Interactive Computing

Authors:
Cogan Noll
Kermen Deol
Praveen Sitaraman

May 04, 2010
List of Figures

1 Haar-like features .................................................. 6
2 Camshift backprojection ........................................... 7
3 Face Recognition Process .......................................... 11
4 Successful face detections with user orientation .............. 12
5 Effect of Face Presence ............................................ 13
6 Open Eye Identifications ......................................... 14
7 Closed Eye Identifications ........................................ 14
8 Open and Closed Eye Images ...................................... 15
9 Example Eigenfaces ............................................... 16
10 Eigenface recognition analysis .................................. 17
11 Eigenface error analysis .......................................... 17
12 Eigenface time analysis .......................................... 18
13 Algorithm Performance Analysis ............................... 19
14 User before alignment ............................................ 19
15 User after alignment .............................................. 20
1 Objective

This project’s main purpose was to create the basis for a secure, innovative, ergonomic robotic computing system that would be able to interact with the user in real-time. The robotic computing system consisted of a standard flat screen computer monitor that was mounted on a mobile chassis with a height adjustment column. The idea is that the monitor should be able to actively track the user’s face in real-time and automatically adjust its height and position to the user's eye-level.

To make the system more secure, face recognition capabilities were added so that the computer system could match up user faces with faces stored in the database. If there is a satisfactory match, the user is able to log in to the system; otherwise, the user is logged out. All algorithms used to make this computing system function were implemented in MS Visual C++ while using the OpenCV library files. A user GUI was also created in Visual Studio to enhance user-system interaction.
2 Procedure

This section describes the design and algorithms that are used in the Project.

2.1 Mechanical Design

The mechanical design of the robot is mainly three different components: the moving base, the monitor tray, and the lift mechanism. The base was built to be wide in order to provide more stability for the monitor. In the rear of the robot is a free spinning wheel that turns with the base which is controlled by its front wheels. Mounted above this rear wheel is the NXT brick. There are two motors on the front that each controls one wheel. This will allow the robot to turn left and right and move forward and backward. The third and last motor on the robot is mounted on the side of the robot. It is used to control the lift mechanism. This motor turns two successive torque converters. These gears turn the motor’s rotation into a linear motion by moving a straight toothed piece. This is known as the rack and pinion and connected to the rack is the tray and back support for the monitor. As the motor turns the pinion, the rack and attached tray will move up and down. This mechanism was mounted in the center of the base providing improved balance and stability. We have found that our robot can effectively lift a large textbook.

This design was inadequate since our original goal was to lift an actual monitor. The first obstacle of this task was to either make the Legos strong enough to hold the heavy weight of a monitor or to make the monitor light enough to be lifted by the Legos. We settled on the latter solution and began removing as much possible weight from the monitor as we could. The monitor was disassembled and separated into the electronics, power supply, heat sink, etc. and the actual display panel with backlight. Only the panel and backlight were lifted while the electronics, power supply and heat sink were placed on the side and did not need to be lifted. After successfully disassembling the monitor we found that we needed to create an extension for the ribbon cable and backlight power wires that connected the electronics to the panel. We cut 20 wires of the same length (a few feet) and soldered the ends between the panel and the electronics to create a much longer ribbon cable. The effect of extending the length of this cable was immediately evident in the picture and there was a lot of interference on the high frequency lines connecting the monitor pieces. In order to counter this interference, we wrapped the cable extensions in aluminum foil as this conductor cut out the interference. After this the interference was gone and we had a clear picture on a lightweight monitor that the Legos had a chance of lifting.
The Lego motors and torque converters that we built were strong enough to lift the panel. However, there were stability issues with an increase in the height of the monitor. The first attempt to correct this was to create a base that was not as tall to lower its center of gravity. This added increased stability but most of the weakness remained in the rack and pinion component. The first design of the robot had a tray attached to a rack which moved up and down when the motor turned. This was reversed so that the rack was stationary and the motor, torque converter, tray and monitor would all move up and down the rack which stayed stationary. The torque converter was housed in piece which was built around the rack so that it could move up and down and the motor was attached to the side of this. The monitor tray was then greatly reduced in size and weight since the housing provided back support for the monitor and the panel itself was so thin. This solution provided the best stability and allowed the monitor to have a full range of motion of 7 inches without losing its balance. A touch sensor at the top and another at the bottom were used to determine if the monitor was at the minimum or maximum height possible so it would not fall of or break the mount. Overall, the mechanical design of the robot required much time and many revisions but proved to be successful in the end as we were able to effectively lift and rotate a computer monitor in real time. The mechanical design of the robot required us to perform work outside of our expertise and use our problem solving skills to construct a robot which could perform the required tasks.

2.2 Monitor Positioning

These are the algorithms that are used to position the monitor.

2.2.1 Haar Face Detection

Haar face detection is used to detect a face in the image. This method works by taking the current camera frame and dividing it into smaller rectangular subsections of the image. The difference between the average of the light pixel values and the average of the dark pixel values in each region is then used to detect if there are any facial features in that region. This is done by comparing the subsection to a set of special rectangular combinations called “Haar features”. If the difference of the average values for the subsection is within a threshold corresponding to a Haar feature that the system knows about then that feature is determined to be present.
This threshold is found by “training” the system to recognize faces using a large number of face images. These images are put through series of different classifiers. At each step of the series the classifier of that step determines whether or not this image is a face. If it is, it moves to the next step in the series. If it is not, the image is rejected as not being a face. Since these are training images, the thresholds are set with these such that all the training images pass. Now, with the thresholds of the features set, the subsection of the image is put through the same series of classifiers. If it reaches the end of the series then this subsection is determined to be a face. If not, it repeats with the next subsection. This entire algorithm is done for all subsections of the image and at different scales. [7]

2.2.2 CamShift tracking

The Camshift tracker works by finding the area of the image with the highest probability of being the object it wants to track. It first takes the area of the image and represents it as a histogram of the pixel hues of that area. This histogram is created once for the object to be tracked. Using the histogram, “face probabilities” can be calculated. This means that using the histogram, the probability that a pixel has a certain hue is calculated. When the system processes each camera frame, it assigns these probabilities to each pixel based on the pixels hue. This is referred to as the “backprojection”. The back-projection is a grayscale image with the black intensity determined by the probability assigned to that pixel. The lightest pixels of this backprojection are where the highest probabilities are and the black pixels are where the lowest probabilities are. With each frame it shifts the area it is following so that the rectangular area is centered of the area with the highest probability. This is called the “mean shift”. The size and scale are adjusted to fit this area correctly and with that information the size and angle of the face are found.[3]
2.3 Robot Positioning

The application communicates with the NXT brick on the robot by using a bluetooth connection. The interface to the connection is achieved by using the C++ communication library [1], and this allows the program to control each motor individually as well as read input from the sensors. The positioning algorithm works by making sure that center of the users eyes are contained within a defined portion of the current camera frame (this works because the camera is located on the top of the monitor, so it will move with the monitor). It is set up so that the robot will move until the eye center point is at 75% of the height of the frame, and also within the horizontal center of the frame +10 pixels. If the user is outside of this area the robot will lower/raise the screen and rotate left and right until the monitor is positioned correctly.

2.4 Sleep Detection

In order to determine if a user is sleeping or not we determine whether their eyes have been closed for a certain period of time (e.g. 10 seconds). To determine the state of the eyes we use a combination of Eye Detection and Eye State Determination.

2.4.1 Eye Detection

The eye detection algorithm uses a method called EigenEyes [8], which is very similar to that of the EigenFaces method used to recognize faces. In the eigenfaces method a database of images of the user faces is used to match the image to a user. In the EigenEyes method, instead of user face images, this database has generic eye images and the process is basically the same.
PCA is used just as in EigenFace method which is explained in section 2.4.2 of this paper. Once the system is trained to detect an eye, the the upper left and right quarter regions of the face area (which is known from the haar face detection) are searched in order to find an eye. At first, we had each eye tracked individually. We found that with this approach the right eye was consistently found and detecting the left eye was much less accurate. This was because the database images used to train the eye detection system were all right eye images and so the shape of the eye was slightly different than the left eye. We changed the eye detection so that it no longer found the eyes independently. The solution was to find the right eye and then determine the proper location of the left eye on the face based on the information we knew about the right eye. In the end, the eye detection system performed well and the EigenEyes method proved to be an effective way of implementing it. Two other algorithms we tried to use to detect eyes were Haar detection and Template Matching. The Haar detection method was similar to that of the Haar Face Detection algorithm but it used eye classifiers instead of face classifiers. The Template Matching method used two eye images and the cvMatchTemplates() function to try and detect the user’s eyes. After implementing and testing both of these methods it was determined that neither of these algorithms were able to deliver an accurate eye detection system and the performance gains were not significant enough. Therefore, the EigenEyes algorithm was chosen.

2.4.2 Eye State Determination

Once the eyes have been detected these regions of the image are analyzed to determine whether each eye is opened or closed. This is essentially performed by searching this area for a circle (representing an open eye). The Canny edge detection algorithm is run on the eye region to find the edges of the eye. Once the edges have been found the program searches for countours, and then tries to fit them to an ellipse. If there is a successful fit an eye has been found, and its coordinates of the center are returned, otherwise the eye is deemed to be closed.

2.5 Security

Before a user can use our software they must be confirmed to be an authorized user. The simplest method for accomplishing this is with a password, but our software is much more advanced and uses face recognition via camera. The algorithm we are using to accomplish this task is called “eigenfaces.” The method was originally described in the 1991 paper “Eigenfaces for Recog-
nition” by Turk and Pentland [9], and today it is probably the most well known face recognition algorithm.

2.5.1 Eigenfaces Overview

Eigenfaces works by matching an unknown target face against a pre-existing database of training faces, whichever training face most closely matches the target face is the user. The idea is simple enough, but how do you determine which image most closely matches the target face? Each image is treated as a vector composed of pixel values. For an NxN pixel grayscale image, the representation is a $N^2$ dimensional vector with values ranging from 0 to 255. This vector can then be treated as a point in $N^2$ dimensional space. If we have a database with 10 training faces, these can all be treated as data points. When we want to match our unknown face, we simply determine whichever face is closest using a simple distance formula, and return that as the answer.

$$d = \sqrt{\text{dim}_1^2 + \text{dim}_2^2 + ... + \text{dim}_M^2}$$ (1)

Although this method is simple, it’s not feasible to do these sorts of computations quickly even for small images. In order to make the eigenface algorithm usable, a technique called Principle Component Analysis (PCA) is used. The idea behind PCA is to project the images in $N^2$ dimensional space onto a subspace with much lower dimensionality, and then do the distance computations in this subspace. It can be shown that the maximum required dimensionality of the subspace is one less than the number of sample faces in the database [2].

2.5.2 Steps in PCA

1. Get some data In this case the data is the set of training images, all of the faces that we want to be able to recognize.

2. Subtract the mean In order for PCA to work properly you have to subtract the mean from each dimension (or each pixel). To do this you create an Average Image that consists of all mean pixel values, then subtract this image from each training image.

3. Calculate the covariance matrix This will be the covariance matrix for the mean-adjusted images. Because each image has $N^2$ dimensions, the covariance matrix is an $N^2 x N^2$ matrix.

$$C_{n \times n} = (c_{ij}, c_{ij} = \text{cov}(\text{Dim}_i, \text{Dim}_j))$$ (2)
4. Calculate the eigenvectors and eigenvalues of the covariance matrix An eigenvector $x$ of a matrix $A$ is a non-zero vector such that

$$Ax = \lambda x$$ (3)

For an $N \times N$ matrix there are $N$ eigenvectors, so for the covariance matrix of the training images there are $N^2$ eigenvectors (or eigenfaces). However, only a few eigenfaces with the highest eigenvalues are required for a good estimate. The amount necessary is equal to the number of data points minus one (i.e. if you have 10 training images only 9 eigenfaces are required). These are the principle components.

5. Project onto the eigenspace The final step is to project the mean-adjusted training images onto the subspace defined by the principle components. The new set of images can be determined by multiplying a row matrix of eigenfaces by a column matrix consisting of the mean-adjusted training images. The rows of the result will be the new images.

$$\begin{bmatrix}
\text{new}_1 & \text{new}_2 & \ldots & \text{new}_{10}
\end{bmatrix} =
\begin{bmatrix}
ef_1 \\
ef_2 \\
\ldots \\
ef_9
\end{bmatrix}
\begin{bmatrix}
im_1 & \text{im}_2 & n & \text{im}_{10}
\end{bmatrix}$$ (4)

2.5.3 Recognizing the face

After PCA has been performed on the training images, the last step is to recognize and unknown face. This is simply a matter of projecting the unknown face onto the eigenspace and calculating the distance between it and the training images. The closest training image should be the same person in the unknown image. Euclidian distance can be calculated by using the distance formula expanded for higher dimensionality computations.

2.5.4 Implementation in Our Application

Because our application uses a live video feed, users must be authorized in real time. When the application is first started it is in a secure mode, no user is logged in. Once the video feed is initialized the program scans each frame for a face using our face detection algorithm. If a face is found that image is converted to grayscale and scaled to match the images in the training database. PCA analysis is then performed on the image and it is
matched against the face database. The face that is closest in the eigenspace is selected as a match, if it is below the minimum threshold. Once a user has successfully been identified they are “logged in”, and the face recognition algorithm ceases to run.

Figure 3: An overview of the process used to identify an authorized user and log them into the application.
3 Experimental Results

3.1 Face Detection

To test the accuracy of our face detection system we measured how often a face was detected in different scenarios for 100 iterations. For these tests the following conditions were maintained throughout: tests were conducted indoors in tungsten lighting and the distance from the monitor was approximately .76 meters.

![Face Detection Accuracy](image)

Figure 4: Successful detections of a face with changing orientation of user’s face.

It is clear from the figure above that if a user’s face is centered then Haar Face Detection will find the face 99% of the time. Additionally, if the user turns their head slightly left or right the face detection becomes very weak detecting a face only 9% or 16% of the time depending on which way the user turns. Haar Face Detection is best suited to users looking head on with the camera an not turning their head. The following figure is an analysis of the system’s false positives. It measures how often a face detected if there is or is not an actual face present.
Figure 5: Shows the Effect of Face Presence on detection

From this figure we see that when a human face is present, the face detection algorithm accurately detects the face 99% of the time. When there is no face or face like object present, the face detection algorithm does not detect a face 100% of the time which is what we would like from the system. We then tested the system with an illustrated picture of a face. The system identified this picture as a face 35% of the time and detected it as not a face 65% of the time. Although 2/3 of the time it does not recognize it as a human face, the other third of the time it does believe the drawn image is a human face. This shows a clear weakness with the haar detection algorithm and face detection algorithms in general. Overall however, the haar face detection method proved to be extremely accurate when the user was facing the camera and gave no false positives when a human face or illustrated face was not present.

3.2 Eye Detection

In testing the EigenEyes method that was implemented we found that 100% of the frames in which a face was detected, the eyes were also detected. In our implementation, finding eyes within the image requires a face to be found first and that region is then searched for eyes. This reduces the processing strain on the program. Since the eye detection relied on a face being found, we tested how often eyes were found given that a face was found first. In every trial iteration the eyes were found. It is possible that the eyes are required to find a face and without those features haar can not detect a face. Thus, eyes must be present for a face to be found so eyes are found 100% of
the time a face is found.

### 3.3 Eye State Determination

To test eye state detection we ran our application for approximately 150 iterations with a user who had their eyes opened, and then with a user who had their eyes closed. We recorded how many times our algorithm identified the correct eye state.

**Figure 6:** Successful and Unsuccessful identifications of the Open Eye state. Open Eyes are successfully identified 25.3% of the time.

**Figure 7:** Successful and Unsuccessful identifications of the Closed Eye state. Closed Eyes are successfully identified 94.0% of the time.
Our results were poor, and indicated our system was not very adept at detecting open eyes. We also had some eyes misidentified as open during the closed eye test.

![Figure 8: Open and Closed Eyes. (Top) Original Image. (Middle) Grayscale Image. (Bottom) Canny Edge Detected Image.](image)

Upon further analysis we decided that the eye images were not of high enough resolutions to accurately determine the edges of an eye. Because the resolution of our camera was only 320x240, and our full image was an entire scene containing the head and surrounding environment, it was impossible to get high resolution imagery of the users eyes.

We do believe that detecting closed eyes is an achievable goal, but not via this method in real time conditions. A template matching algorithm or an eigeneyes algorithm training with closed eye images may have been more effective. However, due to time constraints we did not include “sleep detection” in our final product.

3.4 Face Recognition

When performing PCA analysis the dimensionality of each image is greatly reduced by projecting each training image onto a subspace. The dimension-
ality of the subspace is equal number of eigenfaces used, but there is always a question of what dimensionality is required to accurately perform face recognition. To find out we experimented with a database consisting of 20 users, each with 5 training images of size 92x112. For each user we selected a 6th image not contained in the training database, and then used the eigenface algorithm to identify the faces. We repeated this test for several different amounts of eigenvectors and recorded the number of faces correctly identified and the computation time.

Figure 9: 10 most principle Eigenfaces generated from our training database
Figure 10: All faces were recognized with a subspace of dimensionality $\geq 15$ for this sample set.

Figure 11: Percent Error decreases quickly as the number of eigenfaces increases.
Our results indicate that the eigenface algorithm is very accurate, even for a very low dimensionality subspace. The dimensionality of the original images is $92 \times 112$, or 10304. We were able to obtain extremely accurate results with a subspace of dimensionality of only 15, a subspace less than 600 times the size of the original. We also found that computation time increases linearly with the number of eigenfaces used. This means that a computation in a 15 dimensional subspace would be more than 600 times faster than a computation in the original image subspace!

### 3.5 Algorithm Performance

The performance and additional strain of each algorithm was measured to assess the efficiency of our system. As expected, enabling more algorithms caused the camera feed to slow down by many frames per second. The results showed that Haar Detection with EigenFaces performed the worst out of all the algorithm combinations. Due to the computational intensity of the algorithm, we were only able to process 3.3 FPS whereas combining Haar detection with Camshift tracking and EigenEyes allowed the program to run at 3.7 FPS (a small but measurable difference). With all of the algorithms turned off, the camera feed was approximately 10 FPS. The load of the algorithms is depicted in the following figure.
Figure 13: Performance of algorithms measured in frames per second

3.6 Robot Tracking

Ultimately the robot was able to accurately track the user by aligning the center of the users eyes with the appropriate position on the screen.

Figure 14: User before alignment
The robot moves the monitor up and down so that the dot between the users eyes is vertically even with the blue line across the interface. It also rotates left and right to make sure that users face is in the horizontal center of the screen.
4 Discussion

4.1 Achievements

We were ultimately able to achieve our main goal of building a robot that could lift a monitor and position it for the user. The face tracking and eye tracking algorithms are extremely accurate, and the robot is able to quickly make any adjustments to compensate for the users position.

The face recognition works well in consistent conditions when the user has enough training images. We were able to successfully identify users in real time and ‘log them in’ to our application.

The final goal of determining whether the user was sleeping or not was not accomplished. The technique of using edge detection to detect an open eye or a pupil proved to be impossible on low resolution images coming from a webcam.

4.2 Future Work

The next steps for this project would be improvements that enhance and fine tune functionality.

1. An improved monitor stand built out of metal and servo motors would be much better than the current robot built out of Legos.

2. A face tracker that could track the user off the screen.

3. A face recognition algorithm that continually identifies the user in real time, instead of just logging them in, would offer greatly improved security. This way it would be impossible for any other user to access the application, even if the authorized user forgot to log out when they left.

4. A sleep detection algorithm that could tell the difference between closed and open eyes with high accuracy.
5 Current Trends in Robotics

Robotics technology has captured the human imagination for generations. The possibility of producing machines which exhibit high orders of intelligence has often both fascinated and frightened the masses with their potential. The application for this type of technology is virtually limitless and its implementation currently is only very basic. Currently simple robotics is heavily used in factory production lines such as automobile production lines. However with recent advances in computer vision, sensor, controller and materials technology more complex robotics can now be designed and produced. Among the most intriguing applications of robotics is the idea of using robotics to perform superhuman tasks. When superhuman robotics is discussed usually the topics of exoskeletons and robotic soldiers come up since the potential for military uses are great. However in the medical industry, superhuman robotics also holds a great potential, posing striking implications for the way medicine is practiced in the next decade.

Currently the most prominent use of robotics in the medical world is the rising wave of surgical robots being installed throughout hospitals worldwide. There are two general categories of surgical robots systems, CAD/CAM robot systems and surgical assistance systems. CAD/CAM systems are usually used to design product parts before physically building or manufacturing them. Similarly in the medical domain, the planning done before a medical procedure is considered the CAD system while the medical procedure represents the CAM system. Both elements form a closed system in that first a patient specific treatment model and plan are obtained and then synced or "registered" to the patient in the real 3D world.

One well known example of a CAD/CAM system in surgery is the ROBODOC system which modeled, planned and performed total hip and knee replacements surgeries through the surgeon’s use of a computer software interface. The software and hardware components of the robot can then be registered in the real world by the surgeon. The surgeon would conduct this registration by graphically positioning a 3D model of the prosthesis that needed to be installed on the generated CT image of the body region of interest. Once registration is completed, the robot can carry out its assigned functions such as drilling into bone and affixing plates and prosthesis to bone.

Surgical assistance systems, in contrast to CAD/CAM systems which performs the surgeon’s work for them, is used more as an extra augmentation tool for surgeons to perform his/her work to a higher standard. Two basic tenets exist for designing surgical assistance systems. The first tenet is to improve the surgeon’s sensing and manipulation ability. The second tenet is to increase the number of sensors and manipulators available to the surgeon.
The coupling of these two philosophies can give the surgeon superhuman abilities such as X-ray or infrared vision, elimination of hand tremor and the ability to perform delicate and minute surgical procedures in a very small area successfully. Certain assistance systems can also operate as telesurgery systems, which allows for the surgeon to perform surgery at a remote location from meters away to thousands of kilometers away [5]. Control of the system is usually through fine joystick control, head tracking, voice tracking and robot system AI.

One popular surgical assistance system is the da Vinci surgical system. This system is a telesurgery system that demonstrates both of the previous mentioned tenets for assistance systems. The system consists of a patient-side slave robot and a master control console which is usually controlled by the surgeon. The slave robot consists of four arms that are able to manipulate stereo endoscopes in addition to being capable of manipulating other surgical instruments such as retractors, scissors, grippers and needle holders. The master control console contains joystick-like handles which are attached to two dexterous master manipulator arms which the surgeon can use with limited amount of force feedback. The hand motions of the surgeon are sensed by the master manipulators, which are then sensed and mimicked by the slave arms. From the inserted endoscope (in the patient) the surgeon can have complete view of the site of operation through stereo video transmission into the master control console and can manipulate the arms in the zone of operation while viewing the stereo video transmission streaming from the endoscope. The added advantage of the usage of mechanical manipulator arms is that they can create small incisions in the patient (small enough for the endoscope to go in) while allowing the surgeon to have full view of the operable material under the skin [4].

The minimally invasive technique of the da Vinci surgical assistance system will lead to shorter patient recovery time and shorter hospital stay, yielding a higher patient satisfaction. Currently surgical robotics are very expensive but like all novel devices, as demand grows, prices will decline, eventually making surgical robotic devices, essential equipment for any major hospital to have at their disposal [5, 6].

From this short overview of surgical robotics it can be seen that the influx of robotics into surgery and medicine has only just begun but the potential for robotics in medicine is extraordinary. As robotics technology advances, robotization of medicine will yield faster, more accurate, safer and eventually cheaper results than currently produced; yielding lower post-surgery fatalities or injuries and ultimately creating increased consumer peace of mind and satisfaction.
6 References


7 Appendix

see attached code.
8 Statements of Contribution

Cogan Noll

- wrote GUI for the application
- wrote the code for the face recognition feature
- wrote the code for the eye state detection feature
- wrote the code for final implementation of the robot positioning algorithm
- assisted in mechanical design of the robot
- wrote the corresponding sections of the report

Kermen Deol

- assisted in GUI development
- wrote the code for the face detection feature
- wrote the code for the face tracking feature
- wrote the code for the eye detection feature
- extended monitor cables and assisted in robot design
- wrote the corresponding sections of the report

Praveen Sitaraman

- assisted in mechanical design of robot
- worked on development of a Lucas-Kanade tracker
- wrote the ‘Current Trends In Robotics’ and several other sections of the report