REAL-TIME FACIAL FEATURE POINT DETECTION AND TRACKING IN A VIDEO SEQUENCE

B. B. Abisheva¹, B. M. Baisakov², M. M. Maratov¹

¹International IT University, Faculty of Information technologies
Almaty, Kazakhstan

²Kazakh-British Technical University, Faculty of Information technologies
Almaty, Kazakhstan

Several algorithms for feature detection were compared and the most efficient ones were chosen to detect facial feature points such as eye corners, eyeballs, mouth corners and nostrils. To detect the desired feature points, first face detection was run using Viola-Jones’s algorithm that is based on Haar-like features. Having found a face in an image, the face was searched for eyes, nose and mouth. Having found the necessary facial features, those regions of the image containing them were searched for the specific feature points described above.

For feature tracking several algorithms were compared as well. As a result of the comparison the algorithm of Lucas-Kanade was chosen to be performed on the detected points.

The resultant algorithm detects the feature points to be tracked in the first frame of a video-sequence and then uses these points in the Lucas-Kanade algorithm for tracking. In case more than two points are lost during tracking, feature point detection is run again. This way the algorithm tracks feature points accurately, without lagging in real-time.

Keywords: video sequences, real time tracking for the image points, Lucas-Kanade tracking algorithm

1. Introduction

Object detection and tracking in a video sequence is not a new topic in the area of computer vision, there already are a lot of algorithms proposed to solve this problem. However, a lot of research effort is still being put into this field because of an increased demand in various applications for this technology, such as video surveillance, video-conferencing, information security, access control, video search, etc. Each year more and more efficient and accurate algorithms are being proposed [2]. Mostly these algorithms are either for detecting some feature points in an image, or tracking movement in a video-stream. In this paper we are going to compare several existing algorithms for feature detection in an image, select the most suitable ones, add some modifications and present our own version of an algorithm for facial feature point’s detection. This algorithm will be used to detect feature points of a human face in the first frame of a video-stream. After that detected points will be fed to the modified version of Lucas-Kanade algorithm, which was found to be the most useful for our purposes [1], for tracking.

2. Detecting Facial Feature Points

A feature point is a point that constitutes an “interesting” part of an image. It can be either a corner, or an edge, or a blob etc. For our purposes, facial feature points are the corners of eyes and mouth, eyeballs and nostrils.

First of all, our algorithm starts by detecting a face in an image using the famous Viola and Jones face detection algorithm [3]. This algorithm is based on Haar-like features, which encode the existing oriented contrast between regions in an image. For our purposes we use a cascade of boosted classifiers working with Haar-like features that was already trained to detect front faces. In the computer vision library OpenCV there is a function suitable for this problem.

```cpp
CvSeq* cvHaarDetectObjects(const CvArr* image, CvHaarClassifierCascade* cascade, CvMemStorage* storage, double scaleFactor, int minNeighbors, int flags, CvSize minSize)
```

Parameters:
- image – Image to detect objects in
- cascade – Haar classifier cascade in internal representation
- storage – Memory storage to store the resultant sequence of the object candidate rectangles
scale_factor – The factor by which the search window is scaled between the subsequent scans, 1.1 means increasing window by 10%

min_neighbours – Minimum number (minus 1) of neighbour rectangles that makes up an object. All the groups of a smaller number of rectangles than min neighbours-1 are rejected. If min neighbours is 0, the function does not any grouping at all and returns all the detected candidate rectangles, which may be useful if the user wants to apply a customized grouping procedure

flags – Mode of operation. Currently the only flag that may be specified is CV_HAAR_DO_CANNY_PRUNING. If it is set, the function uses Canny edge detector to reject some image regions that contain too few or too much edges and thus cannot contain the searched object. The particular threshold values are tuned for face detection and in this case the pruning speeds up the processing

min_size – Minimum window size. By default, it is set to the size of samples the classifier has been trained on (for face detection) [10].

Having detected a face in an image, we define areas of the most probable locations of eyes, nose and mouth using existing anthropometric proportional dependencies of a human face [4], [5], [6]. Then these locations are scanned using appropriate Haar classifier cascades to detect the exact positions of facial features. From this point every facial feature is processed separately in order to find the necessary feature points for subsequent tracking (i.e. eye corners, eyeballs, nostrils and mouth corners).

For nostril detection we use the fact that nostrils differ with their darkness from the surrounding area [6]. The area where the nose was detected is searched for the two darkest patches and then the centres of these patches are identified as the nostrils.

For eye areas several algorithms, including Moreira algorithm [4], Canny edge detection and FAST corner detection algorithm [7], [8], were tested. Moreira’s algorithm introduced too much noise if images were not of a good quality and the eye corners could not be found accurately. However slight modification to the algorithm turned helpful in eyeball detection. Firstly non-skin pixels of the eye areas were highlighted by inverting the red component of every pixel. Then the area wasbinarized. The binary image $B_w$ was found using the proposed equation:

$$
B_w = \begin{cases} 
1 & {I' > (I' + Z\sigma)} \\
0, & \text{otherwise}
\end{cases}
$$

where $I'$ and $\sigma$ are the average and the standard deviation of pixels’ intensity of $I$ respectively, and $Z$ is a score that controls the amount of pixels that will be selected to belong to the binary image $B_w$. The parameter $Z$ was calibrated, as advised by the authors, in order to achieve the best results [4]. Then this binarized image was used to define the biggest blob in the middle and its centre was identified as the eyeball.

In order to find the eye corners, Canny edge detection was found unsuitable, as it did not give enough information and not all corners could be found. When trying FAST algorithm [7], [8], it showed accurate results in corner finding. Out of all corners retrieved by the FAST algorithm, the farthest left and farthest right ones were identified as eye corners.

For mouth corner detection we also used FAST algorithm for the same reasons as for the eye corner detection.

Figure 1 and Table 1 shows the results of running the algorithm on 48 2D images of 50 human faces.
Table 1. Results of running the algorithm on 2D face images

<table>
<thead>
<tr>
<th>Feature being detected</th>
<th>Correctly detected</th>
<th>Incorrectly detected</th>
<th>Undetected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face</td>
<td>49</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Eye</td>
<td>96</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Eyeball</td>
<td>91</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Eye corners</td>
<td>70</td>
<td>19</td>
<td>11</td>
</tr>
<tr>
<td>Nose</td>
<td>39</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>Nostrils</td>
<td>67</td>
<td>4</td>
<td>29</td>
</tr>
<tr>
<td>Mouth</td>
<td>48</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Mouth corners</td>
<td>89</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

3. Tracking Feature Points in a Video Sequence

Feature point tracking is performed using the optical flow algorithm. In the computer vision library OpenCV there are several functions that perform this task (void cvCalcOpticalFlow*).

Having compared three most popular algorithms for feature point tracking in a video sequence [1], it was decided that for the given task, a modified version of Lucas-Kanade algorithm [9] was the most suitable.

This algorithm is widely used in computer vision, also known as a local method of differential optical flow. The basic equation of optical flow comprises two unknowns and cannot be resolved unambiguously. Algorithm of Lucas-Kanade avoids ambiguity by using information of neighbouring pixels at each point. The method is based on the assumption that the local neighbourhood of each pixel value has the same optic flow, so you can write down the basic equation of optical flow for all pixels in the neighbourhood and solve the resulting system of equations by the method of least squares.

Lucas-Kanade algorithm is less sensitive to the noise in the images than point wise methods. However, it is purely local and cannot determine the direction of the pixels within the homogeneous regions. Let \( x \) – a feature of the first function \( F \), to find a point \( x + h \) function \( G \), that the difference in area of these points as a minimum. The distance between the surroundings can be written as:

\[
E = \sum \left[ F(x + h) - G(x) \right]^2,
\]

where you can get the value for \( h \):

\[
h = \left[ \sum \left( \frac{dF}{dx} \right)^T \left( G(x) - F(x) \right) \right] \left[ \sum \left( \frac{dF}{dx} \right)^T \left( \frac{dF}{dx} \right) \right]^{-1}.
\]

In OpenCV, this algorithm is implemented by function:

```c
void cvCalcOpticalFlowPyrLK(const CvArr* prev, const CvArr* curr,
CvArr* prevPyr, CvArr* currPyr, const CvPoint2D32f* prevFeatures, CvPoint2D32f* currFeatures, int count,
CvSize winSize, int level, char* status, float* track_error,
CvTermCriteria criteria, int flags)
```

**Parameters:**
- `prev` – First frame, at time t
- `curr` – Second frame, at time \( t + dt \)
- `prevPyr` – Buffer for the pyramid for the first frame. If the pointer is not NULL , the buffer must have a sufficient size to store the pyramid from level 1 to level level; the total size of (image_width+8)*image_height/3 bytes is sufficient
- `currPyr` – Similar to prevPyr , used for the second frame
- `prevFeatures` – Array of points for which the flow needs to be found
- `currFeatures` – Array of 2D points containing the calculated new positions of the input features in the second image
- `count` – Number of feature points
- `winSize` – Size of the search window of each pyramid level
- `level` – Maximal pyramid level number. If 0, pyramids are not used (single level), if 1, two levels are used, etc
status – Array. Every element of the array is set to 1 if the flow for the corresponding feature has been found, 0 otherwise

track_error – Array of double numbers containing the difference between patches around the original and moved points. Optional parameter; can be NULL

criteria – Specifies when the iteration process of finding the flow for each point on each pyramid level should be stopped

flags –

Miscellaneous flags:
CV_LKFLOWPyr_A_READY pyramid for the first frame is precalculated before the call
CV_LKFLOWPyr_B_READY pyramid for the second frame is precalculated before the call
CV_LKFLOW_INITIAL_GUESSES array B contains initial coordinates of features before the function call [9].

The first image in the video sequence is scanned for the feature points and then Lucas-Kanade algorithm is fed the defined points for tracking. In the case of losing two or more feature points, the next frame of the video-sequence is used for detecting the feature points from scratch. This way, the window for Lucas-Kanade algorithm is reduced to produce more efficient results.

Figure 2. Testing the algorithm on a video sequence

4. Conclusions

As a result of testing several algorithms for feature detection, it was found that the best results are achieved using the combination of modified versions of several of them. It was found that FAST corner detection algorithm was the most accurate in finding the corners in an image. In order to increase efficiency the FAST algorithm was used only on previously reduced areas of interest. For nostril and eyeball detection modified versions of Chen [6] and Moreira [4] algorithms were used.
For feature tracking a modified version of Lucas-Kanade algorithm was used. It was given the specific points, which were detected in the first frame only, to track. This way the algorithm tracks the feature points without lagging.

In the end we received an algorithm that detects facial feature points and tracks them in a video sequence in real time.

References