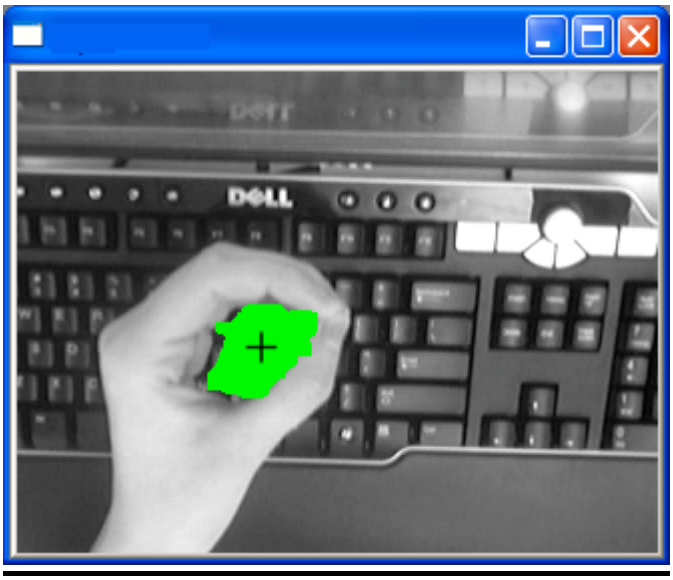
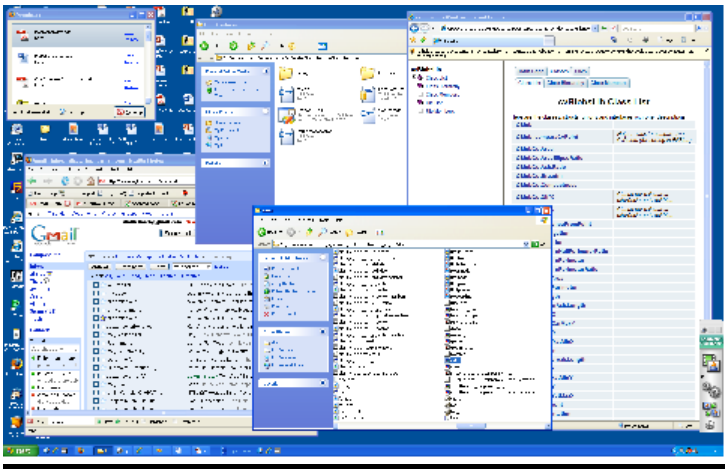
Enabling Simple Window Management Using Pinch Gesture Recognition





ComS 401 Final Project

Benjamin Baldus

Debra Lauterbach

Juan Lizarraga

Abstract

In this project, we developed a machine-user interface which implements pinch gesture recognition using simple computer vision techniques. The interface allows the user to use their hands to control the mouse as well as easily move and resize windows that are open on their screen. This interface is simple enough to be run using an ordinary webcam and requires little training.

1. Motivation & need for the application

The use of only basic input devices such as the mouse and keyboard is limiting the creativity and capabilities of the user. Our goal with this project was to expand the ways that people are able to interact with their computers. Namely, we wanted to enable users to interact more naturally with their computer through using simple hand gestures to move the mouse and perform tasks.

One task which is especially cumbersome to perform with the mouse is moving and resizing open windows. This is something that many computer users do quite often - opening several applications, folders, browser windows, etc. on their screen, and then switching between them frequently. But, on most computers, the only way to move a window is by clicking and dragging it by its header/title bar, and the only way to resize is by clicking and dragging on the bottom-right corner. This can be annoying to do for managing multiple windows, which is the problem that our application aims to solve.

Performing gestures with the hands over the keyboard is a natural place for interaction, since the hands are often already over the keys. That way, the user can quickly transition from typing to performing gestures. The gesture we use, of bringing the thumb and forefinger together (“pinching”) is a simple gesture that can be easily recognized using computer vision techniques. The hole created by a pinch can be detected using simple image segmentation and connected components analysis, instead of having to use complicated hand-tracking algorithms and hand shape recognition techniques.

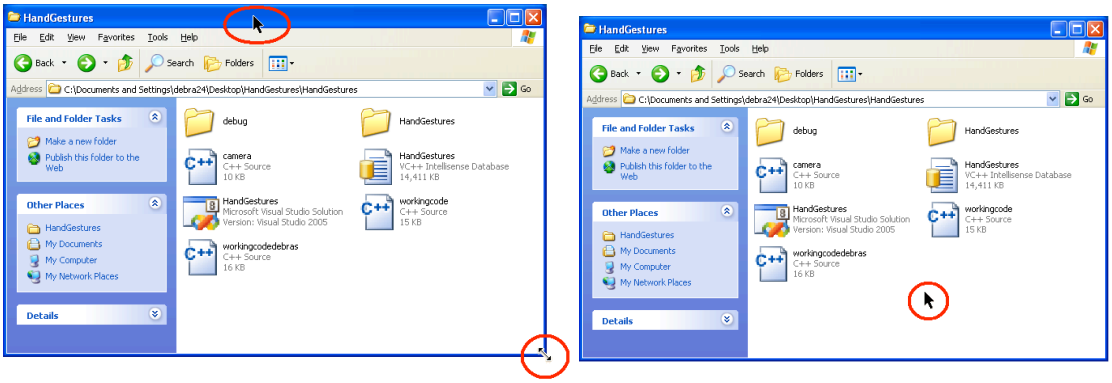


Figure 1: Traditional moving and resizing of windows requires clicking in a specific place in the window (Left image). Our project allows moving and resizing by clicking anywhere in the window (Right image).

These hand gestures can be recognized in real time, and once a pinch is recognized, measurements can be found for its movement and orientation. These measurements can then be translated into mouse events, letting the user control the cursor and easily move and resize windows by selecting *anywhere* in a window, instead of just the title bar or bottom-right corner.

1.1. Target audience

Anyone with a computer and a camera should be able to take full advantage of this program. However, our main target audience is advanced computer users, since these are the users who are most likely to have many windows open, requiring frequent window management. They are also probably the most likely to want to use a cutting-edge application such as this.

However, since this application has a relatively small learning curve, it should be easy for any type of user to master and use it effectively.

1.2. Previous approaches

Hand gesture recognition for computer control is a popular research topic in computer vision. The first research on in the area began in about 1992, when it first became possible to grab images from a camera in real time, and thus enable effective human-computer interaction. Research in the area has typically fallen within the categories of applications for pointing, presenting, digital desktops, virtual workbenches, and virtual reality. In all cases, the

motivation has been to create convenient, intuitive, powerful means of interaction with a computer that can serve as an alternative to the traditional keyboard and mouse.

Approaches to recognizing hand gestures have usually been divided into two types: *model-based* and *view-based* approaches. Model-based approaches use a 3D hand model for tracking, which has many more potential applications, but is a challenging task. The main problems are that it works well only for relatively slow hand motions, and in constrained environments (Stenger, 2006). View-based approaches, however, use pattern classification to identify hand features and determine the current hand pose. The main challenges in this case are to find how to segment the hand from the background, and how to determine which features to extract from the segmented region. Segmentation is often done using skin color detection and segmenting this from a uniform background.

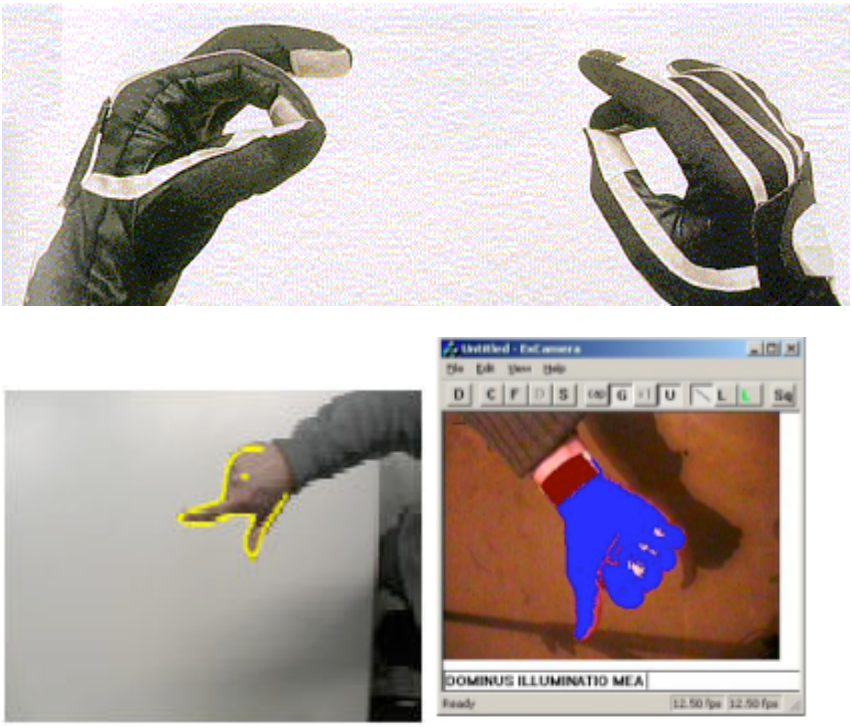


Figure 2: Some previous research approaches, using data gloves, edge detection, and skin color segmentation.

For both approaches, tactics to help in recognizing hand gesture input include:

- using props or input devices, such as a pen or data glove
- restricting the object information, as through a silhouette of the hand
- restricting the recognition situation, by using a uniform background
- restricting the set of gestures made, and using only one hand (Lenman, Bretzner & Thuresson, 2002)

Cursor Control

Several applications for cursor control have been created using 2D tracking. These methods all use a single camera to track a hand, and have typically used pointing gestures as the means of replacing the mouse. An early example is *Finger Mouse* (Quek, Mysliwjec & Zhao, 1995), which used a camera to view the hand from above and recognize hand gestures to control the mouse. One drawback of this system was that the user had to press the shift key with their other hand in order to 'click', making the system not completely gesture-based. Later applications found ways to solve this problem and create click gestures, such as O'Hagan (1997) who defined clicking as when two fingers were extended, and von Hardenberg and Berard (2001) who defined clicking as when a one second delay in hand movement occurred.

1.3. Related work

While previous applications have used complicated model or view-based approaches to do hand gesture tracking, the goal of our project is to implement this in a simpler, more intuitive way. We also are to studying strictly hand gestures that are done over the keyboard instead of away from the computer, since this is a more natural and easy place to use gestures. To fulfill these goals, we based our project off of the research of Andy Wilson of Microsoft Research, who has created a way to detect pinching gestures over the keyboard using simple image processing techniques.

Wilson's technique makes use of the fact that the keyboard (in his case, a common black keyboard model) is much darker than human skin tone. The camera, set on top of the computer monitor looking down, records a picture of the keyboard when the application is first started. When the hands appear

over the keyboard, it then uses this image to be able to distinguish when the user, pinching their thumb and forefinger together, has created a ‘hole’ over the keyboard. This hole is distinguished in several steps, including obtaining a binary segmentation of the scene, computing the connected components of the background pixels, and identifying components of significant size, which are the ‘holes’ of a pinching hand.

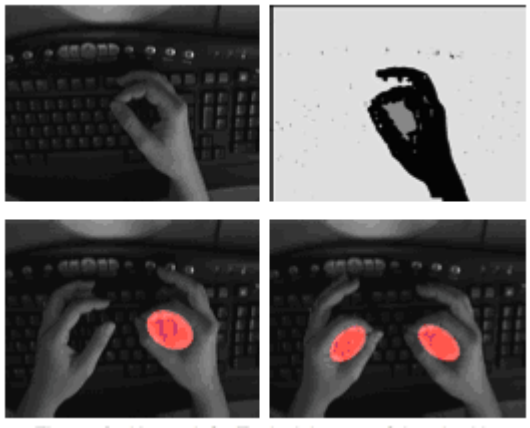


Figure 3: View of the webcam over the keyboard, binary segmentation, and connected component detection of one and two hands.

The centroid of this hole is the point used for cursor control. Cursor movement is enabled when the user first pinches their fingers, and ends when the pinching stops. To create mouse clicks, the interface uses the rapid closing and opening of the thumb and forefinger as the mouse-down and mouse-up events. Double clicking can also be done this way, by creating the motion twice. For dragging, a ‘dragging mode’ must be initialized by a quick open-and-close gesture of the thumb and forefinger, and is ended when the fingers are separated.

Wilson’s technique also allows for more complicated interaction, such as using two-handed pinch detection for translation, rotation, and scaling, which he uses in a map application to zoom in and out, rotate, and pan the map. These changes can be computed from the ellipsoidal hole created when pinching. Scaling, for example, is done when the size of the ellipsoid gets larger or smaller as the hand moves either toward or away from the camera.

Wilson's method has several advantages over other more complex hand tracking methods. For one, the pinch gesture is more precise than tracking an extended index finger, since it is less ambiguous as to when the gesture has been made. Also, the simple image processing done for this method is more straightforward than other techniques, in that it does not need to know anything about the precise hand shape or where the fingertips are.

1.4. Previous experience

Ben Baldus is a senior in Computer Science. Over the past four years he has gained experience programming in Visual Basic, C, and C#. Ben has been programming in C++ for over 5 years. Though he has been involved in several unique projects and created several windows applications in his free time, he was excited to bring his skills to this interesting and novel project.

Juan Lizarraga has successfully completed the first 4 out of 6 years of Telecommunication Engineering at the *Universidad Pública de Navarra* (Spain) and he is now in his 5th year at Iowa State University. During this time he has attended courses in several fields such as Circuit Design (analog and digital), Microwave Devices, Fiber Optics, Computer Networks and Signal Processing. The latter has made him gain experience in MATLAB programming through several projects on audio signal processing, digital filters and FFT algorithms. He also has some experience in C, Pascal and Java programming.

Debra Lauterbach is a senior Computer Science and Psychology. She has experience programming in C, C++, and Java, as well as knowledge of web programming languages. She has been involved with the Human-Computer Interaction program since her sophomore year, doing research related to eyetracking and later with OpenGL graphics programming for the C6, and taking classes and seminars in HCI. She plans to go on to graduate school in HCI to study user interface design.

2. APPROACH

2.1. Equipment

- **Webcam**

We used a standard USB webcam at a resolution of 320x240. This viewing area proved to be sufficient, though a larger area would have advantages as far as having more room for pinches to be tracked.

This device will be installed on the top part of the monitor pointing toward the keyboard. The relative position of the camera and keyboard must be as perpendicular as possible, in order to reduce the tilt angle of the camera.

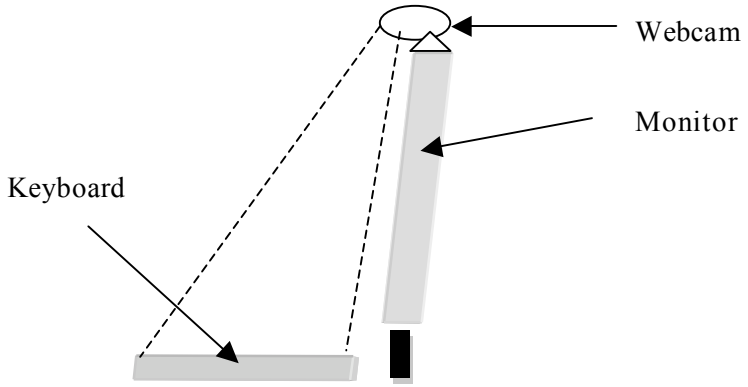


Figure 4: Example Setup

The scene captured must fit the keyboard.

- **Software**

  - Windows XP: this operating system developed by Microsoft is a stable and trustworthy platform for running applications made with Microsoft Visual Studio.

  - Microsoft Visual Studio 2005 is Microsoft's software development product for programmers. It includes several programming languages such as Visual Basic, C++, C# and J#. In our case the project was developed in C++.

  - OpenCV is a C++ based computer vision library developed by Intel. It focuses mainly on *real-time* image processing, and gives the user a large amount of tools that provide extensive functionality on this field. In our project we used OpenCV for capturing frames from the webcam and processing the images using morphology

- cvBlobsLib is a library for OpenCV which performs binary image connected components labeling. It also has additional functionalities such as manipulating, filtering, and extracting blob features. This library is used in our project to determine when a pinch is being made, and to determine the center of the blob for moving the mouse.
  The website for cvBlobLib is:
  http://opencvlibrary.sourceforge.net/cvBlobsLib

- Windows Mouse DLL is an open-source code library for "X-Windows-Like dragging and resizing of windows". This project uses Windows hooks to be able to easily move and resize windows by clicking and dragging anywhere in the window area. This code was tied in to our project by running it as a separate process, and using pinch gestures to do the dragging and resizing instead of using a mouse.
  The website for Windows Mouse is:
  http://www.codeproject.com/dll/wm.asp?df=100&forumid=4491&exp=0&select=1219138

- **Ordinary PC**

  Any standard PC running Windows XP is capable of running our project. The minimum hardware requirements are those of the operating system and the required software.

Figure 5: Real-life setup example.

- **Dark colored keyboard**
  A dark keyboard is necessary to make the algorithm more accurate. This is because binary segmentation is done to distinguish the keyboard from the user's skin. If the color values are too similar, the segmentation will not be as accurate, which will decrease the usability of moving the mouse.

Since a skin detection algorithm is also applied, it also required to work over a non-wooden surface so the algorithm keeps stable and does not confuse the working surface with skin.

2.2. User Interface

When this program is run, the user interface consists of just a window displaying the hands over the keyboard, and any pinches that are detected. Pinches with the left hand are colored green, and pinches from the right hand are colored red.

The gestures that the user can make to interact with the computer include:

NORMAL MODE:	
Action	**Gesture**
Clicking	Pinch one hand and release
Moving the mouse	Pinch one hand and then move hand. Mouse movement ends when the pinch ends.
Double Clicking	Two fast pinch-and-releases of one hand
DRAGGING MODE:	
Action	**Gesture**
Dragging	Dragging mode is initialized by two fast pinches of one hand, and ends when pinch ends
Moving the mouse	Once dragging mode is initialized, moving the hand moves the mouse and drags any windows that the cursor is over. Dragging ends when the pinch ends.
RESIZING MODE:	
Action	**Gesture**
Resizing	Pinch both hands, and move the right hand to resize any window which the cursor is over. Resizing ends when one of the pinches ends.

2.3. Algorithms

- **Binary segmentation algorithm**
  There are several image segmentation techniques that we could have used to segment the hand from the background, but the one implemented is comparing the current image to a stored background image of the keyboard. This image is taken when the program first starts running, at a time when the hands are not over the keyboard.

  To segment the image, we compared the pixel values between the background image and the current frame image. If the pixel had changed significantly in value then, it is be labeled as 'hand' and set to be white. If the value has not changed significantly, then it can be labeled as 'background' and set to be black.

- **Skin detection algorithm**
  Our program hinges on obtaining a quality image segmentation to be able to detect pinch gestures. After much trial and error, we discovered that using just binary image segmentation was not enough, since the light-colored key labels interfered with the segmentation.

  As a solution, we implemented a second method to analyze the image by using skin detection to detect color values within a certain range. This analysis was done before the binary segmentation, and the results of the two processes were added together to produce a final segmented image, with all background pixels being black, and all hand pixels being white.

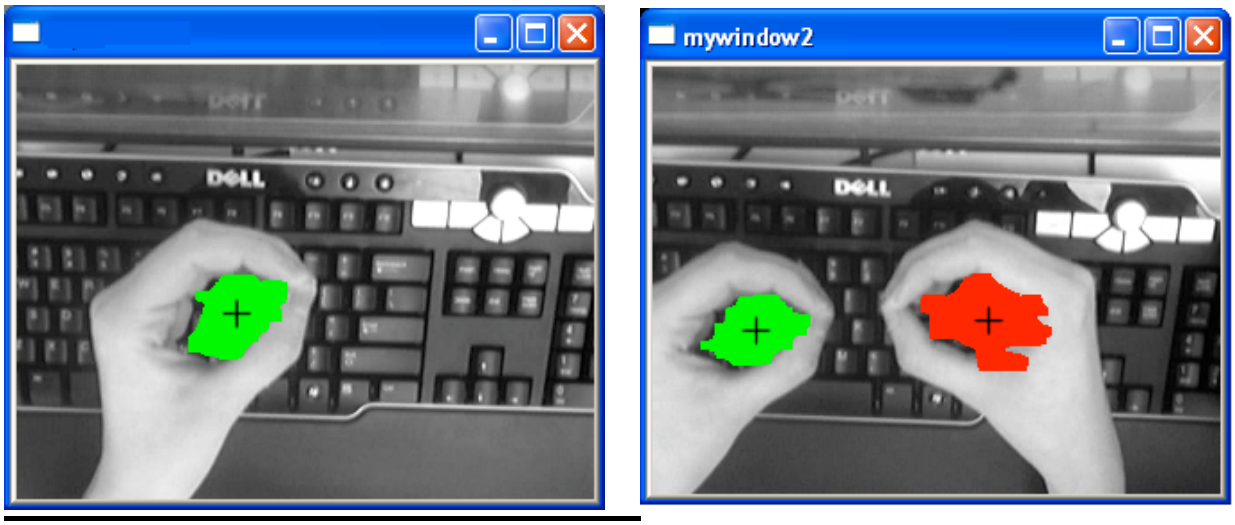


Figure 8: Pinch detected on one hand, pinch detected on both hands.

- **Blob Detection / connected components algorithm**
  In Andy Wilson's pinch detection method, a connected components algorithm is used to find the shape created by pinching the thumb and forefinger. However, in our project we decided to use the cvBlobsLib library to perform our connected components labeling for us. This library is easy to use, and allowed us to detect blobs and then filter them by a certain size range.

  The largest blob or connected component is the background of the image. But, when the hand is pinched, this splits the background into two connected components: the hole formed when the thumb and forefinger are brought together and the rest of the background. Filtering out the background component, this leaves us with just the blobs produced by pinching one or both of the hands.

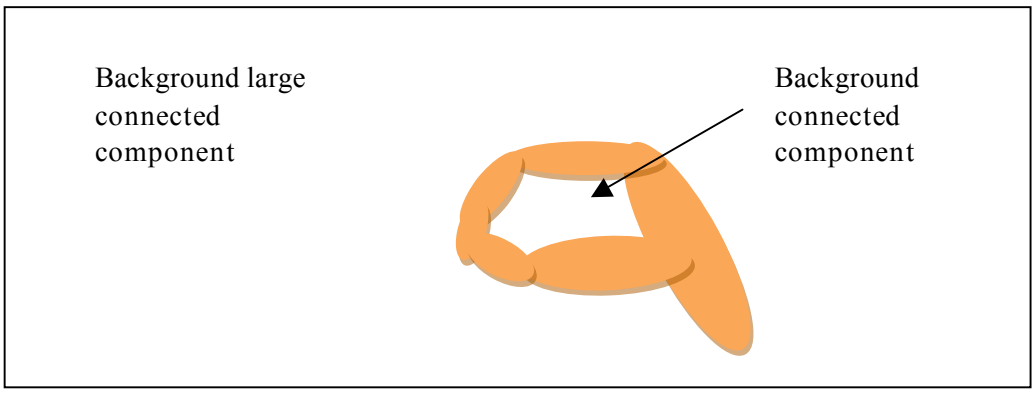


Figure 6: Connected Components Analysis

We also have to take into account that a connected component could occur at the edge of the viewing area, if the hand segments out a corner of the image. In this case, we do not want this to be detected as a connected component.

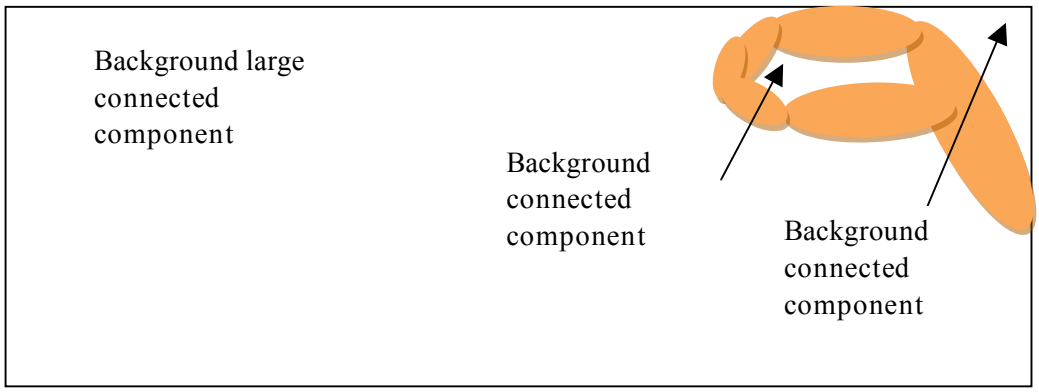


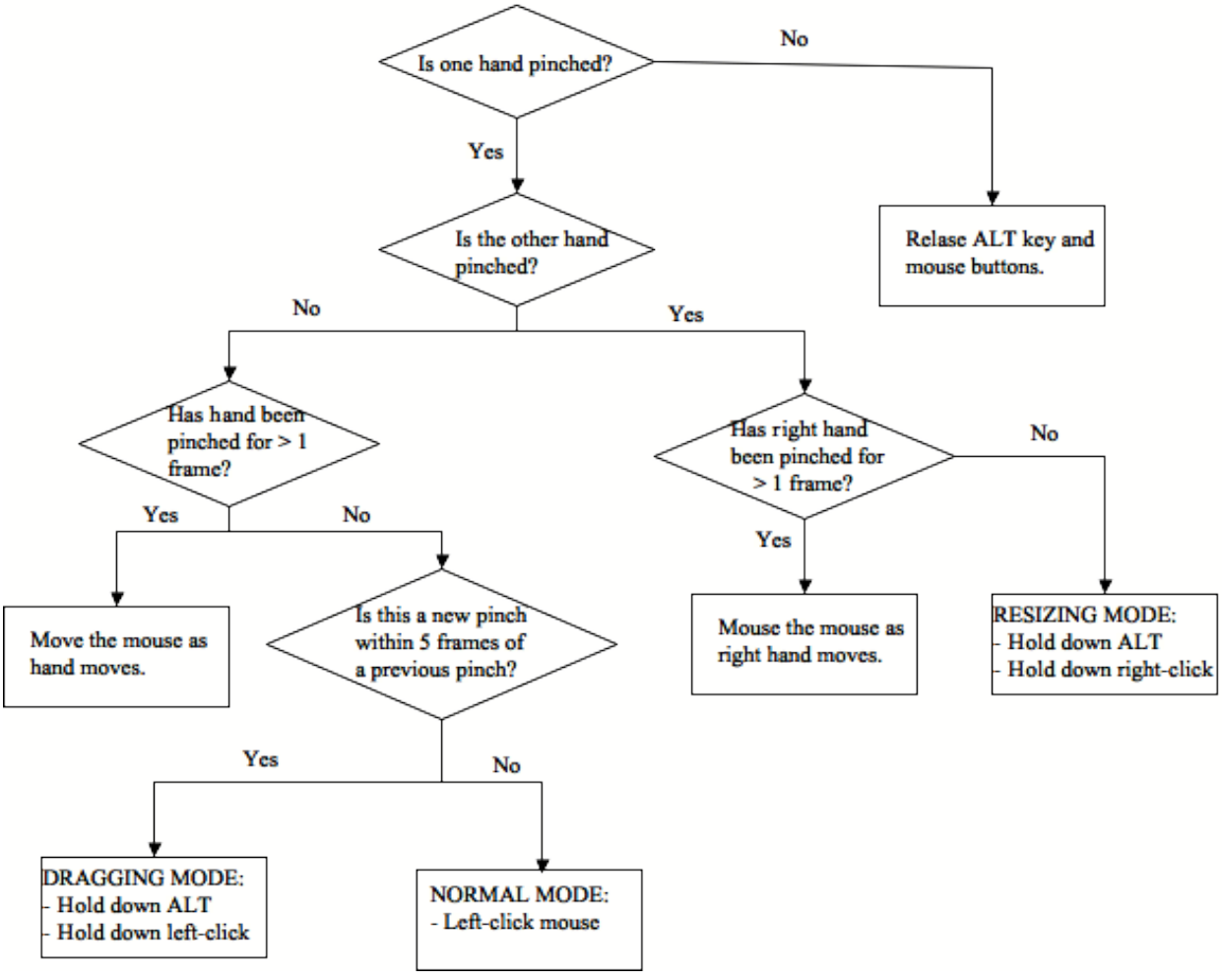
Figure 7: Eliminating connected components on the image edge

To solve this problem, we checked the Min and Max X and Y values for each connected component to ensure they were not on the border of the viewing area, which is set to 320x240 pixels (camera resolution).

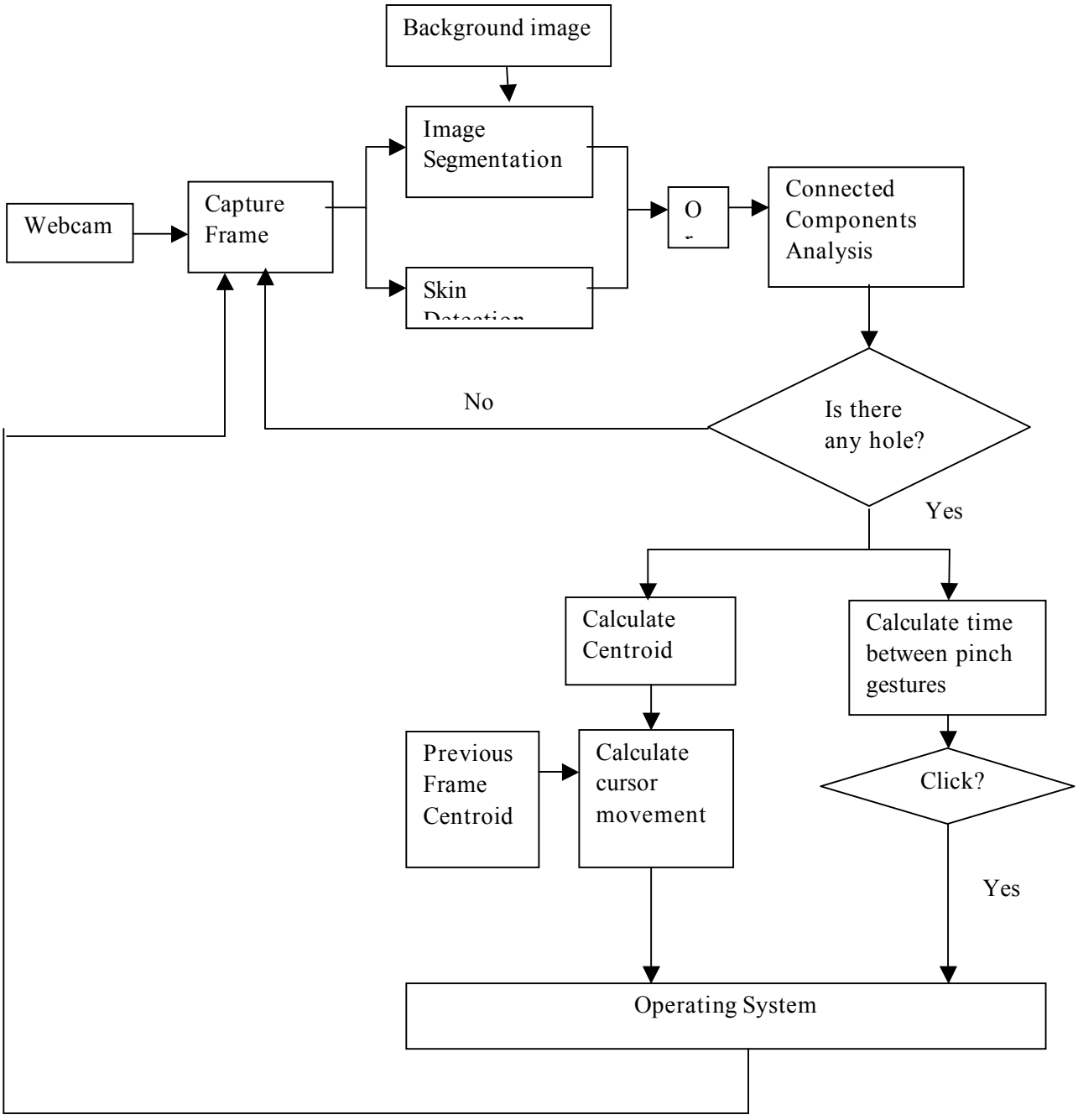
- **Mouse movement algorithm**

  The cvBlobsLib library can be used to obtain statistics about the blobs, such as their centroid, which we use as the point of reference from which to move the mouse.

  The algorithm to determine when to move the mouse and when to change states is shown in the decision tree below:



2.4. Dataflow


Background image
Image Segmentation
Webcam
Capture Frame
Skin
Connected Components Analysis
No
Is there any hole?
Yes
Calculate Centroid
Calculate time between pinch gestures
Previous Frame Centroid
Calculate cursor movement
Click?
Yes
Operating System

3. Evaluation Methodology

3.1. Tests

Our test consisted of taking test subjects and after a basic explanation of how the program works, asking them to perform certain tasks such as moving the mouse, clicking, double clicking, resizing and dragging (See copy of questionnaire attached).,

After the subjects finished playing with the interface, they were given a questionnaire. The questionnaire had several Likert-style questions asking the user to rate their experience on a scale of one to five for several aspects we found important, including usability, usefulness, learning curve, and overall enjoyment. (See copy of questionnaire attached). Afterwards they were asked if they would use it in case it was made available. An optional comment field was also provided.

While the user tested the interface, a test observer was taking notes about which of the tasks were successfully completed. The test observer also took notes about the causes of the difficulties encountered by the user.

3.2. Conditions for Success

Our main conditions for success were that the user would be able to quickly learn how to interact with the computer and more importantly that they would enjoy themselves.

3.3. Test Conditions

For our test, we focused on the ability of the user to perform the following tasks: Clicking, Moving the Mouse, Double Clicking, Resizing, and Dragging. The user was required to perform these functions within a five minute time period. The opinions of the test subject were also a major factor in testing the usability, usefulness, learning curve, and overall enjoyment of interacting with the computer in this fashion.

3.4. Test Subjects

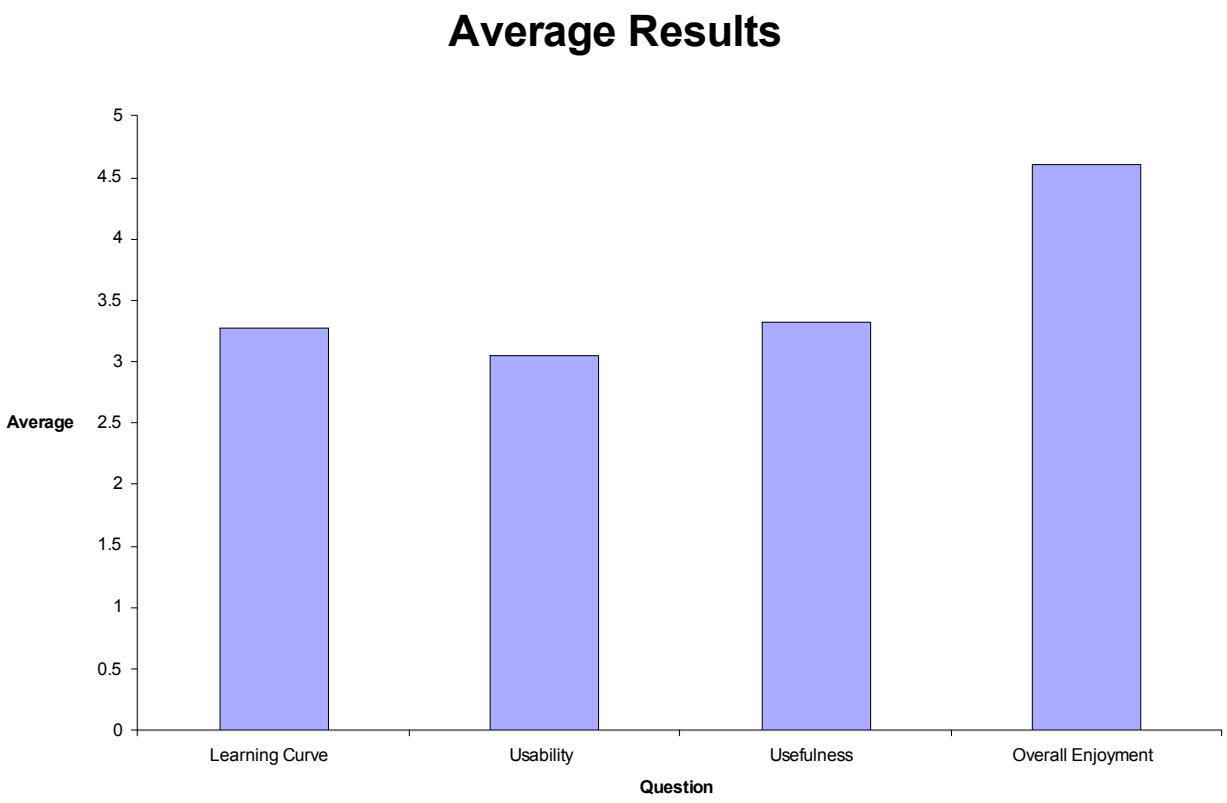
Test subjects were a random group of our peers at Iowa State University. Since our target audience for this project is advanced computer users, we obtained our 10 volunteer test subjects from computer science students who were working in the computer lab in Pearson.

The project could benefit from future testing of a wider range of computer users of different skill levels; however for the current project these test subjects will suffice.

3.5. Results Evaluations

We evaluated our results based on two things: our notes from observations during testing, and the questionnaire filled out by the test subjects after testing.

We computed statistics for these results over all users to gain an understanding of the average opinion of the system. Combined with our notes from observations during testing, these results help provide a clear picture of whether gestures are an appropriate alternative to the mouse.



Test Subject #	1	2	3	4	5	6	7	8	9	10	Average
Learning Curve	3.5	3	4	4	3	2	3	3	4	3.5	3.3
Usability	2.5	3	4	4	3	3	2	2	4	3	3.05
Usefulness	3	2	5	3	3	2	4	4	4	3.5	3.35
Overall Enjoyment	3.5	5	4	5	5	4	5	5	5	4.5	4.6

From these results we can say that the users liked the interface and 9 out of 10 people tested expressed their interest in using the interface if made available.

Test also showed that we need to improve the usability by implementing a more stable solution. With a greater degree of usability we can assume that the usefulness of this type of computer-human interaction will improve as well.

3.6. User Feedback

Some feedback we received was the need of a more stable cursor (mouse precision) as well as a wider working area.

The users also expressed their interest in the new way to interact with their PC. Some of them found it really fun and stated their willingness to use it at home. One of them even mentioned that they think the mouse is out of date and and should be reinvented.

3.7. Future improvement

We envision several possible future improvements for this technology. For starters, in his research, Wilson notes that there are several areas for improvement with his interface, including providing better cues to the user that a gesture has been recognized, such as visual or audible cues, as well as investigating better methods for implementing clicking and dragging than the current pinch-release-pinch gesture.

Our experience has shown us that the portion of the application that needs the most improvement is the image segmentation. We attempted to resolve this problem by implementing skin detection. Though an improvement was noted, the results still are not perfect. Additional work could be done to fine-tune a more accurate segmentation algorithm.

This program could also be extended by implementing new gestures and functionality. There are many applications to which this technology could be applied, other than just simple cursor control. Some ideas are gestures that would show or hide windows in Windows XP/Vista, or gestures to control browser actions when using the Internet. If appropriate means are found which could implement these applications, they would certainly be feasible to create.

Sources

Lenman, S., Bretzner, L., Thuresson, B. (2002): Computer vision based hand gesture interfaces for human-computer interaction. *Technical Report TRITA-NA-D0209*, 2002, CID-172, Royal Institute of Technology, Sweden.

O'Hagan, R. & Zelinsky, A. (1997) Finger Track – A Robust and Real-Time Gesture Interface. *Australian Joint Conference on AI,* Perth.

Quek, F., Mysliwjec, T. & Zhao, M. (1995). Finger Mouse: A Freehand Pointing Interface. *Proceedings of the International Workshop on Automatic Face and Gesture Recognition.* June 26-28, 1995, pp. 372-377.

Stenger, B. (2006). Model-based hand tracking using a hierarchical Bayesian filter, *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 28 (9), pp. 1372–1384.

von Hardenberg, C. & Berard, F. (2001) Bare-Hand Human-Computer Interaction. *ACM Workshop on Perceptive User Interfaces,* Orlando, Florida.

Wilson, A. (2006). Robust Computer Vision-Based Detection of Pinching for One and Two-Handed Gesture Input. *UIST '06*. October 15-18, 2006, pp. 255-258.

Wu, Y. & Huang, T.S. (1999). Vision-based gesture recognition: A review. In A. Braffort et al., editor, *Gesture-Based Communication in Human-Computer Interaction,* 102-116, 1999.