

Carnegie Mellon University

Department of Electrical and Computer Engineering

18-551: Digital Communication and Signal Processing Design

Final Report: Real-time Unobtrusive Sleep Detection

Group 2 Fall 2009

Daniel Burrows (dburrows@andrew.cmu.edu)

Jessica Liao (Jessica@cmu.edu)

Justin Scheiner (emdeefive@gmail.com)

Matthew Wagner (mlw1@andrew.cmu.edu)

1. Problem

A person who experiences short episodes of sleep while driving is sleepy and therefore shouldn't be operating a motor vehicle. The National Highway Traffic Safety Administration estimates that motor vehicle accidents caused by driver fatigue are responsible for 1,550 deaths, 71,000 injuries and \$12.5 billion in monetary damages each year [1]. If drivers are notified each time they experience a short episode of sleep, the number of accidents caused by driver fatigue may decrease. We propose a system that detects when a driver is sleeping. Video of a driver will be analyzed to determine if the driver's eyes are closed in real time.

2. Solution

We simulate the sleepy driver scenario using a web camera. A driver must turn and move his head when operating a car. The Viola-Jones algorithm [5] is used to determine the location of the driver's face in each video frame. This algorithm is implemented on the DSK. When a face is detected, the region containing the face is analyzed to determine if the eyes are open or closed. We cross-correlate an open eye ASEF filter [2] with every frame that contains a face and analyze the correlation output to determine if the eyes are open or closed. The ASEF filter is trained using the CMU PIE database [4]. The cross-correlation of the filter and the video frames is computed on the DSK. The system operates in real time (2 frames per second).

3. Novelty

3.1 Sleepy Head – EYE Can See You

Our project relates closely to Spring 2004 Group 10's project: Sleepy Head – EYE Can See You. This project implemented an open eye detection system on the EVM. A UMACE filter was constructed for each of the five subjects using six to seven images of each subject's open left eye. The test images always contained a centered face on a white background with small in-plane and out-of-plane rotations. Our system runs Viola-Jones face detection before determining the state of the eye. This is the first 18-551 project that implements real time face detection on the DSK. We construct one open eye ASEF filter for all subjects instead of a UMACE filter for each subject. We are the first group to use an ASEF filter. Finally, the input signal is video instead of still images because our system operates in real time.

3.2 FaceL

FaceL is a face recognition and classification system that labels faces in live video. The system was developed at Colorado State University and is explained in [3]. OpenCV is used to detect faces (a computer vision library developed by Intel). The face detection algorithm implemented in OpenCV is based on the Viola and Jones object detection algorithm introduced by [5] and improved by [6]. We implemented the same algorithm on the DSK. FaceL uses an Average of Synthetic Exact Filters (ASEF) filter that was trained using the CMU PIE database to determine

the location of the eyes. We also train our filter on the PIE database. It then uses eye locations to align and scale the face for face recognition. We use one ASEF filter to determine if the eye is open, in addition to locating the eye.

4. Algorithms

4.1 Frame Preprocessing

Frame preprocessing is done on the PC using OpenCV. Each frame is retrieved from the webcam, converted to 8-bit grayscale, cropped from 640 x 480 pixels to 480 x 480 pixels and resized to 64 x 64 pixels using bicubic interpolation. This 64 x 64 pixel image is sent to the DSK for face detection. We thought we should use histogram equalization to increase the global contrast of the frames before face detection because the MATLAB wrapper for OpenCV did so. After examining the OpenCV source code and documentation we don't believe this is necessary. When a face is detected, the 480 x 480 pixel image is cropped and resized to 256 x 256 pixels using bicubic interpolation.

4.2 Viola-Jones Face Detection Algorithm

The algorithm introduced in [5] and improved in [6] and implemented by OpenCV is implemented on the DSK to detect faces. This Viola Jones algorithm uses directional binary filters of varying scale and positions that fit within the detection sub-window. AdaBoost is used to select the best set of directional filters. The five types of directional binary filters used in our implementation are shown in Figure 1. Additional directional filters have been proposed by [6] but determined to not significantly improve face detection. Each filter results in a single value, which is calculated by subtracting the sum of the pixels in the white rectangles from the sum of the pixels in the black rectangles. The sum of pixels in a rectangular region can be computed in four memory accesses, two additions, and one subtraction using the integral image. The integral image contains in each pixel the sum of all pixels above and to the left inclusive. The sub-window is shifted across the image one pixel at a time and can be scaled to detect different sized faces instead of scaling the input image. In our implementation we don't scale the sub-window or image. Our sub-window is fixed at 20 x 20 pixels and the image is fixed at 64 x 64 pixels. This is the case because we assume that the distance between the face and the camera is fixed.

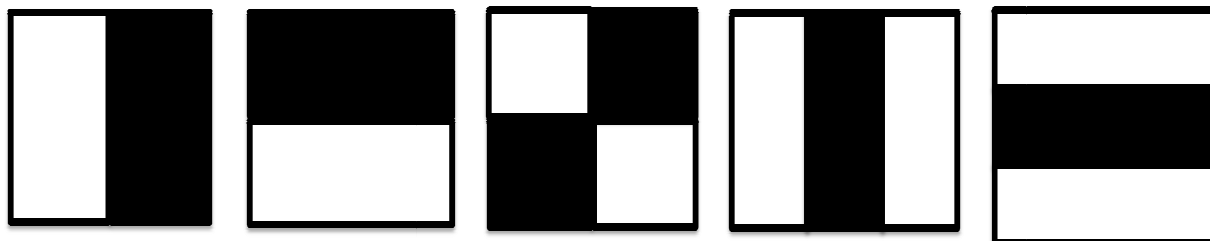


Figure 1: The five types of directional binary filters used in our implementation.

4.2.1 The Integral Image

To compute the integral image, store the sum of the pixels above and to the left of the target pixel including the target pixel. Calculating the sum of pixels in a rectangular area using the integral image is shown in Figure 2. Computing the sum of pixels in any rectangular area can be done in four memory accesses.

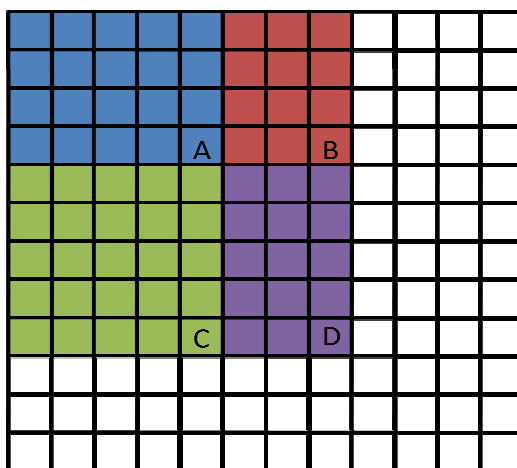


Figure 2: The integral image. Each element is the sum of the elements to the left and above that element inclusive. Element A is the sum of all pixels in the blue region. Element D is the sum of all pixels in the blue, red, green, and purple region. The sum of pixels in the purple region is $D - B - C + A$.

4.2.2 Modified ADABOOST Algorithm

The set of directional filters used to detect a face are chosen using a modified version of ADABOOST. The ADABOOST algorithm used is modified to only select the best filters, polarity, and thresholds by computing the weighted error. The best filters can be selected by analyzing the weighted error produced by each filter. After a filter is selected, the weight of a misclassified training image is kept constant while the weight of a correctly classified training image is decreased. This means that the next filter selected must be more accurate at classifying the training images that were misclassified by the previously selected feature. As the number of filters that have been selected increases, it becomes more important that the remaining candidate features accurately classify the training images that are misclassified by the current set of selected features.

4.2.3 Cascade

The detector is a hierarchical cascade shown in Figure 3. The cascade arranges the filters in stages. At each stage non-face images are rejected because detecting a face is harder than rejecting a non-face. Each stage contains several directional filters. At each stage, if the sub-

window is a non-face, it is discarded immediately and if it is a face, the sub-window is passed onto the next stage in the cascade. The first classifier omits approximately 50% of the non-faces and therefore, there are many false positives in the first stage. This is acceptable since the subsequent stages are expected to reject these false positives using a larger set of directional filters.

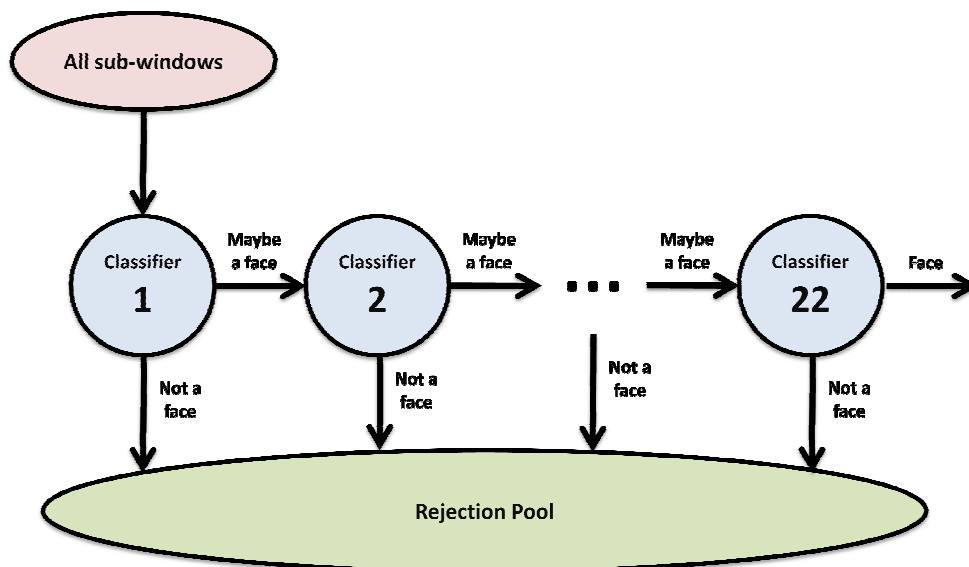


Figure 3: Cascade of classifiers

Training decides the number of stages and which scales, positions and types of directional filters are used in each stage. The training portion of the algorithm will not be implemented on the DSK because it takes on the order of weeks to train a detector. Several frontal face detectors are packaged with OpenCV. OpenCV contains all the information that is needed to implement the detection portion of the algorithm. This face detection algorithm was selected because it is fast.

4.3 Average of Synthetic Exact Filters

After detecting a face, an open eye Average of Synthetic Exact Filters (ASEF) filter is used to determine if the eyes are open or closed. ASEF filter construction is explained in [2] and shown in Figure 4. The desired correlation output for each training image is modeled using a bivariate Gaussian peak centered at the location of the eye with variance of one. An exact synthetic filter is computed for each training image by dividing the Fourier transform of the training image by the Fourier transform of the desired correlation output. To create a filter that generalizes across the entire training set, the exact filters are averaged. The cross-correlation of the ASEF filter and each video frame will be computed on the DSK. We will use the radix 4 FFT code provided by TI to compute the cross-correlation.

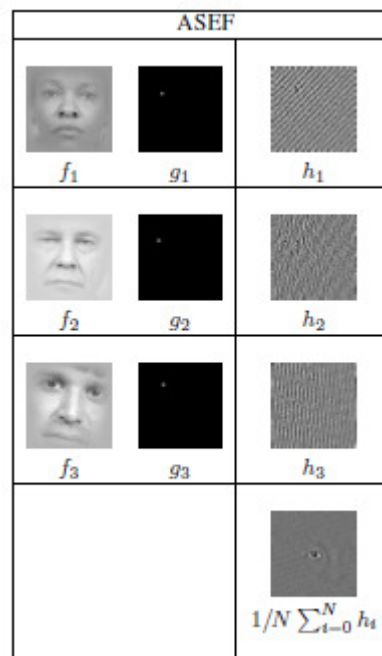


Figure 4: ASEF filter construction [2]

5. Data Flow Graph

Frame cropping, resizing and ASEF filter construction are done on the PC. Face detection and eye detection are done on the DSK. The eye detection algorithm is most suited for the DSK because assembly optimized FFT libraries are available. Frames will be sent to the DSK from the PC using TCP network transfers. The face location and the correlation output will be sent to the PC from the DSK using TCP network transfers. Figure 5 shows the order in which each step occurs.

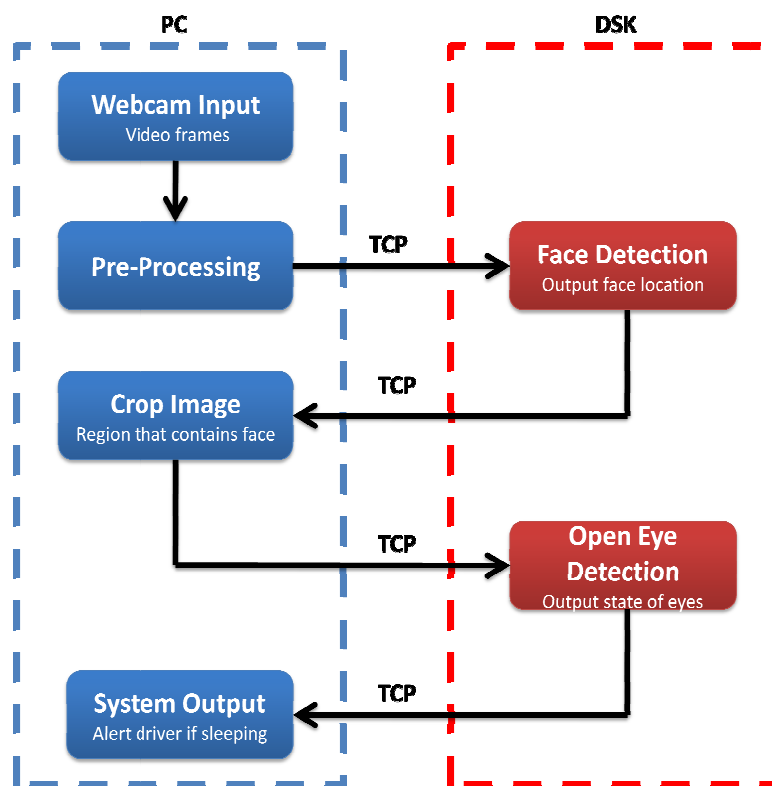


Figure 5: Data flow diagram

6. Databases

6.1 OpenCV

OpenCV contains four frontal face detectors that were trained by the authors of [6]. We will use one of these detectors instead of training our own detector because training can take on the order of weeks. Because Gentle AdaBoost outperforms other boosting algorithms [6] and the optimal input pattern size for detecting faces is 20x20 pixels [6] we will use the stump-based 20x20 Gentle AdaBoost frontal face detector that is packaged with OpenCV. Tree-based detectors are better than stump-based detectors [6] but we will not use the tree-based detector that is packaged with OpenCV because it uses more memory and is harder to implement than the stump-based detector. The detector we chose has 22 stages and 2135 directional filters. OpenCV also contains upper body, lower body, full body, profile face, mouth, nose, and eye detectors. We didn't use any of these detectors.

6.2 Carnegie Mellon University (CMU) Pose Illumination and Expression (PIE) Database

We will use the CMU PIE database to design the open and closed eye filters and test the face detection. This database contains 41,368 images of 68 people in 13 different poses and under 43 different illumination conditions and with four different expressions [4]. The images were

collected in two different sessions: October 2000 - November 2000 and November 2000 – December 2000. The images are 640 x 486 pixels. The 13 poses are shown in Figure 6. The four expressions are shown in Figure 7. This database contains the location of the eyes for each image. We will use poses 27, 9, 7, 5, and 29 to train the ASEF filter. We will use poses 27, 5 and 22 to test face detection. We chose this database because it contains subjects with closed eyes. Initially we were going to make both an open eye and closed eye filter but the closed eye filter was not used for reasons discussed in Section 8.2.2

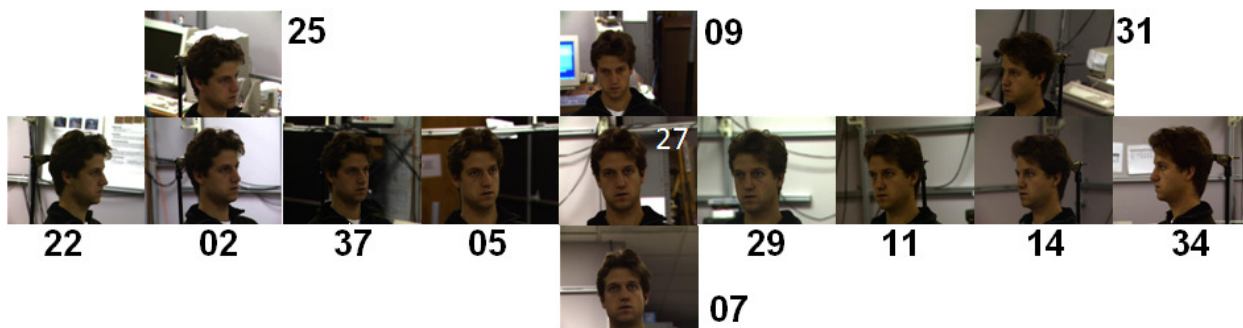


Figure 6: Examples of each pose in the CMU PIE Database.



Figure 7: Examples of each expression in the CMU PIE database. From left to right: neutral, blinking, smiling, and talking.

6.3 Faces in Places Blog

The Faces in Places Blog contains thousands of images that don't contain faces but look like faces [7]. The face detection algorithm is tested on 105 of these images. Three images from the blog are shown in Figure 8.



Figure 8: Images from the Faces in Places blog

7. Hardware

We purchased a Logitech Webcam Pro 9000 for \$86.99 shown in Figure 9. The camera has a 2MP HD Sensor with autofocus. The camera supports frame rates up to 30 fps and can be connected to the PC via a USB 2.0 port. Instead of using the software that came with the camera, we used OpenCV to pull frames from the device. This camera adjusted for changes in illuminations but glare from the computer monitor and other light sources still affected the accuracy of our eye detection algorithm. Illumination issues are discussed in Section 8.2.2.



Figure 9: The webcam

8. Testing and Training

8.1 Viola-Jones Face Detection Algorithm

8.1.1 Training

The Viola-Jones face detection algorithm has been trained by the authors of [6]. The training set consists of 4,000 images of which 1,000 are faces and 3,000 are non-faces. The training is not described in much detail in [6]. The training information is contained in an xml file packaged with OpenCV.

8.1.2 Testing

We tested our implementation of the Viola-Jones face detection algorithm using face images from the PIE database and non-face images from the Faces in Places blog. The face test set is

composed of 15 subjects in three different poses under five different illumination conditions. We tested on three different size faces: 15 x 15, 20 x 20 and 25 x 25 pixels. In total, we tested on 675 images that contained exactly one face. The three poses we used are shown in Table 1. The first pose is facing straight forward. The second pose is a 22.5 degree out of plane rotation. The third pose is a 90 degree out of plane rotation.



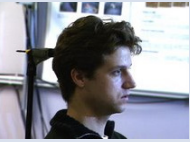
	15x15 pixels	20x20 pixels	25x25 pixels
	77%	100%	99%
	86%	90%	78%
	0%	0%	0%

Table 1: Face detection results

The face detection algorithm was the most accurate on the set of 20 x 20 pixel faces. However, it is unclear whether the algorithm is more accurate on faces that are smaller or larger than the target size. Illumination variations didn't significantly affect the detection rates. No faces were detected in the set of 90 degree out of plane rotated faces because the detector was trained to detect frontal faces. It is assumed that when driving at any substantial speed, the driver should be facing forward. The non-face test set is composed of 105 images of non-faces. Our face detection algorithm reported that 20% of these images contained faces. These 21 images are shown in Figure 10.



Figure 10: Images from [7] that registered false positives.

8.2 Average of Synthetic Exact Filters (ASEF) Filter Design

8.2.1 Training Set

To create a diverse training set for the ASEF Filter, we used a large number of subjects in a variety of poses under a few different illuminations. We used the set of 52 subjects that were imaged during the months of October and November. This set contains variations in race and gender. We used the five most frontal poses of the database which are poses 05,07,09,27, and 29. These poses are shown in Figure 6. These are the same five poses used to train the open eye filter in [3]. These correspond to the subject looking left and right at 22.5 degrees, looking straight ahead, and looking up and down. We used a set of illuminations that could capture high and low illumination and illumination variation over the face. To do this, we used illuminations 06,08,13,16, and 21 as shown in Figure 11. This training set gave us a total of 1300 training image.



Figure 11: The illuminations that the ASEF filter was trained on

8.2.2 Testing Set

We tested on 15 subjects that were imaged for the PIE database at a time after those of the training set. Different subjects and illuminations than training were tested but we kept the poses the same as they enumerate the possible poses we expect to handle. We randomly chose three images per subject to test from the illumination and pose combinations. An eye is detected when the highest peak in the top half of the face is above the detection threshold. The test results for open and closed eyes are shown in Table 2. We achieved enough of a difference in the correlation to detect an open eye against a closed eye and against outside noise. We didn't make a closed eye filter because a closed eye filter has a high false positive rate while the open eye filter seldom detects a closed eye.

Eye Status	Left Peak	Right Peak	Outside Region Peak
Open	3.0755	3.3605	1.6725
Closed	0.9978	1.0435	1.23

Table 2: Eye detection test results

This filter doesn't work well when there are shadows on the face. An image with poor illumination on the left side of the face is shown in Figure 12. Due to low illumination in the left half of the image, there is no correlation peak at the location of the left eye. There is a correlation peak of approximately three at the location of the right eye. Another image that demonstrates poor illumination is shown in Figure 13.

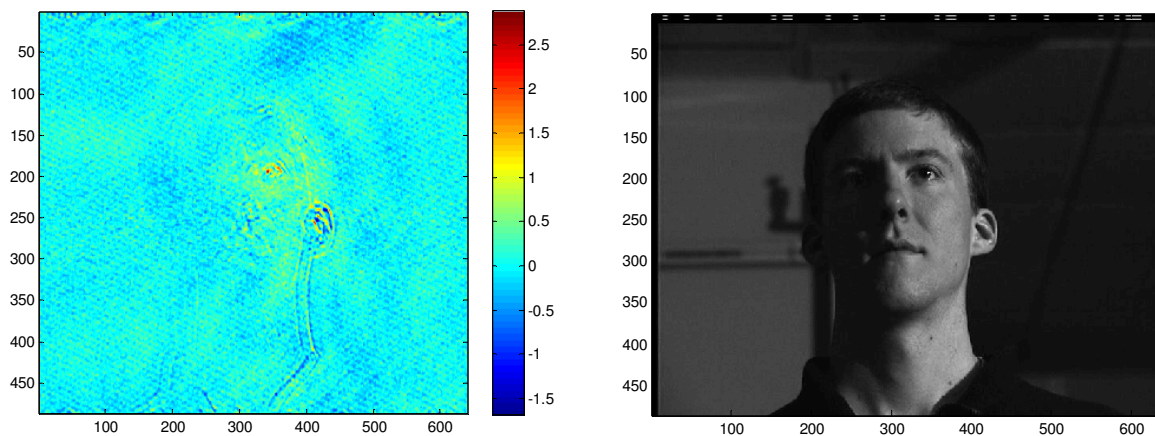


Figure 12: The left eye isn't detected.

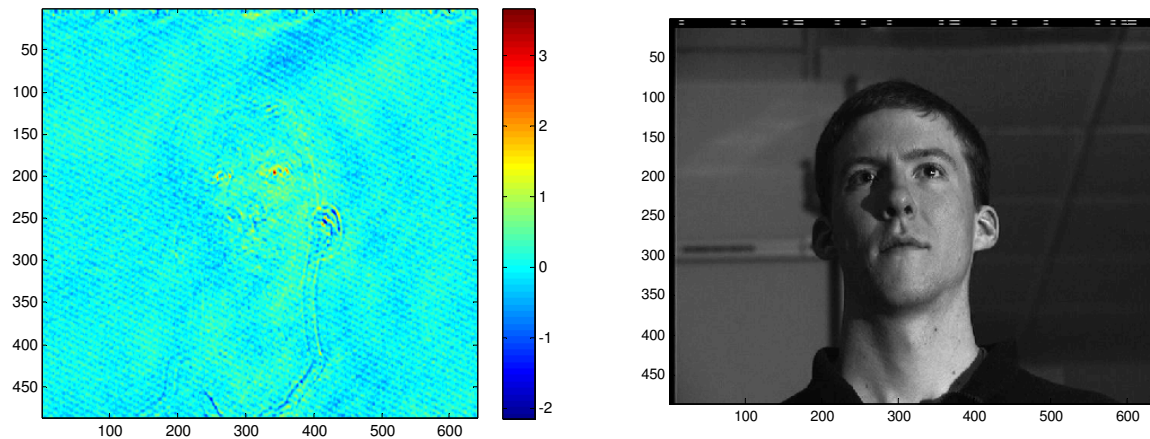


Figure 13: The left eye isn't detected.

In Figure 13, there is a correlation peak in the location of the left eye, but it is not as strong as the peak in the location of the right eye. Because this peak isn't greater than the selected threshold, the left eye isn't detected. The ASEF filter worked satisfactorily in the lab using the webcam. When we integrated the ASEF filter with face detection, the performance of eye detection was poor because the ASEF filter wasn't trained properly on the output size of face detection. Eye detection only worked when the face was in a near perfect position.

9. Demonstration

The subject sits in front of a web cam that is mounted on the monitor and pretends to be a drowsy driver. The face should be approximately two feet from the webcam. The live video stream (the input signal) is displayed on the monitor. A square is drawn around the face when detected. The correlation output is displayed in a second window and circles are drawn in the location of the eyes when open eyes are detected. A loud sound was not emitted from the PC (this was suggested in the project proposal) when the eyes were not detected for longer than three seconds because the eye detection algorithm was so inaccurate that this would have been obnoxious. The demo is shown in Figures 14, 15, and 16.

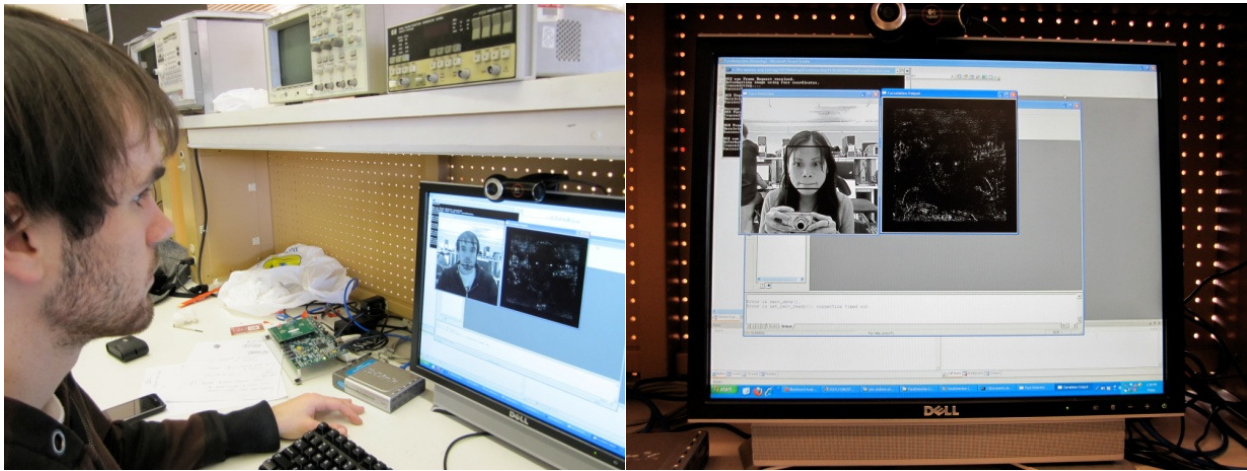


Figure 14: The demo in the lab.



Figure 15: Face detection output.

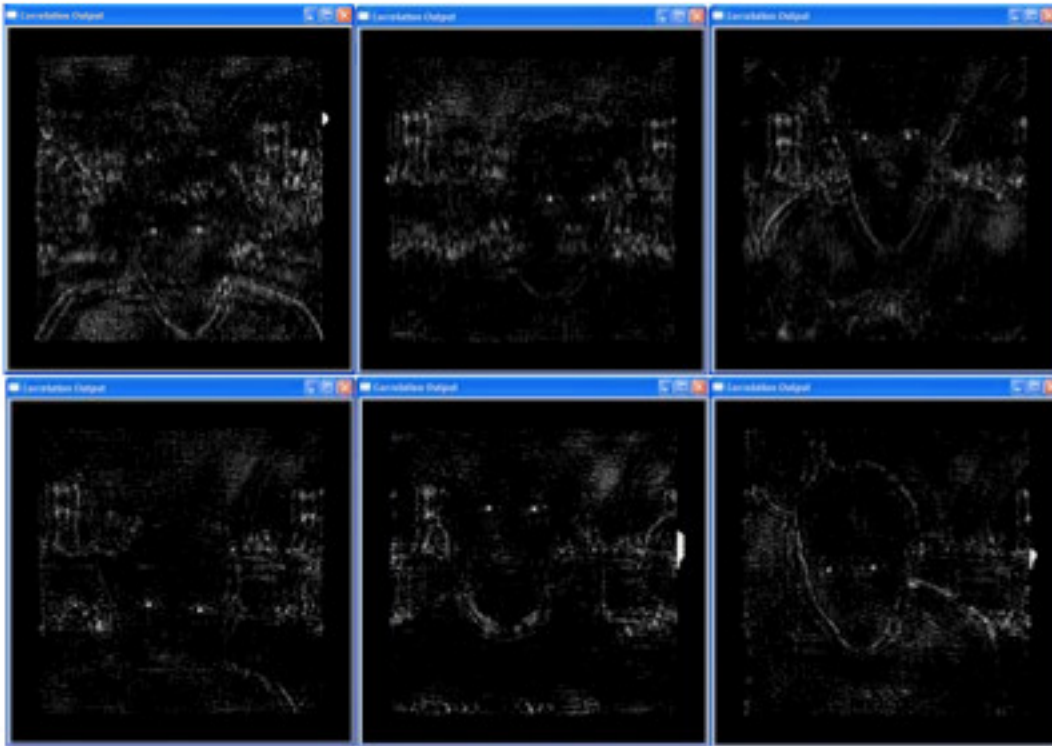


Figure 16: Eye detection output.

10. Timing, Speed, and Optimization

10.1 Viola-Jones Face Detection Algorithm

10.1.2 Optimizations

Our face detection implementation takes 21.9 ms on the DSK to process one frame. Transferring and storing the input image in internal memory instead of external memory decreased the speed of the algorithm from 71 ms to 46.5 ms. Using inline functions to evaluate the direction filters reduced the speed of the algorithm from 46.5 ms to 25.7 ms. Configuring the compiler to assume no memory aliasing decreased the speed from 25.7 ms to 21.9 ms. These times do not include the network transfers.

10.1.1 Timing and Speed Analysis

The face detection input is a 64 by 64 8-bit unsigned integer array. The input is sent from the PC to internal memory on the DSK using a TCP network transfer. All data structures and variables used by the face detection algorithm are in internal memory. The cumulative sum and cumulative squared sum data structures are 64 by 64 32 bit unsigned integer arrays. The internal memory usage of the algorithm is shown in Table 3. The size of the xml file packaged

with OpenCV that contains the description of the frontal face detector is 899KB. We designed a data structure that stored the same data in 49KB.

Data Structure	Size
Input image	4.1 KB
Cumulative sum image	16.4 KB
Cumulative squared sum image	16.4 KB
Cascade and Filter data structures	49 KB
Code Size	3.7 KB
Total	89.6 KB

Table 3: Face detection internal memory usage.

10.2 Average of Synthetic Exact Filters (ASEF) Filter Design

10.2.1 Optimizations

In order to compute the 2D FFT's required to quickly filter the image, we transform the rows, and then transform the columns. To perform the 256 point FFT we use the TI assembly optimized function DSPF_sp_cfftr_dif and to perform the bit reversal of the output we use the TI library function, digit_reverse, with pre-computed indices. To improve the performance of the FFT of the rows, we use a double buffering scheme (ping-pong) to perform transfers from external to internal memory while the FFT is being computed. Each buffer is sixteen rows of the image (a total of sixteen transfers occur). To perform the transform of the columns, it is required that you transpose the image. This ended up being a surprisingly large penalty (approximately nine times longer to compute than the FFT's!), due to terrible cache performance for the naive version of the transpose. A better version of the transpose could be implemented purely by using the DMA controller on the DSK, which has the capability of reading at offsets and writing at offsets. In our case, this would entail reading at an offset of 256, the number of columns, and writing at an offset of two (for the real and complex parts). Each column would then need to be moved into internal memory with a separate transfer (512 separate transfers in total). In this way, with a good double buffering scheme, the penalty of transferring into internal memory and performing the transpose could be virtually zero.

10.2.1 Timing and Speed Analysis

Each FFT takes $(14 \cdot 256 / 4 + 23) \cdot \log_4(256) + 20$ cycles = 3696 cycles
 This is done 256 times for the rows and 256 times for the columns. This gives us an expected 946,176 cycles for the FFT of the rows. We have to bit reverse each of these which will add $256(5 \cdot 256 + 33)$ cycles = 336,128 cycles to the total. To transfer the memory from external memory into internal memory, we require 2 concurrent DMA transfers. Only the first DMA

transfer adds overhead because we have to wait for it to complete. This transfer is of 2 rows of 512 floats which is 4096 bytes. This transfer will take 30 cycles for initialization and 0.5625 cycles for every byte. This totals to 2304 cycles per DMA transfer. There are 64 of these that will count towards our total for writing to internal memory, and 64 for writing to external memory after the FFT is computed. The total cycle number for DMA transfers is 294,912 cycles. Our transpose was the operation that killed us because of its terrible cache performance. For this, we swap two elements in external memory. Each of these accesses takes 7.125 cycles per word and the floating point complex interleaved structure gives us 2 words in each element. This gives us $4 * 7.125 * 2 = 57$ cycles per element swap. We have $256 * 256$ element swaps which runs the total up to 3,735,552 cycles. The overall cost of the FFT2 will therefore be $2 * \text{TransposeCost} + 2 * \text{FFTCost} + 2 * \text{BitReverseCost} + 2 * \text{DMACost}$. This gives us $3,735,552 + 1,892,352 + 672,256 + 589,824 = 6,889,984$ cycles for taking the 2D FFT. The major problem that we have with these estimates is that our transpose function does not obey these predictions. Instead of taking the expected 3,735,552 cycles, it takes over 9 million. Upon investigating this further, the assembly showed it using 2 loads and 4 stores per element swap, which will put our estimate higher than previously, but not by enough to account for the slow speed that we encountered.

Instead of the result of 7 million cycles for this operation, we observed roughly 20 million cycles. Besides the mentioned differences in the speed of transpose, the speed differences are beyond us. It's possible that the indexing into memory adds some overhead, but it doesn't seem like it would be enough to account for the large difference in speed.

To correlate the new image with the filter, we actually convolve it with the conjugate of the filter to save time. This operation takes 4 multiplies and 2 divides for each element and the same 5.625 cycles for memory access due to spatial locality. We have 2 words for each element so overall we take $4 * 5.625 * 256 * 256 = 1,474,560$ cycles to load the elements. Then we have to store the elements back into external memory, so we require $2 * 5.625 * 256 * 256 = 737,280$ cycles. The element multiplication is hard to predict because these boards do not have a hardware floating point divide. We can assume that a software floating point divide will take an extremely long time. The four multiplies give us roughly 16 cycles of computation per element which will total up to $4 * 256 * 256 = 262,144$ cycles. We can therefore put a lower bound on the convolution to $1,474,560 + 737,280 + 262,144 = 2,473,985$ cycles

We assume that our 2D IFFT will be almost identical in time to our 2D FFT, so overall we have an estimate for correlation of our image with the ASEF filter to be roughly 16 million cycles.

11. Schedule and Milestones

In order to complete all parts of our project in one semester, we established three milestones shown in Table 4. We assigned tasks and deadlines to each group member which is shown in Table 5. We underestimated how long it would take to port the eye detection code to the DSK. It took us a long time to get the FFT working because we had issues with bit reversal. Implementing the Viola-Jones face detection in C took longer than expected. Once our face

detection code worked on the PC, it took less than an hour to port to the DSK. We were warned by many that the algorithm would require more memory than the DSK had available but this isn't true because we implemented the entire algorithm using only internal memory. Managing memory on the DSK was difficult because both face detection and eye detection required a considerable amount of memory. We had to sacrifice optimization of the code in order to finish on time and have a real-time demo.

Milestone	Due Date	Tasks
1	November 13 th , 2009	All algorithms working in C or MATLAB
2	November 16 th , 2009	All algorithms working on DSK
3	November 27 th , 2009	System operates in real-time

Table 4: Project Milestones

Week	Task	Team Member
November 2 nd , 2009	- Webcam input and pre-processing - Face detection algorithm in C - ASEF filter design	Justin Dan, Jes Matt
November 9 th , 2009	- Finish face detection algorithm in C - 2D Fourier Transform on DSK	Dan, Jes Justin, Matt
November 16 th , 2009	- Face detection on DSK - Cross-correlation on DSK	Dan, Jes Justin, Matt
November 27 th , 2009	- Code optimization	Everyone
November 30 th , 2009	- Final presentation and demo	Everyone
December 7 th , 2009	- Finish final report	Everyone

Table 5: Group member assignments

12. Future Work

12.2 Viola-Jones Algorithm

The face detection algorithm could be optimized further by converting it to fixed point because the evaluation of each directional binary filter requires a floating point divide. The DSK doesn't have a floating point divide assembly instruction. Our implementation uses a fixed size sub-

window. The input image could be scanned using multiple scales to improve detection rates. OpenCV contains detectors for other objects including eyes, nose and mouth. We suggest that future groups use these because the speed of the algorithm makes it well suited for DSK projects. However, don't underestimate the complexity of the implementation of the Viola-Jones algorithm in OpenCV. Several small details should not be overlooked: the filter values must be divided by the area of the sub-window and the filter thresholds (not stage thresholds) must be multiplied by the standard deviation of the sub-window.

12.2 ASEF Filter

Because the face detection and eye detection were developed without collaboration, the face detection output and eye detection input sizes didn't match. We didn't have enough time to train the ASEF filter on a new input size so the accuracy of eye detection suffered. The cross-correlation should have been normalized. The transpose and inverse FFT were not paged from external memory to internal memory (paging was only implemented for the FFT). An infrared camera should be used for eye detection.

13. References

[1] NHTSA. *National Survey of Distracted and Drowsy Driving Attitudes and Behaviors*. 2002

- Motivation for project

[2] D.S. Bolme, B.A. Draper, and J.R. Beveridge. *Average of Synthetic Exact Filters*. Computer Vision and Pattern Recognition. June 2009.

- ASEF filter design

[3] D.S. Bolme, J.R. Beveridge, and B.A. Draper. *FaceL: Facile Face Labeling*. International Conference on Vision Systems. October 2009

- Face labeling system that uses ASEF filters to locate eyes
- Prior work in the field

[4] T. Sim, S. Baker, and M. Bsat. The CMU pose, illumination, and expression (PIE) database. In *Proceedings of the 5th IEEE International Conference on Face and Gesture Recognition*, 2002.

- PIE database details

[5] Paul Viola, Michael Jones, "Rapid Object Detection using a Boosted Cascade of Simple

Features," cvpr, vol. 1, pp.511, 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'01) - Volume 1, 2001

- Viola and Jones face detection algorithm

[6] Lienhart, R., Maydt, J.: An extended set of haar-like features for rapid object detection. In: ICIP. (2002)

- OpenCV frontal face detector comparisons

[7] Faces in Places, <http://facesinplaces.blogspot.com/>

- Non-face images that look like faces