|  |
| --- |
| COSC 6380 dIGITAL iMAGE pROCESSING |
| Iris Detection |
| Hough Transform and Topographic Approaches |
|  |
| **Dat Chu, Michael Fang, Paul Hernandez, Homa Niktab and Danil Safin** |
| **12/14/2009** |

|  |
| --- |
| This report describes the approaches we have tried and the problems we have encountered along the line towards realizing real-time iris detection and tracking |

Literature review

The problems of Iris detection, segmentation and tracking have been studied extensively, and there is a good amount of work that has been done towards this end. Hence we narrowed down our scope for literature review to more recent advancement and only those that do not require special instrument, i.e., IR illumination.

|  |  |  |
| --- | --- | --- |
| Year | Authors | Title |
| 2002 | Kashima et al. | **An Iris Detection Method Using the Hough Transform and Its Evaluation for Facial and Eye Movement** |
| 2003 | Kawaguchi et al. | Iris detection using intensity and edge information |
| 2004 | Cui et al. | An Appearance-Based Method for Iris Detection |
| 2004 | Perez et al. | Real-time Iris Detection on Faces with Coronal Axis Rotation |
| 2005 | Peng et al. | A Robust Algorithm for Eye Detection on Gray Intensity Face without Spectacles |
| 2006 | Niu et al. | 2D Cascated AdaBoost for Eye Localization |
| 2007 | Wang et al. | **Using Geometric Properties of Topographic Manifold to Detect and Track Eyes for Human-Computer Interaction** |
| 2007 | Akashi et al. | Using Genetic Algorithm for Eye Detection and Tracking in Video Sequence |
| 2008 | Chen et al. | A Robust Segmentation Approach to Iris Recognition Based on Video |
| 2009 | Xu et al. | Real Time Detection of Eye Corners and Iris Center from Images Acquired by Usual Camera |

\*The papers that we chose to implement are marked in bold.

Part I: The Hough Transform Approach

**Investigators: Dat Chu and Paul Hernandez**

# Outlines of the method

Given an input video the algorithm of our method is

1. Capture one frame of the video
2. Perform Viola-Jones eye localization
3. Discard the top 40% of the image
4. Perform binary thresholding using Otsu threshold
5. Perform edge detection on the thresholded image with Canny edge detection
6. Perform Hough circle detection and pick the most likely circle
7. Process the next frame (back to 1)

# Advantages & Disadvantages

The initial goal of our project is to create an algorithm that will perform in real-time and segment the iris from the face in the video. With this goal, our Hough transform-based method achieves the following advantages and disadvantages:

## Advantages

* Fast: the algorithm performs at almost real-time (30 frame per second) in Release mode without parallel implementation
* Easy implementation: eye detection and edge detection methods are readily available from OpenCV.

## Disadvantages

* Does not cope well with non-frontal irises (eye become elliptical instead of circle): this is an inherent drawback of our current Hough Transform step. Please see discussion for future work for our thoughts on fixing this matter.

# Discussions of implementation

## Eye detection using Viola-Jones algorithm

The trained Haar classifier included in OpenCV allows for quick and easy detection of eyes. It works well under our indoor lighting video sequences. However, using it directly will include the eye-brows. One can employ a heuristic approach: removing the top 40% of the detected region in order to remove the eye brows. This approach works well for our video sequences. We did not experiment with re-training of the Haar classifier for only the eye.

However, we hypothesize that eye-brow information is useful in detection of the eye. Thus, using an eye detection which are trained with eyes including eye-brows then removing the eye-brow section should be a preferred approach.

## Employing a tracker (e.g. Kalman filter)

We considered adding a Kalman filter step in our algorithm. However, since we want our algorithm to be real-time, adding an extra Kalman filter will slow it down below the real-time threshold. Adding a filter also mean our algorithm is not easily parallelizable as one frame need to be processed prior to the processing of the next.

## Using original images without thresholding

Our original implementation which results was showed during the demo requires quite a bit of parameter tuning which suggests that the approach is not robust to different images and settings. The original method does not take advantage of the skin color typically lighter/different from the eye.

## Using thresholding with Otsu theshold (targeting skin segmentation)

Otsu thresholding is a brute force method to search for the best threshold which minimizes the total intra-class variance. Otsu thresholding minimizes the following term:

Where weights are the probabilities of the two classes separated by the threshold t and sigma2 are the variance of the two classes.

Otsu thresholding removes the requirement of picking the right parameter for our binary thresholding step. However, it doesn't work well with skin under arbitrary lighting.

|  |  |  |
| --- | --- | --- |
|  | Good case | Bad case |
| Thresholded image with Otsu | D:\dc\Dropbox\My Dropbox\Courses\DIP\uhiris\ForReport\Skin segmentation\input_12.png | D:\dc\Dropbox\My Dropbox\Courses\DIP\uhiris\ForReport\Skin segmentation\input_1.png |
| Edge map of thresholded image | D:\dc\Dropbox\My Dropbox\Courses\DIP\uhiris\ForReport\Skin segmentation\edge_12.png | D:\dc\Dropbox\My Dropbox\Courses\DIP\uhiris\ForReport\Skin segmentation\edge_1.png |
| Found Hough circle | D:\dc\Dropbox\My Dropbox\Courses\DIP\uhiris\ForReport\Skin segmentation\iris_12.png | D:\dc\Dropbox\My Dropbox\Courses\DIP\uhiris\ForReport\Skin segmentation\iris_1.png |

One can see that bad results from thresholding using Otsu (i.e. including regions outside of the eye), do not mean the end result will be bad.

## Constraining the results of Hough transform

A typical image will return in several high peaks in the Hough space. It is important that we constrain the accepted peaks given our knowledge of the input. In our method, we employ the following constraints when searching for possible peaks in the Hough space:

1. Only allow circles that are 10 pixel apart between their centers (to suppress spurious circles)
2. Only allow circles which radius is no more than 1/5 of the image height (remove big circles which correspond to the eye lids)

## Using really high quality images

Using a very high quality image, we get a rather interesting (but bad) output (Figure 1). Using a high quality mode in our Logitech Quickcam Orbit AF, we can get a really high resolution of the eye. In the image on the right, the user holds the camera as close to the eye as possible and still get the eye in focused. This creates a problem since the iris will reflect the scene in front of the subject. Such reflection is then detected by Viola-Jones detection algorithm. The algorithm will then give a bad region detection.

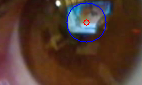


Figure : Eye detection get confused by reflection in the iris

# Future works

## Handling non-frontal irises

Using the average iris for human, then perform matching while considering iris as a patch lying on a perfect sphere.

### Handling higher quality images

We can use the information reflected from the iris for other purposes. (quote that paper) One can use artificial light in order to get information (i.e. artificial known lighting arrangements). Then we can use this information to detect the gaze of the user assuming that the light fixture does not move in space.

## Coping with non-frontal irises

Coping with non-frontal irises require a more flexible model than the Hough circle model. We are planning on investigating the ellipse Hough Transform approach in a future work. Another way is to treat the iris as the overlapped region of two circles and a line. Using this approach one can better segment iris in situation where the upper eye lid obstructs about half of the eye.

Part II: The TOPOGRAPHIC Approach

**Investigators: Michael Fang, Homa Niktab and Danil Safin**

# Outlines of the method

Given an input video the algorithm of our method is

1. Treat the intensity image as a 3D surface and fit a continuous function at each pixel location
2. Topographic labeling based on local gradient and principal curvatures
3. Apply SVM to identify eye-pairs
4. Keep track of the eye locations in the subsequent frame through mutual information matching

# Advantages & Disadvantages

## Advantages

* Does not rely on a face detector
* Handles certain amount of illumination/pose changes
* Fast once in tracking stage

## Disadvantages

* Initialization seems to be slow
* Difficult implementation comparing to the Hough transform method since pretty much everything needs to be written from scratch - no existing libraries that does topographic classification to date.
* Sensitive to parameter/threshold selection

# Discussions of implementation

## Least-square fitting

In order to perform topographic classification on the image we have to compute the local gradient and principal curvatures, which are information only available on a continuous surface. Therefore the first thing we want to do is to fit a continuous surface at each pixel location.



**Figure 2: input image and its gray values**

Assume that the continuous surface takes the form of two variable polynomial of some degree, a surprising fact is that the polynomial coefficients can be computed with a linear filter, independent of the actual data. The Savitzky-Golay filters are used towards this end. We found two implementations of Savitzky-Golay filters in MATLAB [Luo 2005][Krumm 2001] and implemented our own in C++ using Chebyshev polynomials [Meer and Weiss 1990]. Although they generate different filter values but all three give desirable surface fitting results. Figure 2 shows an input image and its corresponding gray-level image. Before computing the polynomial coefficients we first need to smooth the surface otherwise the presence of noise will make the surface fitting error-prone. Figure 3 shows a selected region with its surface before and after the smoothing. In our experiment, the Gaussian smoothing filter is 15-by-15, and=2.5, and is applied twice to get our smoothed result.

|  |  |
| --- | --- |
| C:\workspace\uhiris\ForReport\area.png | C:\workspace\uhiris\ForReport\noisy.png |
| C:\workspace\uhiris\ForReport\smooth.png |

**Figure 3: selected region and its surface before and after Gaussian smoothing**

The Savitzky-Golay filters we used are of size 5 and we assume the polynomials to be up to the second order, hence we have:

 (1)

We thus have:

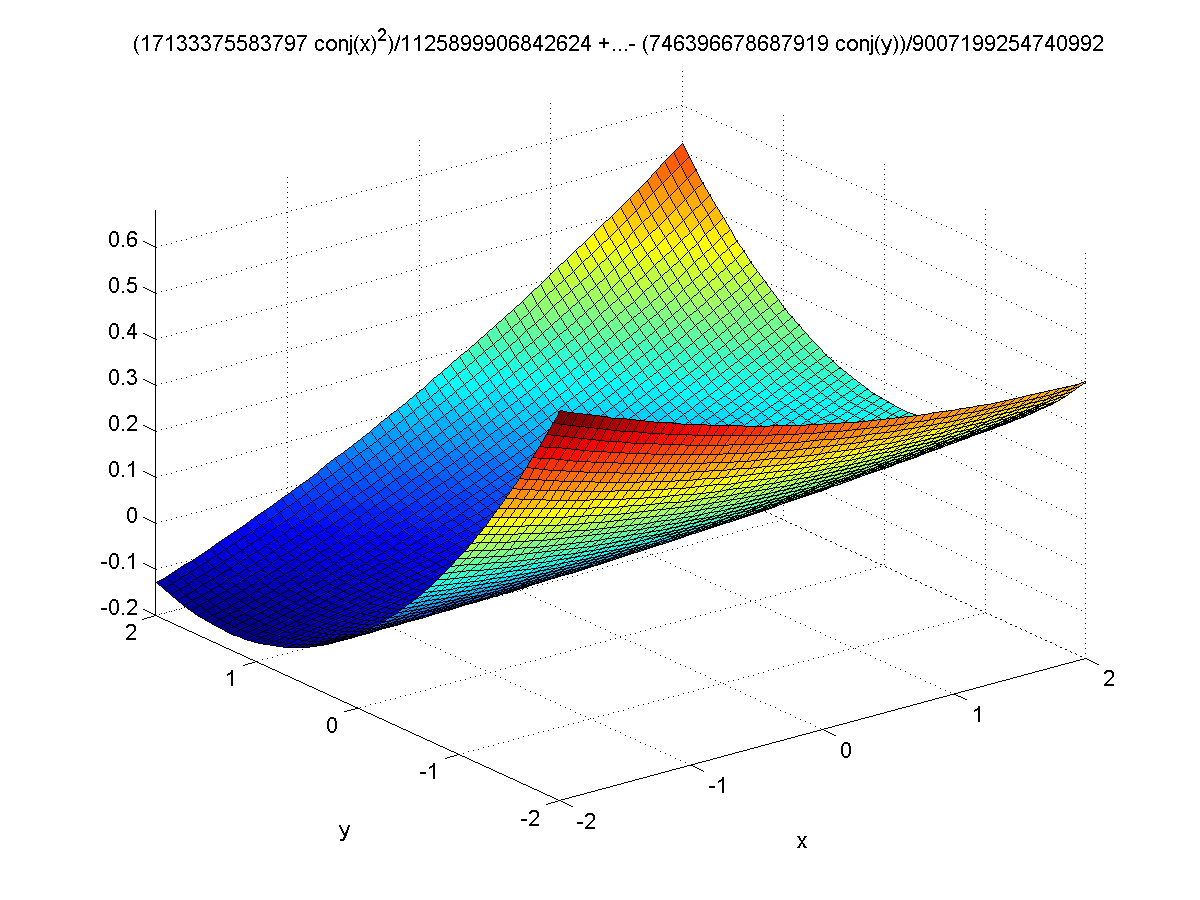
,,,, (2-6)

Since we are evaluating the derivatives at the center of the patch, i.e.,=0,=0, we can analytically reconstruct the continuous surface (shifted by) from the first and second order partial derivatives, which can be obtained by simply convolving the smoothed image with the corresponding filters. The resulting partial derivative images are shown in Figure 4.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C:\Users\airfang\Desktop\f10x.png | C:\Users\airfang\Desktop\f20x.png | C:\Users\airfang\Desktop\f01y.png | C:\Users\airfang\Desktop\f11xy.png | C:\Users\airfang\Desktop\f02y.png |
|  |  |  |  |  |

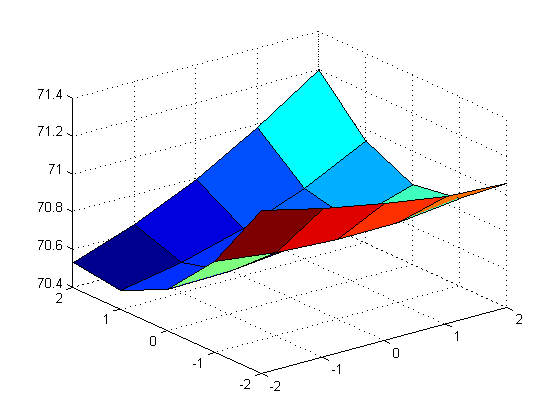
**Figure 4: partial derivatives of the input image**

To verify the correctness of the coefficients, we plot at a random pixel the continuous surface against the discrete data, as shown in Figures 5 and 6.



**Figure 5: continuous surface centered at some pixel**

The fit is pretty good if we ignore the offset along z-axis.



**Figure 6: actual data around the pixel**

## TOPOGRAPHIC CLASSIFICATION

The facial regions of images exhibit different geometric properties when regarded as the topographic manifold. The eye regions in particular have a pit in the center of the surface patch, surrounded by hillsides [Wang et al. 2007]. Here we briefly present the mathematical definitions of each type of the terrain feature. From the continuous surfacecomputed from the previous step, we obtain the Hessian matrix as follows:

 (7)

Applying Eigenvalue decomposition of the Hessian matrix, we get

 , (8)

whereandare the Eigenvalues and,are the orthogonal Eigenvectors. We also defineto be the local gradient:

 (9)

According to [Haralick et al. 1983], the topographic primal sketch includes the following features: *peak*, *pit*, *sloped/curved ridge*, *flat ridge*, *sloped/curved ravine*, *flat ravine*, *saddle*, *flat*, *slope hill*, *convex* *hill*, *concave* *hill* and *saddle* *hill*. [Trier et al. 1995; Wang and Pavlidis 1993] have further broken down *saddle hill* as *concave saddle hill* and *convex saddle hill*, and *saddle* as *ridge saddle* or *ravine saddle* but considered only one type of ridges and ravines. We adopt the latter convention and the classification rules are enumerated in the table below.

|  |  |
| --- | --- |
| Feature | Conditions to be satisfied |
| Peak | * ,, |
| Pit | * ,, |
| Ridge | * ,, * ,, * ,, * ,, |
| Ravine | * ,, * ,, * ,, * ,, |
| Saddle | * ,   Ridge saddle if  Ravine saddle if |
| Flat | * ,, |
| Hillside | * , * , * , * ,,   Slope hill if  Convex hill ifandorand  Concave hill ifandorand  Saddle hill if  Convex saddle hill if  Concave saddle hill if |

With this set of rules in place, we can easily classify each pixel in the smoothed image into one of the 12 categories. But in reality, custom thresholds found empirically have to be used. For instance, the gradient magnitudewas never zero even if there is a pit. Furthermore, the number of false positives increases rapidly as the thresholds for the Eigenvalues approaching zero. We speculate that it may be due to the presence of noise and excessive amount of details. Recall that we already applied Gaussian smoothing twice. More smoothing does help in removing these unwanted aspects but on the other hand the robustness against pose changes is compromised. That is, if the eye is close to the facial boundary, it will be smoothed out (blend into the background).

We also noticed that [Haralick et al. 1983] described the steps for subpixel calculation of gradient:

1. Estimate the surface around each pixel by local least square fitting
2. Use the estimated surface to find the gradient, gradient magnitude, and the Eigenvalues and Eigenvectors of the Hessian at the center of the pixel’s neighborhood (0,0)
3. Search in the direction of the eigenvectors calculated in Step 2 for a zero-crossing of the first directional derivative within the pixel’s area. If the Eigenvalues of the Hessian are equal and nonzero, then search in the Newton direction
4. Recompute the gradient, gradient magnitude, and the values of the second directional derivative extrema at each zero-crossing. Then apply the labeling scheme

Following [Haralick 1984], we tried to accomplish Step 3 and 4 in hope of finding the true 0-magnitude location. The directional derivative ofat the pointis defined as

, (10)

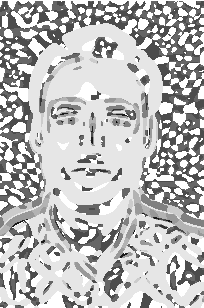
whereis the clockwise angle from the y axis. Substituting (2) and (4) into (10) yields:

 (11)

Since we wish to only consider pointson the line in direction, we can letand, thus we have:

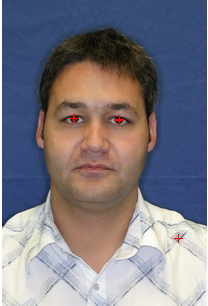
 (12)

The zero crossing then can be obtained by settingand solve for. Discarding values that are outside the pixel’s area, we then have a set of subpixel coordinates. However, after implementing this step, we found that the gradient magnitude computed by these subpixel coordinates is still non-zero and we are actually quite confused with the last step. Unfortunately, [Haralick et al. 1983] does not provide any detail explaining these steps. On the other hand, this subpixel calculation is quite computational demanding. We finally abandoned this idea and just set the thresholds empirically to obtain the label map, as shown in Figure 7. The 12 gray levels represent each of the terrain labels.



**Figure 7: topographic label map**

Possibly due to the empirical thresholding, our label maps look nevertheless quite different from the ones shown in [Wang et al. 2007] and the pits detection is not very reliable. An example of good detection is shown in Figure 8.



**Figure 8: example of good pit detection**

But in a real application scenario, the detector sometimes picks up nose or eye corners instead. In a sense, they are more “pit-like” to the detector than the irises. Background also matters if particular pattern occurs frequently, see Figure 9 for fun. These false positives may be eliminated later on by the SVM but they nonetheless pose considerable computational burden.



**Figure 9: background causing a problem**

## SVM CLASSIFICATION

Given a rectangular image patch of topographic labels, the task is to determine whether the patch contains an eye or not. We used SVMLight implementation of support vector machine classification because it allows adding custom kernels, such as Bhattacharyya kernel used by the paper authors.

### Sample feature vectors / Training data generation

Given a set of points corresponding to pits in the topographic label image, we perform some simple pre-processing. Points that are too close together are combined into one. Points that do not have a pair within certain predefined distance are discarded. For each pair, we extract a rectangular patch oriented at an angle parallel to line passing through both points.

We manually label a set of patches as "eye" (1) or "non-eye" (-1) to create SVM model training/testing data. The size of our training set is 500, and a different set of 150 patches was used for testing. Care was taken to keep two class sizes approximately the same to avoid bias in the model.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Eye: |  |  |
| Non-eye: |  |  |

**Figure 10: Training data: patches of topographic labels**

**Figure 11: Mean and variance of topographic labeled patches does not provide enough discrimination for classification purposes.**

### Bhattacharyya affinity

We started out using the Bhattacharyya affinity measure as a kernel function, as described in the paper. Bhattacharyya affinity is a measure of distance between two probability distributions, defined as. The authors assume terrain labels in a patch are normally distributed, and derive explicit expression for the kernel function in terms of distribution mean and variance. However, the mean and variance of topographic patches we obtained did not provide enough discriminating power to classify the patches, as shown in plot above.

For this reason, we decided to use Bhattacharyya affinity directly on the scaled patch histogram (12 features) in the SVM custom kernel function, without approximating label distribution. We also to tried the RBF kernel with same 12 histogram values as features.

### Parameter optimization

To find optimal SVM model, standard parameter search procedure is used. First, we create a model for each point on exponential grid of parameters (for RBF, gamma = 1e-5, 1e-4 ... 1e+5). The best-performing parameters are selected, and a regular grid is placed around that value. The best model from step 2 is selected. This is repeated for every kernel.

### Resulting models

|  |  |  |
| --- | --- | --- |
|  | **RBF** | **Bhattacharyya** |
| **Accuracy (%)** | 88.62 | 87.43 |
| **True positive (%)** | 89.74 | 92.31 |
| **True negative (%)** | 87.64 | 83.15 |

During deployment, the RBF model performed slightly better than Bhattacharyya model.

## TRACKING

Due to the time limitation, we were not able to implement tracking based on mutual information. We simply search around a neighborhood of a previously detected point for pit locations. If the tracker somehow lost all of the candidates, then it will reinitialize and perform pit detection over the whole frame.

The whole application does not work very well as we desired: accurate and real-time. We speculate that the fundamental problem lies in the classification of topographic features. The empirical set thresholds render the classification error-prone. For different subjects or different image resolutions we may need to change the thresholds accordingly, otherwise either the eyes don’t get picked as pits at all or too many pits may be present.

On top of this instability, our SVM model does not work as we desired. The model by itself offers competitive accuracy but somehow when it is plugged into the system, the performance drop is significant. We barely have some frames with irises correctly classified.

We figure that there is little use going forward without resolving these two issues. These are also the reasons why we did not carry out any quantitative measure of the performance of current system.

## Running the Code

The demo requires a webcam capable of acquiring video streams at VGA resolution. This project also requires OpenCV 2.0. During CMake setup, set the library file and the include directory of SVMLight (they are included in the source code for convenience). Copy RBFmodel.txt and SVMLightLib.dll to where the binaries will be executed. When starting the application, try to look at the webcam and try not to move for a few seconds. The initialization takes some time since the whole image needs to convolve with 5 filters (7 counting the 2 Gaussian smoothers). If eyes are not detected correctly, try to move away from the camera or use hands to block the false responses until a re-initialization is forced, which should take place as soon as the number of candidate points fall below two).

## Final Thoughts

It takes a lot of guesses/speculations to implement a paper. Even though [Wang et al. 2007] has almost specified all the parameter settings, we still run into situations where we don’t fully understand the authors’ intention and the parameters might not work as desired.

OpenCV is a really nice library, especially version 2.0 after they have included C++ wrappers for their structures. But it does have quite a few limitations comparing to MATLAB, which makes it non-ideal for prototyping algorithms. For example, we could not get the new wrapper function calcHist() work. Even the example code in the programming guide does not work. To check the matrix values we have to write routines to dump them to screen or to a file. With MATLAB, error checking for intermediate steps is so simple. Visualizing current matrix values or plotting surfaces is only a matter of one line or a few mouse clicks. Another limitation with OpenCV is that it only takes 32-bit floating-point valued filters, which may lead to some round-off errors.

## References

* Luo 2005: http://www.mathworks.com/matlabcentral/fileexchange/9123-2-d-savitzky-golay-smoothing-and-differentiation-filter
* Krumm 2001: http://research.microsoft.com/en-us/um/people/jckrumm/SavGol/SavGol.htm
* Meer and Weiss 1990: Smoothed Differentiation Filters for Images
* Wang, Yin and Moore 2007: Using Geometric Properties of Topographic Manifold to Detect and Track Eyes for Human-Computer Interaction
* Haralick, Watson and Laffey 1983: The Topographic Primal Sketch
* Trier, Taxt and Jain 1995: Data Capture from Maps Based on Gray Scale Topographic Analysis
* Wang and Pavlidis 1993: Direct Gray-Scale Extraction of Features for Character Recognition
* Haralick 1984: Digital Step Edges from Zero Crossing of Second Directional Derivatives
* T. Joachims, Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning, B. Schölkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999