Computer Control Through Computer Vision

Andrew Fager, Brian Holland, Alex Stefanescu, Joshua Warner

Abstract

For this project the goal is to create a software platform that allows users to use hand gestures and computer vision algorithms to control a computer. We are creating a software package that allows someone to type, move the cursor, and click with simple hand gestures using a webcam. The focus of this project is using computer vision to create a keyboard-less and mouse-less environment for data input. We use a simple version of object analysis to track the object (such as the finger or hand) that performs the movement as well as a motion capture library called OpenCV through a C# wrapper library called EmguCV.

Table of Contents

Introduction to the Project

Computer vision is a very active field of research. It has many applications, ranging from driving a car to military surveillance systems. With digital cameras becoming increasingly pervasive, computer vision can be applied to make these cameras more useful than for simply taking pictures. Thus TR Services, LLC envisioned a project for experimental development of video capture of hand motion for computer control. For this project, we are researching how to use computer vision to interact with computers and building computer control software based upon what we find. We also look at multiple ways in which to analyze video from web cameras (webcams).

Requirements and Specifications

The system is to be centered around the use of a single, off the shelf webcam as the sole input device. For this project, the camera is assumed to be in a fixed position, pointed down at a flat surface so that the camera captures a static background. The user is not required to use recognition aides such as colored gloves, tags, or styluses.

At a minimum, this project is intended to produce an implementation of mouse input and either a basic gesture recognition or keyboard input. As this system should be designed as the primary control of a standard personal computer, analysis of the video must be accomplished in real time on the computer.

Design

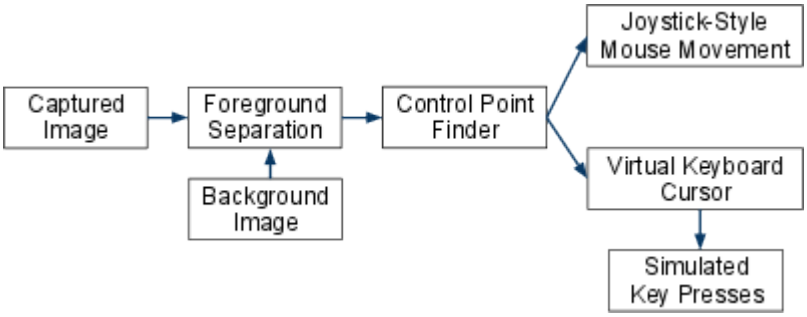

Figure 1. Overall architecture.

The system consists of several stages (see Figure 1). First, the outlines of foreground objects are detected using a foreground-separation algorithm and a static image of the background with no objects. The output of this step is a mask that denotes which pixels in the image belong to foreground objects. Next, a "control point" finder algorithm is run to track a small number of important points in the image. Ideally these control points line up with things in the image such as finger tips. Finally, the movement of these control points is fed into the mouse-movement and keyboard-control systems, which are expected to produce mouse movements and keystrokes (respectively), and feed them to the active window.

Implementation and Results of the Project

Before deciding which video processing library to use, we collected a variety of libraries to use and compared them to one another (shown in the table below). We chose to use C# because of its many productivity advantages, as well as the availability of an OpenCV wrapper library (EmguCV).

	OpenCV/EmguCV	Aforge	VisionLabs
Machine Learning	Good	Great	Bad
Vision Processing Functionality	Great	Good	Great
Support Documentation	Great	Great	Good
Variety of Examples	Great	Good	Good
Open Source	Yes	Yes	Requires Paid License

Table 1. Video processing library comparison chart.

After trying many different potential solutions, we settled on a primitive object tracking algorithm which uses a background / foreground separation algorithm. The user removes all foreground objects from the camera's view, then presses a "Capture Background" button. Our program uses this captured background to compute areas of subsequent frames that should be considered as foreground objects. Using this data, it derives "control points" by detecting singly-connected areas in the input directly below background pixels. For a system that needs only one control point, the system falls back to detecting the topmost foreground pixel in the image. After experimenting with several approaches to translating this control point to cursor movement on the screen, we went with a “Joystick” style of mouse motion (see Figure 2), where displacement of the control point from the center of image produces a constant cursor movement in that direction, where velocity is proportional to the distance from the center.

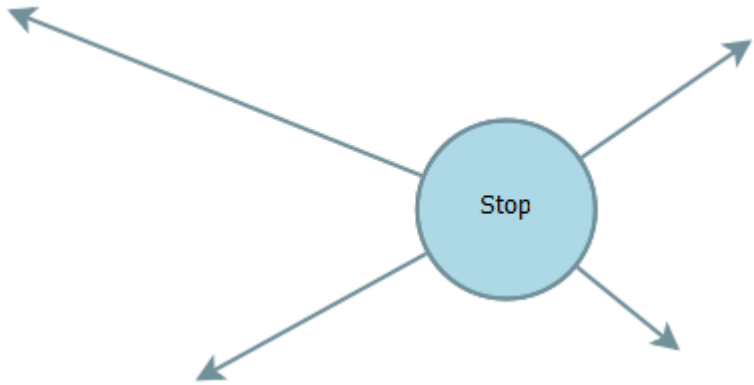

Figure 2. "Joystick" control scheme

In order to achieve clicking, support for multiple control points (Figure 3) was added. When the computer detects more than one control point the button-pressed mouse signal is sent (Figure 5). Then when the number of control points returns to one the button-released mouse signal is sent (Figure 4).

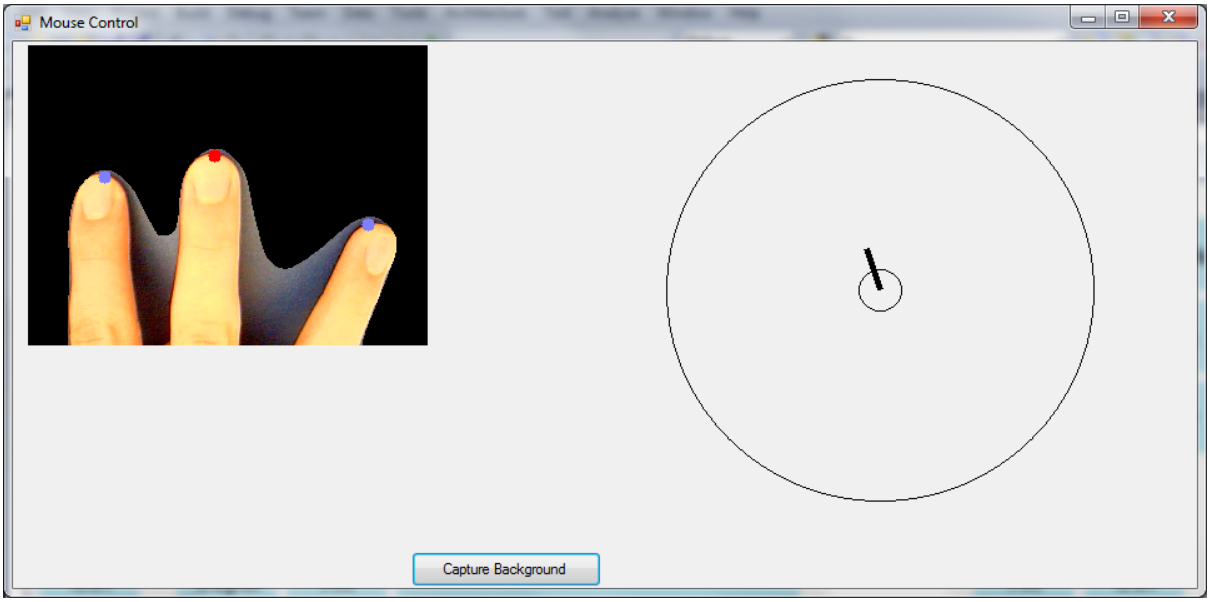

Figure 3. Mouse control in action. Note circle visualizations of the detected control points.

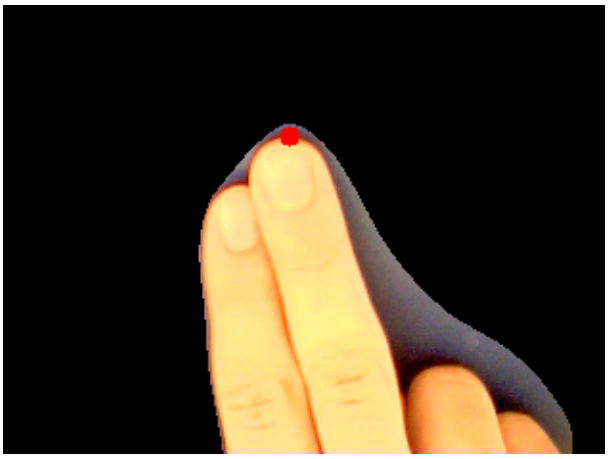

Figure 4. Mouse button up.

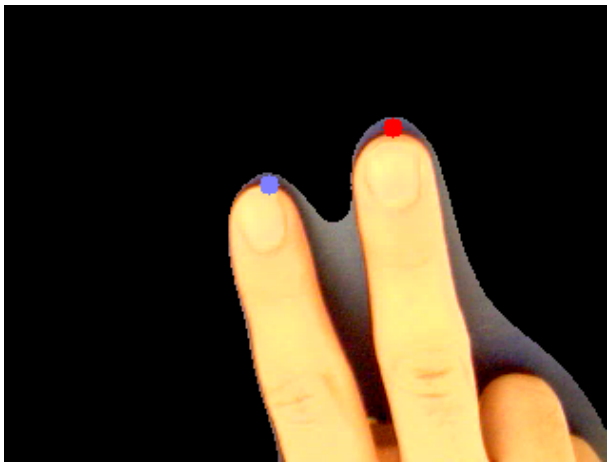

Figure 5. Mouse button down.

For the keyboard we originally thought that more precise object tracking was necessary, however, in the end the idea came that the keyboard could be done in a style similar to the mouse, using only a single control point (see Figure 6). When the control point's velocity drops below a certain threshold, a key press event is generated and sent to the active application. Then the user can move on to the next key.

Instead of using the same joystick-style movement to move the cursor across the virtual keyboard, our keyboard system directly maps the primary control point position to the virtual cursor position on the virtual keyboard. Because of the relatively small size of the keyboard

(compared to the entire screen), this method doesn't suffer from the same problem that led us to joystick-style movement: accuracy. Noise is visible in the cursor movement but not problematic.

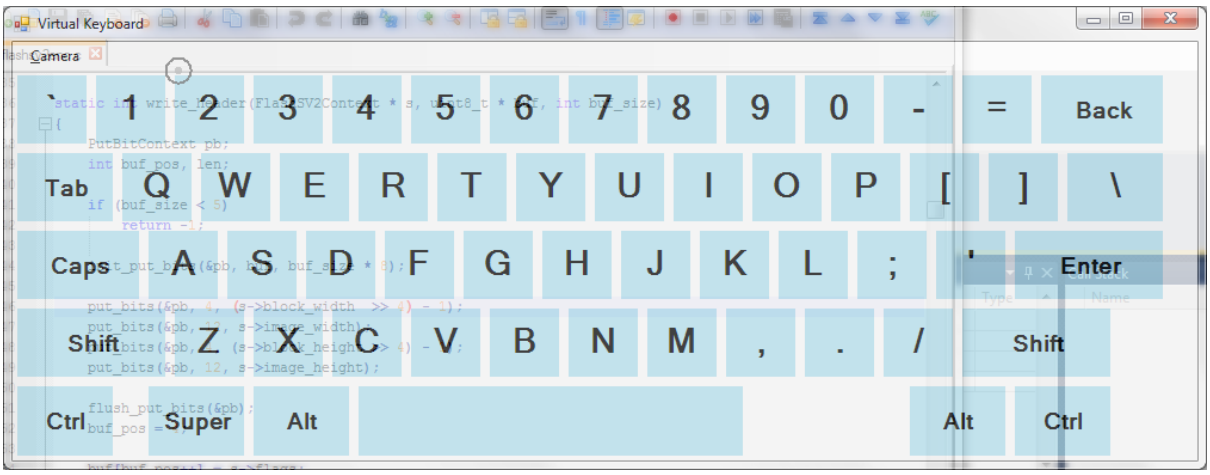

Figure 6. Virtual keyboard in action. Note the semi-transparent interface and target-like "virtual keyboard cursor".

The results of this implementation are that the keyboard and mouse are usable through a single well-placed webcam. The mouse control is easy enough to grasp in a short amount of time. Clicking with our system requires a different mindset from pushing a physical button, but is still intuitive. The keyboard is simple enough not to require any extra time to learn. Overall, this implementation illustrates that it is feasible to control a computer through a single webcam.

Scope and Project Progression

This project had a highly flexible scope, but there were three specific goals: crude gesture recognition, a functional mouse, and a functional keyboard--all using the camera. Due to our unique approach, the project began with research and investigation about feasibility. As research progressed and the attainability of these goals became clear, we moved away from research and towards implementation.

Previous Directions

Our project required intensive research, and we had to abandon many paths that didn't appear to be close to being practical. These paths include basic object tracking through segmentation, shadow removal, k-NN AI trainers, and image matrix manipulation (See Appendices). Of these, we thought that segmentation and shadow removal were paths that might eventually be successful, but were too time-consuming. Shadow removal required a deeper understanding of signal processing mathematics that we didn't have. We also dropped the development of a coarse gesture detection AI because of its inaccuracy: when we experimented with this trainer by teaching it two distinct gestures (left and right), it could only perform correctly about 50% of the time on average. This amounted to the AI producing statistically random outputs (For more information see Appendix A).

Future Directions

In the future, this project could focus on shadow removal, segmentation and AI. The shadow removal is necessary for noise reduction while segmentation allows the computer to identify multiple points (such as fingers). The AI is then needed to interpret the data from each of the multiple control points (fingers). Another future extension is creating a pointy-object recognizer that is able to find the control points from any arbitrary angle or from different and potentially opposing angles. For instance, in Figure 7 the control points are not correctly placed, but if pointy object recognition were used, the control points would be placed on the tips of the fingers.

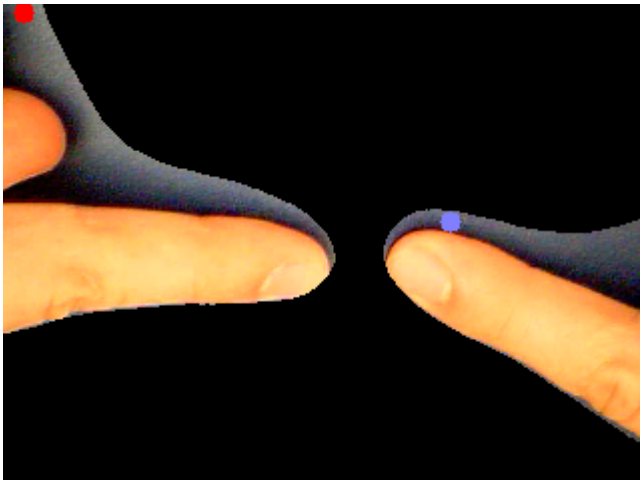

Figure 7. Scenario for better control-point detection.

Conclusion

Overall, the project was able to accomplish the required basic mouse motion, clicking, and minor keyboard functionality.There has also been headway on gesture recognition and a more advanced keyboard setup. In the research and development aspect, there is the potential for a large amount of extension of the current system.

Glossary

- Coarse Gesture - any broad sweeping gesture similar to a wave
- Control Object - any object that is tracked with a control point
- Control Point - a point in the frame (found with our algorithm)
- Fine Gesture - any small fine tuned gesture similar to sign language
- Frame - a single image from the video stream
- K-NN (k-nearest neighbor) - an AI learning algorithm that compares inputs based on a distance metric
- Webcam - a digital camera designed to be able to stream its output across a network.

Appendix (Test/Experimentation write-ups)

These appendices are the compilation of the experimentation of each of the group members. Many of the research topics we felt needed inclusion to illustrate the path to our current solution. Also for future extension, these appendices could be useful in determining what can be done and what paths have already been attempted.

Appendix A: Joshua Warner's experimentation

Tracking "Good features to track"

OpenCV comes with an algorithm to find "good features to track," which boils down to areas of high change, based on a very specific metric. After seeing the disappointing performance of the OpenCV optical flow algorithm (designed to give an approximation of the motion vectors at every pixel), I thought that computing these "good features to track" at every frame and tracking the results through time might yield more useful information that could be plugged into a pattern recognition system. After a little experimentation, I found that this approach was not feasible in the allotted time because of how inconsistent the results of the GoodFeaturesToTrack function were. Oftentimes, the size of the set of returned points would vary dramatically from frame to frame, which made it nearly impossible to track the points through time.

Background separation / Segmentation

Having heard about an algorithm for turning point clouds into meshes, and computing likely segmentations of them in Computer Graphics II, I thought it might have an immediate use here. If it were possible to separate the foreground and background of the current frame reliably and generate a set of points that was consistently on the border, this algorithm could be run to distinguish features such as fingers, and palms. The algorithm starts with a Delaunay triangulation and iteratively merges the two neighboring regions (which are initially defined as the triangles) that are "closest together". The output is a set of sets of points, which describe likely segments in the point set. In the paper, the authors show that the algorithm can reliably distinguish fingers, palms, etc.

Unfortunately, we had problems reliably separating the foreground and background of the frame. The primary problem was making sure that shadows, both diffuse and sharp, under that hands weren't considered as part of the foreground. We abandoned this line of work after assessing that it would take longer to complete than the allotted time.

Center of gravity movement of frame difference

When we tried to use a (potentially down-sampled) version of the absolute-value difference of consecutive frames, I tried tracking the "center of gravity" of the absolute difference of frames. I computed an average of the positions of all pixels in the frame, weighted by the amount of movement in them. This turned out to be not only incredibly noisy, but incredibly inaccurate. Although the cursor did move in the general direction of the movement in the frame, it jumped back to center as soon as the movement stopped. We quickly abandoned this idea as a complete failure.

Nearest neighbor algorithm for training coarse gestures

To get some form of coarse gesture recognition working (gestures such as scrolling and browser controls), I tried recognizing down-sampled versions of the absolute-difference-of-

frames image. In order to make the gestures time-independent (so that fast and slow versions of the same gesture will be recognized consistently), I computed a polynomial regression of each pixel in the gesture video clip, then selecting the nearest neighbor (based on the polynomial coefficients) from a database of prerecorded gestures. This test application included a convenient interface for testing and training the system. We decided to abandon this line of research because of its inaccuracy: on average, it only got 50% of the gestures correct (given two sample gestures) - which amounts to random guessing.

Direct mapping from camera coordinates to screen coordinates

After successfully using the smooth median to get a relatively noise-free foreground mask (though far from shadow-free), and finding a "control point" based finding the top-most foreground pixel, I thought that the easiest way to map the resultant point onto the screen would be a direct mapping, scaled appropriately to fill the whole screen. This turned out to be much less accurate and more noisy than anticipated, to the point of being completely unusable (though, significantly better than anything else we had at the time).

Touchpad-style movement

Following my attempt at direct mapping from camera to screen coordinates, I tried to implement a touchpad-style interface that operated very similarly to the touchpad on a laptop. That is, movement of the control point would correspond to movement of cursor on the screen. Instead of mapping coordinates directly, I decided to low velocities to small amounts of cursor movement, and larger velocities to much larger cursor movement (in a quadratic relationship). This worked well enough, and was respectably accurate. However, when compared to joystick-style movement (where displacement from the center of the camera view translates to constant velocity of the cursor), this was much less usable because of the unnatural gestures required to reach far parts of the screen.

Appendix B: Brian Holland's experimentation

Background Masking

After our initial and promising demonstration with InitialMouseTest, the way forward would take elements of InitialMouseTest and capture more useful data. The simple problem of determining what is the background and what is the foreground, or control object, lead to an initial, naive demonstration of simply determining mathematically the difference between the background frame and the newest frame captured by the camera. The background would be captured after the camera initializes and any subsequent frame would be compared to it. The intent was to use an image segmentation algorithm on the control object that had been successfully separated from the background. Individual control points would be determined from the segmented image and used, ultimately, to build keyboard functionality.
The first results were encouraging. A hand can be easily extracted from a busy background when the light did not change very much. This problem and the problem with shadows caused the reevaluation and eventual abandonment of this approach. I made several attempts to mitigate the effects of shadows and light changes, but these problems proved to be insurmountable within the time span of the project. There are many papers on the subject of removal of shadows which suggested methods to progress, but these did not prove to be feasible.

Mouse Control Using Boxes

Previously, the client suggested we look at the motion data to determine the control of the computer. The result was a down-sampled matrix that we used to attempt to control the cursor on computers. My attempt centered upon tracking the rows and columns with the highest average value. The point at the intersection of the first highest and second highest rows and columns would form a box, the center of which would be used to control the cursor. The first iteration only implemented the highest values. When the intersection between the highest rows and columns was found, the point stayed mostly consistent within a moving object.
Unfortunately, it was also susceptible to noise, so much of the outlying data had to be smoothed, which usually resulted in ignored data. The mouse moved reliably, assuming a consistent environment, but the movement was not smooth. When changing environments, the various thresholds had to be tweaked in order to have consistent movement sensitivity. While I believe that this system may have promise, especially if box tracking were implemented, it was abandoned in favor of a more reliable approach.

Gesture Recognition

By the end of this project, I was tasked with using a neural network to be able to recognize gestures sent from the points captured by the 'mouse'. It would be used with the current version of the ComputerControl software and should be useful for typing. Results have been promising, but it has not reached a level of reliability to be useful to integrate into the final release. Work will continue.

Appendix C: Alexandru Stefanescu's experimentation

Matrix Display:

We decided that we would use OpenCV and EmguCV to accomplish the goal of motion capture. The first thought was to use a scaled down matrix of the difference between two frames to detect motion. We decided to find a way to display the Matrix. After attempts to just calculate the matrix and output the values to the screen, I realized that rendering text on the screen would take too long and it affected the frames per second that were being captured. Since it was necessary for the frames per second to be as high as possible so that more comparisons could be made and more accurate movements could be interpreted, we decided to output the matrix to a text file or a serialized file.

Matrix Movement:

With a way to record the matrix and see how the camera handles and records different motions, the focus of the project became a way to interpret and compute the motion. An approach that was taken was to assign a true/false value to the matrix. If there is no motion in a section of the matrix then the matrix values is considered false. If there is motion in a section, then the value is true. The trick in this approach is how to define motion. There is noise in the images and a difference might occur when no movement occurs. For this reason motion must be defined as a value that is larger than another value or by using thresholding. I kept track of the last few true/false values for a section and then track the direction in which it moves. The problem with this approach is that tuning it is very difficult and must be changed from iteration to iteration because of light changes. For example if the threshold is set too high then different part of the image might be missing and give a false positive. The same could happen if the threshold is to low. This approach was very ineffectife so it was abandoned.

Dynamic mouse:

Once we abandoned the matrix movement approach, we made a joystick system that controlled the mouse very accurately, the client asked us to implement a mouse system that resembled a touch pad. One of the approaches that I attempted was to keep the joystick approach but move the center of the joystick to your finger. The center of the joystick is the area of the camera frame where motion is not transfered into mouse movement. It uses simulated spring physics in modeling the way the center aproaches and reaches the finger. As I implemented this approach I came across quite a few issues. The biggest of these issues was what happens when you reach the right side of the frame but you still want to go right. On a touch-pad, you can lift your finger to reset it. With the camera you cannot do so, because it sees it at all times. Also the way the mouse is implemented causes the mouse to jump in to the side of the frame from which your finger enters. I attempted to create a pause that would allow the center area to go to your finger. This pause starts when your finger first enters your the frame and ends a short period of time later. The problem with this approach is that it creats lag which makes it inconvinient for the user.

Appendix D: Andrew Fager's experimentation

Good Features to Track (frame to frame):

The experimentation started with the idea of experimenting with the vision library. For this the `SURFTracker` class was researched. This function is supposed to take two images, an input image and a frame, and find a set of features that is constant and present in both images. In our application it was used to track the hands, from frame to frame and then from an image of the hands to the frames of the sequence. This path proved a dead end, as the features to track created a point array that was inconsistent as things moved. Also, there was little data produced in order to analyze the motion.

Artificial Intelligence (k-NN):

From this failure, I continued experimenting with EmguCV/OpenCV. Moving on to the AI that would be required I began to work with the nearest neighbor algorithm that was built into the library. The built-in algorithm would classify points based on a training set that allows the AI to map the input points to a point in the training set based on a Euclidean distance. We thought that this type of AI would be necessary in order to categorize different gestures. This would allow us to develop a way for the computer to learn what an action meant and what that action would look like relatively, so as to not force exact motions for the gestures. This path was a letdown. The library's distance algorithm was not good with points and was not expandable. After the initial experimentation we decided that it would be easiest and fastest to implement our own nearest neighbor algorithm. However, we based our AI on a set of matrices of image data.

Maximum Motion Mouse Movement:

After the unsuccessful AI, I began experimenting on manipulating and utilizing the data in a matrix in order to control the mouse. I experimented with having the mouse follow the direction of greatest motion that is to follow the maximum value present in the matrix. This method worked but did have some erratic behavior if long sweeping gestures weren't used. Also, fine gestures were challenging because of this erratic behavior. Due to this behavior, the maximum motion mouse movement experiment was dropped.

Shadow Reduction and Removal:

The final phase of research that I undertook was looking into shadow removal. We as a team, decided that, for good motion tracking of objects for the possibility of a keyboard, being able to identify only the fingers would be necessary. I read multiple papers about shadow reduction and removal. After researching I began to implement a form of shadow reduction. This attempt proved unsuccessful; however, based on the papers it can be done. The primary concern when stopping the development was the time consumption involved. More time devoted to shadow reduction could be necessary for an ideal solution.

References

Dey, T., Giesen, J., & Goswami, S. (n.d.). Shape segmentation and matching with flow discretization. Retrieved from https://blackboard.mines.edu/courses/1/spring2010-CSCI498B-CSCI598B-MATH498B-MATH598B/content/_255071_1/feature.pdf?bsession=4993596&bsession_str=session_id=4993596,user_id_pk1=25052,user_id_sos_id_pk2=1,one_time_token=C7CE848538E6D14FEE514B84288AA591

Emgucv. (n.d.). Retrieved from http://www.emgu.com/wiki/index.php/Main_Page

Finlayson, G. D. , Hordley, S. D. , & Drew, M. S. . (n.d.). Removing shadows from images. Retrieved from http://www.cs.sfu.ca/~mark/ftp/Eccv02/shadowless.pdf

Hardesty, L. (2010, May 20). Gesture-based computing on the cheap. *MITnews*, Retrieved from http://web.mit.edu/newsoffice/2010/gesture-computing-0520.html

Jiang, H., & Drew, M. S. (n.d.). Tracking objects with shadows. Retrieved from http://www.cs.sfu.ca/~mark/ftp/Spie03/spie03.pdf

Matsushita, Y., Nishino, K., Ikeuchi, K., & Sakauchi, M. (2004). Illumination normalization with time-dependent intrinsic images for video surveillance. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, *26*(10), http://www.cvl.iis.u-tokyo.ac.jp/papers/all/650.pdf

"Minh" (Designer). *Podtech ms research clip* . [Web]. Retrieved from http://www.youtube.com/watch?v=AtmwQnUlEmc&feature=player_embedded

Novák, O., Ondřej, F., & Ondřej, J. (n.d.). Gesture recognition. Retrieved from http://ibm-cvut.felk.cvut.cz/srp2/gesture_recognition/doc/doc.pdf

Opencvdotnet. (n.d.). Retrieved from http://code.google.com/p/opencvdotnet/

Prasan, M. (2008, March 18). *Shadow Removal with Opencv*. Retrieved from http://maleesh.wordpress.com/2010/03/18/shadow-removing-with-opencv-2/

Wang, R. Y., & Popovic, J. (Designer). *Real-time hand-tracking with a color glove*. [Web]. Retrieved from http://www.youtube.com/watch?v=3JWYTtBjdTE&feature=player_embedded