GHENT UNIVERSITY

Assignment 2: Shot Detection

Design of Multimedia Applications

Group 1
Rian Goossens, Bart Middag & Bart Vetsuypens
1st Master of Science in Computer Science Engineering
Academic year 2014-2015

1. GENERALIZED SHOT DETECTION METHOD

The generalized method was based upon the motion detection method. Instead of looking at color values however, we use edge information extracted from the frames. Both the strength and the direction of an edge are taken into account. The advantage is that objects can be accurately tracked and various types of fades can be detected, because a lot of fades significantly change edge information. The following is the pseudocode for this method:

```
1. Calculate edge information for current frame and next frame.
2. For each block of size blockSize in the current frame:
     a. Sarch for block in next frame within windowSize that matches
        edge information the most.
     b. Calculate difference in edge information of current block and
        best matching block
3. Sum differences calculated in 2a, and rescale this difference in
   range [0,1]
4. If global difference is bigger than threshold, then we have detected
   a new shot.
```

2. SHOT DETECTION PARAMETERS

2.1 Pixel difference

For this method, we have used 2 parameters: a pixel-level threshold and a frame-level threshold.

The *pixel-level threshold* can be any number from 0 to 765. Knowing that a color value can range from 0 to 255, we can define a pixel as significantly different if the sum of all 3 red, green and blue value differences between two consecutive frames is larger than or equal to this threshold.

The frame-level threshold ranges from 0 to 1 and represents the fraction of pixels that have to be significantly different from the previous frame for the frame to be recognized as the start of a new shot. For example, with a value of 0.5, half of the pixels have to have a total colour difference over the pixel-level threshold.

2.2 Motion estimation

We use 3 parameters: a threshold, the block size, the window size.

The *threshold* represents the minimum difference to detect a shot. This can range from 0 to 1: the difference has been scaled to this range for convenience.

The *block size* is the size of each block used. This can range from 1 to 2^n. Preferably, this is a divisor of both the width and the height of the video so no information gets lost.

The *window size* is the pixel-wise maximum distance to look for a matching block in the next frame. This can be any non-negative integer. 0 will produce the same result as using pixel difference. Making the window size too big will cause the window to be too large for the video.

These parameters are the only constants used in the algorithm that affect the outcome when changed. Making the block size bigger makes it harder to find a matching block because more pixels need to be similar. Depending on the type of motion, smaller block sizes could give better results. Making the window size bigger produces better results when motions are faster. With fast motions, objects move a lot between frames – a bigger window size allows us to still track this. The threshold determines when a shot should be detected: if the value is too low, it produces too many false positives. If it's too high, it produces too many false negatives.

2.3 Global histogram

We have used 2 parameters for this: the threshold and the number of bins.

For the *threshold*, we first calculate the histogram of the total frame and subtract from this the histogram of the former frame as described in the assignment.

We divide this D factor (by height times width of the frame) so that we get the number per pixel. There is a mistake in the GUI of the program: it says integer only, but double values are allowed too, and the default value should be 2,0.

The *number of bins* is the amount of divisions pixels are categorized into. It can be a number from 2 to 8, which is used as a power of 2. If we set the parameter to 8, we get a total number of 256 bins which is the maximum.

2.4 Local histogram

The local histogram method uses mostly the same parameters as the global histogram method. However, there is one additional parameter: the *number of splits*.

The *number of splits* defines the amount of times we subdivide both height and width of the frame, so we get a total of this parameter squared number of areas to calculate the DP array with, as described in the assignment. The range is actually from 1 (the same as global histogram) to the minimum of height and width of the frames.

It is worth noting that the *threshold* parameter is different from the global histogram method: we first multiply the D number by 300 and then divide it by height times width of the frame. This is done to give better resolution to the parameter, so that when we fill in an integer we can tune over a broad range instead of over 1 to 3. The default value we use is 450.

2.5 Generalized

As the generalized method is based on the motion estimation method, it uses the same parameters.

3. RESULTS PER SHOT DETECTION METHOD

The following benchmarks were performed on a Lenovo G780 computer with a Intel i5-3230M processor clocked at 3,10GHz and 8GB of RAM. In the following section, we observe the results of the Star Wars video sequence.

3.1 Pixel difference

Parameters		Results			
Pixel Threshold	Frame Threshold	Precision	Recall	F_1-score	Time
100	0.5	39.1%	81.8%	52,9%	00:08,1
160	0.5	50.0%	36.4%	42,1%	00:03,7
220	0.5	60.0%	27.3%	37,5%	00:02,6
500	0.5	0.0%	0.0%	0,0%	00:01,4
220	0.3	35.3%	54.5%	42,9%	00:05,5
220	0.7	33.3%	9.1%	14,3%	00:01,8
160	0.3	25.7%	91.8%	39,1%	00:11,0
160	0.7	50.0%	18.2%	26,7%	00:02,3

For the Star Wars video sequence, the highest F_1-score is reached for a pixel threshold of 100 and a frame threshold of 0.5. We notice some slight fluctuations in the timings resulting from the amount of shots detected: the creation of shots and sending them to another thread takes a small amount of time that becomes visible when performed a lot.

3.2 Motion estimation

Parameters			Results			
Threshold	Block size	Window size	Precision	Recall	F_1-score	Time
0,15	16	6	90,9%	45,5%	60,6%	01:17,0
0,18	16	6	63,6%	53,8%	58,3%	01:15,0
0,25	16	6	27,3%	50,0%	35,3%	01:13,2
0,15	8	6	90,9%	55,6%	69,0%	01:22,2
0,18	8	6	36,4%	50,0%	42,1%	01:19,5
0,15	32	6	81,8%	45,0%	58,1%	01:07,7
0,18	32	6	63,6%	63,6%	63,6%	01:05,6
0,18	32	3	72,7%	44,4%	55,2%	00:20,7
0,2	32	3	45,5%	55,6%	50,0%	00:19,0
0,15	32	12	81,8%	60,0%	69,2%	04:09,3
0,18	32	12	36,4%	50,0%	42,1%	04:06,8

Using motion estimation, we reach optimal results with a threshold of 0.15, a block size of 8 and a window size of 6. There are parameters yielding a slightly higher F_1-score, but looking at the execution times, this result is clearly optimal. The execution time largely depends on the window size. With a larger window size, this method will have to take more pixels into account per pixel in the video.

3.3 Global histogram

Parameters		Results			
Threshold	Number of bins	Precision	Recall	F_1-score	Time

1,6	8	63,6%	41,2%	50,0%	00:05,6
2	8	54,5%	46,2%	50,0%	00:04,3
2,4	8	45,5%	62,5%	52,6%	00:03,2
2	6	45,5%	50,0%	47,6%	00:03,8
2	4	45,5%	50,0%	47,6%	00:03,9
2	2	36,4%	50,0%	42,1%	00:03,3
1,6	6	54,5%	40,0%	46,2%	00:05,3
1,6	4	45,5%	38,5%	41,7%	00:04,3
1,6	2	36,4%	50,0%	42,1%	00:03,4

We notice that for the global histogram method, the results stay very similar for various parameters. In this case, the best parameters were a threshold of 2.4 and 2^8 bins.

3.4 Local histogram

Parameters			Results			
Threshold	Number of bins	Number of splits	Precision	Recall	F_1-score	Time
450	8	35	90,9%	37,0%	52,6%	00:20,6
475	8	35	72,7%	47,1%	57,1%	00:17,2
500	8	35	54,5%	54,5%	54,5%	00:16,3
450	8	25	54,5%	40,0%	46,2%	00:09,3
450	8	30	72,7%	38,1%	50,0%	00:14,6
450	8	40	100,0%	33,3%	50,0%	00:27,8
500	8	25	27,3%	60,0%	37,5%	00:07,0
500	8	30	27,3%	50,0%	35,3%	00:10,5
500	8	40	54,5%	50,0%	52,2%	00:21,2

Here, we used the optimal results from the previous method and tested with various numbers of splits. The optimal result was obtained using a threshold of 475, 2^8 bins and 35 splits. The number of splits also has a clear influence on the execution times.

3.5 Generalized shot detection

Parameters			Results			
Threshold	Block size	Window size	Precision	Recall	F_1-score	Time
0,035	16	3	54,5%	42,9%	48,0%	04:48,9
0,04	16	3	27,3%	50,0%	35,3%	04:45,2
0,03	16	3	81,8%	40,9%	54,5%	04:49,3
0,03	16	2	27,3%	33,3%	30,0%	02:12,8
0,03	16	5	54,5%	54,5%	54,5%	13:53,2
0,03	8	3	54,5%	50,0%	52,2%	05:26,5
0,04	32	3	36,4%	21,1%	26,7%	04:07,3

The optimal result for this method was obtained with a threshold of 0.03, a block size of 16 and a window size of 3. There is another result with the same F_1-score, but the execution times show which result is optimal. It is clear that the window size has a significant influence on the computational complexity.

4. COMPARISON OF DETECTION METHODS

4.1 Precision and recall

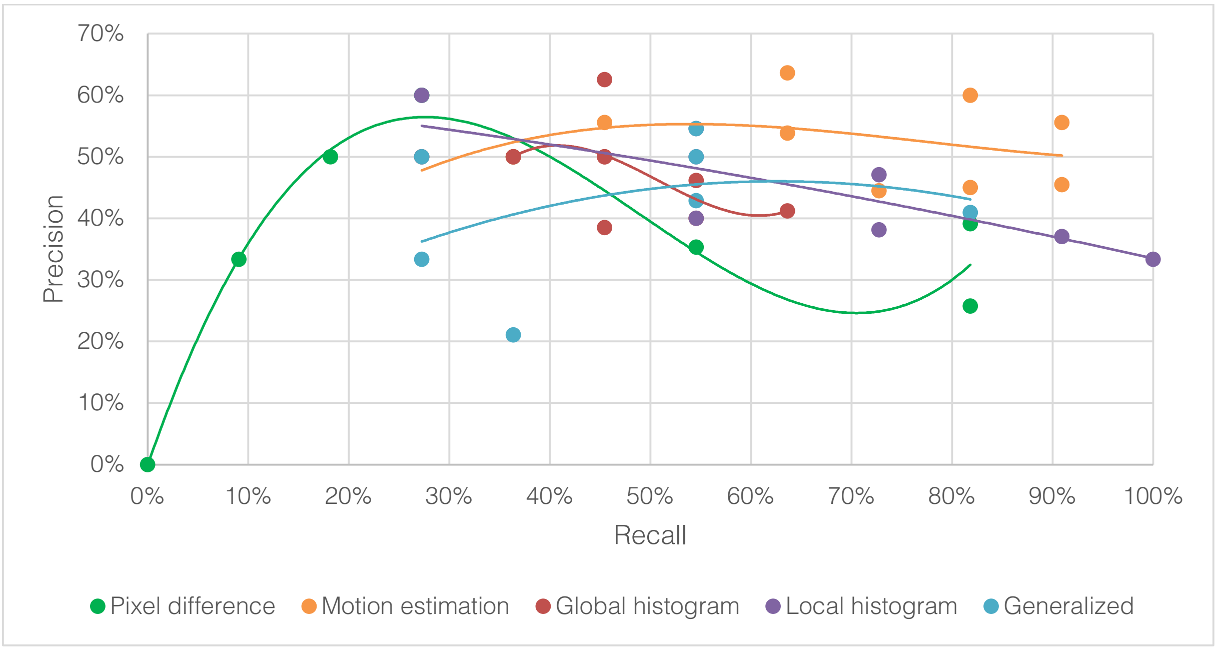
It seems that the more shots a method detects, the more likely it is that they are wrong shots. This explains the seemingly inversely proportional relation between precision and recall values. In terms of F_1-score, the local histogram method shows promising results, but motion estimation is just ahead with its optimal parameters.

4.2 Computational complexity

We see that the global histogram and pixel difference shot detection methods, two most simple algorithms, both take very little time to execute. However, they don't bring the best results either. Motion estimation is the second most computationally complex algorithm we have implemented, because the larger the search window, the more pixels it has to analyse per pixel. The generalized method is based on this and makes the execution even slower due to the extra calculations it has to make.

5. ROC CURVES

This is the ROC curve we obtained for the Star Wars video sequence:



6. CONCLUSION

As we expected from the F_1-scores, we see that motion estimation performs better than all other methods.

7. QUESTIONS

7.1 A clear winner

Among all different shot detection methods, the local histogram and motion estimation methods both show very promising results, with motion estimation coming out on top.

7.2 Improving the clear winner

In terms of precision improvements, we have thought of using edge information instead of colour information for the generalized method, but our implementation is not optimal.

In terms of speed, we could improve the algorithm by coding it for a GPU. For configurations with multiple cores, parallelizing the algorithms would also help to speed up shot detection. Lastly, since DirectShow is not being developed (nor optimized) any further, we might bring some last slight improvements by using the current state-of-the-art video capture technologies.

7.3 The obvious shortcoming

All implemented shot detection methods have one shortcoming in common: there are no optimal parameters for every type of video. The parameters and results we listed were for the Star Wars test video, but the obtained results were very different between the provided test videos. It is not possible to obtain optimal shot detection without first testing various shot detection methods with various parameters. Due to the differences between videos, it is very hard to prevent this.

7.4 Video shot detection for wearable devices

While video footage for films or TV series does not often contain sudden quick movements, a human often makes quick turns. When using Google Glass as an example, even just looking sideways for a split-second can cause a jump in the recorded video that will likely be detected as the start of a new shot.

Even if the implemented shot detection algorithms would somehow be resistant against this, there would still be the problem that footage recorded by a wearable device is normally only one shot per recording. To be able to annotate parts of the video, it would be possible to detect the content of the frames and divide the video into annotatable parts based on the content.

7.5 The importance of metadata for video content

With the increasing amount of video content available on the web, the need to search for videos or specific shots or parts of videos has become apparent. With metadata, this becomes possible: we simply search the metadata.

6.6 Problems

Apart from the scalability of some of the video shot detection methods, the biggest problem is that users would have to manually annotate the shots. It is possible to add feature recognition to automatically annotate the shots, but this was not implemented in our application.

Our method isn't future-proof either: machines still cannot properly browse video content. This can be improved by the addition of an ontology to describe the metadata.