CS221 Project Report:

Webcam-based Gaze Estimation

Luke Allen (source@stanford.edu) and

Adam Jensen (oojensen@stanford.edu)

Our project uses a laptop webcam to track a user's gaze location on the screen, for use as a user interface. At a high level: we use a variety of image-processing techniques to produce probability estimates for the location of each pupil center. We overlay and multiply these probability matrices (effectively applying direct inference at every pixel) to estimate the most likely pupil position given the probabilities from both eyes. We then use least-squares to train a transformation from pupil position to screen coordinates. Our algorithm can estimate gaze location in real time, within approximately 3cm in favorable conditions.

Roughly Locating Eyes in the Original Image

Our algorithm begins by using the OpenCV library's implementation of the Haar cascade detector [1] to locate a face and the two eyes in a video frame:

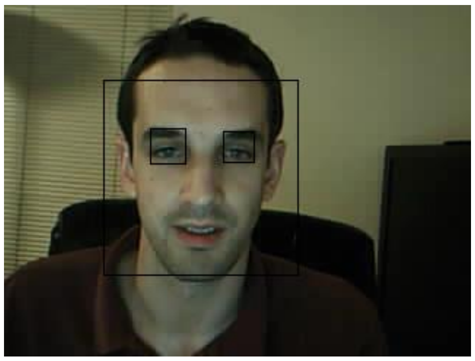


Figure 1: Haar Cascade detection of face and eyes

We use known face geometry to remove spurious eye and face detections, producing very reliable bounding boxes around the user's eyes.

Finding Pupil Centers

To estimate gaze direction, we must find the pupil center within the bounding box for each eye. For this, we have implemented a gradient-based method [2] [3] which uses the fact that image gradients at the border of the iris and pupil tend to point directly away from the center of the pupil. It works as follows:

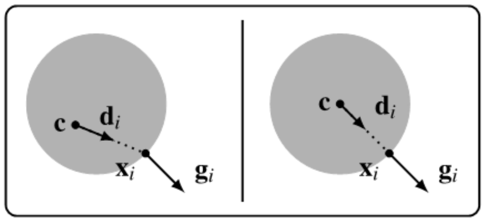


Figure 2: Evaluating candidates for pupil center

If we make a unit vector d_i which points from a given image location to a gradient location, then its squared dot product with the normalized gradient vector g_i, expressed as $(d_i^{\mathrm{T}} g_i)^2$, will be maximized if we are at the pupil center. We do this calculation for each image location and each gradient. The pupil center c^* is the location that maximizes the sum of these dot products:

$$\mathbf{c}^* = \arg\max_{\mathbf{c}} \left\{ \frac{1}{N} \sum_{i=1}^{N} \left(\mathbf{d}_i^T \mathbf{g}_i \right)^2 \right\} ,$$

$$\mathbf{d}_i = \frac{\mathbf{x}_i - \mathbf{c}}{\|\mathbf{x}_i - \mathbf{c}\|_2} , \quad \forall i : \|\mathbf{g}_i\|_2 = 1$$

For computational efficiency and accuracy, several refinements were necessary:

We only do the pupil-center-probability calculation in the dark parts of the image, because the pupil is known to be dark. (To do this, we threshold on pixel brightness, then dilate the image to expand the allowable areas slightly. We must dilate so that our search region includes parts of the pupil where there are bright reflections.)

Eyelashes and shadows produce unwanted gradients, which can produce spurious peaks in the pupil-center probability image. Our desired gradients are fairly strong, so we discard all gradients whose magnitude is below the mean.

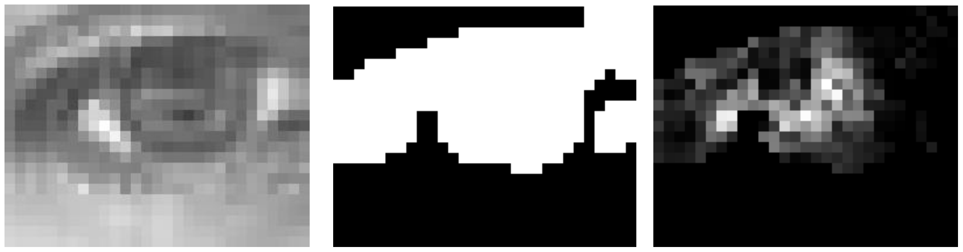


Figure 3: Left to right: A blurred eye image; the areas considered dark enough to be pupil candidates; our estimates of center probability in the regions that were dark (white=highest probability). Note the spurious peak on the left.

Additionally, we enforce the prior knowledge that the pupil is circular: when calculating pupil-center probability at each pixel, we build a histogram of distance to each gradient location in the image. Only gradients near the most-common distance are included. Thus, only the dominant circle of gradient locations around a candidate point contributes to its pupil-center probability. We also discard gradients which are outside the expected min and max iris radius (based on the size of the eye, as found by the Haar cascade detector).

This method finds the center of the iris with fairly high reliability, and acceptable accuracy (usually within 3 pixels). However, it is vulnerable to regions which look similar to a pupil in low-resolution images, such as the shadowed blob on the left of the eye in Figure 3.

Joint Pupil Probability from Two Eyes

Based on discussions with our project mentor (Arun Chaganty), we improved the reliability of our estimate by combining multiple probability estimates. Specifically, we designed a method to combine the estimates from the two eyes: we overlay the probability image from the right eye onto the left eye and multiply the two probabilities. Eye pupils move together, and therefore appear at the same place in the two images. But other (noise-producing) parts of the eye are mirrored about the centerline of the face, and therefore appear at different places in the two images.

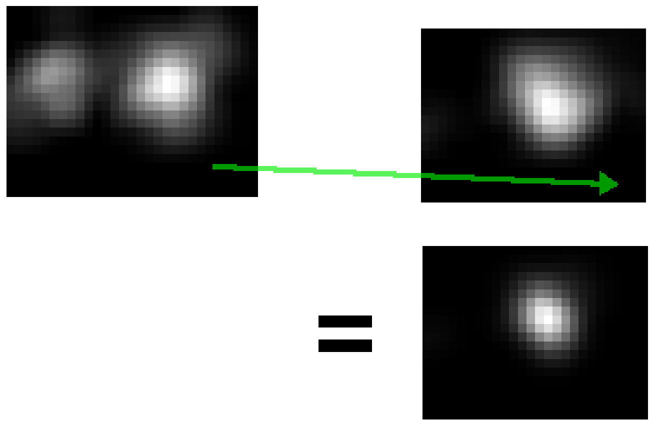


Figure 4: Camera-left eye probabilities are overlaid on camera-right probabilities using the average pupil-to-pupil vector from previous frames. Their product has much less noise.

Combining the two eye estimates was not straightforward. Because the eye bounding boxes are inaccurate, they cannot be used to align the probability images. We used the insight that, if head position is steady, the vector from the left eye's pupil to the right eye's pupil is almost constant as the pupils move around. (Especially when the eyes are focused on a plane, such as a computer monitor.) Therefore, if **any** image region that includes the left eye is shifted by that vector, the left pupil will end up on top of the right pupil. Thus, we can combine the eye probability estimates without requiring any absolute reference points on the face. (We blur both images slightly before the multiplication, so that an error of a few pixels in the pupil-pupil vector still yields a strong peak near the true pupil.)

The pupil-to-pupil vector can be obtained from the noisy estimates of individual pupil positions. We simply applying a very slow moving-average filter to reject noise in the estimate of the vector.

This method proved very effective. If the pupils are visible to the camera, the joint pupil estimate is within the iris nearly 100% of the time.

Reference Point for Eye Direction

Knowing the coordinates of the pupils is not enough, by itself, to find gaze direction. In order to attach meaning to the pupil coordinates, we need to express the pupil coordinates in terms of an offset from some reference point on the face. The Haar cascade bounding boxes for the face and eyes are not steady enough to serve as reference points, even when averaged together. So we experimented heavily with several other possible reference point methods:

Our initial approach was simply to draw a blue dot between the eyes with a dry-erase marker, and apply our single-pupil-finding code to find the center of the blue dot.

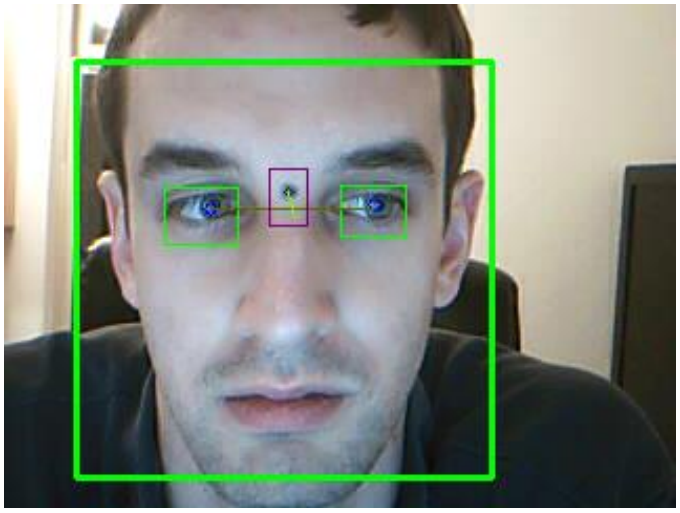


Figure 5: The debugging blue-dot reference point on one of the authors. It is found within 1 or 2 pixels of accuracy, and is useful for testing. Though I blue myself for testing, I would not expect users of a system to do so. Thus, a more natural reference point is required.

The corner (tear duct) of the eye would be good reference point geometrically, because its distance from the camera is the same as the pupil, so head rotation would affect it less than other points. Unfortunately, our implementation of an eye-corner detector was very unreliable. Building an eye-corner detector is nontrivial, because the corner of the eye looks very different depending on whether the test subject is looking left or right. Its appearance also varies greatly with light direction. We also tried to build an eyebrow detector, but since that was also unstable and error prone, we tried to invent a better approach.

A perfect reference point should be as stable as possible, but a key insight (which has not yet appeared in any paper we are aware of) is that the facial reference point does not need to be based on any specific facial feature. With that in mind we implemented a “virtual reference point” tracking algorithm, based on many image keypoint descriptors. At a high level, our virtual reference point is measured relative to many distinct points on the face, and in each frame, each of these points 'votes' on where they expect the virtual reference point to be. We weight the votes according to stability and then take a weighted average over the keypoints.

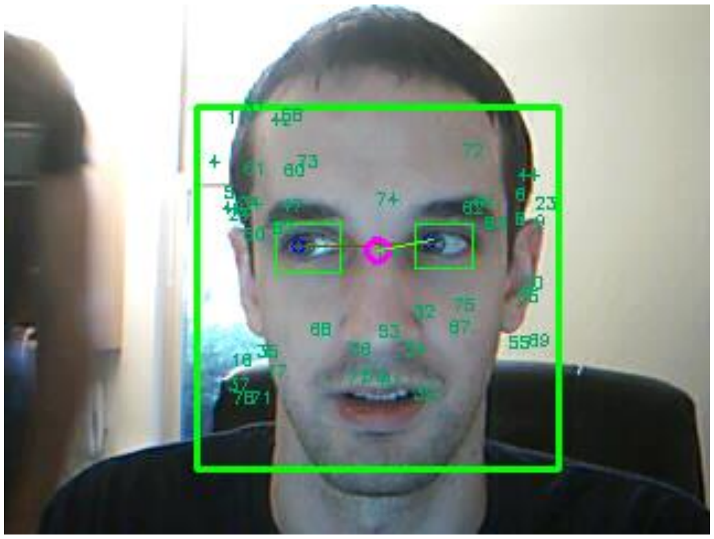


Figure 6: The magenta circle is the virtual reference point. The numbers are unique IDs of SURF keypoints.

A more technical description follows:

We arbitrarily choose a 'virtual reference point' on system startup (we chose to use the average of the two eye bounding boxes). Then, we locate and uniquely identify image “keypoints” using the popular SURF algorithm [4]. We only use keypoints which are inside the face bounding box and outside the eye bounding boxes, to discard the pupils and the background. For each new keypoints, we record the vector from each keypoint to our virtual reference point (this vector will be immutable for the rest of the session). In each frame after that, if the same keypoint appears again, we compute its “vote” for the position of the virtual reference point by taking the keypoint’s instantaneous position plus its saved immutable vector. The votes for the virtual reference point are combined in a weighted average. Each keypoint’s vote is weighted by the number of consecutive frames that we've found that keypoint. (This is a simple way to ensure that the virtual reference is based on stable keypoints.)

This method of averaging the *keypoint position + immutable vector*, rather than directly averaging the keypoint positions, is important because keypoints appear and disappear due to noise and minor head movements. This would cause a direct average of keypoint positions to jump around. Our method produces a very steady virtual reference point even while keypoints are appearing and disappearing.

Aside: For matching keypoints between frames, we use the standard image-descriptor matching guidelines pioneered by David Lowe [5]. *At each keypoint (distinctive blobs or corners* *in the image) a descriptor vector is computed which characterizes the appearance of that image region. The keypoint descriptors from the current frame are compared to all previously-seen descriptors using Euclidean distance. Two SURF descriptors 'a' and 'b' are considered to match if a (from set A, the current frame) is closer to b (from set B, the history) than to the next-closest keypoint in B, by some margin. This is also applied the other way around (by requiring bijective matching in this way, we reduce false positives).*

Training the transformation from eye to screen coordinates

With a steady reference point, we can reliably find the eye gaze direction relative to the reference point. We must next convert gaze direction to a pixel position onscreen. We used the Pygame library to produce a data-gathering tool in which the user looks at the mouse cursor and clicks, and the algorithm records the crosshair location and the current estimate of pupil offset to produce one data point for the training set.

After taking several such data points, we use the Numpy library's least-squares solver to produce a transformation matrix which relates pupil offset (x,y) to screen position (*pixelX*, *pixelY*). We also input quadratic features, because the relationship between eye offset and screen location is not entirely linear (as the eye looks more to the side, small movements of the eye correspond to larger movements on the screen). So our input feature vector is $[x, y, 1, x^2, y^2, xy]$. We use least squares to train a 2 x 6 matrix H such that

$$feature * H = (pixelX, pixelY)$$

Unfortunately, the least squares fit is strongly degraded if there are outliers in training data (such as when the user's eyes were closed and the pupil position is incorrect). Therefore, we use the RANSAC (Random Sample Consensus) algorithm to reject outliers. RANSAC works by choosing a random small subset of the training data, training an H matrix, and using only the training data that is well-described by H to refine H. Many such H candidates are computed (800 in our case), and the best-performing one is chosen.

We retrain the algorithm after each click by the user. After approximately 10 to 20 clicks, our algorithm can estimate gaze along the top of the screen very well. When looking at the very bottom of the screen, a user's pupils are usually not visible to a webcam, so the algorithm fails. We therefore ignore the very bottom region of the screen in the testing below.

Measurement of Results

In adequate lighting conditions, the system performs quite well, as shown in the plot below.

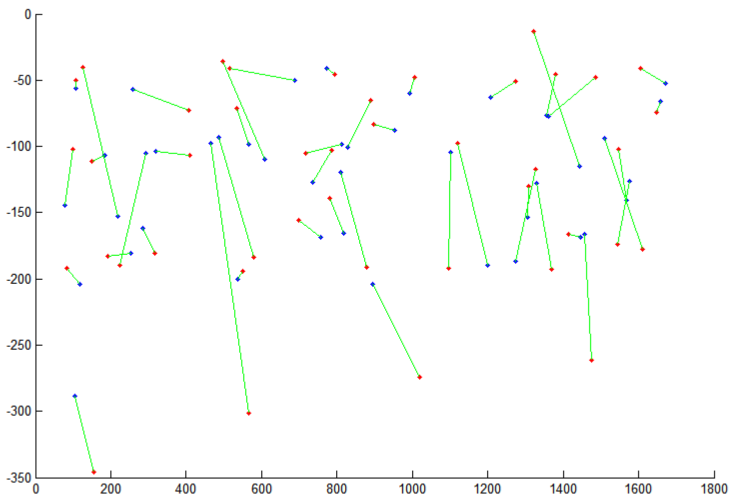
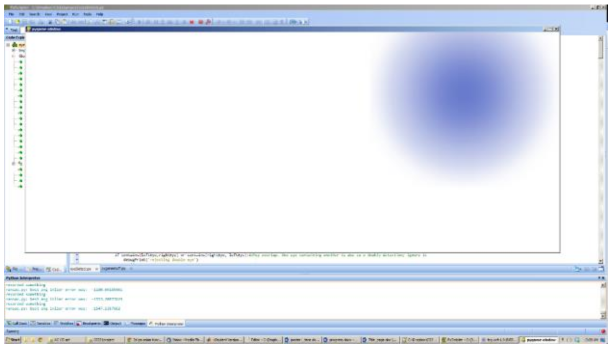


Figure 7: Red indicates true click position, and blue indicates the system's predicted eye gaze location at the time of that click. The prediction is made by transforming pupil offset with the H matrix discussed previously.

In a test of the system with 43 clicks using a webcam with resolution 640 x 480, the median error between predicted and actual click position was 69 pixels, and the mean was 79 pixels. Most of the error was in the vertical direction as evident from Figure 7. This is because the system still has trouble locating the exact center of the pupil vertically. The horizontal median error was only 28 pixels.

With our test setup, the overall mean error of 79 pixels corresponds to 2.8 degrees of angular error in estimating gaze direction. Commercial services which use webcams claim accuracies of 2 to 5 degrees. So, our system seems to be comparable in quality to commercial eye-tracking systems. After training, our system can predict gaze location quite well in real time:



Future Work

The biggest opportunity for improvement is to allow head movement without retraining. (Neither our algorithm nor commercial systems can handle head movement.) One possibility for this is to estimate head position and apply geometry. Side-to-side head movement is detected by the current system, and could be used to modify the model in real time. Distance is not directly sensed by webcams, but we believe the length of our pupil-to-pupil vector is a very promising signal for estimating distance changes.

Head orientation changes would be harder to correct for, as head rotation produces nonlinear changes in the reference point's position relative to the pupils. One option might be to use a Haar cascade to detect the user's nose and infer head orientation, and then apply either machine learning or facial geometry to correct for head orientation's effect on the reference point.

Another promising possibility for user interfaces, which could be used in combination with geometry, is to continuously train the system by assuming the eye is likely to look at salient features on the screen [6] or at mouse click positions during normal computer use. Such a method could provide very large amounts of (noisy) calibration data without requiring the user to manually train the system.

A demo video of our pupil tracking was submitted in our "more.zip" file, as "demo.avi." The green line indicates the final pupil offset.

References

[1] V. a. Jones, "Robust Real-time Object Detection," 2001.

[2] V. a. Gevers, "Accurate Eye Center Location and Tracking Using Isophote Curvature," *IEEE Conference on Computer Vision and Pattern Recognition,* 2008.

[3] F. T. a. E. Barth, "Accurate Eye Centre Localization by Means of Gradients," [Online]. Available: http://www.inb.uni-luebeck.de/publikationen/pdfs/TiBa11b.pdf .

[4] A. E. T. T. L. V. G. Herbert Bay, "SURF: Speeded Up Robust Features," *Computer Vision and Image Understanding,* vol. 110, no. 3, 2008.

[5] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," 2004.

[6] J. C. a. Q. Ji, "Probabilistic Gaze Estimation Without Active Personal Calibration," [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5995675.

[7] D. H. a. Q. Ji, "In the Eye of the Beholder: A Survey of Models for Eyes and Gaze," [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5995675.