18-551: Final Report

Fall 2005

Group 4, Final Report

The Handimouse

Wilson Chai (wchai)

Wai Cheung (Eric) Chu (wcc)

Ting (Patty) Pun (tpun)

Motivation

As computers become more and more involved in all aspects of our lives, there is an increasing number of computer-related disorders such as RSIs (repetitive stress injuries) and RMIs (repetitive motion injuries). The main causes of these injuries are often repetitive, forceful and awkward hand movements due to bad designs of computer accessories. Mouse as an input device is convenient and intuitive, however the physical implementation of traditional mouse constraints user's hand movements and postures. As a result prolonged mouse usage, unhealthy postures and bad mouse design will increase the risks of such types of injury.

Solution

Our goal is to design a user-friendly mouse that helps to prevent hazards such as RSIs and RMIs. The solution that we come up with is to utilize the user's own hand to act as the mouse. The advantage of such a design is that the hand of the user will be placed naturally on a flat surface, and hence the pressure on the hand will be spread uniformly on the palm instead of concentrating most of the pressure on the wrist. This alleviates stress on the hand and helps to prevent the user from suffering muscle fatigue as a result of prolonged usage. We named our design Handimouse. Besides the fact that Handimouse can help to prevent RSIs and RMIs, it is also easy and simple to use. It requires no significant adjustment or training from the user side. Thus, it is very intuitive for the users to switch from traditional mouse to Handimouse.

Overview

The set-up of Handimouse includes a webcam, a piece of blue paper, EVM, and a PC. The user can define his workspace by using a black marker to draw a rectangle grid on the blue paper; only movements and actions within the workspace will be detected and executed. Webcam should be placed at an elevated position in order to capture the entire workspace. This completes the initial hardware set up of the device.

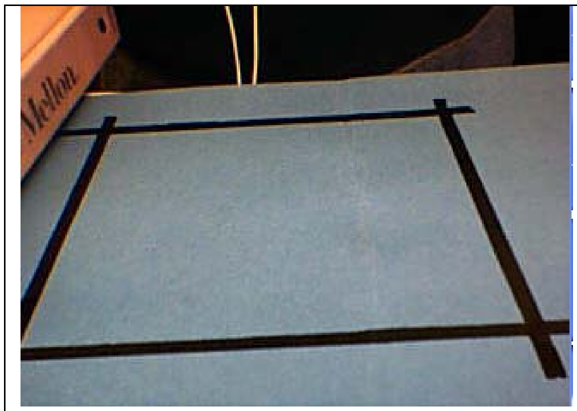
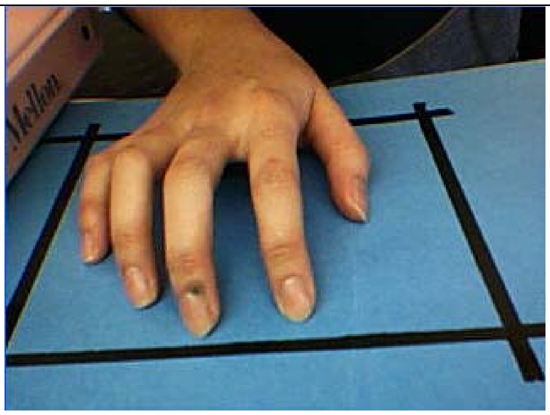
	

Figure 1. Image captured by the Webcam: workspace (the rectangle grid) on a blue paper	*Figure 2. Image captured by the Webcam: Handimouse in operation*

The following describes the detail operation of Handimouse. Once the application starts, Handimouse will be activated when user place his hand inside the workspace. User's thumb position in the workspace will be mapped to the cursor position on the computer monitor in a one-to-one scale. For example, if the user places his thumb in the lower left hand corner of the grid, the mouse cursor will carry out the corresponding movement. Our program will detect the motion of the index finger and the middle finger to control the left and right mouse button respectively. Since webcam cannot detect pressure applied on a surface, mouse-button-down is signaled by holding up the finger, while the mouse-button-up is signaled by putting down the finger in the workspace. This may be unnatural for dragging action, since the user will need to hold up his index finger instead of pressing down. However clicking action and double-clicking action would still be the same as using a traditional mouse, i.e. bringing finger up then putting down defines a button click.

Division of Labor

The following data flow diagram shows the interaction between the user interface, PC, and EVM.

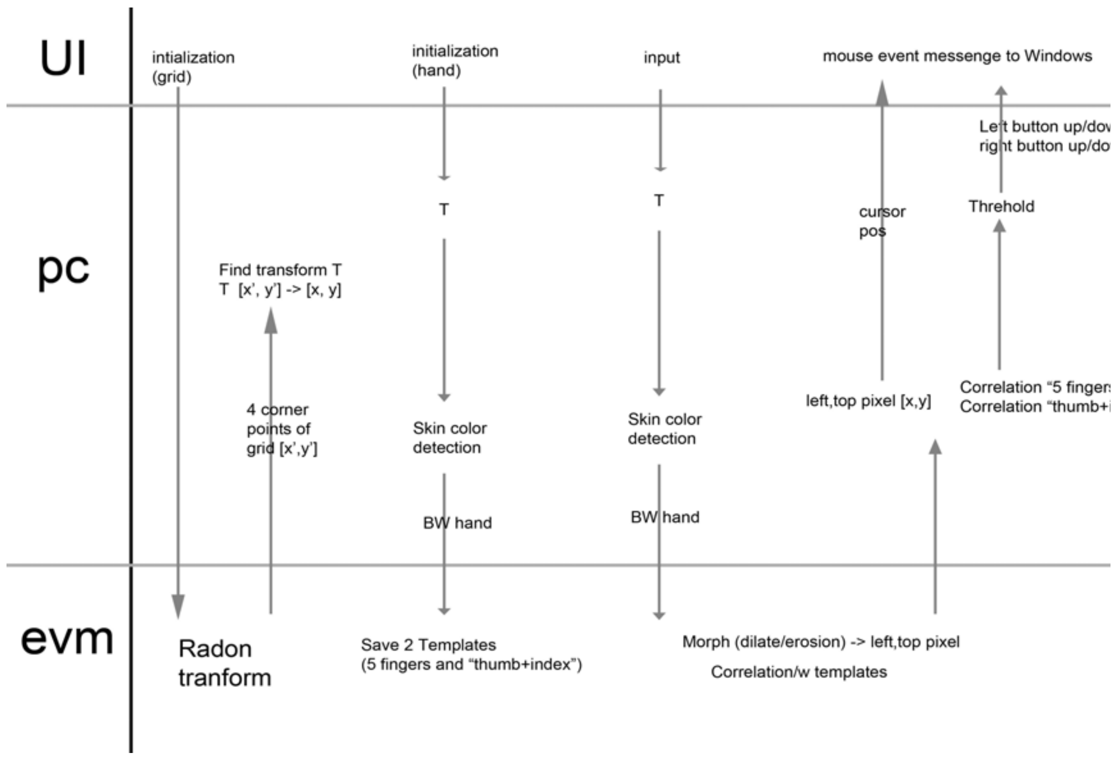


Figure 3: Application Flow: Division of Tasks among UI, PC and EVM

User Interface and PC Side:

The User Interface is on the PC Side as well. It guides the user through and handles the initial set up of the Handimouse, i.e. it collects the data for the transformation matrix from image-space to monitor-space, and it also collects the templates for defining the right and left mouse button click.

The PC handles most of the static computations. The tasks of the PC include:

1. Handling data transfer between Webcam and EVM, and also data pre-processing before sending to EVM;
2. Downsampling of the images;
3. Conversion of RGB to HSV, and determination of Luminosity Threshold for defining the workspace in image-space (refer to the Morphological Operations in the Algorithm Matrix for procedure details);
4. Obtaining the Transformation Matrix for the workspace between image-space and monitor-space, and also the workspace mapping between image-space and

monitor-space (refer to Transformation Matrix in the Algorithm section for procedure details);

5. Skin Detection on the raw input images (refer to Skin Detection in the Algorithm section for procedure details);
6. Defining the templates for correlation of hand postures (refer to Pattern Recognition/Correlation in the Algorithm section for procedure details);
7. Defining the correlation input from the raw image (refer to Pattern Recognition/Correlation in the Algorithm section for procedure details).

We will discuss the detail of the above procedures and algorithms in the later Algorithm section.

EVM Side:

The EVM handles most of the dynamic computations and intensive image processing. Following is the list of tasks that EVM performs:

1. Downsampling on inputs to Radon Transform (refer to Radon Transform in Algorithm section for procedure details);
2. Morphological Operations (i.e. erosion, dilation and edge detection) (refer to Morphological Operation in Algorithm section for procedure details);
3. Radon Transform (refer to Radon Transform in Algorithm section for procedure details);
4. Calculation of the Transformation Matrix (refer to Transformation Matrix for procedure details);
5. Defining the Mapping Equations of the workspace between image-space and monitor-space (refer to Transformation Matrix for procedure details);
6. Locating the position of the thumb in the workspace;
7. Pattern Recognition (refer to Pattern Recognition/Correlation in Algorithm section for procedure details).

Previous Project Compare and Contrast

Our project has some similarities to a previous 551 project. Group 13 in Spring 2000 purposed using an Eye-Tracking System for the eventual mouse-input removal. However, in their eye-tracking system, no clicking action was implemented. This makes the human-computer interaction system incomplete. In our project, we will complete the human-computer interaction system by implementing clicking and dragging motions of the mouse. And also, the superiority of our Handimouse when comparing to the previous 551 project is the fact that it is more simple, easy and intuitive to use.

To our best knowledge, there is no commercial product that does the same thing as Handimouse. However, similar products like EagleEye's CamMouse, Handieye from MouseVision exists. These products use the motion of the face or the nose as input to the devices. These products (trial version) lower user performance, and their accuracy is unsatisfactory. Moreover, none of the above products implement the mouse dragging activity. We expect our design to be more complete and user-friendly, and to provide higher efficiency.

Webcam Input Formats and Conversions

All raw image inputs from the webcam are in Bitmap format. Multiple conversions are needed for the following purposes:

1. To define the workspace on the image-space:

Obtain the V (i.e. V as in HSV format, the luminosity) values of the pixel from its R G B values that obtained from the bitmap. The V value will then used to determine the workspace location.

2. To accomplish faster computations in EVM:

Since EVM does not offer great processing power, fast computations are required in order to achieve the real time response that we desire in our device. In order to accomplish faster computation in EVM, all the input images that are needed by EVM will be converted into binary format. Based on the R G B values of every pixel that obtained from the raw bitmap input, the corresponding pixel in the binary output will be determined as skin pixel or non-skin pixel. In the raw bitmap input, all the skin pixels will assign a value 1 to the corresponding pixel in the output, while all the non-skin pixels will assign a value 0 to the corresponding pixel in the output. As a result, all input images can be represented by binary matrices. These conversions greatly improve the speed of image processing, and also reduction in memory usages.

Algorithms

The following section presents an in-depth discussion on the algorithms that we have implemented in Handimouse.

1. ***Radon Transform***

Radon Transform is used to determine the workspace boundaries in the image-space; its outputs are then used to determine the four corner points of the workspace from the image-space, so that the transformation matrix for workspace mapping between the image-space and the monitor space can be calculated based on the information of the four corner points.

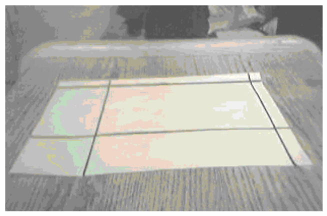
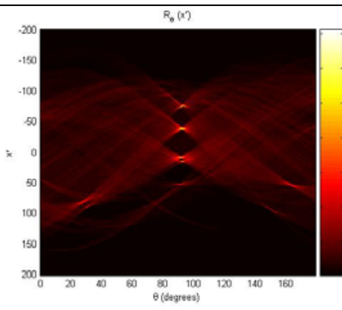
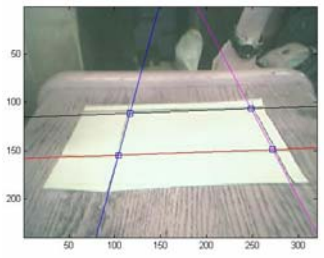
		

Figure 4. Image that includes the workspace	*Figure 5. Result of the Radon Transform of the workspace boundaries*	*Figure 6. Locate the four corners and the boundaries of the workspace*

We first convert the image includes the workspace into HSV format and use the V value of the pixels to determine the block that represents the workspace in the image. Radon Transform is able to transform two dimensional images with lines into a domain of possible line parameters (in our case we will be using ρ and θ, where ρ is the distance from the origin of the image to the line, and θ is the angle that ρ makes with the positive x-axis), where each line in the image will give a peak positioned at the corresponding line parameters. Radon Transform is used in the initial set up for defining the workspace boundaries of Handimouse in image-space.

After we have obtained the line parameters, we can now define the workspace of Handimouse by locating the four corners of workspace. This is done by finding the four line equations and their intersections. Since the output of the Radon Transform is given in terms of ρ and θ as defined above, a quick way of obtaining the line equations that define the bounties is to use the polar form representation of a line equation. The polar form is given by $x*\cos(\theta) + y*\sin(\theta) = \rho$, where x and y is the coordinates of the image space.

2. *Transformation Matrix*

Transformation Matrix is used for mapping the workspace between image-space and monitor-space. The mapping algorithm is obtained from an online source listed in the reference. The implementation is quite straight forward. First we need the location of the four corner points, which we can calculate from Radon Transform detailed above. Then we are able to construct two transformation matrices, one for forward mapping and one for backward mapping. For examples, if we want to go from the image-space to the monitor-space we can use the following matrix multiplication formula, $u = x*A$, where u is the coordinate of the monitor-space and x is the coordinate of the image-space. (Refer to reference [1] on the reference page for algorithm/equation detail)

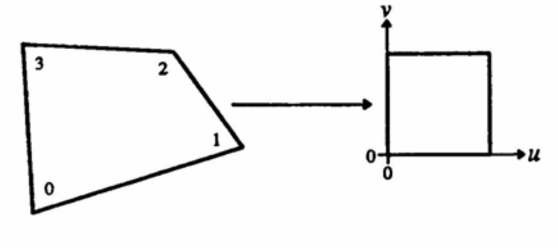
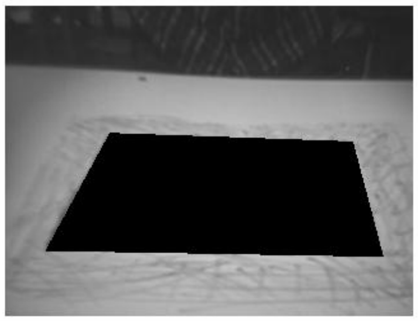
	

Figure 7. workspace mapping from image-space to monitor-space	*Figure 8. the resulting area after mapping the complete monitor-space back to the workspace in image-space (the whole monitor-space is the transformed workspace)*

3. Skin Detection

In order to segment the hand of the user from the background, we need to apply Skin Detection algorithm.

We have investigated several color spaces for detecting skin color in the image. Here are some scatter plots of the skin color distribution in different color spaces that we obtained from the skin samples:

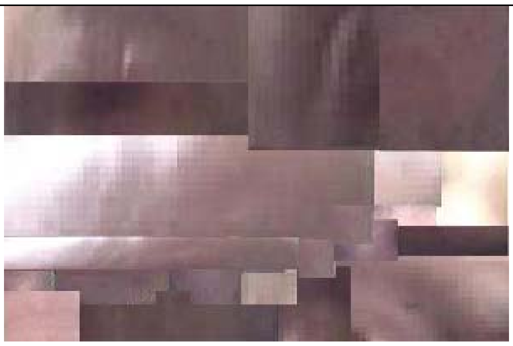
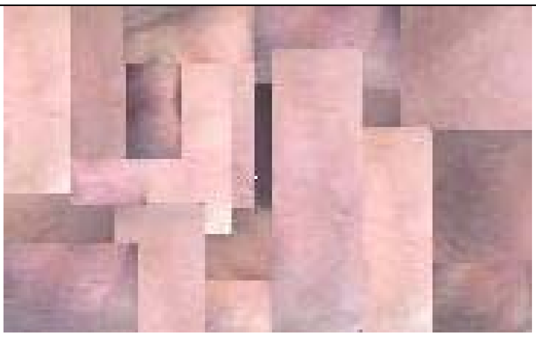
	

Figure9. Skin Sample 1 used	*Figure 10. Skin Sample 2 used*

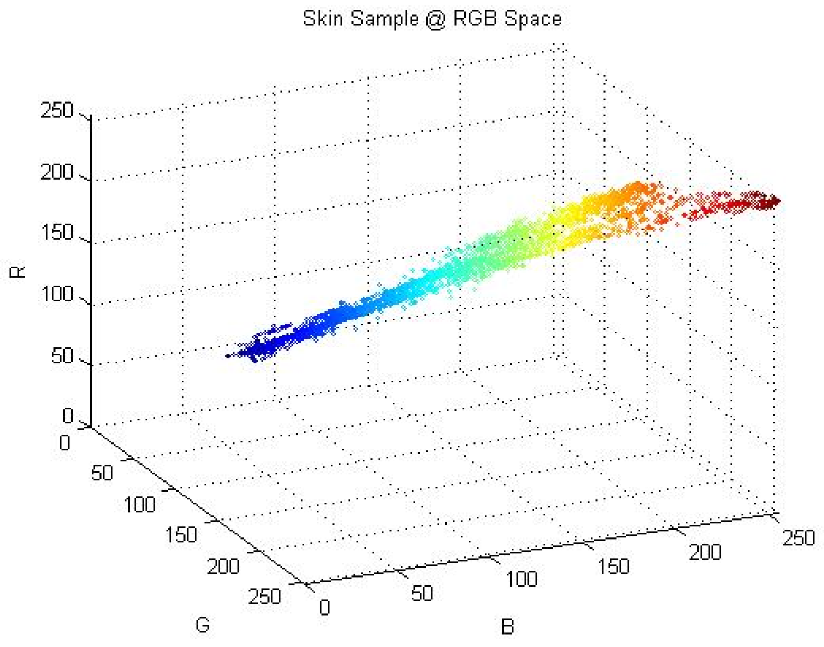


Figure 11. Color distribution of Skin Samples in RGB space

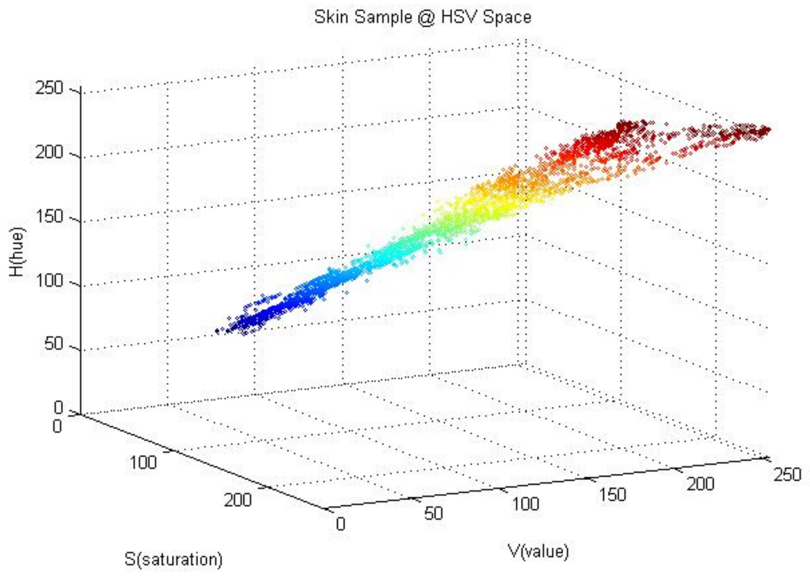


Figure 12. Color distribution of Skin Samples in HSV space

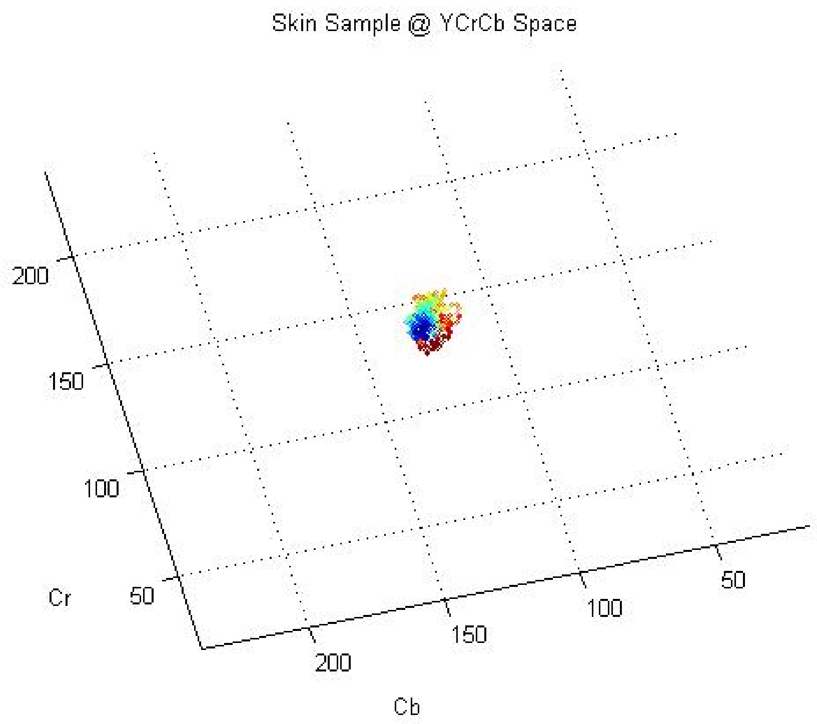


Figure 13. Color distribution of Skin Samples in YCrCb space: the color represents the Y values ranged from 0 to 1; the red represents the highest Y values while the blue represents the lowest Y values

As seen above, the skin color cannot be determined by solely restricting the extent of individual dimensions in the color spaces. From observation, the color distribution is most uniform in the YCrCb space as seen above; ignoring the Y(luminosity) values, the color distributions are concentrated within a small extend of Cr and Cb values. Yet, when tested thoroughly, if we define the skin color by solely setting a threshold in Cr and Cb values, it include some non-skin pixels as skin-pixels from time to time; hence we decided that we cannot obtain a robust skin detection by solely constraining the Cr and Cb values in the YCrCb space. From inspection, the best way to define skin color is using principal component analysis, since the color distribution in the color spaces are pretty linear. However, we do not want to spend too much machine power on segmentation of the hand, as it is a procedure that will be required in every cycle in real time, and especially when the complex computation takes a relatively much longer time to complete in the EVM than in the PC.

As a result, we choose to define the skin pixels using the relationship between the R G B values of the pixels, as they are the first color information we can obtain from the raw image input. We are able to obtain a fairly robust skin color detection algorithm based on R G B relationships from the Internet. Although the calculation is done in the RGB color space, it defines the skin pixels depending on the relationships among all RGB values instead of solely depends on the individual thresholds of R, G and B values. The details of the algorithm can be found in the reference. Please note that the accuracies of our skin detection are greatly affected by other outside issues as well: we have encountered digital camera noise problems when applying the skin detection on the webcam inputs. It will be discussed in detail later in the conclusion section. This skin detection logistic is not perfect, though it serves the purpose of our Handimouse.

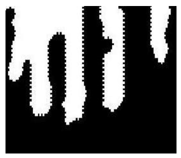


Figure 14. Handimouse Skin Detection output

4. Morphological Operations

Morphological operations can eliminate the errors/noise in image generated by skin color detection or luminosity threshold.

We were considering a lot of different combinations of morphological operations, but in the end we have decided on using a combination of only erosion and dilation, since the other morphological operations are expensive and do not help too much in improvement and thus are decided as unworthy to implement.

Erosion:

It is a morphological operation that usually decreases the number of pixels in a black and white picture. We used a 2D convolution kernel of dimension 3x3, whose entries are all 1. For each convolute location, if the value is lower than 9, that pixel will be set as a black pixel next generation.

In other words, for a given pixel, if it or any of its neighbors is not white, then the pixel will be black in next generation.

Dilation:

It is a morphological operation that usually decreases the number of pixels in a black and white picture. We used a 2D convolution kernel of dimension 3x3, which entries are all 1. For each convolute location, if the value is larger than 0, that pixel will be set as a black pixel next generation. In other words, for a given pixel, if it or any of its neighbors is white, then the pixel will be white in the next generation.

When the operations are used in “erosion, dilation” order, the output image will include less noisy pixels while the general shape and size of the main features preserves. When we use the morphologies in higher orders (for example erosion twice then dilation twice), we are able to remove noises in larger area. However the main features will be distorted or even disappeared if the morphologies are used in a very high order. Therefore, we chose the order of 2 (erosion twice then dilation twice) based on experience of the noise we encounter. These noises mostly from the error in skin color detection, which recognize some parts of the lines of grid as skin. These noise are at most 3 pixels wide, hence morphologies at order of 2 serves the purpose for Handimouse.

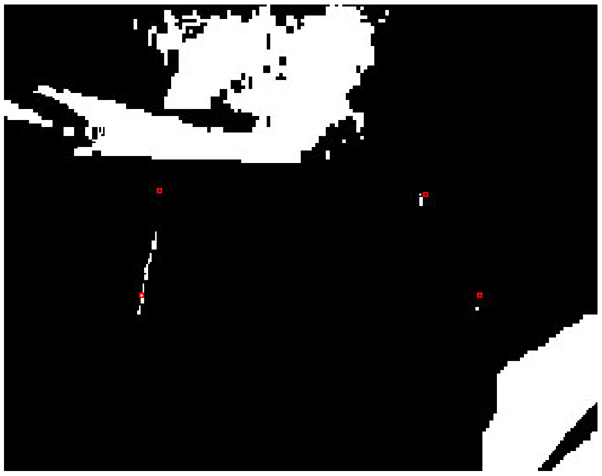


Figure 15. Sample Skin Detection that contains noise; in such case, morphological operations are needed

Moreover, Radon transform needs further noise reduction. To generate a sensible result from radon transform to obtain information of the grid lines (workspace boundaries) from the image, we need to isolate the edges of the grid from the image, and thus we need to first locate the position of the grid (workspace), and then extract the edges of the grid (workspace boundaries).

In the following paragraph we will describe the method we obtain the location and shape of the workspace. Firstly, we obtain a luminosity threshold image which indicates any pixel is almost as bright as white. We assume that the whole workspace is bright enough and also the workspace covers the middle point of the image. From the middle point of the image we check if there is any point around it on the image is a white. We will mark the pixels and all its connecting neighboring white pixels as white on the corresponding pixels in another image. The result is a connected blob that has a marked pixel in the middle point of the image. If the workspace boundaries are marked black enough and thick enough, the workspace can be clearly separated from the background in the luminosity threshold image; the blob should represent the workspace.

Furthermore, we need to perform edge detection on the grid space blob before radon transform, so that the result of radon transform will achieve satisfactory accuracy. We have chosen to implement Sobel edge detection. [7] This simple edge detection consists of two convolution kernel (Gx and Gy) representing horizontal and vertical edges representatively. After convolution of the blob image with these 2 kernels, we obtain a black and white image by applying a threshold. (i.e. if the sum of the two convolutions is greater than 3, the pixel will be set as white, else set as black) As a result, only the edges of the grid are visible, hence the result of radon transform can be interpreted easily.

$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad \mathbf{G_y} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

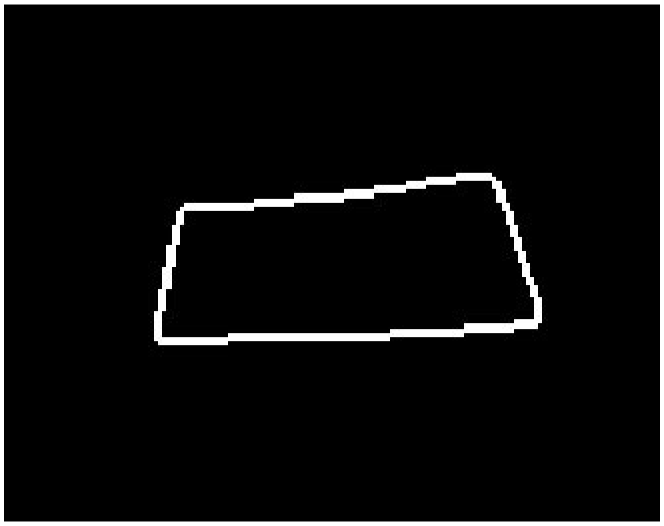


Figure 16. Handimouse output of Edge Detection

5. Locating the Thumb

Originally we decided that the leftmost bottom pixel can be seen as the location of the left side of user's thumb. However, during testing phase, we discovered that different user has different posture of hand, and sometimes our algorithm locates the joint that connect thumb and hand, instead of locating the thumb position. Hence we abandoned the previous idea and tried to locate the right side of the thumb. Our new idea is the following. We first locate the leftmost bottom skin-pixel and then traverse from that pixel downward and rightward until we cannot go any farther. This way, we are able to obtain correct and very stable thumb position. However, since the right side of a right handed user's thumb is the inward side of the thumb, which means that the thumb is usually shadowed by itself and the index finger, and thus it lower the accuracy. Moreover, because of the memory effect that we encounter (will be discussed later in the conclusion section), our accuracy of locating the thumb is greatly reduced.

As we can see from the picture below, the red dots indicate where the grid corners are. Since some of the grid lines (workspace boundaries) reflect ambient light as well that the skin color detection cannot tell it apart from the skin color.

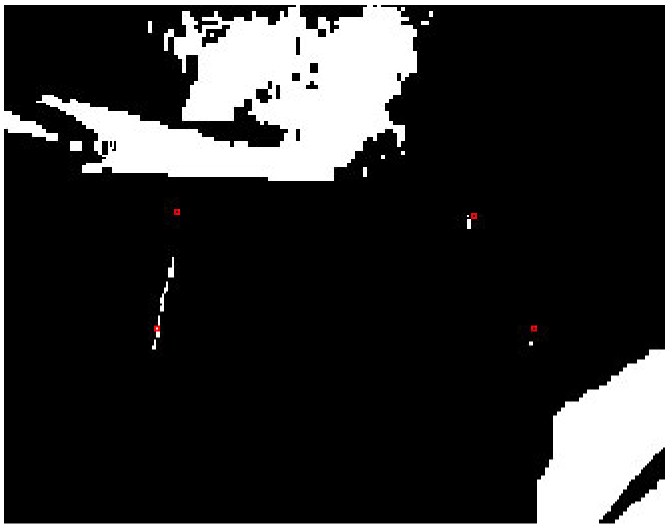


Figure17. Sample Image that demonstrated the Error encountered when tracking the thumb

6. Pattern Recognition/Correlation

Pattern Recognition is implemented to determine the finger activities that trigger the mouse button action, i.e. whether the index finger is holding up or down, whether the middle finger is holding up or down. The basic algorithm used for pattern recognition is 2-dimention-correlation.

The following describes the implementation of our pattern recognition. During the initial phase of the program, the UI (user interface) at the PC side will request the user for 3 picture of his hand: picture of his hand in relaxed posture, picture of his hand with index finger holding up, and picture of his hand with middle finger holding up. The images are first transformed into grid space in PC. As the positions of the thumbs are found, we can crop a template starting from the thumb points. The size of the crop templates is equal to the size of the workspace.

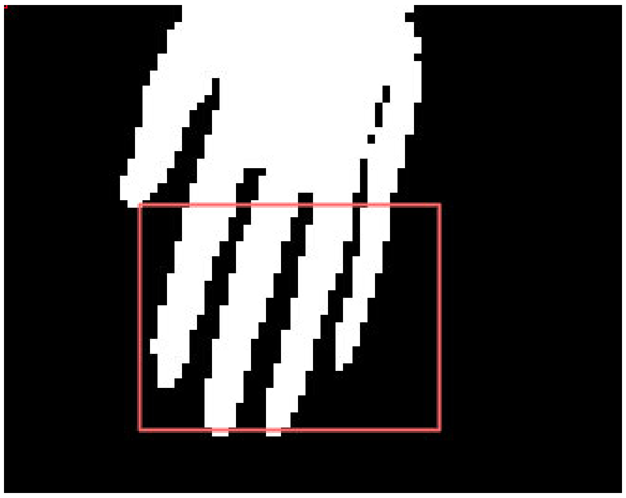


Figure 18. Sample image that demonstrates the template that is cropped from the picture that is taken by the webcam at Handimouse initialization

These templates are stored in the EVM with a downsample factor of 8. Since we know the maximum correlation value is obtained when the hand in the templates overlap with the hand in the current image, we can deduce that the maximum correlation value for a template is obtained when the thumb in the template is aligned with the thumb in the current image. From the values of the three template correlation, we can determine if the user is holding up any finger, and which finger it is.

Our correlation achieves high processing speed, though it suffers from the accuracy issue of thumb location. And also, the result template correlation is not rotational invariant, and thus we cannot consider the case when user rotates his hand during Handimouse operation.

The Handimouse

In this section, we will discuss the operation procedures of Handimouse in detail. To start Handimouse, the user first need to run the GUI at the PC side that guides the user through the initial set up procedures. The initial set up includes defining the workspace and obtaining various templates for pattern recognition. The initial setup steps are illustrated as following:

<u>Step 1: Defining the workspace</u>

First we start the GUI, and make sure the camera can see the entire workspace. Click on the "Initialize" button.

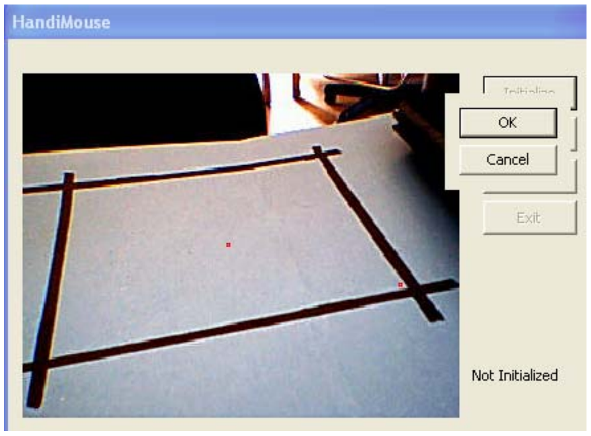
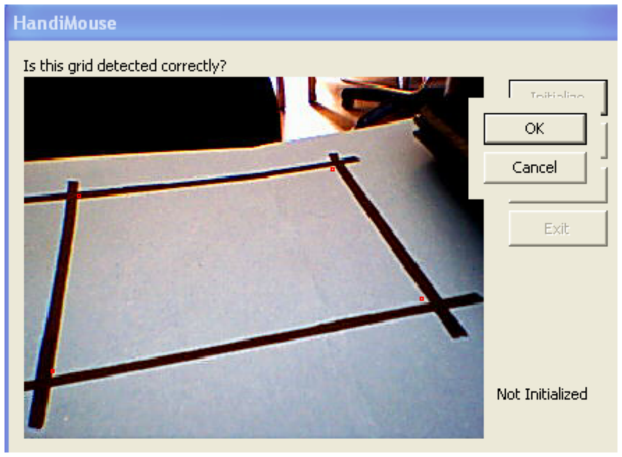


Figure19. Handimouse GUI at start up: place the workspace so that the red dot is inside the workspace

Place the workspace (the rectangle grid) so that the red dot is inside the workspace. Click on "OK" when done. The application will then start performing Radon Transform on the input image to obtain the four corner points of the workspace in image-space.



Is this grid detected correctly?

Figure19. Handimouse GUI after detecting the four corner points of the workspace using Radon Transform

Above is the output image after the Radon Transform. The four red dots in the figure above indicate the four corner points of the workspace points found and calculated by our program. Click "OK" if the corner points are detected correctly, or click on "Cancel" and redo the Radon Transform if the corner points obtained are incorrect. After the "OK" button is clicked, the four corner points will be passed to the EVM to determine the transformation matrix for workspace mapping between image-space and monitor-space.

Step 2: Obtaining the four-finger-template

After the transformation matrix is defined, the GUI will then ask the user for the templates for later pattern recognition purpose. At first, it will request the user to place his hand flat showing all his fingers except for the thumb. If all four fingers show up in the GUI correctly, click on "OK" to confirm on the template, or click on "Cancel" to redefine the template.

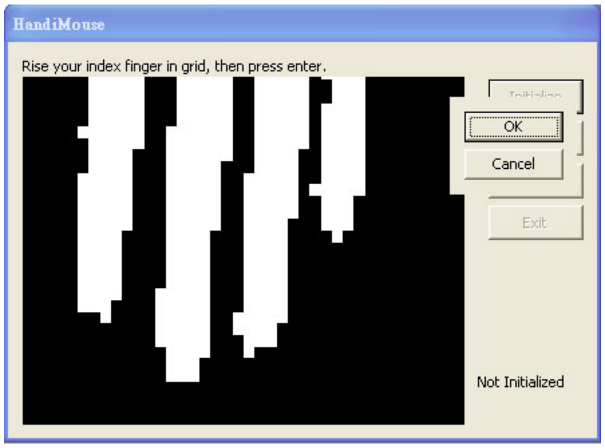


Figure20. Handimouse GUI showing four-finger-template of a user

The figure shows a four-finger-template of the user. The four-finger-template is defined as the "default" state. It is used to compare with the incoming image to determine if the index or middle finger is raised.

Step3: Obtaining the raise-index-finger-template

After the four-finger-template is confirmed, the GUI will then ask for the user to raise his index finger.

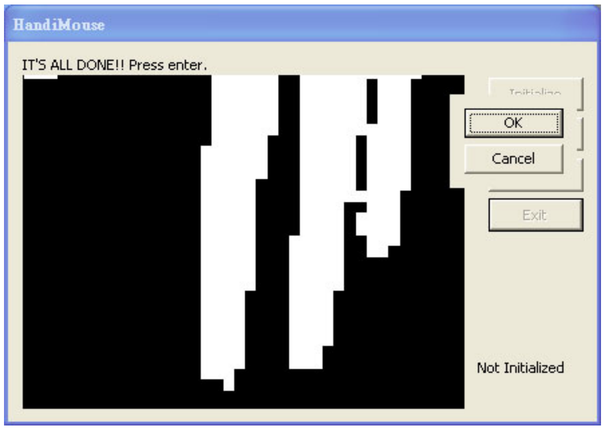


Figure21. Handimouse GUI showing index-finger-template of a user

The figure above shows the index-finger-template of a user. The index-finger-template is used to detect left-mouse-button-down action of Handimouse. Click on “OK” to confirm the template, or click on “Cancel” to redefine the template.

Step 4: Obtaining the middle finger template

After that, the GUI will request the user to raise his middle finger for the last template.

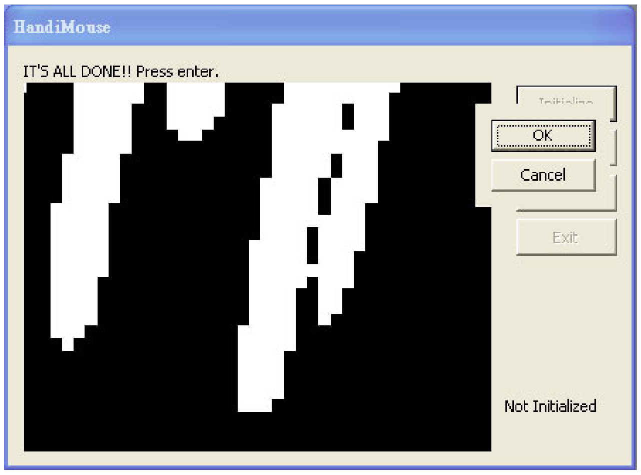


Figure22. Handimouse GUI showing middle-finger-template of a user

The figure shows the middle-finger-template of a user. The middle-finger-template is used to detect right-mouse-button-down action of Handimouse. Click on “OK” to confirm the template, or click on “Cancel” to redefine the template.

After completing this simple initialization procedure, Handimouse is ready to go. User can then click on the "Start" button on the GUI to start using Handimouse. The basic operations of Handimouse are exactly the same as any traditional mouse, except for the dragging motion. Basically, the index finger controls the left mouse button, while the middle finger controls the right mouse button. The mouse-button-down actions are signaled by holding up the finger, while the mouse-button-up is signaled by putting down the finger in the workspace. In other words, left click is accomplished by putting the index finger up and then down in the workspace; left double click is accomplished by putting the index finger up and then down twice; and similarly for right click. Dragging is accomplished by holding the index finger up and moves the hand to the desire position and put the index finger down. Since the program is not rotational invariant, we overlay the user's template onto the user's hand to remind him of his template orientation and shape. The blacked out pixels are the places the hand matches template. The four red points indicate the position of the grid, and the green dot is the detected thumb position.

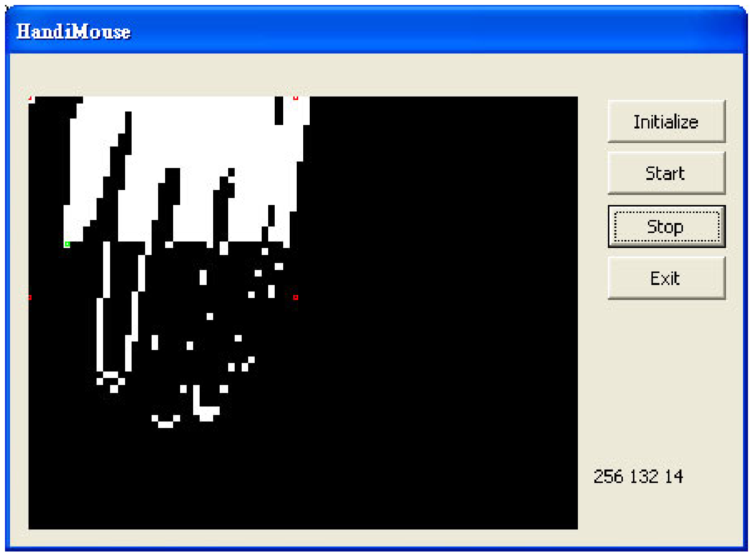


Figure23. Handimouse GUI demonstrating Handimouse in operation

Clicking on the "Stop" button terminates the Handimouse and switches the window mouse input to the default traditional mouse, while clicking on the "Exit" button on the GUI will exit the Handimouse program completely.

Optimization

In order to fully utilize the EVM, we have tried to perform many intensive image processing computations on the EVM instead of on the PC. As a result, we are required to speed up the EVM process as much as possible. Thus, we had applied several bitwise manipulations and tricks to replace some algorithm that require brute force calculation. No memory optimization is needed (e.g. paging, ping pong memory) because all of the memory we needed are allocated in the 64kb on-chip memory.

1. Erosion and dilation

   These two operations are used extensively in our program, requires multiple access of a single memory address if we use naïve method to calculate. This is because of the manner of our storage (a single memory address contains 32 bits, each correspond to one pixel). However, if we use bitwise shift operations on a memory address, each bit in the memory address we would have to consider separately will be aligned with its neighborhood. This is very convenient for calculating erosion and dilation because these functions are determined by the disjoint or union of neighborhood of a single pixel.

2. Radon Transform

   Discrete Radon transform is a very calculation expensive algorithm. Naïve implementation of it has the efficiency of O(n^3). Also since we have to store a multi-bit value for the radon transform output, the on-chip memory will not be able to hold all the output value hence the performance will be greatly reduced. Fast discrete radon transform has the efficiency of O(n^2 log(n)) but require 4 times the memory than naïve implementation needs. We discard the idea of using fast discrete radon transform because that will require more memory than on-chip memory provides. Fast discrete radon transform will therefore greatly decrease the performance. Since the usage of radon transform is the detection of the grid, which is used only to reveal the perceptive distortion the image is undergoing, sub pixel error in radon transform is tolerable. We decided to reduce the size of the radon transform by a half (n/2) and implement the radon transform using naïve method.

What is more, we have successfully speeded up the Radon transform process by another factor of 1/8 by creating a lookup table for storing the repetitive cosine and sine values (reduce the operations by 1/2) and by combining similar mathematical operations (operations that are negation of each other) in a single for-loop instead of repeating the same process four times.

Memory Usage

Since the program is intended to be real time operation, we allocate all the memory from on-chip memory, which is of the size 64kilobytes. Because of this memory constraint, all the input images to EVM are binary (black and white) images. These binary images are produced by either a threshold of luminosity of the consisting pixels, or the skin detection condition that have been discussed above.

The major memory allocations are these:

1) Buffer for accepting incoming

   Size of buffer: 320*240*1/8 bytes = 9.375kilobytes

2) Buffer for calculation of radon transform (which value is recorded as an 8-bit value). Since the accuracy of the position of the grid is not very important, we down sampled the image before we perform radon transform.

   Size of buffer: 320/2*240/2 bytes = 18.75kilobytes

3) 3 Buffers for template storage (each of them is down sampled by a factor of 8)

   Size of each buffer: 320/8 * 240/8 * 1/8 bytes = 0.1465kilobytes

   Total size of buffers: 28.56Kilobytes

Note: 320 = pixel width of the input image, 240=pixel height of the input image

Conclusion

The performance of Handimouse is only satisfactory. We are now going to discuss the problems that we encountered.

The most serious error that we have encountered is an image artifact known as memory effect (ME). Pattern of the resulting noise image is called banding. This pattern is a pattern consists of alternating lighter-darker horizontal stripes that is of constant frequency. Memory effect is known to be caused by circuitry contained in the pre-amplifiers immediately following the detectors in the instrument electronics. This is primarily due to a portion of feedback circuit that contains a resistor/capacitor combination. And also, because of the data structure of the image memory, in which data are set by the order red followed by green followed by blue, the memory effect of the red value leads the memory effect of blue value by a significant amount that some of the colors are distorted. The problem can be illustrated by the following web cam inputs:

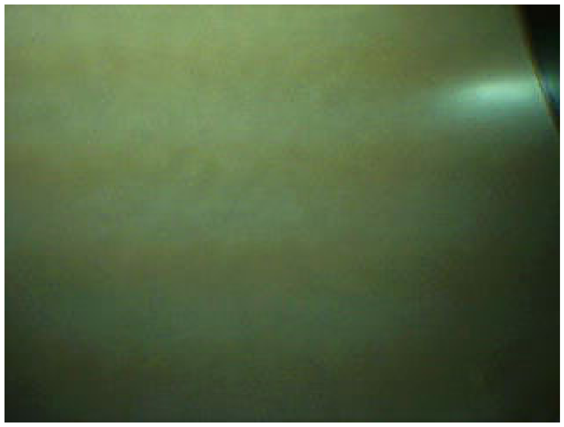


Figure24. raw image from the webcam demonstrating the banding effect

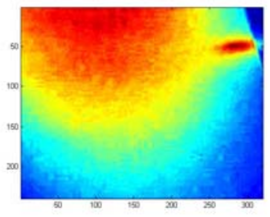
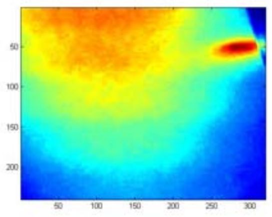
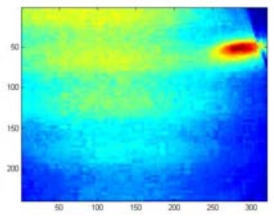
		

Figure 25. Red component of Figure 24	*Figure26. Green component of Figure 24*	*Figure 27. Blue component of Figure 24*

From the pictures above, we can see that all color domains suffer from memory effect. The effect is more dominating at where amplitude changes rapidly. For instance, when we take a picture of the user's hand, the resulting banding pattern at the edge of his hand will affect the skin color detection result, and results in fluctuations in measurement. The problem is worsened by the pattern traversing in the image.

This is a core problem for our project, as our skin detection outputs all strictly depend on the relationship among the RGB values that obtained to determine skin pixel or non-skin pixel. As the RGB values of the pixels cannot be accurately obtained from the raw image input, the accuracy of our mouse cursor is greatly decreased. Moreover, because the banding pattern is not a standard pattern, the thumb location that we obtain is constantly changing even when the user's hand is not moving. So far, we had tried to reduce this problem with the following 3 methods:

Elimination of the specific frequency of banding pattern:

We tried to take the Fourier transform of the image and expecting to determine the specific frequency of the banding, so that we could apply a combination of sine and/or cosine waveforms to cancel out the banding effect. However, we have figured out that there is no specific frequency of the banding since the amplitude of banding pattern depends on the amplitude of the original pixel. The banding pattern consists of many different frequency components. Within the constrained time, we are not able to figure out a mathematical model of the frequency of the banding that occurs.

Stabilizing the moving stripes by changing sampling rate

Since the major problem is that the banding pattern travels in the image in time, the problem is solved we can stop the pattern moving. We were hoping to stabilize the moving of the stripes by sampling images at an integer multiple of the frequency of banding to make the image seem to stand still. The best sampling rate we obtained to avoid the banding is 1/161(samples/ms), but at this frequency the banding pattern only stand still for about a second until it jumps to another position. We suspect that multiple of the frequency of banding cannot be converge to a certain integer, and/or the sampling time of C++ is not a constant. As a result, we are not able to obtain a satisfactory improvement from this method.

Research for skin detection that do not require great accuracy of all R G B values

After the above two trials, we realized that we could avoid the errors generated by these artifacts without develop our own mathematical model of the banding frequency of our web cam. We try to solve the problem in a more efficient one by researching another option of skin detection that does not strongly depend on the accuracy of all R G B values. We have investigated the following options:

1) Do not depend on color space

    This is impossible to achieve, since skin only can be determined by its color from the image

2) Try to convert the detection depend on solely one or two values for the three R G B values to reduce the error produced by the artifacts

   All other color space, i.e. HSV, YCrCb, NTSC, etc, are calculated on the relationship among the R G B values; in other words, switching to another color space will not improve the situation. Moreover, as mentioned in algorithm section, the skin color distribution spans all R G B dimensions, and as a result, the detection requires all three of the R G B values. Even if we consider using principal component analysis as proposed before, it will requires all R G B values to obtain the eigenvalues and eigenvectors.

To conclude, there is still much room left for improvements. Further work can be conducted on how to improve the accuracy, performance, and usability of Handimouse. Rotational invariant, shape invariant and remedies to memory effect are the major improvements we look forward to.

Reference

1. *Paul S. Heckbert, Fundamentals of Texture Mapping and Image Warping, http://www.cs.cmu.edu/~ph/texfund/texfund.pdf*
   The paper provided us with a straight forward implementation of projective transformation matrix. The transformation matrix is necessary to convert image from workspace to monitor space.

2. *Paul F. Whelan and Derek Molloy, Machine Vision Algorithms in Java, Springer-Verlag, 2001.*
   The book discussed some of the basic image processing algorithms.

3. *A Survey on Pixel-Based Skin Color Detection Techniques, Vladimir Vezhnevets, Vassili Sazonov,Alla Andreeva, http://graphics.cs.msu.su/en/publications/text/gc2003vsa.pdf*
   The paper is a survey on various skin colour detection algorithms. It discussed the pros and coins of a number of skin detection schemes and their implementation. The skin detection algorithm that we used in our project is obtained from this paper.

4. *Lab 3-filterevm.c, Rohit Patnaik, 18-551 2005 fall*
   The lab handout provided us with C codes to implement data transfer between EVM and PC.

5. *Behnhard Behn, Jeffrey Lee, Bharat Bhat, Ain't Gonna Micky Mouse 'Round No More!, 18-551 2000 spring*
   The paper provided us with some information on a previous project that had some similarities to ours.

6. *NASA*
   http://ltpwww.gsfc.nasa.gov/IAS/handbook/handbook_htmls/chapter7/chapter7.html
   The web source provided us with the explanations on the noise in our system.

7. "Sobel." Wikipedia. <http://en.wikipedia.org/wiki/Sobel>