THE PENNSYLVANIA STATE UNIVESRITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

AN INTRODUCTION AND OVERVIEW OF A GESTURE RECOGNITION SYSTEM IMPLELMENTED FOR HUMAN COMPUTER INTERACTION

ISAAC D. GERG

Summer 2004

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Engineering
with honors in Computer Engineering.

Approved: _____ Date: _____
Dr. Richard Tutwiler
Thesis Supervisor

_____ Date: _____
Dr. Chita Das
Honors Advisor

_____ Date: _____
Dr. Raj Acharya
Department Head

ABSTRACT

The following paper develops the theories and methods used in a gesture recognition system implemented as a human-computer interface (HCI). The gesture recognition system recognizes four fundamental static hand gestures and variations for a total of nine gestures. The system uses a variation of the CAMSHIFT algorithm for hand tracking and a minimum distance classifier for classification. The Win32 API is utilized to perform the desired actions, which are determined by a microstate/macrostate architecture by which contextual information is used to correct any falsities in single frame classification. The state oriented model uses order statistics to provide corrections. The software, named MTrack, is implemented using Borland Delphi 7.0 and DirectX and is specifically designed for low-end desktop hardware. E.g. A 600MHz Pentium with commercial off the shelf (COTS) camera hardware.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

This paper introduces the motivation, theory, implementation, and applications for a hand gesture tracking and translations system. Furthermore, system architecture and implementation details are presented. The algorithms, implemented in Borland Delphi 7.0, are explained in a manner independent of any programming language. The goal of the explanations is to allow any programmer to understand the algorithms to a degree that allows implementation of them in any language, including C, C++, and Java. However, the implementation of the translation of hand gestures to mouse actions will focus on the interface of the Win32 Application Programming Interface (API). The concepts will be explained in such a way to allow adaptation to any operating system including Linux. Partial source code for the software is provided in the appendix along with a CD containing a demonstration.

The software, named MTrack, is designed to run in any Microsoft Windows environment featuring compatibility with the Win32 API and supporting DirectX 9.0. It can be interfaced with most Universal Serial Bus (USB) camera devices including the popular Logitech QuickCam. The software was designed to run on a 600 megahertz (MHz) Pentium a minimum frame processing of fifteen frames per second (FPS).

This paper introduces the utility, desires, and sample implementations of a gesture recognition system in chapter one. Chapter two describes the motivations and theory behind common recognition and classification algorithms pertinent to this experiment. Chapter three outlines the implementation details needed to construct the gesture recognition software created for this paper. Analysis of the software along with sample execution is described in chapter four. Finally, chapter five concludes the research and is followed by the appendix containing the algorithms used in implemenation.

1.1 Utility of a Gesture Recognition System

The need for a gesture recognition system (GRS) has been very prevalent over the past decade[1][2] [3] [4]. Many different problems have found their solutions in gesture recognition. More so, many unique applications have arisen from this technology. Some of these technologies include: interfaces for smart classrooms [2] and even vehicle interaction [3]. These systems have gone so far as to even develop easy to use interfaces for non-technical users to utilize [4].

The need for a perceptual human computer interface has been around for some time now. With the onset of computer encounters and interactions becoming more prevalent in our daily life, the need for an intuitive interface to interact with a computer has increased. Ultimately, computer scientists work towards harmonizing our lives with our digital surroundings.

Hand gestures are a common way humans communicate with others in a harmonious and informal way. Humans continuously communicate with their hands whether it is to show outward feelings such as disgust or to motion for input from others. Despite the clear interpretation most hand gestures have to humans, gestures are mainly an informal means of communication based heavily on context. Many different versions of a wave can say “Hello.” A single gesture can also have different meanings across cultures.

The interpretation of many hand gestures is purely contextual. For example, the same waving hand motion for “Hello” may also mean “Goodbye.” How does one deal with such differences? Though many differences may be found in gesture subtleties, many gestures put into another context can be reinterpreted differently. The idea of a computer accurately interpreting gestures solely on movement alone is difficult. Thus, contextual information plays an important role in gesture classification.

Many human computer interaction systems involving gestures rely on context for classification. Many systems predict the context of the user by issuing the interpretation of a specific gesture with a predefined meaning. Thus, the user cannot make just any motion to give a command to the computer. These systems prove most efficient and reliable.

Recently, researchers at Penn State [5] have developed a computer kiosk using gesture recognition to help the visitors find their way around campus. Other projects include American Sign Language recognition [6] and the use of the hand as a pointing device similar to a laser pointer [7]. Someday, we may find signaling a traffic light at a crosswalk no longer involves pushing a button, but instead, making a motion.

With computer processing power increasing, the ability to sort and discriminate data improves. This improvement aids the image processing / computer vision community as the complexity of a software can increase and thus provide superior processing. Some of these improvements may lead to longer image processing pipelines including features such as dynamic image filtering and sophisticated tracking methods.

1.2 Past Experiments

Many different gesture recognition systems have been implemented over the years. This section, presents a survey of gesture recognition systems found useful and highlights their significance. Later, an overview is provided of each system and its theoretical framework.

The movie *Minority Report*[1] has inspired the creation of many gesture recognition systems. For a synopsis, the movie features actor Tom Cruise using a pair of "special gloves" by which he

[1] The movie *Minority Report* trademarked and copyrighted 2002 by Twentieth Century Fox and Dreamworks, LLC. All rights reserved. Property of Fox.

is able to interface with a futuristic computer. Mr. Cruise uses the gloves to manipulate data streams of images and movies. The fluency of the special effects depicted in the movies is particularly smooth and very believable given today's technology.

One such Hollywood inspired gesture recognition system is the Mouse WebCam created by E. Garcia and L. Morentin [8]. Their system uses color to track the hand; specifically, a green kitchen mitt for which the application is tuned to use. The system is written in Visual Basic. It uses edge detection to perform the hand extraction from the background and then uses first and second order moment analysis to classify gestures to form three specific mouse actions: left button down, right button down, and no button down. The authors also point out details in the movie stating Tom Cruise's glove is black and contains three blue diodes. As expected, the authors also point out that "[their system] is actually more difficult [to implement than] Tom Cruise's gadget, as detecting [bright blue] lights is easier than detecting colors."

Many gesture recognition systems are used as pointing devices. The system presented by Cantzler and Hoile [7] is an example of this. Their system is designed to replace conventional 2D pointing devices like those such as touchpads, trackballs, and mice. Because of its design, it can work with any visible screened device. The system uses low cost camera hardware to shoot a display, a monitor, find the center of the display, and then implement a pointing device against the scene. Their claim is that the system works as if the user is controlling the mouse with a laser pointer.

The third system to be examined is a mobile robot tracking system by Pylkkö [9]. This system uses color to extract an object from a scene. This system pays special attention to computational complexity as it is implemented for robotic use where resources are limited. The author goes on to develop a notion of color judges, which are used to help achieving real-time

performance. These color judges also make the system highly configurable for many different tasks. The author shies away from pixel based noise reduction schemes because of their computational cost. Instead, markers are used as an interface between the tracking system and robot control. The system has the ability to track many different objects simultaneously.

An interesting way to interpret American Sign Language (ASL) is developed by Wysoski, et al [10]. The authors use histograms based on hand postures for gesture classification. Their algorithm achieves the property of rotation invariance, which is much desired in gesture recognition. The images are preprocessed by a color filter combined with mathematical morphology. Boundary chord size histograms based on Peripheral Direction Contributivity (PDC) are used for feature extraction. Several methods were used for classification including a multilayer perceptron neural network.

A paper by Triesch and von der Malsburg [11] uses elastic graph matching for recognition. The system works well against complex backgrounds (86.2% success rate). More so, the system works with gray scale images and uses no color information. The graphs for classification are formed by local image descriptors, called jets. The jets are based on a wavelet transform along with complex Gabor-based kernels. As mentioned by the authors, it is believed Gabor filtering resembles methods of perception used by humans. Despite their lack of color analysis, the system is able to work against complex backgrounds.

Another ASL recognizer system was implemented by Starner, Weaver, and Pentland [6]. In their paper, they investigate ASL recognition with two different camera positions: hat mounted and desktop. This review only covers their unique cap mounted system. Both of their systems implement a Hidden Markov Models [12] (HMM) for classification. Forty ASL words are able to be recognized with their system. Their system uses hand color and eight-way connectivity

for broad hand extraction. Second order moment analysis is used, along with some tricks, to assemble a sixteen element feature vector. Along with the use of HMM's as a classifier, the system yields high accuracy rates by also including contextual information by using lexicons to form complete sentences.

The last experiment covered is by Guo. et al. [3]. The authors feature a recognition system used for human-vehicle control. They use a neural network for hand segmentation and use invariant moments for gesture classification. They also implement a method to separate the hand from the forearm based on the circularity of the palm. As an example, their system is able to perform turn left, turn right, move forward, move backward, move, and stop.

1.3 System Requirements

A good gesture recognition system must be adaptable to change. As previously mentioned, gestures are heavily dependent on context for proper interpretation. It is often a good idea to predefine gestures and their interpretation. With this information given to the user, the system has *a priori* knowledge of the context and gestures are correctly interpreted and safely executed.

For this project, the following attributes are desired of a gesture recognition system:

a) Insensitivity to changes in scene illumination [1]: the system must respond the same way to different lighting conditions including transient ones.

b) Gesture recognition must be performed in real time: gestures must be processed at a time of 0.067 seconds or shorter. This translates to a video frame rate of approximately fifteen frames per second or better.

c) Distinguish many gestures accurately: the classifier must be able to accurately distinguish a specific gesture from another across different hand shapes, colors, and deformations such as rotation and scaling.

d) Accurately track and recover during loss or occlusion: the system should recognize when no hands are present in the scene. It must track the desired object in a scene with many distractions including other hands or faces (flesh colored objects).

e) Intuitive gesture to action translation: the gestures recognized by the system must be able to be reasonably performed by the user without causing fatigue or discomfort with extended use. The gestures should be natural and intuitive to the function they describe. Transitions between commonly invoked functions should be natural and comfortable.

Guo, et al. [3] present a very concise framework of requirements. In their paper, they desire a gesture recognition system working in different environments including both indoors and outdoors. They conclude that a gesture recognition system should include the following properties (quoted directly from source):

a) **Dynamic background**: The vehicle faces a complex background with a variety of scenarios. A hand gesture recognition system that only works in the uniform and static background may not be flexible enough for outdoor application.

b) **Variable lighting condition**: In the outdoor and dynamic environment, lighting conditions are uncontrolled. The hand gesture recognition system is required to have the adaptability for variations of lighting condition.

c) **Real-time interaction**: It is essential that vehicle could recognize human hand gestures in real time. Slow system response is unsafe and not convenient for users.

d) **Come as you are**: For hand gesture recognition, it is desirable that recognition system is user & device independent, so that different users are able to operate it without wearing special devices (e.g. data gloves, colored markers, special sleeves etc.).

e) **Natural tendencies**: In hand gesture recognition, it is suggested that human's natural intuitive gestures be selected as prototype gestures, which are easy for public users to learn.

1.4 Implementation Overview

This section provides an overview of the requirements used to implement MTrack. This section will lead to the notion of *execution* whereby actions conducted by the user are interpreted and dispatched to the operating system as mouse commands. Thus, when execution is not desired, a computer can interpret a gesture, display the interpretation, but execute no mouse movement or other mouse actions.

MTrack was created to run on Microsoft Windows 32 bit operating systems with DirectX 9.0 or higher installed. The software was created using the Borland Delphi 7.0 development environment. The DSPack [14] library was used to access DirectX functions such as video capture and play. A matrix library from the JEDI Math [15] group was also utilized to provide ease when completing matrix calculations.

Most algorithms in MTrack were designed using Mathworks [16] MATLAB. After qualifying, algorithms were then ported to Object Pascal for use.

The gesture recognition system is able to recognize four fundamental static [1] hand gestures. These are:

- Open hand open fingers
- Open hand closed fingers
- Fist
- Exclusive extension of the index finger (the simple pointing gesture)

The variations of these gestures are formed by rotation of the hand.

The fundamental gestures are translated to computer actions. The gestures translate as follows:

- Open hand open fingers translates to mouse up. A mouse button is considered to exist in two states, up or down. When the user presses a mouse button, that button is considered to be in the down state.
- Open hand closed fingers translates to mouse down.
- A fist translates to active window minimization.
- Index finger extension translates to cycling windows through the Z plane (ALT-TAB).
- Gesture rotation translates to middle mouse button scrolling.

Note that for the fist, an action performed from rotation does no really make sense corresponding to the action of the fist and therefore the rotational data is ignored. It is also not physically comfortable to rotate the fist accurately over long periods of time. With this in mind, the translation schema was chosen for maximum hand and arm comfort.

The gesture recognition system is composed mainly of two subsystems, the hand extractor/tracker and the classifier. The hand extractor focuses on segmenting a human hand from the video stream. It relies on the hand's color, or hue, to determine its location in an image. The RGB (red, green blue) input image from the camera is converted to HSV (hue, saturation, value) space whereby the hand is then segmented from the rest of the image. The user can

control parameters such as the hue and even help filter out errors in the hue plane due to very high or low saturation or value values. During the segmentation process, a feature vector is generated which is passed along to a high-level classifier. The high-level classifier function looks at the feature vectors and determines what the best course of action is for the mouse. It uses a statistical filter to help detect and correct misclassified gestures if any should arise.

Some of the functions available to the user are:

- Ability to ignore pixels based on extreme values of saturation and value.
- Adjustment of the height to width ratio of the hand.
- Hand color (hue) thresholds.
- Expansion of search window between continuous frames to track the continuity of the hand through time.

The hand is then tracked through time using a variation of the CAMSHIFT algorithm. This algorithm uses an adaptation of the MEANSHIFT algorithm to track color objects based on a histogram of the desired object color. CAMSHIFT is useful as it is computationally efficient and is very resistant to distracters such as a second hand or the face appearing in a scene. This algorithm is used to construct binary images or blobs of the hand.

A feature vector is constructed from the binary images created during the CAMSHIFT algorithm. First and second order invariant moments are the main methods used as a discriminant to construct a feature vector. These moments are based upon normalized central moments and invariant from rotation, scale, and translation [17]. From these feature vectors, a minimum distance classifier is used in conjunction with a small database of template images to classify an unknown hand gesture. The minimum distance classifier uses the Mahalanobis

distance as it's distance metric. This system has been proven to work in many other areas of recognition including optical character recognition [18] [19] [20] [21].

Mouse action is not translated on a frame-by-frame basis. A new framework is constructed for the actual decision making process using the notion of *microstates* and *macrostates*. The method is similar to playing poker. Each card (frame of video) in a hand by itself does not amount to much, but in the presence of other cards and the order of their arrival, one can conclude the best way to play the hand (the macrostate). MTrack works similarly. It constructs the macrostate (action to take) from microstates using order statistics. This state architecture is designed to correct misclassifications by injecting contextual information into the data stream via filtering.

2.0 MOTIVATIONS AND THEORY

Since gesture recognition's start by Myron W. Krueger during the mid seventies [3] many practical applications and theories have arisen. Gesture recognition has adapted to many different forms from facial gestures [22] to complete bodily human action [23]. Since then, several more applications have arisen and created more of a need for this type of recognition system. Today, a movement towards hand controlled vehicles [3] and kiosks [5] is prevalent.

2.1 Theoretical Framework for Comparison

Generally speaking, a gesture recognition system is broken down into a few fundamental components. These components are:

1. Noise reduction system
2. Object extraction engine
3. Discriminant
4. Classifier
5. High level abstraction
6. Action engine

Each piece plays a key role in the systems that preclude and follow it in the pipeline. Often, these systems are quite tightly intertwined and almost indistinguishable from one another. A description of each component of the system follows next.

<u>Noise Reduction System</u>

Almost all mechanical or electrical systems are highly susceptible to noise or signal distortion. This is very prevalent in most low end and low cost commercial off the shelf camera devices. In the systems studied in this paper, camera noise was a problem. This noise is usually caused by inconsistencies in the capture elements or noise added to the signal via wire transmission. In any case, this noise is often salt and pepper flavored or some form of Gaussian. Worse off, the noise is not consistent, meaning it does not strictly follow the same distribution exclusively. The distribution of noise can change under different lighting conditions or in the presence of infrared of ultraviolet light; in which case, the camera may flicker appearing to loose frames.

Very few systems reviewed in this paper attempted to remove noise in a general sense by means of Wiener filtering or kernel convolution such as averaging. In some sense, these types of filters may drastically improve recognition and especially tracking. A scene better suited for tracking could be formed by region segmentation using a split and merge algorithm. Systems such as these are often not robust enough to be exclusively used for general use. Therefore, in this particular application, image filtering was simply beyond the scope of the study.

Object Extraction Engine

This is where the heart is of a gesture recognition system lies. In most cases, the results of the feature extractor are only as good as the data given to it. Many different types of hand extraction algorithms are used but they almost all involve color image analysis. There are some that involve difference images [24] and even feature points [11].

The most common method used is extraction by color. There are two types of methodologies used. One assumes the tracking of human flesh. The other assumes the user has

some type of glove or object attached to the hand. The use of gloves brings up an interesting point about human flesh tracking. Often, solid colored gloves are used to help the computer better isolate the hand from the background. Gloves also provide a much more uniform color distribution of color across the hand thus, the acceptable color tolerances are smaller, making the hand easier to extract.

Discriminant

The discriminant separates most gesture recognition system designs from others. Given a scene, the discriminant is the characteristic by which an attempt is made to separate data by similarity or dissimilarity. A feature vector is created uniquely defining data with the properties of stability where by small changes in the data realize small, but predictable changes in the feature vector. It is this property that allows for classification.

Discriminant functions can be created in many ways. In the case of GRS's, researchers have used Gabor filters [11], KLT trackers, graphs [11], moment analysis [18], and principal component analyses [26] to name a few. Since the discriminant function is heavily dependent on the data coming from the object extraction function, the extraction function plays a large role in the choice of feature extractor. For example, in the case where a hand is segmented into a binary image from a background, principal component [25] or moment analysis [6] is commonly used. In cases where color is not present, the use of Gabor based filter kernels [11] or KLT tracking is sometimes used.

Classifier

The classifier is an essential part of a gesture recognition system (GRS). The classifier is responsible for determining the class or type of a gesture. In the case of GRS's, the input of a classifier function is a feature vector describing an unknown gesture [26]. The output of the function is the class, or type, of gesture from which the feature vector was derived.

Generally, a good discriminant separates feature vectors into mutually exclusive subspaces in feature space. The classifier functions by creating a decision surface or manifold for each subspace by which to determine the class of an unknown gesture. Sometimes these decision surfaces are generally unknown at runtime and the classifier must create them as data arrives. This type of configuration is known as *unsupervised learning*. In this project, the decision surfaces are formed before runtime and unknown feature vectors are measured against known data samples. A gesture is classified with the class it is closest to in feature space. This type of classifier is known as a *minimum distance classifier*. Often, this type of classifier always returns an answer. However, thresholds may be set to simply declare a gesture as unknown if it is too far from any known subspace.

Many systems actually involve a number of different classifiers and other mechanisms to weights their results to reach a highly confident decision. Examples of such systems are ones utilizing fuzzy logic of voting scheme for classification. The use of neural networks with such fuzzy logic and voting schemes is often used as they have the ability to be trained and retrained. In these systems, if a particular classifier of a set does not work well in the given situation, its contribution to the decision is lowered accordingly.

High Level Abstractor

The high level abstractor is the last step to correct any classifier errors. Safety is of the utmost importance during this stage as this is the last step before executing an action. This is important for many situations involving heavy machinery [3].

Classification on a frame-by-frame basis is not always reliable. Often, it is desired to introduce contextual information about a scene into the final class assignment. It is possible to correct frames which otherwise may result in misclassification by introducing high-level contextual information into the function. In the case of ASL, contextual information can improve classification simply by studying the grammar formed by the user. For instance, in an ASL recognition system the user forms a gesture falling into one of two classes. One of the classes is for a verb and the other a noun. By analyzing the sentence structure, a system can correctly interpret the gesture.

Action Engine

This section actually performs the operation requested by the user. This may be to move a vehicle, crane, or mouse pointer. Safety mechanisms are often also employed here especially in the maneuvering of heavy machinery.

The action engine is likely the most implementation dependent stage of a GRS. Generally, there are few frameworks allowing for generic or inherited behaviors. The execution of this stage is critical though to the user. It is desired that the actions complete in a timely manner, most desirably being in real time. Most GRS's invoke operating system level API calls to accomplish their tasks for this reason.

Some systems may choose to provide feedback from hardware. Such as the case with vehicle control, feedback from motors can aid classification or other stages of gesture recognition by their state.

2.2 Overview of Theoretical Practices

This section provides a mathematical synopsis of relevant theories to gesture recognition.

Noise Reduction Systems

Noise reduction systems are not usually covered in gesture recognition studies. Because of this, only a short overview will be provided. Most noise begins at the camera and can sometimes be removed by adjusting the camera appropriately. Each camera is different yielding no absolute way to remove all noise uniformly. Based on an estimate of a camera's point spread function (PSF), blur is often eliminated with generic filtering. Other techniques for filtering involve thresholding and mathematical morphology.

Mathematical morphology is a way to analyze data by its shape. More so, it can be used remove unwanted shapes or enhances others. Mathematically speaking for binary images, morphology is the convolution of a binary mask across an image for which each pixel is of the output is determined by some function of the mask and the original image.

Two fundamental operations of morphology are shown below [27].

A binary image X is represented by the set:

$$X = \{z : f(z) = 1, z = (x, y) \in \Re^2\} \tag{2.1}$$

X is *added* (termed Minkowski addition) to binary image B as such:

$$X \oplus B = \bigcup_{b \in B} X_b \tag{2.2}$$

B is *subtracted* from X (termed Minkowski subtraction) as such:

$$X \oplus B = \bigcap_{b \in B} X_b \tag{2.3}$$

These fundamental operators, (1) and (2), are constructed to create the operations of erosion, dilation, opening, and closing.

Mathematical morphology often works well for most binary filtering situations. However, it often performs poorly with different or dynamic contexts. For example, a system performs thresholding of an image to isolate a hand (See Figure 1). It also notices that some of the hue values are incorrect and speckles or other appendages are visible on the hand. To remove these aberrations, morphological opening is performed. The mask, termed a *structuring element*, is created to be just bigger than the unwanted speckles. This method works great when the hand is close to the camera. All the digits are visible and the binary image clearly depicts the hand silhouette with no other noise. However, when the user moves their hand further from the camera, the speckles are soon as large as the fingers. This causes the fingers to be removed during the opening operation.

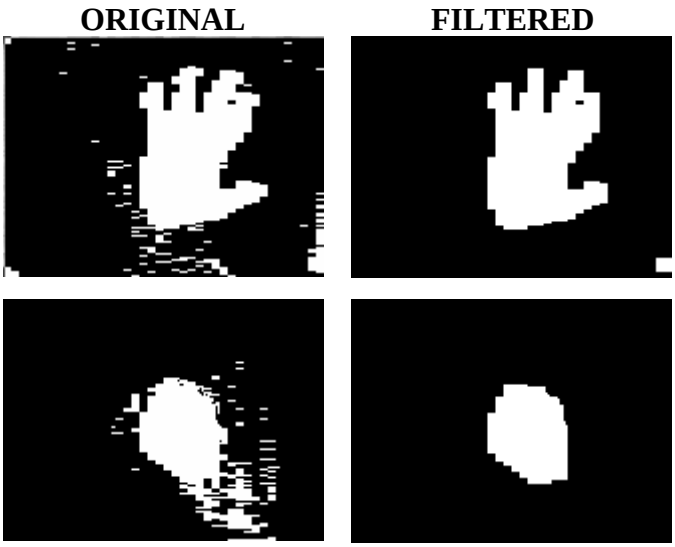


Figure 1. Two noisy hand gestures filtered via morphology. Each image is eroded three times, opened, and then dilated three times. A 3x3 mask was used in all operations.

A way to solve this problem is to use a smart or adaptive filter. This filter can be based on: the distance the hand is from the camera, connected component analysis, or conditional dilation.

Though these algorithms may work very well, they consume a lot of CPU cycles and are therefore not used with standard, low end PC hardware. More so, noise reductions algorithms excel best when specialized hardware is designed for their use like associative memories and such.

Such hardware is now available on desktop computers. This allows these more complex morphological operations to perform efficiently. An example of such hardware are the MMX extensions available on the Intel Pentium architecture. These extensions contain instructions allowing for single instruction multiple data (SIMD) execution.

Object Extraction Engine

The object extraction engine extracts the hand from the scene and also tracks it. Almost all hand tracking algorithms examined use some sort of color extraction scheme. The extracted color is often formed into a probability density function or in the case of MTrack, a binary image.

Common ways to extract color information is by converting from RGB (red, green, blue) colorspace to HSV (hue, saturation, value) colorspace. Most camera input is easily available to the user naturally through RGB or YUV (luminance and chrominance, component video) colorspace. Thus, methods using these streams are preferred but however are often not the best choices for describing color as humans think of it, hue.

HSV colorspace separates color into the humanistic terms of color, lack of color, and brightness. This separation creates an easy to define subspace of acceptable ranges of color to track. The RGB to HSV colorspace calculated conversion is as follows [28]:

$$H = \begin{cases} \theta & \text{if } B \le G \\ 360 - \theta & \text{if } B > G \end{cases} \tag{2.4}$$

where theta is:

$$\theta = \cos^{-1}\left\{ \frac{\frac{1}{2}\left[(R-G)+(R-B)\right]}{\left[(R-G)^2+(R-B)(G-B)\right]^{1/2}} \right\} \tag{2.5}$$

$$S = 1 - \frac{3}{(R+G+B)}\left[\min(R,G,B)\right] \tag{2.6}$$

$$I = \frac{1}{3}(R+B+G) \tag{2.7}$$

Often, the saturation and value are used to remove pixels lacking color and those too bright or too dark. As saturation and value reach their upper and lower bounds, the accuracy of the hue component decreases. Due to the orientation of the HSV space, it is easy to define a subspace of acceptable ranges of saturation and value. When this is done, pixels with a saturation and value below a predetermined limit are not considered part of the object of interest, even if their hue is acceptable. Similarly, this is also done for pixels with too much color or brightness.

There are many schemes for object extraction that do use RGB and others that do not even use color altogether. MTrack chooses to track based on color partially due to the efficiency the HSV model has on the CAMSHIFT algorithm. More advanced techniques including those involving Gabor filtering [11] and filter banks have been studied. These techniques are more complicated than what is used in MTrack and can perform with the absence of color information.

CAMSHIFT (Continuously Adaptive MEANSHIFT) is a variation of the MEANSHIFT algorithm. The MEANSHIFT algorithm is an algorithm, which simply climbs the peak of a probability density function given a specific constraint window. This window is called the *search window*. The MEANSHIFT works well for analysis of static distribution through time. However, it fails when the distributions change.

CAMSHIFT picks up where the MEANSHIFT fails. CAMSHIFT adaptively modifies the search window each of frame until a steady state results and adjusts the window for the next frame. By doing so, CAMSHIFT is able to track continuity of an object through time.

By its nature, CAMSHIFT is well suited for colored object tracking as long as the underlying probability distribution is “good enough.” CAMSHIFT is also not very sensitive to noise or occlusions. Thus, even if an object is occluded for an instant, CAMSHIFT is able to

quickly recover and lock on to the target. The algorithm is fast and runs in $O(\alpha n^2)$ time and its parameters can be changed instantaneously. The algorithms efficiencies and performance have been studied in detail in [13] [29].

Discriminant

The discriminant function's purpose is to generate a unique vector for a given object. A good discriminant function is well behaved in the fact that small variations in the input result in predictable and small variations in the output. Thus, two objects with dissimilar features should have very different or even orthogonal feature vectors.

A classifier is really only as good as the discriminant used to generate its feature vector. This presents a unique problem for gesture recognition: gestures are never "perfect" in that they vary in scale and rotation to the camera. Thus, a discriminant insensitive to these features is desirable [18].

It is possible to construct a seven dimensional feature vector using normalized central moments, which is translation, rotation, and scale invariant [30]. This set of seven numbers with these characteristics are called the Hu invariant moments [17]. These moments are constructed from normalized centralized moments up to the third order [28]. Figure 2 depicts two images along with their seven Hu invariant moments.

The normalized central moments are constructed from the central moments through the following relationships:

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q f(x, y) \tag{2.8}$$

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\lambda}}; \ \lambda = \frac{p+q}{2} + 1 \tag{2.9}$$

$$\phi_1 = \eta_{20} + \eta_{02} \tag{2.10a}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{2.10b}$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{2.10c}$$

$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \tag{2.10d}$$

$$\begin{aligned}\phi_5 = {} & (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] \\ & + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]\end{aligned} \tag{2.10e}$$

$$\phi_6 = (\eta_{20} - \eta_{02})\left[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})\right] \tag{2.10f}$$

$$\begin{aligned}\phi_7 = {} & (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} - \eta_{03})^2\right] \\ & + (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} - \eta_{03})^2\right]\end{aligned} \tag{2.10g}$$

$$feature\_vector = \left[\phi_1\ \phi_2\ \phi_3\ \phi_4\ \phi_5\ \phi_6\ \phi_7\right] \tag{2.11}$$

These moments are not strictly invariant for our discretized environment. This results in slight variations of the feature vector in rotation, scaling, and even translation. However, the variation is so small that does not affect classification greatly. There are ways to help improve this error and one of them relies on the use of Greene's Theorem [31]. Greene's Theorem allows calculation of an integral by boundary information. Similarly, this theorem is used with binary images and contour following algorithms to compute the data necessary for moment analysis

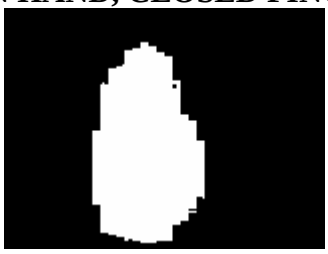
OPEN HAND, OPEN FINGERS	OPEN HAND, CLOSED FINGERS
	
$\phi_1 = 8.1142\text{e-}004$	$\phi_1 = 7.5239\text{e-}004$
$\phi_2 = 6.5596\text{e-}008$	$\phi_2 = 1.5620\text{e-}007$
$\phi_3 = 3.5793\text{e-}011$	$\phi_3 = 2.2224\text{e-}011$
$\phi_4 = 5.9157\text{e-}012$	$\phi_4 = 2.8286\text{e-}012$
$\phi_5 = -4.6247\text{e-}023$	$\phi_5 = 2.2298\text{e-}023$
$\phi_6 = 1.4187\text{e-}015$	$\phi_6 = 1.0771\text{e-}015$
$\phi_7 = 7.2604\text{e-}023$	$\phi_7 = -2.4073\text{e-}024$

Figure 2. Two binary image representations of gestures and their associated Hu invariant moments.

In addition to the information given by the Hu invariant moments, the objects rotation can also be extracted. It turns out the moment calculations can be directly used to calculate the rotation of the object, which is now a binary blob. The equation to calculate rotation is given below.

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2\mu_{11}}{\mu_{20} - \mu_{02}}\right) \tag{2.12}$$

Classifier

The classifier assigns a feature vector to a class. There are many ways to determine which class to assign to a feature vector.

The class of a feature vector can be determined by measuring it against sets of samples of which the class is known. Several metrics can be used for measurement. A common metric used is the Euclidian distance given by:

$$D^2(x_t) = (x_t - x)^2 + (y_t - y)^2 \tag{2.13}$$

where:

x is a classified feature vector.

x_t is an unclassified feature vector at time t.

The Euclidian distance metric does not work well for all data. Most data separates but does so in a way by which the subspace forms in a characteristic manner. This could be a linear or exponential dependence for example. To account for this characteristic variance the Mahalanobis distance metric is often used.

The Mahalanobis distance metric takes into account information about a class' variance in the feature space. Thus, any vector following the trend is measured as being close to the sample space. Vectors orthogonal to the vector space are measured as far from the sample space.

Figure 3. A plot of data points of a known class. The circle and ellipse represent the distance metrics of the Euclidian and Mahalanobis distance respectively. Every point on each of the respective lines is considered the same distance from the centroid marked with an X.

As shown in Figure 3, any point along the lines is the same distance from the X. From the figure, we can see for the Euclidian distance, the red line, feature vectors orthogonal to the "flow" of the known samples could be classified with the depicted class. However, it is easily seen that an orthogonal data point would simply not be a part of the current data trend. It is easy to see how the Mahalanobis distance, the blue line in Figure 3, compensates for the "flow" of data and would continue to properly classify feature vectors following this trend.

The Mahalanobis distance [32] is defined by the following equation:

$$D^2(x_t) = (x_t - m)S^{-1}(x_t - m)' \tag{2.14}$$

where:

x_t is a feature vector of unknown class at time t

m is the mean vector of the sample vectors

S is the covariance matrix of the sample vectors

The values calculated from the sample space can be quickly computed. Thus, when a feature vector of an unknown class is classified, it can become part of the sample space and used to classify future feature vectors. With a large enough number of samples, a defined class subspace and decision surface forms thus improving classification.

High Level Abstraction

This section of the architecture is often not presented independently in many recognition systems. This system is often integrated into the classification scheme and not formally defined. However, such a formality is used here and its use described in detail.

The high-level abstraction function is the last component data visits before going to the action engine. It is one of the last times software can detect and correct any misclassifications. In concern of safety, this section plays a critical role. In the case where a gesture recognition system can control heavy machinery such as a car, safety is of concern. A misclassification could result in injury or death. It is often desired then to place limits on the capabilities of the system or the execution order.

The high-level abstraction engine provides this contextual information to correct for misclassifications. In MTrack, this is implemented using order statistics to help filter out false alarms. The concept of collating feature vectors and classifications independent of each other and providing contextual information to them forms the microstate/macrostate architecture.

Microstates are made up of data independent of other microstates and time. The microstates are then assembled to form a macrostate, which provides a general description of a system. The macrostate engine may look at the order which microstates arrive, the number of

identical macrostates, etc. The important concept is that high-level information is injected into the classification scheme to correct for errors.

Action Engine

The action engine is responsible for providing feedback to the user to reinforce their gestures. This is often implemented directly through the operating system. This provides the most control over the desired actions. This method also provides the least response time without directly interfacing with hardware.

The Win32 API is a framework used to manipulate a users desktop. The API provides many convenient functions desirable for a human computer interface with gesture. Some of these functions are given in Table 1.

FUNCTION	**ACTION**
Move_Event( )	Mouse and wheel movement, mouse button action.
GetForgroundWindow( )	Provides access mechanism to the current foreground window.
GetWindowLong ( )	Provides characteristics of a window such as its display state.
SetForegroundWindow ( )	Bring a window to the foreground.

Table 1. Some common functions from the Win32 API used to create a users desktop experience.

2.3 Requirement Analysis

To this point, the theories and methods used to implement a gesture recognition system have been presented. A set of fundament requirements for a gesture recognition system must now be specified.

Guo, et al. [3] describe goals for an ideal gesture recognition system. MTrack is able to meet these system requirements by the following methods:

1. Track and distinguish hand gestures by skin color alone. Thus, moderate illumination changes do not effect the software.
2. Utilizes optimized algorithms and little memory to process and classify a scene in under 0.067 seconds.
3. A minimum distance classifier that is translation, rotation, and scale invariant provides robust gesture classification.
4. A variation of the CAMSHIFT [13] algorithm is used to reliable track and separate a particular hand in a scene.
5. Gestures utilized provide little fatigue and are fairly intuitive to the user.

2.4 Justification of Selected Methodologies

In this section, the theoretical underpinnings utilized in MTrack are described.

There exists signal filters and/or processor algorithms well suited for removing noise from video. However, these algorithms are often computationally intensive and are often not robust enough to handle dynamic contexts. Morphological operators are great for filtering, but often have the same downfall as frequency based filters such as the Wiener filter; but can be computationally expensive.

Based on the analysis performed, there is no frequency or morphological based filter with real time performance specification that stands out as an optimal filter to use for gesture recognition. Because of this, it was chosen to not implement an initial noise recognition stage as part of MTrack.

MTrack solely relies on the user to properly adjust the camera parameters correctly to achieve the best image. Many different types of imaging sensors are used in digital video cameras. The two main types are CCD and CMOS arrays [29]. The tuning of video equipment is left to the users discretion. It is assumed they know their equipment best. Additionally, a filter can be programmed and inserted into a video stream's filter graph. The filter can be loaded by the operating system at run-time and filter video in real time.

A variation of CAMSHIFT is used for object extraction and tracking. CAMSHIFT is robust and provides a generally good extraction based on color. CAMSHIFT is very fast allowing room for more complex features like noise reduction or feature extractors to be implemented yielding real time performance globally.

CAMSHIFT relies so heavily on color making global illumination and color models important. The use of the HSV colorspace allows an acceptable color subspace to be easily described and implemented. Mainly, simple thresholds on each dimension are all that are needed to realize a great track. Other color models often used with this algorithm are normalized RGB also known as chromacity space [33].

Colorspace is sensitive to the light used to illuminate a scene. CAMSHIFT does not take into account relative color differences to form an absolute color scale. MTrack performs no color constancy computations.

The discriminant function is often the most complex part of the recognition system. Hu invariant moment [24] [17] analysis was chosen to assemble feature vectors. Hu moments are rotational, scale, and translation invariant. The output of the object extractor, a variation of CAMSHIFT, is a binary 'blob' and moment analysis works best for such types of input. In fact, results computed during the CAMSHIFT algorithm are used to determine Hu moments.

Not all the Hu moment generated are necessary in classifications. Often, the lower order moments alone server as a sufficient classifier. This is true for this system. The lower order moments proved to be insensitive to noise [34] [35] and robust enough to classify well. The higher order moments were found to be sensitive to noise and detrimental to classification.

After creating a series of test images, which were outputs of the CAMSHIFT algorithm, a proper classifier was selected. It was noticed that all the gesture classes had similar characteristics where by each feature space spread elliptically over the subspace. Few outliers resulted from the discriminant analysis. Adjacent classes, which may cause confusion, were orthogonal to each other.

The data often fit an elliptical manifold. Therefore, it was decided to use the Mahalanobis distance as the classifying means for an unknown vector. This allowed outliers following in the trend of a class to be correctly classified. The use of simple norms, such as the Euclidian norm, yields unsatisfactory results. A shortcoming of the Mahalanobis distance as a metric is it must be trained.

In order to handle some of the outliers that resulted, high-level abstractions were analyzed. The abstraction used slightly resembles models used in statistical physics, but has little relation. The model is called the microstate/macrostate model. Essentially, each frame is assigned a microstate, which is derived from itself and only itself. Then, these microstates are fed into a macrostate engine where future classification is conducted.

The macrostate is another opportunity to help filter out false classifications. It uses contextual information to do so, mainly gesture continuity. The macrostate engine uses order statistics to correct these falsities.

Finally, the Win32 API was used to execute the classification results. The API gives a very close handle on the desktop environment and executes in a timely manner. The framework is portable across Windows systems (ME, 98, 2000, XP) such that MTrack can run on any of these operating systems and a user can manipulate the environment uniformly.

3.0 IMPLEMENTATION

The overall architecture of MTrack is now presented. The system consists of a six-stage process. It starts by taking a video stream input and finishes by transforming it into desktop actions as dictated by the user. Each stage is presented in detail.

3.1 System Architecture

The system architecture is composed of a six-stage process (Figure 4).

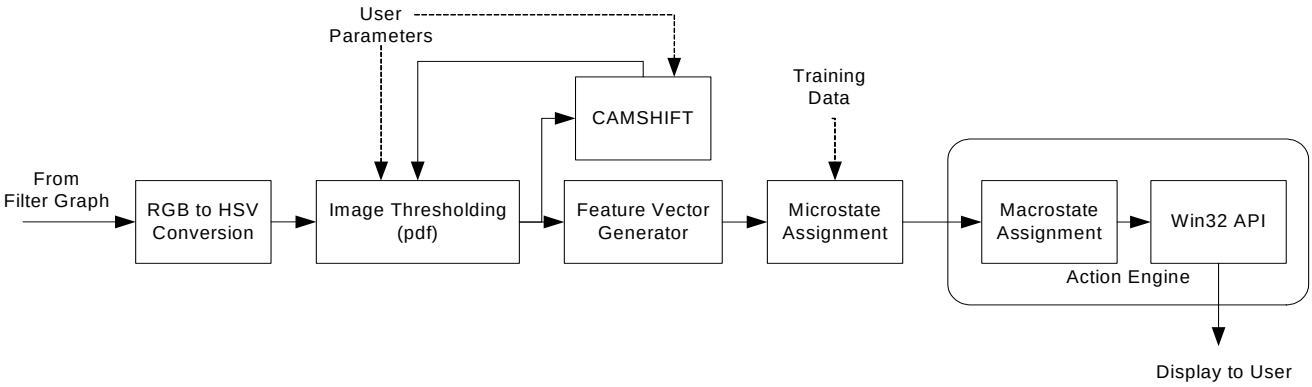


Figure 4. A diagram of MTrack's six-stage architecture.

The process begins by constructing a filter graph acquiring a video stream using DirectX. The filter graph used is depicted in Figure 5.

Figure 5. The filter graph of MTrack. The Color Space Converter filter is not the RGB to HSV converter.

The video is broken down into frames. Each frame is passed into the gesture recognition system (GRS) to be interpreted.

Stage 1 – Colorspace Conversion

Figure 6 depicts the RBG to HSV conversion function. Each frame is stored in a word aligned structure contain the red, green, and blue color components. The frame is processed pixel-by-pixel converting each RGB frame to HSV colorspace. The image is stored in the same memory space as the RGB image maintaining the word alignment for processing speed.

Each pixel of RGB is one byte-worth of color data. Thus, each color, red, green, and blue, has a range of [0, 255]. Since the same memory space containing the RGB image is used to store the HSV image, the HSV values also have the same domain. This may seem strange because the hue component naturally has a domain of [0, 360] degrees, a cycle around the colorwheel. This value is normalized to a range of [0, 255]. This results in a slight loss of color granularity, but none significantly affecting the recognition process.

Figure 6. RGB to HSV Conversion.

Stage 2 – Image Thresholding (pdf construction)

Thresholding is performed on each HSV frame (Figure 7). Using the user-adjustable parameters of hue, saturation, and value bounds, pixels are converted to two colors, black or white. If a pixel is deemed to be part of the desired object, it is converted to white, otherwise, it is converted to black.

Thresholding forms a binary image *blob*. This blob is viewed as a probability density function by the CAMSHIFT algorithm. Typically, CAMSHIFT assigns probabilities based on a

look up table generated from a histogram. In this case, all pixels are of equal probability. This shifts the focus to not only obtaining a track, but to creating a silhouette of the object. This shape is used in the next stage where a discriminant function will operate upon it.

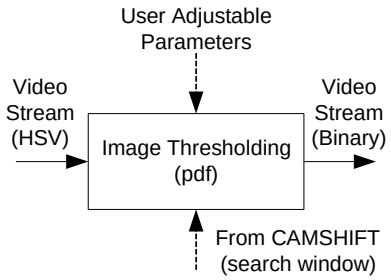


Figure 7. Threshold function.

Stage 3 – CAMSHIFT and Feature Vector Generation

During this stage, the output of the thresholding function is simultaneously used (not in parallel) to create a feature vector and track the object. As it turns out, some of the calculations used in CAMSHIFT can be used to create the feature vector. Thus, CAMSHIFT runs first in the sequence.

CAMSHIFT works by performing the MEANSHIFT among a search window while continually modifying this window. This search window is local to the object. Thus, it prevents distracters from being tracked. The MEANSHIFT algorithm repeatedly runs updating the centroid each time. The algorithm halts when the centroid has reached a steady state defined by the user-adjustable tracker parameters. CAMSHIFT then resizes the search window by a user-adjustable amount, which becomes the initial search window at the start of the next frame.

The feature vector is generated after CAMSHIFT completes. Lower order moments are used to compute the centroid during each iteration of CAMSHIFT. During these iterations, this information is also used to compute higher order moments.

The feature vector also contains rotational information about the object. This information is also derived from the computations from CAMSHIFT. An object's rotation is computed after the Hu invariant moments are computed. The Hu invariant moments along with the object's rotation are stored in a feature vector, which in this case is called a microstate.

Stage 4 - Microstate Assignment

The microstate is created from the Hu invariant moments and the object's rotation. It is also at this time the minimum distance classifier assignes the feature vector to a preliminary class. All of this information is stored in a structure composing the microstate.

Stages 5 and 6 - Action Engine

The action engine is composed to two parts: macrostate assignment and execution. The macrostate determines the actions to be executed by the software. Execution of the decision follows using the Win32 API.

The macrostate and microstate are similar as they both describe an action of the user. The macrostate differs by containing high level, contextual, information about not just one frame, but several. This information aids in correcting any misclassifications should they arise.

The action engine has a three-frame microstate buffer. It contains the microstates for the last three frames of video in order of arrival. The macrostate is computed by examination of the buffer and nonlinear filtering.

The macrostate is computed by examination of the buffer. If all three microstates states in the buffer are identical, the microstate's gesture classification becomes the macrostate. This is denoted through order statistics as:

$$X_{(1)} = X_{(3)} \tag{3.1}$$

where:

$X_{(1)}$ denotes the first order statistic of X.

$X_{(3)}$ denotes the thrid order statistic of X.

The macrostate does not change if the contents of the buffer are not identical. The effectiveness of this filter does generally rely on the effectiveness of classification at microstate assignment. However, upon inspection of the microstate stream, this filter aids in stability by removing outliers in a local window of the stream. As the gesture changes, what appears to be an outlier, in one scenario, begins to dominate the buffer and thus the macrostate changes.

Potentially, the microstate stream could be thoroughly analyzed and a more exact filter be constructed for macrostate determination. Probabilities of filter effectiveness can be constructed [27]. For example, consider a noisy microstate stream x_i given by

$$x_i = c + z_i \tag{3.2}$$

where:

c is constant representing the ideal gesture in some interval in time.

z_i is uniformly distributed noise over [0, 1].

The noise cumulative density function (cdf) is given by

$$F(x) = \begin{cases} 0, & x \le 0 \\ x, & 0 \le x \le 1 \\ 1, & 1 < x \end{cases} \tag{3.3}$$

Using a three-point window, a 3rd order-statistic filter is applied to x_i. This filter is described as

$$y_i = X_{(3)} \tag{3.4}$$

where the window of points used at index is *i* is

$$\{x_{i-1}, x_i, x_{i+1}\} \tag{3.5}$$

The cdf of $F_3(x)$ can be computed from the general equation $F_r(x)$

$$F_r(x) = \sum_{i=r}^{n} \binom{n}{i} F^i(x)[1 - F(x)]^{n-i} \tag{3.6}$$

where:

n is the size of the filter's window.

$F(x)$ is the noise cdf.

From these equations, it also possible to calculate the minimum and maximum values of the filter output. This information could be used to restrict specific gestures from executing based on context.

As the macrostate is computed, the Win32 API is called to execute the decided actions. It is at this time, rotation information is used to manipulate the middle-mouse-button scroll. The action, along with the scrolling information, is sent back to the GUI where it is displayed to the user.

3.2 Implementation Details

The algorithms used in MTrack were first designed using MATLAB. Upon verification of the algorithms, they were then ported to Object Pascal. MATLAB provided an effective means by which to quickly design and test possible algorithms to be used by MTrack.

MTrack was developed on a standard desktop machine. The machine is a 600Mhz Pentium processor with 512 megabytes of RAM. The development environment, Borland Delphi 7.0, operated in Windows 2000 environment. The main camera used for testing was a

Logitech Quickcam Pro USB utilizing software version 5.4.1. A DirectX wrapper, named DSPack, designed for use in Delphi was used. Its version is 2.31 with a patch provided by: Paul Glagla[2].

The sample space to calculate the Mahalanobis distance was defined by recording a video of the known gesture for five seconds. During this time, the gesture was moved towards the camera, away form the camera, and then rotated ninety degrees in each direction. Twenty-five frames were selected from each video to be used as ground truth. Figure 8 depicts the feature vectors created for ground truth.

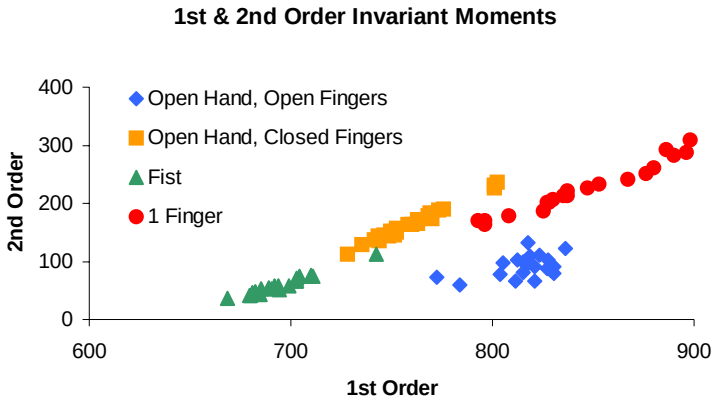


Figure 8. 1st and 2nd order moment plots of the four fundamental gestures.

Source code for algorithms is provided in Appendix B.

2

http://www.progdigy.com/boards/index.php?act=ST&f=1&t=1340&hl=christmas+gift&s=5c6d37d66d2ed18f1d54ffda5b2f9f09

4.0 ANALYSIS

This section provides analysis of the usability of MTrack as human-computer interface software. It describes the capabilities of the system, provides screenshots of sample execution, and provides directions for future improvements.

4.1 System Capabilities

Video

MTrack supports any camera interface that can be recognized and processed by a filter graph through Microsoft's DirectX framework. It includes easy selection of video source if many cameras are connected to a system. It also provides support for loading video files of the correct size for repeatable analysis.

MTrack features two video displays, one is the raw stream from the camera, the other is the rendered stream upon which recognition is conducted. By right clicking on the rendered video stream window, one can take a screenshot of the current frame and save it to bitmap, switch between different views including RGB, hue, and the binary image, and turn the on screen display on or off. Figure 9 depicts the four fundamental hand gestures and their associated binary image.

Hand Extraction and Tracking

MTrack supports numerous tracking features that can be tuned for better recognition and tracking performance. This includes hue thresholds, saturation thresholds, value thresholds, height to width ratio of hand, mean threshold, window expansion size for the CAMSHIFT

algorithms and the area the window must be greater than before the system declares that the track was lost. Optionally, theses parameters can be stored and recalled in a *.mtk* file which uses XML format to store these parameters for different environment profiles.

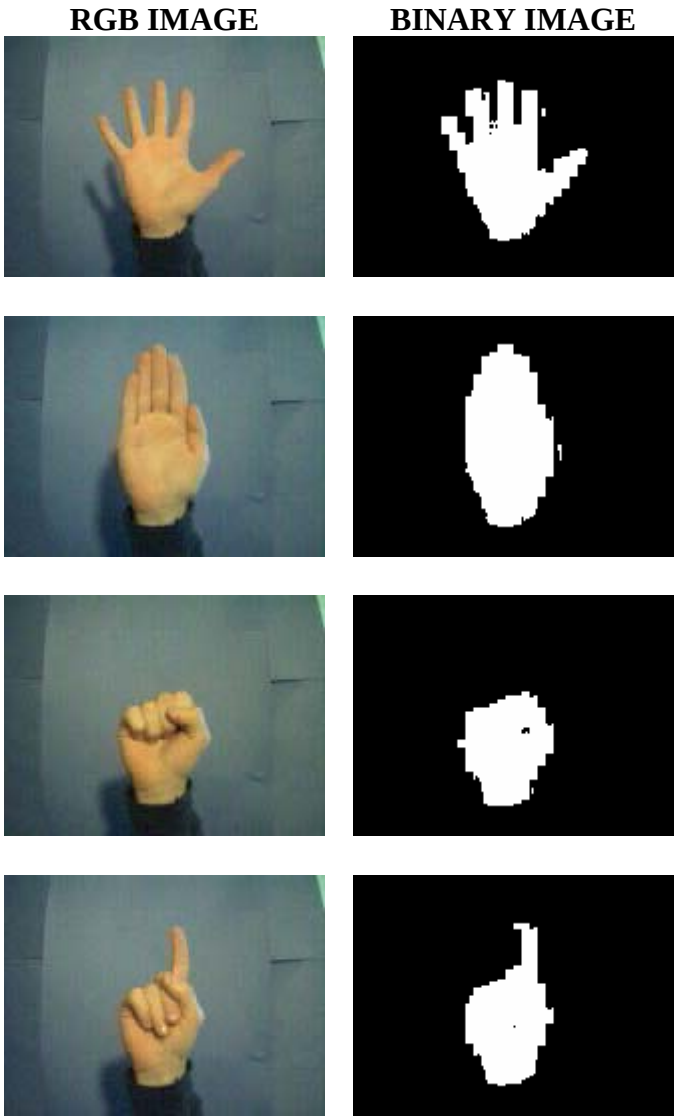


Figure 9. The four fundamental gestures and their associated back projections. These binary images are interpreted as pdf's by CAMSHIFT and are used to generate feature vectors.

Translation can be turned on or off. When translation is turned on, a large stop button is displayed for easy system shutoff (Figure 10). MTrack always remains the top window in the Z-

order in Window thus making it always visible to the user. It has a large display showing the current mouse action and the display turns read when the track goes out of the video window to denote that the track may no be correct as the hand is not fully in the scene.

Table 2 depicts the gestures and translations supported by MTrack. These set of

GESTURE	ACTION
Open hand, spread fingers, little rotation	Mouse up
Open hand, closed fingers, little rotation	Mouse down
Index finger extended	ALT-TAB, more specifically, iterating through the z-order.
Fist	Minimize current window
Open hand, spread fingers, rotation <-25, >25 degrees	Mouse up, middle mouse button scroll down.
Open hand, closed fingers, rotation <-25, >25 degrees	Mouse down, middle mouse button scroll down.
Index finger extended	ALT-TAB with middle button scroll
Index finger extended	ALT-TAB with middle button scroll
Hand movement	Direct translation to mouse movement

Table 2. The complete set of gestures and their associated actions as supported by MTrack.

fundamental gestures can be used to perform other computer tasks such as:

- Double clicking
- Click and drag
- Selection

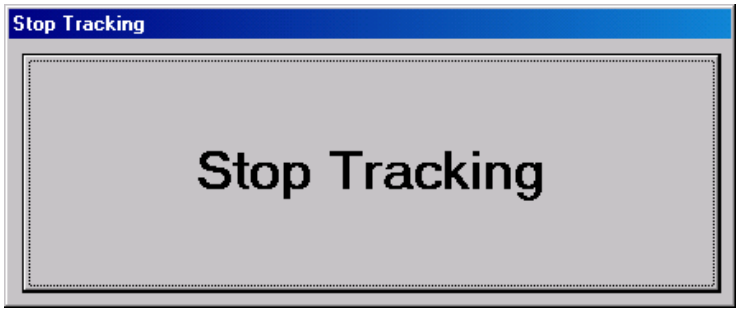


Figure 10. The Stop button. This button allows the user to easily turn off the translator.

MTrack does not currently support features such as resolution change, multi-hand tracks, and a training program. However, the software was created in such a way that these features can be added with ease.

4.2 Sample Execution

Figure 11 below depicts the six prominent gestures and their translations. The red box in the figure is the search window determined by CAMSHIFT. The lower left-hand corner denotes the frames per second of the video stream. The text at the top center of the image denotes the action interpreted by the system. The underscore and caret denote scroll down and scroll up, respectively.

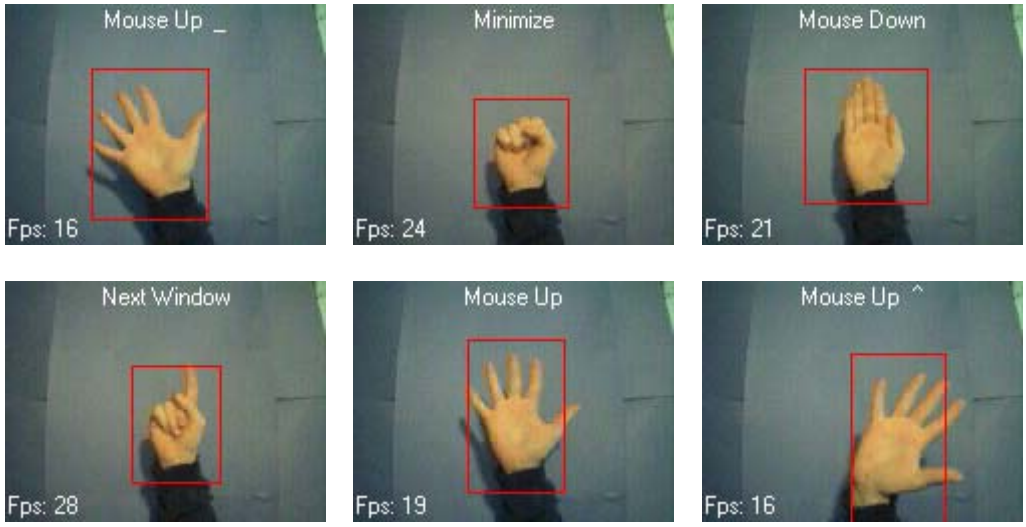


Figure 11. Six gestures captures from the Rendered Video Display. The red box denotes the tracked object. The top, centered text denotes the action. Fps is an abbreviation for frames per second.

Each frame of video is decomposed from RGB to HSV colorspace and then a binary image is created (Figure 12). This binary is created by thresholding using the user adjustable parameters. This forms a binary 'blob' or pdf as it is viewed by CAMSHIFT.

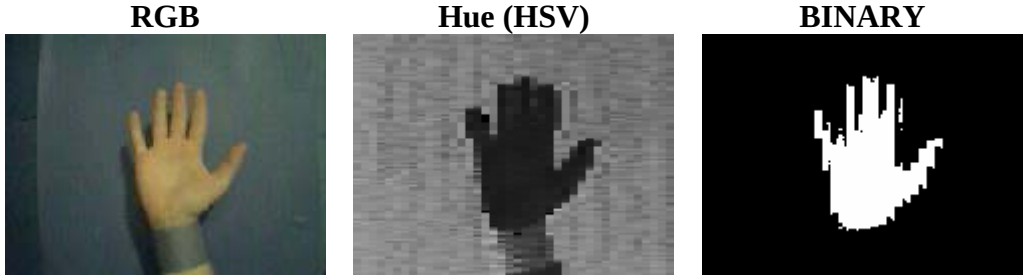


Figure 12. A gesture shown as it is decomposed into a binary image.

MTrack does not always classify correctly. Usually, this is due to the hand being incorrectly extracted from the scene (Figure 13). Because the system relies on color, if the user does not wear a wrist band or a long sleeved shift, the arm may be extracted also, thus, yielding an incorrect extraction. Algorithms have been developed to correct such incorrect extractions.

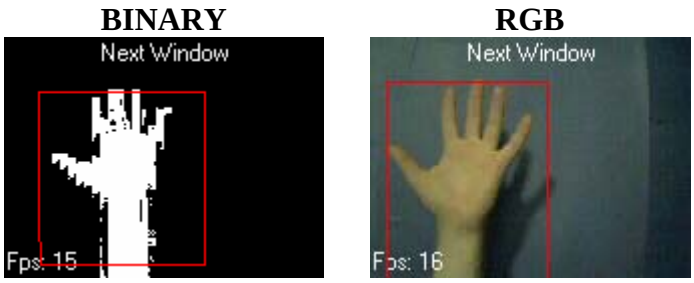


Figure 13. False positive due to the arm being included during thresholding.

MTrack is able to recognize gestures despite translation, rotation, or scale invariance. This is due to the properties of the discriminant, the Hue invariant moments. A sample of this property in use is depicted in Figure 14.

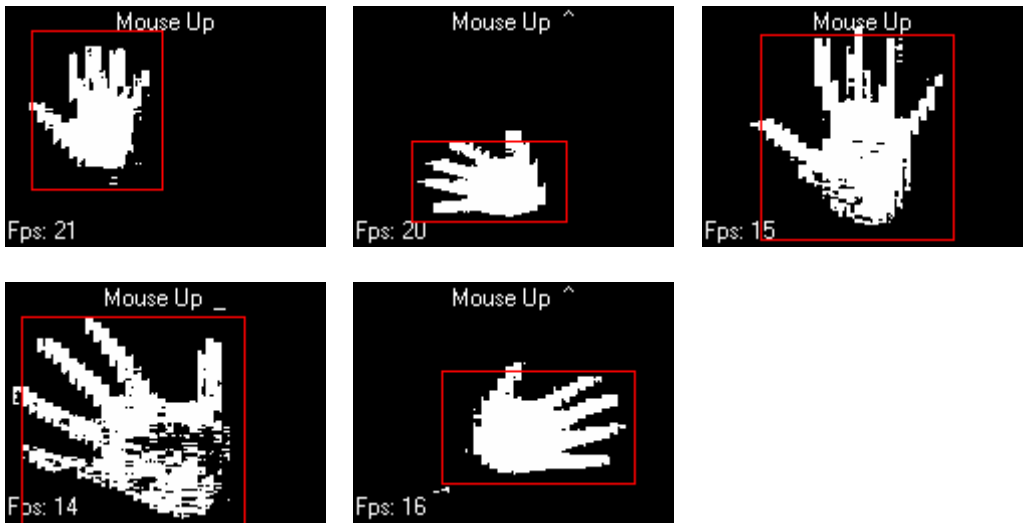


Figure 14. Five images of the same gesture displaying scale, rotation, and translation invariance. Despite the noise in some of the images, the Hu invariant moments are focused on shape. The 1st and 2nd order moments not very sensitive to noise.

4.3 Future Improvements and Direction

Much can be done to improve the MTrack software. Limits on the amount of development time restricted the initial release's supported features. The development of features and improvement upon those that already exist could easy make for another thesis. I will outline some of the improvements of interest to myself if given more time for the project.

MTrack is multithreaded and currently has in place a generic system for hand tracking. The hand tracker class used in MTrack is inherited from a generic tracker class. Thus, a programmer can inherit from this class and a user can seamlessly use the new tracker without modifying any code.

More than one instance of the tracker class can be instantiated at one time. Thus, two hands can be simultaneously tracked. More so, the Window message passing system can be utilized to exchange messages between the two classes in an effort to coordinate their movements. Such a procedure would allow for more complex actions by the user.

Another such improvement to MTrack concerns the classifier. A simple application or dialog can be constructed which asks the user to perform a different gesture in an effort to train the classifier. Once the training has finished, the results could be stored in an XML file and recalled then at any time. This allows different users to easily use MTrack without suffering classification problems.

The classifier can also be improved by retraining itself in real time. When the macrostate engine finds a misclassified frame, it can add the frame's feature vector to the correct class and then retrain the space. This will help tune the classifier quickly. Again, the classifier's parameters can also be saved to disk to be loaded again when needed.

Morphological filtering can be performed on the threasholded video stream. This would help remove artifacts resulting from camera noise or a poorly tuned camera. This also helps to give the binary hand blobs a more natural look better resembling the hand. This then provides then better input to the classifier for improved classification.

Thresholding can be improved by utilizing the concept of color constancy. Currently, thresholding is done on relative color as changes in light can slightly affect the performance of this function. Color constancy would be an effort to improve this and create a more robust thresholding function.

Currently, a non-flesh colored wristband or long sleeve shirt must be worn to prevent the arm from interference with the hand thresholding. A way to remove this has been studied [3] and could be implemented in MTrack (Figure 15). This would eliminate the use for colored gloves, wrist bands, and other unnatural devices.

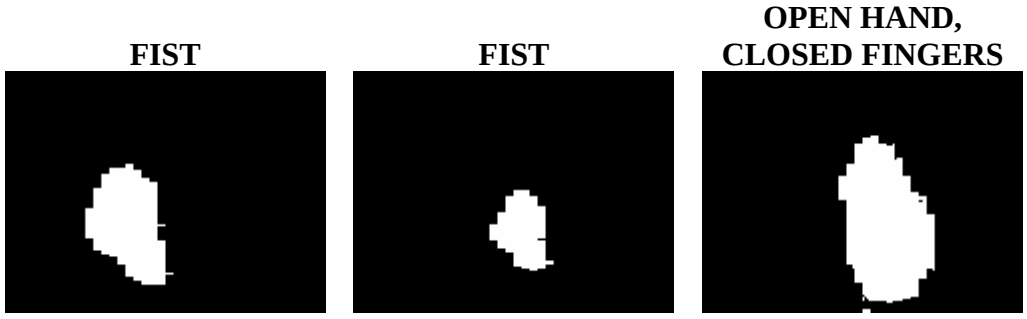


Figure 15. Two images of a fist but one has parts of the wrist in it. This makes the fists appear similar to the open hand, closed fingers gesture.

More gestures could be supported by MTrack. The ability to detect numbers or hand curves could be implemented. Instead of performing analysis on the whole hand, morphological skeletonization could be performed and then the resulting data put into graph form. This graph could better describe hand gestures as it could closely resemble the mechanical framework contained in the hand to form the gesture.

5.0 CONCLUSION

MTrack is a gesture recognition software developed to recognize simple gestures and perform a task from them in real-time. The system is designed to run in Windows on a mid level desktop platform. The system performs in real time using low-cost commercial off the shelf (COTS) camera equipment.

MTrack is designed to recognize four fundamental gestures and variants to execute a total of nine actions. The system responds the same to gesture rotation, scale, and translation. This provides a natural interface for easy use.

MTrack uses a variant of the CAMSHIFT algorithm for object extraction and tracking. Using calculations from CAMSHIFT, the system is able to discriminate each frame through shape analysis via Hu invariant moments. Each frame is then classified against a small sample set of known gestures through a minimum distance classifier. Each frame is then passed into a high level filtering framework attempting to correct misclassifications. The framework uses a notion of microstates and macrostates to utilize contextual information for filtering. The final frame classification is an action executed via the Win32 API.

MTrack continually provides feedback to the user. Gestures can be interpreted, but not executed, by the computer at a user's wish. The system also provides video displaying a track of the hand, the current gesture, and system performance.

Despite MTrack's successful application, room for improvement exists. Many methodologies exist for color tracking, discriminating, classifying, and execution. These topics are often studied independently due to their complexity. The software's generic framework provides a plugable interface by with to experiment with these different methodologies.

ACKNOWLEDGMENTS

Thank you to Dr. Richard Tutwiler for guiding me along this project by providing encouragement and insight. His help has been selfless, motivating, and inspiring. Thank you to Dr. Gary Weisel whom has always believed in my intelligence and hard working ability. Thank you to my family; you have supported everything I have ever done and I am eternally thankful for it. And finally, thank you Dear Old State.

REFERENCES

[1] Freeman, W., and M. Roth, "Orientation histogram for hand gesture recognition," In Int'l Workshop on Automatic Face and Gesture Recognition, 1995.

[2] Ren H., and Xu G, "Human action recognition in smart classroom," Proceedings of The Fifth IEEE International Conference on Automatic Face and Gesture Recognition. Washington, DC, May 20-21, 2002, pg. 4.

[3] Guo D., Yan Y H. and Xie M., "Vision-based hand gesture recognition for human-vehicle interaction," Fifth International Conference on Control, Automation, Robotics and Vision, Singapore, Vol.1, Dec 8-11, 1998, pp. 151-155.

[4] Francisco J. Perales López, Ramon Mas, M. Mascaro, P. Palmer, A. Igelmo, A. Ramírez. "A colour tracking procedure for low-cost face desktop applications." First Iberian Conference, IbPRIA, Puerto de Andratx, Mallorca, Spain, June 2003.

[5] Sharma R, "Research on vision-based gesture analysis and multimodal interfaces," [Online Document], [citied 2004 June 16], Available HTTP: http://www.cse.psu.edu/~rsharma/imap1.html

[6] Starner T., Weaver J., and Pentland A. "Realtime American Sign Language recognition using desk and wearable computer-based video," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 20, No. 12, Dec 1998. pp. 1371-1375.

[7] Cantzler H., and Hoile C., "A novel form of a pointing device," Proc Vision, Video and Graphics (VVG), Bath, UK, July 2003, pp 57-62.

[8] Garcia E., and Morentin L., "A EyeToy/WebCam Mouse in Visual Basic," [Online document], [cited 2004 June 18], Available HTTP: http://dftuz.unizar.es/~rivero/alumnos/vmouse.html

[9] Pylkkö H., "Real-time color-based visual tracking," Master's Thesis, Findland: Department of Electrical Engineering, University of Oulu, 2000.

[10] Wysoski S., Lamar M., Kuroyanagi S., Iwata A., "A rotation invariant approach on static-gesture recognition using boundary histograms and neural networks," ICONIP, Singapore, Nov 2002, pg. 2137.

[11] Triesch J., and von der Malsburg C., "Robust classification of hand postures against complex backgrounds," Proc. Second International Conference on Automatic Face and Gesture Recognition, Killington, VT, Oct. 1996.

[12] Tanguay Jr. D. O., "Hidden markov models for gesture recognition," Masters Thesis, Massachusetts Institute of Technology, Cambridge, MA, June 1993.

[13] Bradski, G. R., "Computer vision face tracking for use in a perceptual user interface," Intel Technology Journal, No. Q2, 1998.

[14] Gourvest H., "DSPack," [Online Document], [citied 2004 June 17], available HTTP: http://www.progdigy.com/dspack/index.html

[15] Bhatt A., Muench R., Fursov V., et al, "JEDI Math," [Online Document], [cited 2004 June 17], available HTTP: http://jedimath.sourceforge.net/

[16] The Math Works, Inc., 24 Prime Park Way, Natick, MA 01760-1500.

[17] Hu M. K., "Visual pattern Recognition by moment invariants," IEEE Transactions on Information Theory, vol. IT-8, 1962, pp. 179-187.

[18] Kwatra V., "Optical character recognition," [Online Document], [cited 2004 February 24], available HTTP: http://www.cc.gatech.edu/~kwatra/computer_vision/ocr/OCR.html

[19] Flusser J., and Suk T., "Affine moment invariants: A new tool for character recognition," Pattern Recognition Letters, vol. 15, Apr. 1994, pp 433-436.

[20] Alt F.L. "Digital pattern recognition by moments," Journal of the Association for Computing Machinery, vol. 9, issue 2, Apr. 1962, pp 240-258.

[21] Romantan M., Vigouroux B., Orza B., Vlaicu A., "Image indexing using the general theory of moments," 3rd COST #276 Workshop, Budapest, Hungary, Oct. 2002.

[22] Zhu Y., DeSilva L. C., Ko C. C., "Using moment invariants and HMM in facial expression recognition," University of Singapore, 4th IEEE Southwest Symposium on Image Analysis and Interpretation, Apr. 02-04, 2000, Autsin, TX.

[23] Elgammal A., Shet V., Yacoob Y., Davis L. S., "Gesture recognition using a probabilistic framework for pose matching," The Seventh International Conference on Control, Automation, Robotics and Vision, ICARCV 2002, Singapore in December 2-5, 2002.

[24] Rosales R., "Recognition of human action using moment based features," Technical Report BU 98-020, Boston University Computer Science, Boston, MA 02215, Nov. 1998.

[25] Gouaillier V., and Gagnon L., "Ship silhouette recognition using principal components analysis," Applications of Digital Image Processing XX, SPIE Proceedings, vol. 3164, 1997, pp. 59-69.

[26] Yang, J. Xu, Y., "Hidden markov model for gesture recognition," The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, May 1994.

[27] Pitas I., Venetsanopoulos A. N., Nonlinear Digital Filters: Principles and Applications. Kluwer, 1990.

[28] Gonzalex, R., Woods R. E., Digital Image Processing Second Edition, Prentice Hall, 2002.

[29] M. Boyle, "The effects of capture conditions on the CAMSHIFT face tracker," University of Calgary, 2004.

[30] Flusser J., "On the independence of rotation moment invariants", Pattern Recognition, vol. 33, 2000, pp. 1405–1410.

[31] Yang L., and Albregtsen F., "Fast and exact computation of moments using discrete Green's theorem", Proc. NOBIM Conf., pp. 82-90, 1994.

[32] P. Hong, M. Turk, and T.S. Huang, "Gesture modeling and recognition using finite state machines," Proc. of the Fourth IEEE International Conference and Gesture Recognition, March 2000, Grenoble, France.

[33] Caetano T.S., Barone D.A.C., "A probabilistic model for the human skin color", IEEE International Conference on Image Analysis and Processing, Palermo, Italy, 2001, pp. 279–283.

[34] Belkasim, S. O., M. Shridhar, and M. Ahmadi, "Pattern recognition with moment Invariants: A comparative study and new results," Pattern Recognition, vol. 24, num. 12, 1991, pp. 1117-1138.

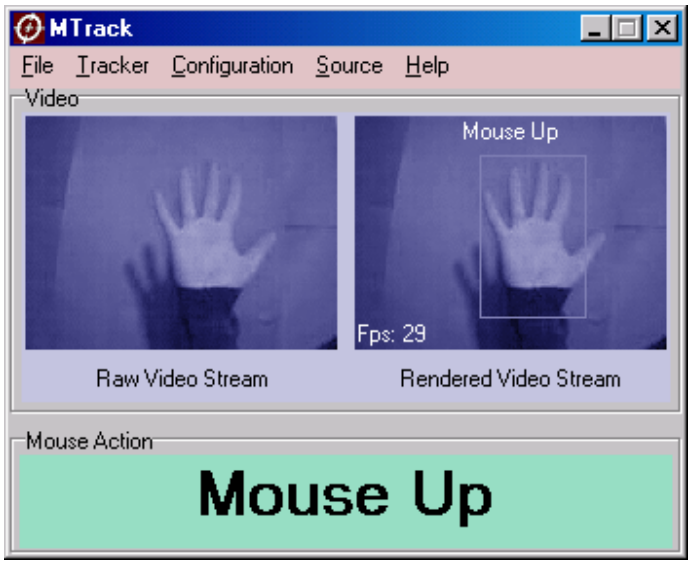
[35] Nikolova M. and A. Hero, "Noisy word recognition using denoising and moment matrix discriminants," IEEE Workshop on Higher Order Statistics, June 1999.

APPENDIX A: SOFTWARE MANUAL

This section describes how to use the MTrack software.

A.1 Software Components

MTrack is composed of three sections: menu (red area), video (blue area) and mouse action (red area).



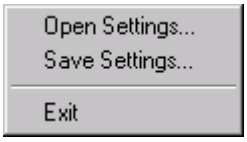
The menu provides an interface to the features of MTrack. The Video frame provides output of two video streams, the Raw Video Stream coming from the camera and the Rendered Video Stream which contains tracking and gesture information. The Rendered Video Stream window is also equipped with a popup menu, which allows for some additional features for analysis. The Mouse Action frame has the current mouse action as interpreted by the system.

A.2 Menu Options

This section contains the majority of features offered by MTrack. This section is organized by function starting with the File menu.

With the DirectX architecture, video is streamed from a source to an application by means of a filter graph. This graph represents the filtering performed to convert a video stream from an arbitrary format to something usable by any application. A filter graph consists of a series of nodes forming a directed unclyclic graph. Each node has inputs and outputs and performs some function.

The File Menu



The File menu contains three options:

1. Open Settings… – Opens an mtk file which contains parameters for the tracker as shown in the Tracker Settings window. This window is available in the Configuration menu. Mtk files are stored in simple XML format.
2. Save Settings… - Used to save tracker settings to an mtk file.
3. Exit – Stop the current filter graph, deallocates program memory, and closes the software.

The Tracker Menu

This menu is used to enable translation of the interpreted hand gestures to mouse actions. The Tracker menu has the following options:

1. Start Tracking – When a video stream is active, this performs the actions denoted in the Mouse Actions frame.
2. Stop Tracker – Discontinues translating the actions denoted in the Mouse Actions frame.

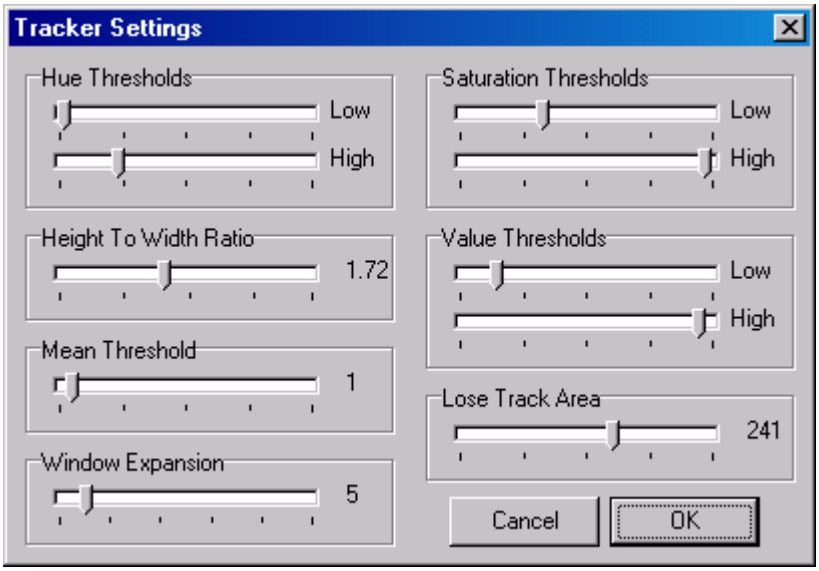
When translation is started, a large stop button is provided to the user to easily stop translation.

The Configuration Menu

The Configuration menu contains one option:

1. Adjust Tracker Parameters… - Launches the Tracker Settings window, which allows real time adjustment of tracker parameters.

The Tracker Settings contains performs the following functions:



1. Hue Thresholds – Any pixel in an image with a hue value above the low threshold and below the high threshold becomes part of the back projection image. All other values are rejected. The system has no support for circular ranges such as those less than 10 and greater than 250 which is supported by a colorwheel.
2. Height To Width Ratio - Governs the height to width ratio of the search window. This allows for different shaped hands to be accurately tracked.

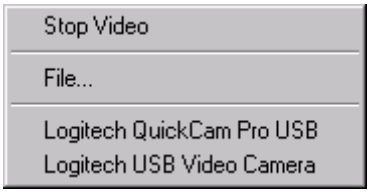
3. Mean Threshold – Denotes the maximum centroid movement in either the X or Y direction between MEANSHIFTS during CAMSHIFT. A higher value usually corresponds to less iterations, faster processing, but worse tracking.
4. Window Expansion – Denotes how many pixels in every direction (-X, X, -Y, Y) the search window is to be expanded during the CAMSHIFT algorithm. A high number helps to track a fast object but is more susceptible to locking on to distracters in a scene.
5. Saturation Thresholds – Used in conjunction with Hue Threshold in construction of a back project. Any pixel below the low threshold and above the high threshold is rejected during thresholding. This is used to filter out colorless objects from a scene.
6. Value Thresholds – Same as the Saturation Threshold by concerning the value component of the HSV colorspace. Saturation and value constraints help to remove dark and bright areas of a scene which often contain incorrect hue values which distract the tracker. Much noise can be accurately removed with adjustment of these thresholds.
7. Lose Track Area – If the area of the search window becomes less than this threshold, the tracker assumes the target is occluded or no longer in the scene.

The Cancel button restores the Tracker Settings to their original values. The OK button saves the current tracker settings.

At any time when these parameters are changed, the tracker immediately implements their modifications.

The tracker parameters are also stored in the Window registry upon exit and are reloaded when the software is started. This is provided for convenience.

The Source Menu

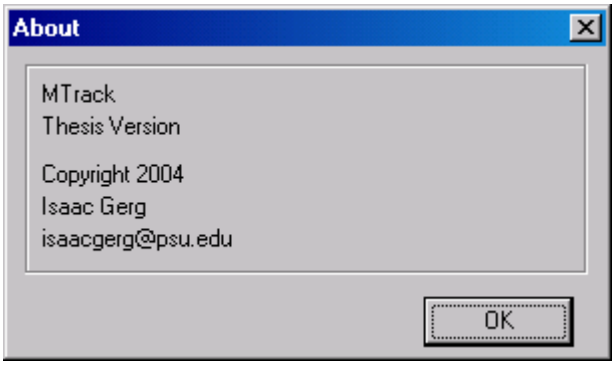


The Source menu contains three sections:

1. Stop Video –Stops the processing of the video stream.
2. File… - Allows selection of a video file and processes the video stream. The video must be of size 160x120 pixels or the stream will not render correctly.
3. Upon program startup, MTrack enumerates all camera devices and displays a list of them in this section. Select a camera from this list to begin processing its stream. The camera must support RGB 24-bit native output.

The Help Menu

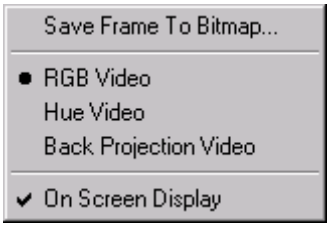
The Help menu contains one option which is to display the About dialog.



This dialog is a way to verify the version of MTrack you are using.

The Rendered Video Stream Menu

This menu is displayed by right clicking on the Rendered Video Stream window. The menu has threes sections. The first section allows a user to save the current video frame to a bitmap file. The option is designed to allow successive frame captures and automatically saves the frames in bitmap format with numerical naming. The second section allows the user to view the video in different colorspaces, RGB, Hue from HSV, and the binary image generated from thresholding. Finally, the last section provides and option to toggle the on screen display.



The Mouse Action Frame

The mouse action display will turn red in the instant the window falls outside of the scene. This is to denote that the track may not be accurate as information may be missing.

APPENDIX B: ALGORITHMS USED IN IMPLEMENTATION

During action execution, the mouse pointer is moved to follow the hand. The mean centroid of the microstate buffer saved is also part of the macrostate. This information is then translated to screen coordinates and then executed using

```
mouse_event
(
      MOUSEEVENTF_ABSOLUTE || MOUSEEVENTF_MOVE,
      Centroid.Y,
      Centroid.X,
      0,
      0
)
```

The middle-mouse-button is scrolled using

```
mouse_event
(
      MOUSEEVENTF_WHEEL,
      0,
      0,
      WHEEL_DELTA,
      0
)
```

The left middle mouse button is manipulated using

```
mouse_event
(
      MOUSEEVENTF_LEFTDOWN or MOUSEEVENTF_LEFTDOWN,
      Centroid.Y,
      Centroid.X,
      0,
      0
)
```

Sending a message to the foreground window minimizes it. Note that Windows provides two ways to communicate a message through a window's queue: `SendMessage` and `PostMessage`. `SendMessage` returns only after the message sent has been executed. This is desired for MTrack as it ensures the video processing does not proceed until the action is performed. This is opposed to `PostMessage` which returns immediately leaving the target window to execute the message upon convenience.

Windows maintains an imaginary plan extending outward from the users screen. This plane, called the *Z-plane*, contains information about the order of windows on the screen. This order is known as the *Z-order*. The Z-order is not simply the list of applications running when a user presses and holds ALT & TAB. A full discussion of the Z-order is beyond the scope of this paper. However, a generic algorithm is presented for implementation.

Windows are cycled through the Z-order using the code below.

```
LONG lStyle;
LONG lCaptureWindowStyle;
HWND hCaptureWindow = GetWindow(GetActiveWindow(), GW_HWNDFIRST);
HWND hForegroundWindow = GetForegroundWindow();

while (hForegroundWindow <> 0)
{
      lStyle = GetWindowLong(hForegroundWindow, GWL_STYLE);
      if (
            (lStyle and WS_VISIBLE) == WS_VISIBLE)
            &&
            ((lStyle and WS_POPUP) <> WS_POPUP)
         )
      {
            hCaptureWindow = hForegroundWindow;
            lCaptureWindowStyle
            =
            GetWindowLong
            (
                  hCaptureWindow,
                  GWL_STYLE
            );
      }
      hForegroundWindow = GetNextWindow(hForegroundWindow, GW_HWNDNEXT);
}
SetForegroundWindow(hCaptureWindow);
```

RBG to HSV Colorspace Conversion

For each RGB pixel of an image, perform the following calculations [28]:

1. $\theta = \cos^{-1}\left\{ \dfrac{\frac{1}{2}\left[(R-G)+(R-B)\right]}{\left[(R-G)^2+(R-B)(G-B)\right]^{1/2}} \right\}$

2. $H = \begin{cases} \theta & \text{if } B \le G \\ 360 - \theta & \text{if } B > G \end{cases}$

3. $H = round\left[255\left(\frac{H}{360}\right)\right]$

4. $S = 1 - \frac{3}{(R+G+B)}\left[\min(R,G,B)\right]$

5. $I = \frac{1}{3}(R+B+G)$

Thresholding

For each HSV pixel of an image perform, the following calculations to form output image I:

1. $I = \begin{cases} 255 & \text{if corresponding } H,S,V \text{ are all within acceptable range} \\ 0 & \text{otherwise} \end{cases}$

Note: Each plane in the colorspace is stored as an 8-bit unsigned integer. Thus, the range of each plane is [0, 255].

CAMSHIFT and Moments [13]

Let *I*(*x*, *y*) be an HSV image.

Let *SearchWindow* be a rectangle. This object is saved between function calls (static).

1. While algorithm has not converged, perform the following calculations:

2. Create a new window, *W*, by expanding *SearchWindow* in every direction (+*x*, -*x*, +*y*, -*y*).

3. Perform Thresholding on image *I*(*x*, *y*) saving only data within W. The output of this operation is stored in *T*(*x*, *y*).

4. For every pixel within W:

5. $m_{00} = m_{00} + T(x, y)$

6. $m_{01} = m_{01} + T(x, y)$

7. $m_{10} = m_{10} + T(x, y)$

8. End For Loop at 4.

9. *OldCentroid* = *CurrentCentroid*

10. *CurrentCentroid.x* = round(m_{10}/m_{00})

11. *CurrentCentroid.y* = round(m_{01}/m_{00})

12. $WindowMultiplier = round\left(\sqrt{\frac{m_{00}}{255}}\right)$

13. If *NewCentroid* is close enough to *OldCentroid* then algorithm converged.

14. End While Loop at 1.

15. Let all $\mu = 0$.

16. For every pixel within W:

17. $\mu_{20} = \mu_{20} + T(x, y)(x - CurrentCentroid.x)^2$

18. $\mu_{02} = \mu_{02} + T(x, y)(y - CurrentCentroid.y)^2$

19. $\mu_{11} = \mu_{11} + T(x, y)(x - CurrentCentroid.x)(y - CurrentCentroid.y)$

20. $\mu_{30} = \mu_{30} + T(x, y)(x - CurrentCentroid.x)^3$

21. $\mu_{03} = \mu_{03} + T(x, y)(y - CurrentCentroid.y)^3$

22. $\mu_{12} = \mu_{12} + T(x, y)(x - CurrentCentroid.x)(y - CurrentCentroid.y)^2$

23. $\mu_{21} = \mu_{21} + T(x, y)(x - CurrentCentroid.x)^2(y - CurrentCentroid.y)$

24. End For Loop at 16.

25. $\theta = \frac{1}{2}\tan^{-1}\left(\frac{2\mu_{11}}{\mu_{20} - \mu_{02}}\right)$

26. Compute *NewSearchWindow* using NewSearchWindowAlgorithm.

27. *SearchWindow* = *NewSearchWindow* centered around *CurrentCentroid*.

28. $\lambda = \frac{p+q}{2} + 1$

29. $\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\lambda}}$

30. $\phi_1 = \eta_{20} + \eta_{02}$

31. $\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$

NewSearchWindowAlgorithm

Given *SearchWindow* and *WindowMultiplier* from the CAMSHIFT algorithm and the gesture rotation θ. Given *HeightToWidthRatio* from user-adjustable parameters.

1. $\theta = abs(\theta)$

2. $\rho = \tan^{-1}\left(\frac{1}{HeightToWidthRatio}\right)$

3. $\theta = abs(\theta - \rho)$

4. $z = \sqrt{\left[(WindowMultiplier)^2 + \left[(HeightToWidthRatio)(WindowMultiplier)\right]^2\right]}$

5. $NewSearchWindow.height = round(z \cdot \sin(\theta + \rho))$

6. $NewSearchWindow.width = round(z \cdot \cos(\theta))$

Microstate Classification

1. $D_{\min} = INFINITY$

2. $Class = UNKNOWN$

3. For each gesture class, *c*, given an unknown feature vector *x*:

    4. D = MahalanobisDistance(x)

    5. If $D < D_{min}$

    then

        6. $D_{min} = D$

        7. $Class = c$

    8. End If at 5.

9. End For Loop at 3.

10. *MicrostateClassification = Class*

Macrostate Classification

Let *s* be a FIFO list containing 3 elements, all microstates.

Upon reception of a new microstate, add it to list *s*.

1. If *s*[0] = *s*[2]

then

    2. *Macrostate* = *s*[0].microstate

else

    2. Macrostate is unchanged.

4. End If at 1.

APPENDIX C: ACADEMIC VITA OF ISAAC GERG

EDUCATION

The Pennsylvania State University, University Park, PA
Major: Computer Engineering

Advanced Digital Image Processing	Design of Digital Systems	Database Management Systems
Computer Vision	Signals & Systems	Operating Systems
Numerical Analysis	Electronic Circuit Design	Computer Architecture
Data Structures and Algorithms	Modern Physics	Technical Writing
	Fluids and Thermodynamics	Probability

CLEARANCES

United States Department of Energy security clearance (Level L). May 02/03 – August 02/03

COMPUTER SKILLS

C, C++, Java, Gdb, Microsoft Foundation Classes, MATLAB, VTK, Object Pascal (Borland Delphi), Win32 API, VHDL, MIPS
RISC Assembly, Windows Sockets, Visual Basic, QBASIC 4.5
ASP, PHP, XML, PL/SQL, JSP, NetBeans, MySQL, Python, VBA, HTML
Synopsis, PSPICE, Latex, Microsoft Office Package, ClearCase
Windows, Linux (Mandrake, Debian, Suse), Solaris, FreeBSD, BeOS, DOS , Macintosh
Adobe Photoshop, Macromedia Flash, extensive web & graphic design skills

PROFESSIONAL EXPERIENCE

5/04 – 8/04 The Pennsylvania State University, State College, PA
Software Engineer – Image Processing / GIS

- Developed a client / server software used to position probe and acquire ultrasonic images used for tomographic reconstruction of human tissues.
- Invented and documented a Java suite to mimic Matlab image processing tools. The suite features dozens of functions including, but not limited to: mathematical morphology, split and merge, and correlation.
- Verified and ported target recognition code from Matlab to Java.

5/03 – 8/03 Bechtel Bettis Atomic Power Laboratory, West Mifflin, PA
Summer Intern – Software Engineering Group

- Acted as a member of a design team in Generic Instrumentation and Control Software Engineering.
- Developed a computer software program that communicated directly with an embedded system. The program collected plant control data for use in calorimetric calibration.
- Software engineering tools used were C++ language, Microsoft Foundation Classes, LabView, and embedded system test equipment.

5/02 – 8/02 Bechtel Bettis Atomic Power Laboratory, West Mifflin, PA
Summer Intern – Quality & Environmental Assurance Group

- Provided analysis to support the issue of a technical information package describing the first canister of spent fuel loaded by the Naval Nuclear Propulsion Program. Specific calculations included radionuclide inventory, decay heat levels, and mass of uranium, heavy metals, and fissile material.
- Created a computer program that processed weather data from Naval Reactors sites. This program provides a saving of three man weeks per application**.** It is estimated that the program will save one man-year of work.

12/01 – 1/02 Community Nurses of Elk & Cameron Counties, St. Marys, PA
Internet Programmer

- Created entire website and set up hosting, domain name, and email accounts (www.communitynurses.org).
- Designed a professional style website with searching capabilities. Implemented an elaborate dynamic content and update system via PHP and MySQL for use by non-technical personnel.

6/01 – 8/01 Anderson & Kime Realty Services, St. Marys, PA
Internet Programmer

- Recreated entire website – www.andreson-kime.com
- Designed and programmed an online real estate entry system. System allowed for non-technical users to dynamically upload property information to website in real time.
- Created a database system with ASP and SQL, which allows website viewers to search or view real estate properties by categories such as price or location.

6/00 – 8/00 SinterFire Inc., St. Marys, PA
Press Operator

- Operated many small moldings presses, which produced ammunition.
- Computed, monitored, and maintained parts within a desired mass, density, and diameter range.

RESEARCH EXPERIENCE

9/01 – 12/01 Dr. Gary J. Weisel, Altoona, PA
Research Assistant

- Assisted in the conduction of a numerical simulation on the nuclear scattering of deuterons with hopes of better characterizing the strong nuclear force.
- Edited and executed scripts, which computed uncertainty values for a Monte-Carlo numerical simulation.
- Installed CD-Burner and burning software on a Linux machine to create backups of data
- Created user accounts and provided secure, remote access on a Linux system via SSH.

LEADERSHIP EXPERIENCE

5/03- 9/03 The 6th Annual Melissa Heydenreich 5k Moxie-Thon
Race Director

- Coordinated over 400 people in a 5k road race through Penn State, University Park.
- Established over $2,500 for race sponsorship.
- Realized over $7,300 in funds from participants. Funds donated to the Leukemia & Lymphoma Society.

00 – Present Lion Ambassador Student Alumni Corps
Various positions including: Moxie-Thon Ad-Hoc, IT Chairperson, Committee Secretary, & Historian

- Responsible for serving as a liaison between prospective students and alumni. Guided over 150 families through a tour of University Park and Altoona campuses. Over 70 hours of public speaking.
- Coordinated campus and community events, which communicate University history and personality.
- Unwavering life style of commitment to the Pennsylvania State University through tradition, excellence, and pride.

ACTIVITIES

00 - Present	Lion Ambassador Student Alumni Corps Moxie-Thon Director, IT Chairperson, Committee Secretary, Historian, Source Book Coauthor
03	Tau Beta Pi – Engineering Honor Society
01 - Present	Lion Scout Student Recruitment Program – Visited high schools to recruit students for Penn State.
00 - 02	Altoona College Pep Band – Drumline Section Leader
01 - 02	PSUnix - Founder & President of an organization for the support of the Unix/Linux operating systems
01 - 02	L.I.F.E. House Resident – Non-Drinking, non-smoking lifestyle residence hall.
01	Freshman Orientation Leader
00	Alpha Lambda Delta – Freshman Honor Society

AWARDS

Recipient of the Jerome J. Kapitanoff Memorial Scholarship
Recipient of Who's Who Among College Students (faculty/staff nominated)
Golden Key Society - Top 15% of 2002 class at Penn State

PUBLICATIONS

Software - *Alphabetixe*. PC Utilities Magazine (http://www.livepublishing.co.uk/content/pcutilities.shtml). Issue 26.