are done with the task, make sure you have some valid data in the variables dx, dy, dt, gradInd (these are set as you execute runMe), and type

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

This should provide exactly the same results as the edge filtering of the previous excercise.
In order to view the edge detection in realtime[4] on the video feed, change the argument in the runMe script:

[3]Verify this easily by typing 'whos' or inspect the workspace variables in your matlab desktop gui

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

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

2.1 Task 1.b: Edge filtering with a twist

Above, you were asked to do edge filtering on the most current frame (indexed by 'gradInd'). Instead, implement edge detection on all frames in the sub-volume, and output the average edge strength[5]. Re-run with 'nofTimeSlices' = 15. This is a filter sometimes used in special effects industry[6], and has applications to numerous computer vision tasks[7].

3 Task 2: Optical flow between 2 images

As before, optical flow is to be calculated by the function 'DoFlow'. Open this function in the editor. All flow methods in this function will be estimated using spectral moments. The spectral moment calculations will now be 3D, so 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'.
Once you are done, the original configuration for running the LK algorithm is with arguments:

```
method = 'LK';
nofTimeSlices = 1;
```

run the LK algorithm on the synthesized sequence. Verify that you get the same results as you did in the previous excercise.

4 Task 3: Optical flow using the 3D structure tensor

The function `DoFlow` should also be able to estimate flow by the structure tensor method. With the structure tensor, we should do filtering along the time axis, not only along x and y. This would call for 3D Gaussian functions(instead of just 2D), and 3D filtering to construct our moments. However, inspecting the code, we see that we are doing only 2D filtering for the construction of moments, and simply summing in the t-direction.

[5]averaging should be done over the temporal component only, so that the output is still a 2D image

[6]this means that viewing yourself in the camera is alot of fun with this one

[7]The edges can be thought of as detected contours of objects. The averaged edges can be seen as holding dynamic information on the object changes

Explain why this is reasonable!
HINT: we do summation along the temporal direction, and we recall there was something special in how the different temporal components in `dx`, `dy`, and `dt` are weighted. We saw this in the introduction of this document, where we calculated `energyOfDxComps`.

Using the moments, form the correct structures in the code for implementing the Structure Tensor(TS) method. If you are unsure about the theory, review the document 'opticalFlowByMoments.pdf'.
Once you are done, a good configuration for running the TS algorithm is with arguments:

```
method = 'TS';
nofTimeSlices = 4;
```

run the 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?

References

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