Detecting and Tracking Eyes By Using Their Physiological Properties, Dynamics, and Appearance

Antonio Haro† Myron Flickner‡ Irfan Essa†

†GVU Center / College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280

‡Computer Vision Enhanced User Interfaces
IBM Almaden Research Center
San Jose, CA 95120

Abstract

Reliable detection and tracking of eyes is an important requirement for attentive user interfaces. In this paper, we present a methodology for detecting eyes robustly in indoor environments in real-time. We exploit the physiological properties and appearance of eyes as well as head/eye motion dynamics. Structured infrared lighting is used to capture the physiological properties of eyes, Kalman trackers are used to model eye/head dynamics, and a probabilistic based appearance model is used to represent eye appearance. By combining three separate modalities, with specific enhancements within each modality, our approach allows eyes to be treated as robust features that can be used for other higher-level processing.

1. Introduction

In this paper, we present methods for tracking and detecting eyes in complex scenes. Robust and reliable eye detection is an important first step towards the development of user interfaces capable of gaze tracking and detecting eye contact. To support concurrent higher-level processing, algorithms for eye detection should be cheap both in cost and computational complexity. For this reason, we have developed an algorithm that runs in real-time on a consumer-end processor using an inexpensive (under $50) black and white camera with structured infrared lighting. Our algorithm does not require any camera calibration and works for all people. We also do not require users to do any pre-registration prior to having their eyes detected.

Our main goal is to detect eyes reliably and in real-time. We establish for a given frame in an incoming video sequence which regions are likely to be eyes as well as a degree of confidence that the region is indeed an eye. In our method, we use the physical properties of pupils along with their dynamics and appearance to extract regions with eyes. The detector gives us a probabilistic measure of eye detection for each region. The probability for each region is weighted with components coming from the appearance and dynamics along with temporal information.

To deal with the dynamics of head/eye movements, we use Kalman trackers with a simple motion model to measure the movement of different candidate regions. A probabilistic appearance based model of the eyes is used to compute statistics of the texture for different regions to aid in our classification. All three processes measuring eye physiology, dynamics, and appearance are merged to achieve robust detection and tracking.

Once we know which regions are likely to be eyes, we undertake higher level processing on these regions. We observe pairs of regions and probabilistically determine which regions are most likely to be faces when paired together. This allows for higher-level processing on the face level instead of the pupil level. We present some higher-level processing applications using this method.

1.1 Previous Work

There is much prior work on finding faces and facial features in complex scenes. Some approaches, like Kothari, *et al.* [3], find eyes in images without finding faces. They observed that the gradient intersection points were good candidate locations for eyes. They removed most false positives by using temporal cues and by finding pairs of candidates that are close given a priori inter-eye distances and have similar numbers of gradient intersections. The physical properties of the eyes are not taken into account nor are dynamics modeled as we propose in our method. In addition, the technique functions on the pixel level so there is no model.

Other approaches like those of [9, 7] find faces first and then process the facial regions to find facial features. Faces are usually chosen to be located first because they occupy more of the image than their features. Our method differs from these approaches in that we use eyes to reliably locate

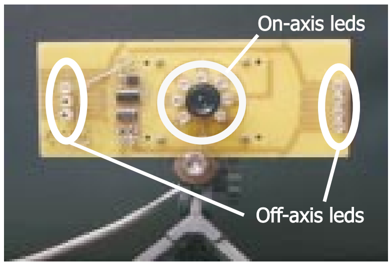

Figure 1. *Our infrared structured lighting camera [6]*

faces.

Scassellati [9] finds faces in cluttered scenes in real-time. Once the faces are located, their system foveates on them and then zooms in on the left eye. However, their system uses ratio templates to find faces and as such is not rotationally invariant. Also, specialized DSP chips are required to achieve real-time performance. Our algorithm does not require any specialized chips to perform tracking in real-time.

Oliver, *et al.* [7] utilize blobs and Kalman tracking to find and track faces as well as facial features in real-time. They initially use blobs and color information to create a mixture model to find the faces and then assign trackers to track each face. The system runs in real-time and achieves good results, but due to the fact that color and anthropometric statistics (to find facial features) drive their system, faces might not always be detected if they are of a size and shape that the system was not trained on. This situation could arise if a face is occluded by an object, or if someone is not completely within the field of view of the camera.

Our work has a lot in common with the work of Rasmussen, *et al.* [8]. In both works, trackers are given multiple sources of information to update their estimates. The main difference is that our algorithm runs in real-time as our model is simpler, without any degradation in performance. Also, like in their work, we utilize a probabilistic framework for our tracker's components. This allows us to combine the information from the components and is what makes our algorithm robust. These components can be interchanged with stronger components for possibly better performance. In section 5. we will discuss how the different components' modalities are combined.

2. Pupil Thresholding

Our system utilizes a black and white camera with structured infrared lighting [6] as a first step in pupil detection. The camera utilizes two concentric rings of IR LEDs (Figure 1), one along the camera's axis, and one off-axis to exploit the *red eye effect*. As the two rings are flipped on and off, they generate two images (Figure 2) for a single frame. The image where the inner ring is on, which we will refer to as the *bright image*, has white pupils. The image where the outer ring is on, which we will refer to as the *dark image*, has dark pupils. While the images look very similar except for the difference in eyes, in practice, the difference image that results from subtracting them from each other is quite noisy. The difference image will contain things like eyes, specular reflections from objects and light sources in housings (e.g. fluorescent lights). We use the difference image despite its noisiness as a preprocessing step since it contains valuable information.

There are several things we can do to get rid of the non-eye pixels in the difference image such as thresholding the image. Thresholding the image has its own problems however, as deciding on a threshold is hard. Even when a good one is found, there is no guarantee that the particular threshold will be good for different environments. While adaptive thresholding could be used, it is very computationally expensive especially when done on a consumer-end CPU as it must be done on every frame and is a slow operation. However, if we have a lot of noise in the difference image, we would like to get rid of as much of it as possible before doing any higher level processing. We want to get rid of the pixels that are obviously not eyes, but we do not want to mistakenly miss out on finding any eyes in the scene.

2.1 Fast Adaptive Thresholding

We have developed an algorithm for doing fast adaptive thresholding which behaves like adaptive thresholding but is much cheaper performance-wise. This algorithm is intended to get rid of as much noise as possible in each incoming frame while not getting rid of any eye pixels mistakenly, so it is conservative. We would rather have excess candidates at this stage than miss a pupil region. The idea behind the algorithm is simple: looking at the histogram for the current frame, we back-integrate the histogram, keeping a certain amount of the brightest pixels (about 1/1000 of the total number of pixels in the frame). The rest of the pixels are then set to black. This algorithm is extremely fast, which is desired as this is the lowest level step in our overall approach and the later steps are more computationally expensive.

Now that we have a smaller group of pixels to work with, we want to group these into candidate regions. Candidate regions are those groups of pixels that we suspect are likely to be pupils. Pixels are put into candidate regions by fitting 16x16 (pixel) windows centered on the brightest pixel in each candidate region. Any pixels that are not sufficiently connected are not turned into candidate regions and are set to black. We also ensure that the regions do not overlap as it is impossible for two pupils to overlap. Since we will be processing regions individually later and collecting statistics on them, removing overlaps at this stage will save us some time in the future.

3. Candidate Region Tracking

Now that we are working with candidate regions instead of just with pixels, we can start to gather some higher level information on these regions. One thing we can do is to track these candidate regions. Tracking allows us to see how each region is behaving over time, which provides additional information. Also, if we track the regions, then we will be able to handle blinks, which would cause significant problems otherwise.

Kalman trackers [12] are assigned to each candidate region. We chose Kalman tracking instead of a more advanced tracking scheme because it can be implemented to update in real-time and provides the necessary functionality we require. Since our algorithm is multi-modal, we can afford to have individually weak sub-parts that working together yield robustness. It is possible to periodically lose track of regions with Kalman tracking, especially if someone is making very sudden head movements because those are not modeled. However, if a tracker goes completely off-track it is removed from the list of active trackers. The region it lost track of can then be tracked again with a new tracker. While statistics that the tracker was gathering are lost when we remove it, that is not a problem because the new tracker eventually acquires the statistics the old tracker possessed. Key to these assumptions are the facts that frames are not dropped and that people cannot suddenly move from one side of the image to the other in one frame. However, as we show later, our algorithm still works even under these conditions.

The trackers are implemented as having four dimensional state vectors: x-position, y-position, velocity in the x direction, and velocity in the y direction. We did not incorporate acceleration information into the state vector as we found it did not improve the performance. The dynamics are modeled simply: we state that the x and y positions of the state vector change by one pixel between frames and that the velocity of the region in the x and y direction changes by two pixels between frames. This model works because we assume that people do not perform sudden, jerky movements in real-life. We make several other assumptions if our model does not hold. If a tracker does not have a measurement near to it, which in our algorithm is a candidate region from the newest frame, then it continues going at the same velocity it was going before. If it does not have a measurement for a number of consecutive frames, then it is made inactive and removed from the candidate list. When new trackers are added, they check to see if any other tracker is tracking that region already; if not they start tracking. Also, if trackers collide with each other, we remove one and keep the other. This is done because there should never be two trackers tracking the same region.

We use the Kalman filter's covariance matrix to give us a measure of similarity between a particular region's motion compared to a pupil's motion. The covariance matrix update equation for the Kalman filter is given by:

$$P_k = (I - K_k H) P_k^{'}, \tag{1}$$

where K is the Kalman gain matrix, H is the connection between the state vector and measurement vector, $P_k^{'}$ is the prior estimate of P_k, and P_k is the covariance matrix at time k. To see how well each individual tracker is doing, we can compute:

$$P(\hat{x}_{k+1}) = N(\hat{x}_k, P_k). \tag{2}$$

This equation yields a measure of how well the previous state estimate and the current state estimate correlate to the covariance of the model. If this result is near 0, it means that the tracker is not sure if it is tracking well because the state changed significantly compared to the covariance between the two frames. If this result is near 1, it means that the tracker is confident that it is tracking well. As a result, we can compute:

$$M = 1 - P(\hat{x}_{k+1}). \tag{3}$$

M gives us a sense as to whether the region is moving as a pupil that is attached to a head should move: if it is near 0, it implies that the region is stationary, because the tracker was confident, and hence the region is not moving like an eye. If it is near 1, it implies that the region is moving like an eye and that the tracker is not that confident of the tracking. It should be noted that these movement cues are clearly not enough to classify the regions yet; someone could be sitting still, or something that is as bright as an eye in the difference image could be in the scene. Also, it should be noted that the strength of the tracker does not affect the above equation. We talk of tracker confidence, but what we are really comparing are the state estimates in the previous and current frames. Even if we were using a perfect tracker, this equation would still hold because the state must either change from frame to frame, or remain stationary. In both cases, the formulation for M is correct.

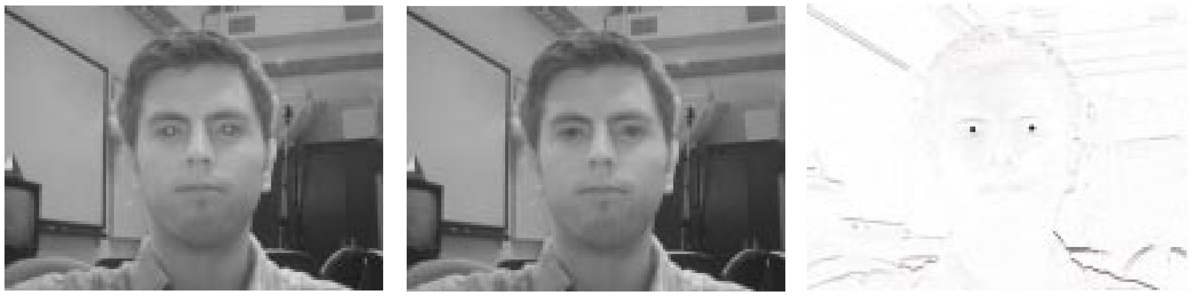

Figure 2. *Left: the bright image, Center: the dark image, Right: Difference Image (contrast enhanced)*

4. Appearance based models for candidate regions

So far, we have not looked at the actual texture belonging to each candidate region. Texture information is taken into account to increase the robustness of the algorithm. We have to do this because something could be as bright as a pupil in the difference image and could move like a pupil, but could in reality just be a moving reflective surface. For example, in Figure 3 we see a set of regions in one frame that are being tracked. Those regions are being tracked because they passed our adaptive thresholding pre-processing and because they are moving in a way that complies with our motion model. However, the majority of the regions that are being tracked are specular reflections off of the glass of water, which we are not interested in.

We perform appearance based matching using principal component analysis (PCA). We create two vector spaces as done in [11]: one for eyes, with 85 training images of pupils from the dark image, and one for non-eyes, with 103 training images of patches that are not eyes from the dark image. The idea is that we will take each candidate region, form a vector from its texture, and project it into both of the spaces. Whichever space the vector is closest to, that will be the space we will classify it as belonging to. This class information is then added to the information being kept for the region.

The problem with PCA itself is that the distances to the spaces are not in any particular scale. This makes it hard to combine this texture information with the tracking information we are already maintaining for each region. This is not good since the ultimate goal is to use this information to help in making a decision as to whether a particular region is an eye or not. Also, if we just use the distances, we have no measure of confidence that something belongs to the eye or non-eye vector space. To alleviate both of these problems, we use probabilistic principal component analysis (PPCA) [10, 5].

4.1 Probabilistic PCA

Probabilistic principal component analysis (PPCA) frames the distances in a PCA vector space probabilistically. PPCA treats all of the training data as points in a probability density with Gaussian distribution. We use PPCA because we want a means to get a probability for a particular region; that is, we want to know the probability that the texture for a particular candidate region belongs to the eye vector space or to the non-eye vector space. If t is the region we are interested in classifying, we want to calculate $P(t)$ for both the eye vector space and for the non-eye vector space. The maximum of these probabilities yields the probability density that the region is most likely a part of. Bishop, *et al.*provide the full derivation [10] for $P(t)$ by using the fact that PCA is a special case of factor analysis. Here we will present a condensed version of their derivation.

Factor analysis is similar to PCA in that N dimensional data is reduced to D dimensional data with $D \ll N$. When dimensionality is reduced with factor analysis, one ends up with:

$$t = Wx + \mu + \varepsilon. \tag{4}$$

This means for an N dimensional t, we can reduce it to a D dimensional vector x, which represents the latent variables. W are the factor loadings, μ is the mean of the training set, and ε is the error, modeled as $N(0, \Psi)$, where Ψ is diagonal. The model for t is also assumed to be normal $N(\mu, C)$ where $C = \Psi + WW^T$. In factor analysis, if μ and C are calculated, then the probability of t belonging to a particular density is given by $P(t) = N(\mu, C)$.

If the factor loadings were indeed the principal components, one would be able to see the similarities to PCA. However, the loadings are not guaranteed to be the principal components so we must calculate them. It is possible to calculate them because in PCA the data is assumed to have a systematic component, an error term for each variable, and common variance σ^2. If $\Psi = \sigma^2 I$ and

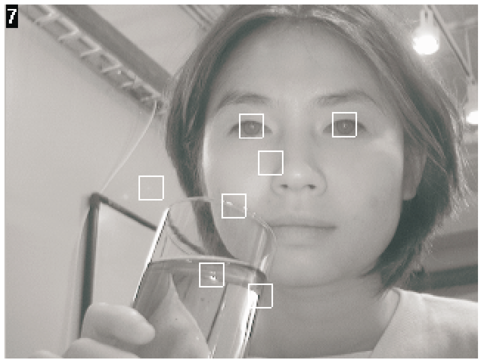

Figure 3. *Tracking all candidates*

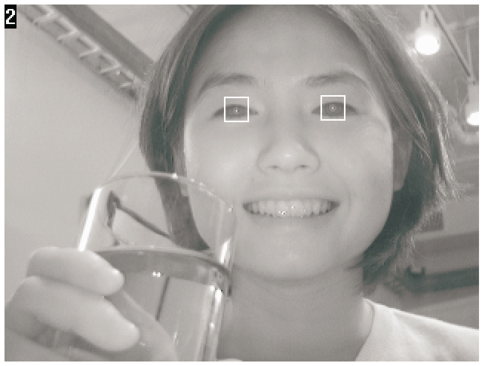

Figure 4. *Classified results*

$S = WW^T + \sigma^2 I$, where S is the covariance of the training data, then PCA can be treated as factor analysis. Therefore, $W = E\sqrt{(\Sigma - \sigma^2 I)}R$, where E are the eigenvectors of the training data, Σ is a diagonal matrix with the eigenvalues, and R is an arbitrary rotation matrix (since we are concerned with WW^T, which eliminates the R). Because we are concerned with maximum likelihood, σ^2 ends up being the average of the lost variance or the average of the eigenvalues of the unused eigenvectors:

$$\sigma^2 = \frac{1}{N-D} \sum_{i=D+1}^{N} \lambda_i. \qquad (5)$$

All of these equations allow us to calculate $P(t)$ given PCA information for a set of training data. For a given candidate region, the probability that it is an eye or not an eye can now be determined. Since these probability densities are in high dimensional space, the probabilities end up being very small, so they are normalized to be in the interval between 0 and 1.0.

5. Classifying Candidate Regions

We use PPCA to give us probabilities of candidate regions being eyes or non-eyes. This is still not enough as the probabilities are not temporally stable and they depend on the regions being centered closely around the pupils. Since the region positions are being driven by the results from the Kalman trackers, they might not necessarily be exactly over eyes at a given time. Moreover, if classification is done just for one frame, no temporal information is used. This is undesirable because the tracker might have drifted off slightly and would be misclassified as a result. Clearly, we would also like to incorporate the movement information that we are calculating for each frame coming from the Kalman tracker's covariance matrix.

For a particular candidate region t, we combine all of the modalities in the following equation which is a weighted probability of all of the statistics we have at time i:

$$P_i(t) = \alpha P_{eye}(t) + \beta P_{noteye}(t) + \gamma M + P_{i-1}(t), \qquad (6)$$

where α, β, and γ vary on the confidence of the results from the PPCA components and the Kalman tracker respectively, M (see section 3) is a measure of whether the region is moving like an eye, and $P_{i-1}(t)$ is the previous weighted probability of the particular region and has an exponential dropoff. All regions have all of these statistics initialized to 0 in the first frame. The motivation of this equation is that if the trackers are somewhat off-center and the probabilities that a region is an eye or non-eye are close, γ will be increased and α and β decreased. Conversely, if the movement information is not very helpful, we rely more on the PPCA component. Temporal information is also included so that single instances of misclassification from the PPCA components do not bias the final classification. Once $P_i(t)$ is calculated, if it is $>$ 0.5, we classify the region as an eye for this frame; if it is ≤ 0.5 we do not.

We chose to compute probabilities for our different modalities so that we could combine them together at this stage easily. If we had not used probabilities in our formulations up to this point, we would have had to rely on heuristics to combine the modalities to make a classification decision.

Figure 4 shows a frame from live video of a user doing some movements in front of the camera. Notice that the eyes are the only things that are classified; even the specular reflections from the glass do not fool the system. While the appearance of the glass is indeed (at some orientations and scales) similar to a pupil due to specular reflections, it is not classified as such because the movement statistics that have been gathered for the region do not match the motion model and the appearance is not consistent over time with that of an eye.

6. Results of Eye Tracking

Our method yields a robust pupil detector. Pupils are found very easily, and false positives are minimized. While false positives occur very rarely, as a result of incorporating temporal information into our classifier, they always decay out quickly (within five frames or less). False positives occur in the current implementation of the system when something reflects light back like an eye, moves slowly like an eye, and actually looks like an eye. For a false positive to occur, all three of these conditions must occur, which is rare. In order for the algorithm to misclassify a region, these three conditions would have to hold over a large number of frames.

Our implementation runs very fast. On a single processor Pentium II, the system runs at about 25 frames per second at an interlaced resolution of 640x480 (320x240 each for the bright image and the dark image). On a dual processor Pentium II, the system runs at about 29 frames per second at the same resolution. This is a result of having our system largely parallelized to take advantage of multiple processors and to fully exploit the processing pipeline.

Several experiments were performed to test the reliability of the pupil regions that are detected and tracked. For detailed evaluation, we recorded two sequences of 30 sec. of video at 30fps. The first sequence had slow head movements with small out of plane rotations. The second had fast head movements with large out of plane rotations. The sequences were chosen to see how the tracking and classification error for Kalman tracking with the adaptive thresholding preprocessing were affected by head speed compared to our method. For each frame in each sequence, the eye positions were manually determined for comparison. Both sequences consisted of one user sitting in front of the camera. The results of these experiments can be seen in tables 1 to 4.

RMS error (900 frames)	**Kalman tracking**	**Weighted prob. (Our method)**
Left eye x	3.06888	2.13689
Left eye y	1.51138	1.49021
Right eye x	2.52898	2.02244
Right eye y	1.46873	1.42833

Table 1. *Seq. 1: Slow and small head movements*

RMS error (900 frames)	**Kalman tracking**	**Weighted prob. (Our method)**
Left eye x	17.56796	13.53245
Left eye y	10.27857	6.10000
Right eye x	17.31363	17.89213
Right eye y	12.78411	5.75922

Table 2. *Seq. 2: Fast and large head movements*

Detected eyes (900 frames)	**Kalman tracking**	**Weighted prob. (Our method)**
None detected in	0 frames	0 frames
Average detected	3.95540	2.10813

Table 3. *Seq. 1: Slow and small head movements*

Detected eyes (900 frames)	**Kalman tracking**	**Weighted prob. (Our method)**
None detected in	0 frames	81 frames
Average detected	4.89608	1.80335

Table 4. *Seq. 2: Fast and large head movements*

In table 1, the RMS error is only slightly better with our method. The reason for this is that for small head movements, the trackers can track fairly well. However, in the faster sequence, shown in table 2, the RMS error is significantly smaller with our method when compared to Kalman tracking with adaptive thresholding. Our method is 0.5 worse than just Kalman tracking in the case of the right eye's x value, but this is most likely due to unsteady hands when selecting the eyes for ground truth.

Also of note are tables 3 and 4, which show that when using Kalman tracking with adaptive thresholding there are about double the amount of detected regions compared to the expected number of eyes (2) in the image. Our method on average detected close to the correct number of regions. However, it did not detect eyes in the fast test case for a small portion of the frames (81/900), but this is due to its inability to exploit temporal information since the head movements are very jerky.

In looking at all of the tables together, we see that our method tracks with a higher accuracy than Kalman tracking with adaptive thresholding and also yields more meaningful regions. Our method consistently finds the right number of pupils in the scene, and as such, is a better foundation for higher level processing.

7. Higher-level Processes

In addition to the above results we developed methods to add higher level processing that rely on our eye tracking methods.

Figure 5. *Facial pairing training data*

Pairing Candidate Regions: While having a list of regions that have high probabilities of being pupils is valuable, having information on how to pair these regions into faces is more so as pairing the pupils off into faces is the first step in any other higher level processing. Since our pupil list is reliable, we can pair pupils very reliably as well.

We use a facial appearance model to classify faces. PPCA is used by creating a vector space of a set of the upper quarter of 165 faces at a low resolution (20x11). Each face training sample consists of the bounding rectangle around the left and right eyes with a small amount of space above and below the eyes as shown in Figure 5. For every pair of pupils in our list of candidates at each frame we figure out all possible pairings such that a priori information on interocular distances is satisfied. We then find the affine warp to normalize the face candidate's texture region so that the left and right pupil positions line up with the left and right pupil positions of our training data.

Once this is done, we can project the candidate face's texture into the vector space to calculate a probability that the pairing constitutes a face. We find the maximum a posteriori (MAP) set of pairings for our set of pupil candidate regions for each frame. The pairings that result are very reliable even for users with their faces close together or at different head orientations. Mis-pairings do not occur if only one eye is visible; single eyes and non-eyes are left unpaired. Figure 6 shows an example of the results of our pairing algorithm. As a result of this pairing step, if users are facing the camera, we can very reliably find an arbitrary number of faces in the scene. Otherwise, we try to pair unpaired regions again in the next frame.

Incorporating Eye Blinks: With faces identified in the image, we can now go after even higher level features. One of these which we use to make our tracking stronger is eye blinks. We identify eye blinks by creating an appearance model for closed eyes with PPCA as we did with faces. After we find faces in the scene, we look at the eye candidates for each potential face. For each pair, we compute the probability that each region constitutes a closed or closing eye. This allows us to increase the confidence in those regions that exhibit eye closing/opening appearance as they should be even more likely to be eyes as a result. When

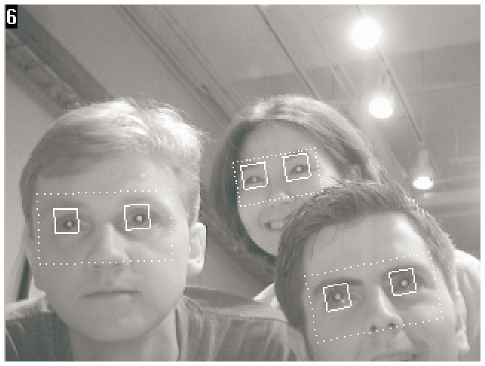

Figure 6. *Pairing found pupils from multiple people*

$P_{eye}(t)$ is computed in equation 6, we take it to be:

$$P_{eye}(t) = max(P_{face}(t,u)_{i-1}, P_{eye}(t), P_{closed}(t)_{i-1}) \tag{7}$$

which is the maximum of the probability that a region, t, is an eye that is open or closing, or was at some prior time a part of a face with an arbitrary region u. In equation 6, we have not done any facial pairing yet so the best we can do is to look at the probability that the region was part of a face in the previous frame, $i-1$. Likewise, at that stage in the algorithm we do not compute blink probabilities since those are only computed for regions that are likely to belong to faces. Therefore, we look at the probability that region t was a closed eye in the previous frame.

Using this formulation, candidate regions that are part of faces or appear to have blinked recently will be less likely to be removed from the candidate list. This provides a mechanism to reliably track an eye region even if the eye is blinking. In our testing, we found this to be a reliable metric, however, we encountered difficulties in creating an adequate and scale invariant appearance model of closed/open eyes.

Testing with an active robot: We also tested the system on a robot head in our lab with the aim of creating an attentive user interface. The robot has a black and white camera with structured infrared lighting in its nose and maintains eye contact with the user closest to the camera. The eye motors are driven by back projecting the pupils that are found with the known camera parameters. This allows us to figure out how to line up the robot's eyes so that they are looking at the pupil positions in the image plane, and hence maintain eye contact. The robot is able to successfully make, establish, and maintain eye contact with different users at distances of up to three feet from it.

8. Discussion & Applications

There are a large number of interesting applications that can use the methods we have outlined here. We can utilize the face positions to count the number of people in a scene, to identify facial expressions, to perform facial recognition, or to estimate head pose for multiple people in the scene, among many other possible applications. We are particularly interested in the last application as we expect to combine our robust face finding method with texture based head tracking to do real-time multiple person pose detection. Pose detection is possible in real-time because only a small number of parameters need to be estimated. Most of the affine parameters for the head positions can be inferred from the orientation and interocular distances of found faces.

With the eyes paired off, we can also do processing to tell if someone is falling asleep by looking at how the rate of their blinking changes as has been shown possible in [1, 2, 4]. Our blink statistics could be used to do this by measuring the delays between blinks or whether blinks have stopped and the eyes are in fact closed.

We have shown that for our system utilizing multiple modalities yielded increased robustness. We are interested in exploring whether multiple simple modalities are always better that single 'strong' components. Another interesting avenue of future investigation is how the performance of our system would be affected by using better appearance based models, such as utilizing support vector machines instead of PPCA.

9. Summary

In this paper we presented a real-time pupil detector and tracker. The system is multi-modal, which adds to its robustness. Since there are plenty of leftover cycles, we can do interesting higher level processing with them, now that eyes are very robust features. With eyes as robust features, we can find faces, which in turn gives us even more information to use in tracking and classifying pupil regions.

The system has been tested on a number of users each with different eye shapes and skin tones. It was able to consistently locate eyes and faces for single and multiple subjects and was able to reliably track these as well.

Being able to find and track eyes reliably in complex scenes has allowed us to do higher level processing, such as maintaining eye contact and finding faces reliably. We foresee many other applications employing this method.

References

[1] M. Eriksson and N. Papanikotopoulos. Eye tracking for detection of driver fatigue. In *IEEE Conference on Intelligent Transportation Systems*, pages 314–319, 1997.

[2] M. Funada, S. Ninomija, S. Suzuki, I. Idogawa, Y. Yazu, and H. Ide. On an image processing of eye blinking to monitor awakening levels of human beings. In *18th Annual International Conference of the IEEE Engineering in Medicine and Biology*, volume 3, pages 966–967, 1996.

[3] R. Kothari and J. Mitchell. Detection of eye locations in unconstrained visual images. In *ICIP96*, page 19A8, 1996.

[4] S. Kumakura. Apparatus for estimating the drowsiness level of a vehicle driver. U.S. patent no. 5786765.

[5] B. Moghaddam and A. Pentland. Probabilistic visual learning for object detection. In *International Conference on Computer Vision*, pages 786–793, 1995.

[6] C.H. Morimoto, D. Koons, A. Amir, and M. Flickner. Pupil detection and tracking using multiple light sources. Technical Report RJ-10117, IBM Almaden Research Center, 1998. http://domino.watson.ibm.com/library/cyberdig.nsf/Home.

[7] N. Oliver, A.P. Pentland, and F. Berard. Lafter: Lips and face real time tracker. In *CVPR97*, pages 123–129, 1997.

[8] C. Rasmussen and G. Hager. Joint probabilistic techniques for tracking multi-part objects. In *CVPR98*, pages 16–21, 1998.

[9] B. Scassellati. Eye finding via face detection for a foveated, active vision system. *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[10] M.E. Tipping and C.M. Bishop. Mixtures of probabilistic principal component analyzers. *Neural Computation*, 11(2):443–482, 1999.

[11] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuro Science*, 3(1):71–86, 1991.

[12] G. Welch and G. Bishop. An introduction to the kalman filter. Technical Report 95-041, University of North Carolina, Department of Computer Science, 1995.