Project report on

REAL TIME HAND GESTURE RECOGNITION AND BLINK DETECTION

Done by
Parneet Kaur
Prateek Prasanna

Electrical and Computer Engineering Department
Rutgers University

Spring 2011

Contents

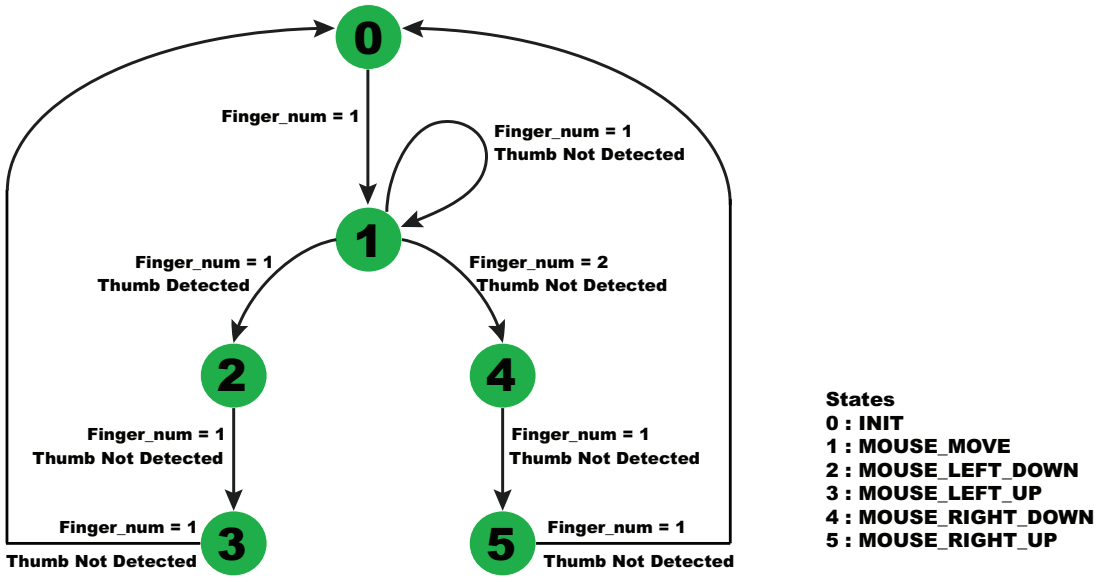
1 Objective

Human Computer Interaction(HCI) plays a vital role in day to day activities. Use of computer vision has taken HCI to an altogether different level. In this project, a simple webcam is used for tracking hand and eye features in real time by incorporating computer vision and image processing techniques. The features to be detected include fingertips and thumb for hand and blink detection for eyes. Our objective is to implement all the mouse tasks without the use of mouse. Fingertip count and hand gestures are used to implement mouse movement and left and right click.

2 Procedure

2.1 Hand Gesture Recognition

For detecting a hand gesture, first the hand needs to be recognized. Then its features can be detected using contours as discussed further in section 2.1.1 . Finally the features extracted are used for making gestures recognized by a system as input. In this project, the designed system is capable of identifying three different gestures, one for each mouse movement, left click and right click. To make sure that only one of the three required tasks is performed, a finite state machine is implemented as in figure 1. The gestures are as shown in figure 4 .As soon as index finger-tip is detected, the mouse movement is enabled and the pointer is updated as described further in section 2.1.4. When mouse movement is enabled and a thumb is detected, a left click is performed. Similarly, a right click is performed when the two fingers detected, provided the mouse movement is enabled. Since a mouse click comprises of a *button down* and *button up*, transition between states is required in order to know that the button has been released. Proper entry and exit conditions in the state machine ensure smooth functioning of the gesture recognition system.



(a) State machine

(b) Different states

Figure 1: Gesture Recognition Implementation

In next sections we will describe how each gesture is recognized to control the mouse pointer.

2.1.1 Hand Detection

Hand detection is most critical for successful gesture recognition. We tried different methods to achieve this goal. In first method, if the background is kept constant, then the hand can be segmented as foreground.This method gives good results but the user's arm is also recognized as foreground which is not as required. To overcome this limitation, HSV(Hue Saturation Value) color space is used to segment the skin color and recognize hand. HSV has better illumination resistance and segmentation capability when compared to RGB and YCrCb color spaces. Results are very good when the light source is behind the webcam but when it is from another direction(specially from above), the result is poor. HSV color method is used to demonstrate rest of the work. The segmented hand, when converted to binary image, looks as in figure 2. For efficient tracking, it is better not to have the other hand and user's face in the image being captured by the webcam. It is noteworthy that this detection algorithm is robust to presence of faces as long as the user's face is not very close to the camera.

(a) Background subtraction. Result is good but arm can not be segregated.

(b) HSV color technique without noise removal

(c) HSV color technique with noise removal. Note that Peak and Valley points are less sharp.

(d) HSV color technique with improper illumination gives inappropriate result

Figure 2: Hand segmentation by Background Subtraction and HSV Color Method

However, we found that this method is also sensitive to illumination conditions and also

depends on the camera being used. Section 4.1 discusses the dependency of HSV color space on cameras.

As another approach, we created a Haar classifier for hand detection. If hand can be detected using a classifier, the region of interest will be small and also user's arm can be effectively segregated from the hand. Face-position constraint would also be eliminated. However, the results were not as desired as discussed in section 4.1.

2.1.2 Finger-tip Detection

After successful detection of hand, the next step is to find the fingers so that they can be used for recognizing gestures. This is done by the method of contour detection. Contours are sequences of points defining a curve in an image. After obtaining the binary image, the contours in the whole frame are found. Since the hand is closest to the webcam, it corresponds to the contour with the biggest area. Using convexity hull and convexity defects functions, the *peaks* and *valleys* of the hand are detected. These functions give multiple points around the hand. To detect the reliable points, those points are selected which are at a distance greater than 25 pixels in both x and y directions. This reduces the number of defects from more than 30 to less than 10. Next, the distance of each *valley* point from hand's center of mass is calculated. The minimum distance is the hand's palm size. To find which of the *peak* points are fingertips, the distance from hand's center of mass to each *peak* point is computed and it is verified that the *peak* points lie above the hand's center of mass. All these features are shown in figure 3. If the distance between the hand and the webcam is kept constant, the peaks with distance greater than a threshold are recognized as fingertips. Initially, we fix this threshold to twice the radius of hand's palm size. However, since the contour size varies rapidly because of the illumination variance, the radius also varies rapidly. So, we removed this check and introduced another check. A bounding rectangle is drawn around the hand contour. The *peak* points corresponding to fingertips lie in the upper region of the bounding rectangle between the topmost part till almost one-third of rectangle's height.

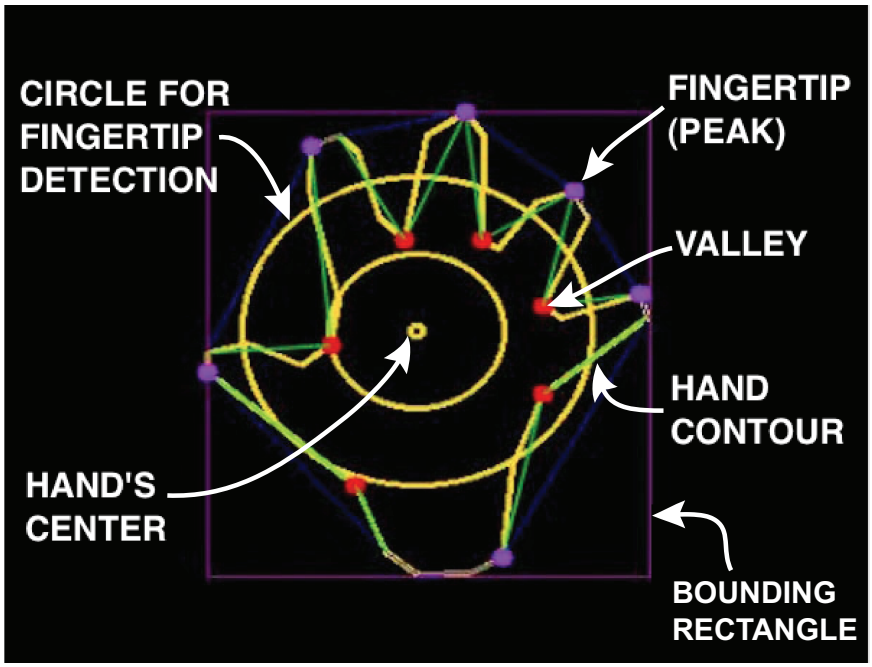


Figure 3: Hand Features. Note that the fingertips lie in the upper part of the bounding rectangle.

2.1.3 Thumb Detection

Thumb detection is required to implement the left mouse click. A bounding rectangle is drawn around the contour as explained in section 2.1.2. The thumb is searched for within this bounding rectangle in the binary image. Starting from leftmost column, each column is scanned for the white pixels. After analyzing the pixel count in each column for many frames, it was found that the thickness of the thumb varies between 15 and 45 pixels and the length of thumb is 70 pixels.Since the bounding rectangle contains only hand region, it is enough to observe 45 continuous columns having pixels between 15 and 45, which avoids computational overhead. It is possible that in the right side of the bounding rectangle fist is detected as thumb, mostly when the hand is not detected completely due to insufficient illumination. To avoid this false detection the bounding rectangle is scanned for finger or fist from left to right direction. It is sufficient to detect 60 continuous columns with pixel values 60 and above to assert that a thumb is not expected after a finger or fist has been detected. This further reduces the computation overhead since only initial few columns of bounding rectangle will be scanned in each frame.
This technique of thumb detection works for right handed person only. For a left handed person scanning for thumb needs to be done from right to left direction.

(a) For mouse movement (b) For left click (c) For right click

Figure 4: Hand Gestures

2.1.4 Mouse Control

Since we are able to detect the required hand features and thumb successfully, these features can now be successfully used to control mouse. Initially hand's center was used for moving the mouse to an absolute position. In order to be able to move mouse with index fingertip, the detection is stabilized by updating the contour computation and detecting finger tip every 3rd frame captured by webcam, rather than updating it every frame. This makes the mouse movement very slow. A smoother mouse movement is obtained by stabilizing the hand's center of mass by averaging it over 30 frames. However this method is dependent on the whole contour being detected, so it is not very reliable. A better option will be to have a point which is relatively stable. Index finger-tip is one such point and it also looks like a good user interface. In section 2.1.2, all the finger tips are detected and counted. If the number of fingers detected is 1, then the finger position is computed. It can be obtained directly by using the *peak* point corresponding to index finger computed by convex hull algorithm. However, it doesn't give a stable result as even a slight change in the index-finger's contour will change this point. To stabilize the point another approach is used. When the number of fingers detected is one, the finger is searched for again in the binary image of hand contour obtained in section 2.1.1. The binary image is scanned horizontally from top to bottom for continuous white pixels in each row in the range 15 to 45 and the first pixel coordinates are stored. If 50 such consecutive rows

are found, this asserts the presence of finger and the saved pixel coordinates are used for mouse position calculation. In case of any discontinuity, the stored pixel coordinates are discarded and the scanning continues.

Once the finger position is obtained, we need to use it for computing mouse pointer position. The captured image and monitor differ in their respective resolutions. In order to cover the complete monitor screen the captured image needs to be mapped to the screen resolution. We implemented two approaches to achieve this goal. In first approach, the captured image is resized to the screen resolution using openCV function *cvResize*. In second approach, we compute the coordinates of the mouse pointer from the captured image as:

$$x_{mouse} = x * \frac{W_S}{W_C}, y_{mouse} = y * \frac{H_S}{H_C}$$

where,
(x, y) are coordinates of the center of the hand from captured image
W_C x H_C is resolution of the captured image
W_S x H_S is resolution of the monitor screen

In our project, the second approach is integrated because we observed that it gives better mouse control.

The whole hand has to be present in the image for detecting the hand features. This further limits the monitor screen that can be covered. To map the whole screen, we divide the image into half by drawing a horizontal line User has to keep the finger-tip above this line. The coordinates of the mouse pointer computed above are multiplied by a factor of 2. This allows the coverage of whole monitor screen.

Next we need to implement mouse clicks which comprises of *button down* and *button up* by setting the required flags in the registry of system. The state machine in figure 1 is used to implement the clicks. When a thumb is detected as described in section 2.1.3, the state changes from *MOUSE_MOVE* to *MOUSE_LEFT_DOWN* and a left button down action is performed. When the thumb detection gives negative results, the state changes from *MOUSE_LEFT_DOWN* to *MOUSE_LEFT_UP* and a left button up action is performed. Similarly, the state changes from *MOUSE_MOVE* to *MOUSE_RIGHT_DOWN* when number of fingers is two and then to *MOUSE_RIGHT_UP* when the number of fingers is one again.

Thus, left and right clicks are successfully implemented using hand gestures.

2.2 Eye Feature Extraction

Eye gaze detection is challenging because it is sensitive to head movement and changes in illumination conditions. To overcome this, in many implementations, Infrared(IR) headmounted systems are used [6] [11]. Because of the system limitations, for now, only eye feature extraction and blink detection are implemented. This part of project is based on the approach described by Ibrahim et al. [4].

2.2.1 Eye Region Detection

To extract the eye region, first the face is detected in the video using OpenCV Haar classifier. Then as in figure 5 the Region of Interest(ROI) for eyes is selected by choosing appropriate height and width specific to user. Only slight variations have to be done to adjust this ROI for another user.

A haar classifier trained on eye images can also be used for detection of eye region. A positive image set of 1000 images and negative image set of 3000 images is used to train the classifier. We are awaiting results for this and hopefully will be able to demonstrate this.

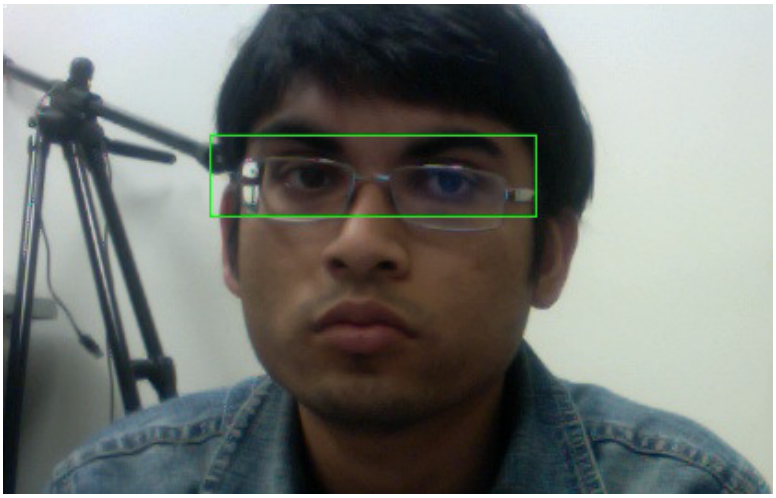


Figure 5: Eyes ROI extraction from face detection Haar classifier

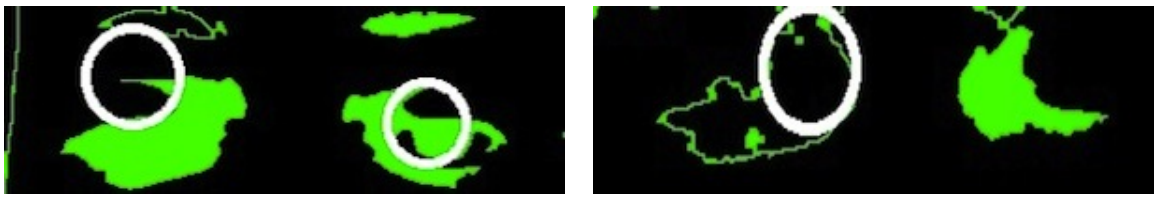
2.2.2 Iris Detection

Histogram equalization is used on the grayscale eye-image from the previous step to achieve contrast adjustment using the image's histogram. This method usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values. This allows for areas of lower local contrast to gain a higher contrast without affecting the global contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values. The grayscale image is converted into binary form by a particular threshold value which is chosen to be 60 in this case but subject to change according to lighting conditions. To obtain the edges of the image, a Laplacian edge-detection algorithm is used with a 3X3 convolution matrix. This method is a high-pass edge-detection method.

Blob extraction function is then applied to obtain the blobs with area smaller than a threshold value. By repeated trials, we fix the threshold area as 1000 pixels. This value is sensitive to illumination conditions. As shown in figure 6, the detected blobs are filled with green color. They also include the iris region.

Figure 6: Blob detection and Hough Transformation applied on open eyes. Note that more green area is enclosed for open eyes.

After detection of blobs, circular hough transformation is applied to the images which finds the circular regions in a given image. Our intention is to detect the exact center and the radius of the iris, thus enabling computation of the green pixels enclosed by the hough circles. But, it is observed that application of this results in false detection of circles at times i.e., the hough transformation detects incorrect circles which don't overlap with the blobs found. Also, when the eyes are closed, sometimes only one circle is detected and the radius obtained is not consistent with the actual radius of the iris. In next section, we propose another method, relating to total blob area computation that enabled us to detect blinks.



(a) Partially closed eyes (b) Closed eyes

Figure 7: Blob extraction (green pixels) and Hough Transformation(white circles) applied on partially closed and closed eyes. Note that as the eye opening reduces, number of green pixels in the circle also reduce.

2.2.3 Threshold Computation

The green pixel count found in section 2.2.2 varies depending on the illumination in the room. Also the percent area covered by green pixels for closed and open eyes varies by a small margin. If we fix a single threshold, the false detection is very high. So, we decided to have a training phase in which two thresholds are computed. Around 10 images are taken with eyes open. The percentage area covered by green pixels is computed and averaged for the 10 images. Same procedure is repeated for closed eyes. The mean of both the values is averaged and two thresholds are defined:
1. $THRESH\_OPEN = mean + 0.5$
2. $THRESH\_CLOSE = mean - 0.5$

If the percentage area lies between the two thresholds, it is considered as a "reject" option. Based on these two thresholds, the flags *Open* and *Close* are set as shown in figure 8 and are used in state machine discussed in next section.

percent area > THRESH_OPEN
=> Open = 1

percent area < THRESH_CLOSE
=> Close =1

THRESH_ CLOSE < percent area < THRESH_OPEN
=> Open = 0, Close = 0

Figure 8: Decision Making for Open and Close Eyes

2.2.4 Blink Detection

Based on the decision whether the eyes are open or closed, the state machine as shown in figure 9 is implemented. A blink is detected only when the state machine goes from state 1 to state 2 i.e., when eyes are open in immediate previous state and closed in present state. Message is displayed on the video till eyes are closed. If the eyes are open or the percent area lies between the two thresholds, we go to state 1 and state 0, respectively. Since we don't go from state 1 to state 2 when the percent area lies between the two thresholds, lot of false detection is avoided.

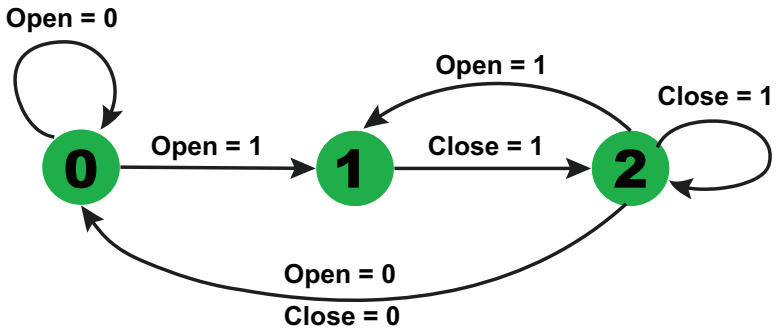


Figure 9: Blink Detection: State Machine

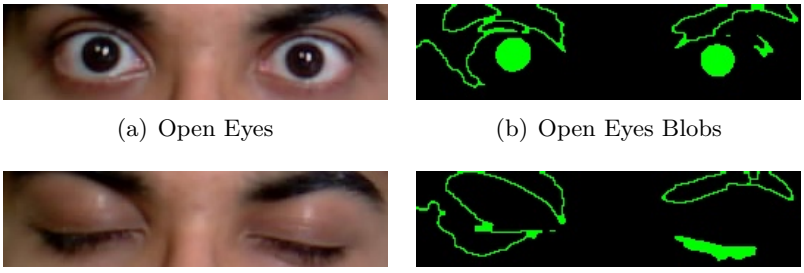
3 Experimental Results

We are successfully able to implement mouse movement, left click and right click by using hand gestures. The response time is good, thus, making it useful for a real time application.The success rate of hand gestures is 100% in good illumination conditions where the hand is detected properly. However, it decreases when the illumination is not good as shown in figure 10.

	Total	Detected	Success Rate
Mouse movement	100	74	74
Left Click	70	50	71.43
Right Click	70	43	61.43
Blink	61	54	88.5

Figure 10: Success Rate in Improper Illumination conditions.

Snapshots of open and close eyes and their corresponding blob outputs are as shown in figure 11. Using the procedure described in section 2.2, we are able to detect voluntary blinks with a success rate as of 88.5%. Since the processing is done at a lower frame rate, the chances of algorithm detecting involuntary blinks is reduced.

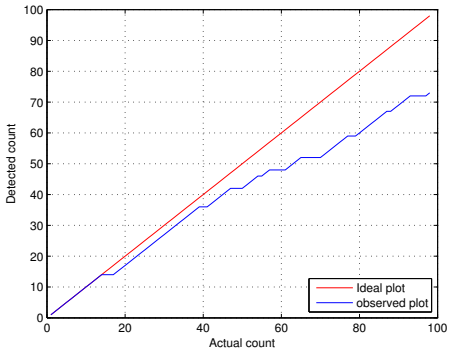


(a) Open Eyes (b) Open Eyes Blobs

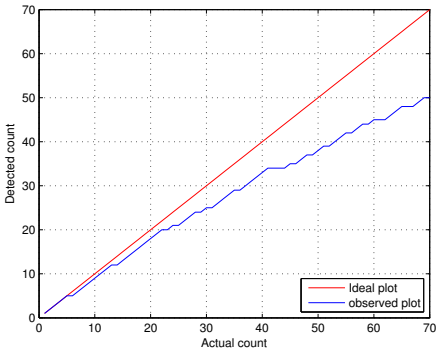
(c) Closed Eyes (d) Closed Eyes Blobs

Figure 11: Blobs for open and close eyes. Note that the area covered by the green pixels is more for open eyes.

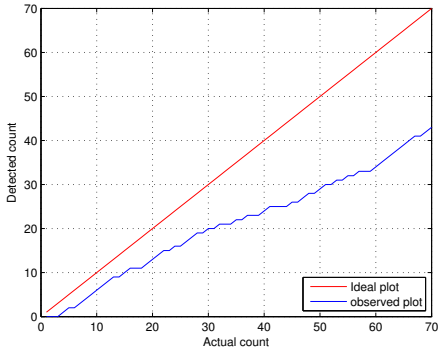
The error analysis for hand gestures and eye blink detection is as shown in figure 12. It is observed that the observed plots are not consistent with the ideal ones. The error is due to the improper illumination conditions. If the illumination is positive, hand is detected properly and then mouse movement and left clicks are 100% accurate. Sometimes false positives occur for



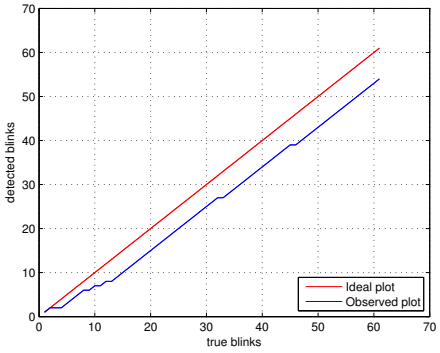
(a) Error Plot for Index Finger Detection for Mouse Movement



(b) Error Plot for Left Click



(c) Error Plot for Right Click



(d) Error Plot for Blink Detection

Figure 12: Error Analysis in Improper Illumination

right click. Their count is negligible in general, but increases with improper lighting. The False positive cases also arise in the blink detection test, but their counts are negligible.

4 Discussion

4.1 Difficulties and Sources of Error

1. Initially, we faced difficulties in deciding the area of the blobs to be considered for their subsequent extraction. This was eliminated by running the code on images taken at different illumination conditions and deciding that value for area that gave the best results in all such conditions.
2. We observed that the same hand detection code, when run with input from different cameras give different results even when the background is same. Figure 13 shows the results from three different cameras. The reason is that the webcams we are using are not high-end webcams made for efficient image processing and they compress the video input before sending it to the machine. This effects video quality and also the HSV color space video.
3. The classifier we obtained detects hand only sometimes and it also detects the background in the positive images in our training set. We suspect that the reason for false detection was a small positive image set (we took 550 images). Also, the background was more in the images, so the classifier got trained on the background partially.

(a) Logitech Camera (b) Dell Laptop (c) Apple iSight Macbook Pro

Figure 13: HSV colorspace images from different cameras.

4. Fingertip detection program was extremely slow in the beginning. So, memory allocation and deallocation was done very carefully to avoid any memory leakage. Resulting program gives a very good time response.
5. When implementing mouse movement, it was observed that the contour size and consequently the center point of hand change very frequently. The frequent change makes the mouse movement very unstable. To overcome this problem, mouse pointer position is updated every 30 frames. Also, before updating it is checked if the movement is above a specified threshold. We specify the threshold as 10 pixels in x-direction and 5 pixels in y-direction. This gives a smooth mouse movement. Later, we used a fingertip for mouse movement stabilizing it in a similar manner by averaging the movement over 10 frames.
6. We were not able to use the OpenCV SET_CAPTURE_PROPERTY functionality to set a lower frame rate because it is buggy. So, we decided to skip some frames inbetween.

4.2 Future Work and Extentions

It was a great learning experience for us to implement this project. In future, we would like to improvise some aspects of this project and extend it further. Segmenting hand from the colored background was the most challenging task and has lot of scope for improvement. The algorithm implemented in 2.1.1 doesn't give appropriate results and effects rest of the algorithms. We tried to implement an online training algorithm for skin detection but it was not successful. It will be a good idea to implement an adaptive method based on skin training using CAMSHIFT algorithm as discussed in [8]. Also, optical flow can be used to determine the motion. One such implementation is done in [5]. In our project, we implemented only simple mouse functionalities. For implementing mouse features like scrolling, zoom in, zoom out etc, position of fingers can be combined with optical flow algorithm to identify the gesture made. For gesture recognition, classifiers can be trained for different hand poses and the recognition can be done using K-Nearest Neighbor method. If poses can be identified correctly, a very useful application will be to be able to understand the sign language.

In the second part of project, blink detection is successfully implemented in proper illumination conditions. It will be interesting to be able to distinguish between drowsy and open eyes. Also, if drowsiness and blink detection can be made robust, a good application will be to be able to analyze driver's drowsiness on highways. One such interesting implementation is discussed by R. Garg *et al.* in [2]

5 Current Trends in Robotics

Robotics in Rescue Engineering

The recent earthquake in Japan has once again brought to limelight the use of robotics technology in disaster and rescue management. Japan's leading experts in rescue robotics deployed wheeled and snake-like robots to assist in search for survivors and control of leakage in the nuclear plants. It in fact is not entirely possible to avoid natural disasters but the sufferings can be minimized to a great extent by developing strategies and being prepared in such a way so as to cope with the immediate aftermath. This is where robotics comes to great asset as they can withstand the hazardous conditions which humans cannot.

The area of Robotics that comes into great use in this field includes, but is not limited to, mobility, sensor networks, navigation and artificial intelligence. Robots employed in such tasks are called *Service Robots*. Different classes of service robots are robots that replace humans in hazardous environments (radioactivity, military, fire-fighting etc), robots that work along with human beings (entertainment, housekeeping, rehabilitation etc) and robots in medical research[3].. Unmanned Aerial Vehicles, Under Sea Vehicles and Unmanned ground vehicles are a few of the robots that are being deployed in rescue missions. Robots can make important contribution and be integrated into a socio-technical system for disaster management. Surface conditions can be monitored and real time information can be collected from a disaster site by robots. This helps assist emergency personnel mobilize an effective response, either autonomously or in co-operation with human operators.

Many countries have been significantly inventing in robotics projects for disaster management. An interesting project named 'Survivor Buddy' has been has been carried out by Stanford and Texas A&M University under the patronage of Dr Robin R Murphy. This project involves victim management and human-robot interaction. A multimedia robot head was created that could serve as an interface between the trapped victim and the outside world. Another project, 'RoboCup Rescue' [10] is one of the pioneers in fostering research and development in this field. It focuses on building robust information systems that is very much inter-disciplinary and aims at building robust systems that enhance victim survival rates, reducing risk to the rescue personnel at the same time. At Germany's Ilmenau University of Technology, research is being carried out to develop flying quad-copter robots. Just like a copter, such a robot can fly to a disaster area, position itself on a higher place and then set up temporary Wi-Fi and mobile networks infrastructure. They are equipped with GPS and radio equipments , but require costly batteries.

In 2002, for the first time, Japan Ministry of Education, Sports, Culture, Science and Technology launched a national project, Special Project for Earthquake Disaster Mitigation in Urban Areas (DDT Project) [9] which aimed at involving advanced robots for disaster response. As part of this project, Serpentine Robot were researched which need not move autonomously but can successfully intrude more than 30 meters, take 3D information and have intelligence for human recognition. Aero Robot (Intelligent Helicopter) which have radiation detector, 3D camera and temperature, inflammable gas and humidity sensors were also in DDT Project's agenda.

The first time ever that robots were used for urban research and rescue missions (USAR) was immediately after the 9/11 attacks on the World Trade Center[7]. CRASAR, (Center for Robot-Assisted Search and Rescue) working independently under Army Reserve National Guard, New York Fire Department and New York Police department, played a very important role in the rescue activities. The robots used for the search operations were developed in the Tactical Mobile Robots program (sponsored by US Dept. of Defense) or used by contractors in the program. Unlike other natural disaster scenarios, the WTC disaster was quite different. As the

rubble was all steel, the voids were very small (less than 1m wide) and traditional approaches (like human or canine search) werent feasible. In order to formulate effective strategies, rescuers were required to see the interior of the rubble. To aid this purpose, small robots that could enter places still on fire or posing risk to structural collapse, were used. Larger bots like Packbot and SPAWAR Urbot were used in buildings where destruction was minimal. The robots with cameras and two-way audio were teleoperated through Operator Control Units. There were many problems such as lack of depth perception in video-cameras, and lack of peripheral vision or feedback. Even though there werent any survivors found, this operation resulted in robots getting increasing acceptance by the rescue community.

Apart form the wheeled and snake bots mentioned above, two PackBot ground robots from iRobot were deployed in the crippled Fukushima nuclear power plant to take readings of temperature, oxygen levels, and radioactivity. Because of the radioactivity levels were impossible for human beings to access the facility, aero robot called 'Monirobo' (Monitoring Robot) and U.S. Air Force drone "Global Hawk" were deployed to observe the situation aerially. Since snakebots are the most widely used robots in disaster scenes, the next paragraphs highlights a few technical aspects of the serpentine robots.

Snakebots are extremely flexible and nimble on wheels and thus can easily ascend obstacles.Unlike rovers which are expensive and large, snakebots are very compact and cost effective. It is more like a caterpillar, apart from turning at corners, can also slither through even inch-wide gaps and climb inclines. Such robots are voice-controlled and emulate the sinusoidal motion of snakes. Speech commands are first converted into motion commands by speech recognition software, and then transmitted to the robot via infra-red link using on-off keying technique. The control PCs parallel port is used to interface the IR transmitter. The IR receiver is located on the tail module of the robot. Thus, the design of the bot provides the following essential features:

a. Combined torque produced by the various modules helps in traversing uneven terrain.
b. The S-shaped curve helps in moving around obstacles
c. The deforming of one module doesnt affect the others which continue to follow their path

The potential of Robots in the field of disaster management hasnt been exploited to the fullest extent because of hinderance by many factors [1]. First and foremost, a major drawback comes in the form of the huge expense involved at least during the initial investment. Setting up of new robotics equipment is usually very costly. According to a report entitled National Robotics Technology Roadmap, US lags behind in robotics research because of insufficient investment. Secondly, such robots are very application and environment-specific. Special robots are required to perform different tasks. This adds to the increase in cost.

Nations, like Japan, Korea and US, currently lead the race in such technology. With rapid industrialization and progress in other developing nations, it has become even more important to have rescue robots that make the social system secure and safer. Mutual collaboration and resource sharing among nations helps a great deal in this respect.

References

[1] A. Boucher, R. Canal, T.-Q. Chu, A. Drogoul, B. Gaudou, V.T. Le, V. Moraru, N. Van Nguyen, Q.A.N. Vu, P. Taillandier, F. Sempe, and S. Stinckwich. The around project: Adapting robotic disaster response to developing countries. In *Safety, Security Rescue Robotics (SSRR), 2009 IEEE International Workshop on*, pages 1 –6, 2009.

[2] R. Garg, V. Gupta, and V. Agrawal. A drowsy driver detection and security system. In *Ultra Modern Telecommunications Workshops, 2009. ICUMT '09. International Conference on*, pages 1 –8, oct. 2009.

[3] Maki K. Habib and Yvan Baudoin. Robot-assisted risky intervention, search, rescue and environmental robot-assisted risk intervention search, rescue and enviormental surveillance. In *International Journal of Advanced Robotic Systems,*, volume 7, 2010.

[4] Ibrahim Furkan Ince and Tae-Cheon Yang. A new low-cost eye tracking and blink detection approach: Extracting eye features with blob extraction. In *Emerging Intelligent Computing Technology and Applications,ICIC 2009*, volume LNCS 5754, pages 526–533, 2009.

[5] M. Kolsch and M. Turk. Fast 2d hand tracking with flocks of features and multi-cue integration. In *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW '04. Conference on*, page 158, june 2004.

[6] Dongheng Li, D. Winfield, and D.J. Parkhurst. Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches. In *Computer Vision and Pattern Recognition - Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, page 79, 2005.

[7] R.R. Murphy. Trial by fire [rescue robots]. *Robotics Automation Magazine, IEEE*, 11(3):50 – 61, sept. 2004.

[8] S.M. Nadgeri, S.D. Sawarkar, and A.D. Gawande. Hand gesture recognition using camshift algorithm. In *Emerging Trends in Engineering and Technology (ICETET), 2010 3rd International Conference on*, pages 37 –41, 2010.

[9] S. Tadokoro. Special project on development of advanced robots for disaster response (ddt project). In *Advanced Robotics and its Social Impacts, 2005. IEEE Workshop on*, pages 66 – 72, 2005.

[10] S. Tadokoro, H. Kitano, T. Takahashi, I. Noda, H. Matsubara, A. Shinjoh, T. Koto, I. Takeuchi, H. Takahashi, F. Matsuno, M. Hatayama, J. Nobe, and S. Shimada. The robocup-rescue project: a robotic approach to the disaster mitigation problem. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 4, pages 4089 –4094 vol.4, 2000.

[11] Zhiwei Zhu and Qiang Ji. Novel eye gaze tracking techniques under natural head movement. *Biomedical Engineering, IEEE Transactions on*, 54(12):2246 –2260, 2007.

Appendix

Handed as hardcopy.