MMX-Accelerated Real-Time Hand Tracking System

Nianjun Liu and Brian C. Lovell

Intelligent Real-Time Imaging and Sensing (IRIS) Group
School of Computer Science and Electrical Engineering
The University of Queensland ,Brisbane,Australia 4072
Email: {nianjun, lovell}@csee.uq.edu.au

ABSTRACT

We describe a system for tracking real-time hand gestures captured by a cheap web camera and a standard Intel Pentium based personal computer with no specialized image processing hardware. To attain the necessary processing speed, the system exploits the Multi-Media Instruction set (MMX) extensions of the Intel Pentium chip family through software including. the Microsoft DirectX SDK and the Intel Image Processing and Open Source Computer Vision (OpenCV) libraries. The system is based on the Camshift algorithm (from OpenCV) and the compound constant-acceleration Kalman filter algorithms. Tracking is robust and efficient. and can tracks hand motion at 30 fps.

Keyword: Real-Time, Camshift algorithm, Kalman filter, moment, HSV color, Gesture Recognition

1. INTRODUCTION

Every interaction with the physical world involves some form of physical manipulation, which may be considered as a gesture. We wish to capture these gestures to form a touch-free computer interface using techniques from computer vision. A major impediments to developing real time computer vision systems such as this has been the computational power and level of skill required to process video streams in real-time. This has meant that many researchers have either analyzed video streams off-line or used expensive dedicated hardware acceleration techniques. While valuable, off-line demonstrations tend to be unconvincing as they are typically performed on just a few sequences. Hardware accelerated real-time systems suffer from high cost, lack of portability, and high maintenance due to the enormously rapid development in image processing hardware platforms. Recent software and hardware developments have greatly eased the development burden of real-time image analysis leading to the development of portable systems using cheap PC hardware and software by exploiting the Multi-Media Extension (MMX) instruction set of the Intel Pentium chip. Notable software packages to aid the development of MMX-accelerated computer vision applications have been released by Intel and Microsoft. Intel has released the Image Processing (IPL) and Open Computer Vision (OpenCV) Libraries to the public domain. Microsoft has released the DirectX Software Development Kit that provides the necessary drivers for a large number of video capture devices including universal serial bus, Firewire connected cameras and streaming video formats.

The final system is a computationally efficient computer vision system for tracking and recognizing hand gestures, which exploits efficient C++ coding and MMX-acceleration to achieve its real-time performance on low cost hardware. The system is the first step towards replacing the mouse interface on a standard personal computer with a touch-less interface to control application software.

2. REAL-TIME SOFTWARE TOOLS

Microsoft DirectX is a set of low-level application programming interfaces (APIs) for creating games and other multimedia applications. Microsoft DirectShow is an API for streaming media on the Microsoft Windows platform. DirechShow simplifies media playback, format conversion, and capture tasks. At the same time, it provides access to the underlying steam control architecture for applications that require custom solutions.

The Intel Image Processing Library provides a set of low-level image manipulation functions in standard DLLs and static library form. The functions are optimized for Intel Architecture processors, and are particularly effective at taking advantage of MMX (Multimedia Extensions) technology, the Streaming SIMD Extensions (SSE) and SSE-2 [6]. The Open Source Computer Vision Library is mainly aimed at real-time computer vision

3. IMPLEMENTATION

Figure 1 is the flow chart of the system.. First we convert the image color model from RGB to HSV, and then convert the image to a skin-colour probability distribution image using a predefined lookup table for skin colour hue. Next, we use eiher the Continuously Adaptive Mean-shift algorithm (Camshift) algorithm [10] or Kalman filter [11] to determine a region of interest (ROI) surrounding the hand in each successive frame. Finally, we use canny edge detector on the grayscale ROI determined in the previous step to extract the contour of the hand, and combine this edge map with the skin-colour probability distribution image to determine a reliable contour.

3.1 HSV Color Transformation and Histogram Based Probability Distribution Image.

The hand region is separated from background clutter by using a discrete probability model of human skin color and typical image background. Skin segmentation is made invariant to illumination changes and differing skin tones by using an illumination normalized color space and a skin color probability distribution that is generated from a range of different ethnic groups. The raw image is converted to a color probability distribution image via a color histogram model of the color being tracked (skin for hand gesture tracking). We use the Hue Saturation Value (HSV) color space and create a model of the desired hue using a color histogram. The HSV space separates out hue (colour) from saturation (how concentrated the colour is) and intensity. The colour model is created by taking a 1-D histograms of the H(hue) channel in HSV space. For hand tracking via a skin color model, skin regions from the users are sampled by prompting them to center their face in an onscreen box. The hues derived from the skin pixels in the image are sampled from the H channel and binned into a 1D histogram. When sampling is complete, the histogram is saved for future use.

Clearly, sampling flesh hues from multiple people may make more robust histograms. However, even simple skin histograms tend to work quite well with a wide variety of people. A common misconception is that different color models are needed for different races of people, for example negroids and caucasians. That is simply not true. All human skin is much the same hue. Dark-skinned people simply have more skin colour saturation than light-skinned people, and these differences are largely removed in the HSV colour system and therefore ignored in our skin-tracking colour model.

During processing, the stored skin color histogram is used to convert incoming video pixels to a corresponding probability of flesh image. This is done for each video frame. Using this method, probabilities range in discrete steps from zero to the maximum probability pixel value (1.0 in our case). We then track using the Camshift algorithm on this probability of flesh image. When using real cameras with discrete pixel values, a problem can occur when using HSV space in low illumination conditions. If intensity is low (V near 0), hue then become quite noisy due to large discretization noise in the RGB values. This then leads to wild swings in the hue values. To overcome this problem, we simply ignore the hue of pixels that have very low intensity values. Thus for very dim scenes, the camera must auto-adjust or be adjusted for increased brightness so that it can track reliably.

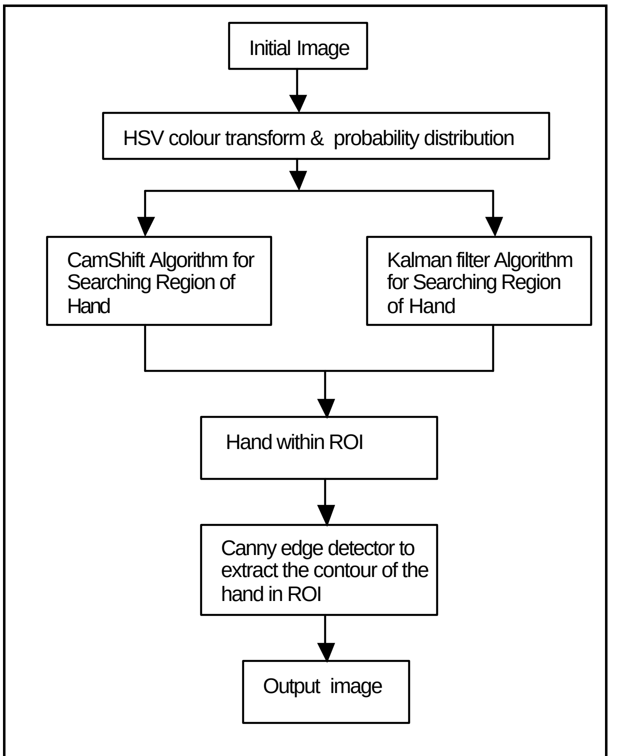

Initial Image
HSV colour transform & probability distribution
CamShift Algorithm for Searching Region of Hand
Kalman filter Algorithm for Searching Region of Hand
Hand within ROI
Canny edge detector to extract the contour of the hand in ROI
Output image

FIgure 1 Flow chart of hand tracking system

4.2 CamShift algorithm

The Continuously Adaptive Mean-shift algorithm, or CamShift, algorithm is a generalization of the Mean shift algorithm which is designed for static distributions. CamShift operates on a 2D color probability distribution image produced from histogram back-projection. It is designed for dynamically changing distributions. These occur when objects in video sequences are being tracked and the object moves so that the size and location of the probability distribution changes in time. The Camshift algorithm adjusts the search window size in the course of its operation. For each video frame, the color probability distribution image is tracked and the center and size of the color object as found via the Camshift algorithm. The current size and location of the tracked object are reported and used to set the size and location of the search window in the next video image. The process is then repeated for continuous tracking. Instead of a fixed, or externally adapted window size, Camshift relies on the zeroth moment information, extracted as part of then internal workings of the algorithm, to continuously adapt its windows size within or over each video frame.

CamShift Algorithm Process (from [10]):

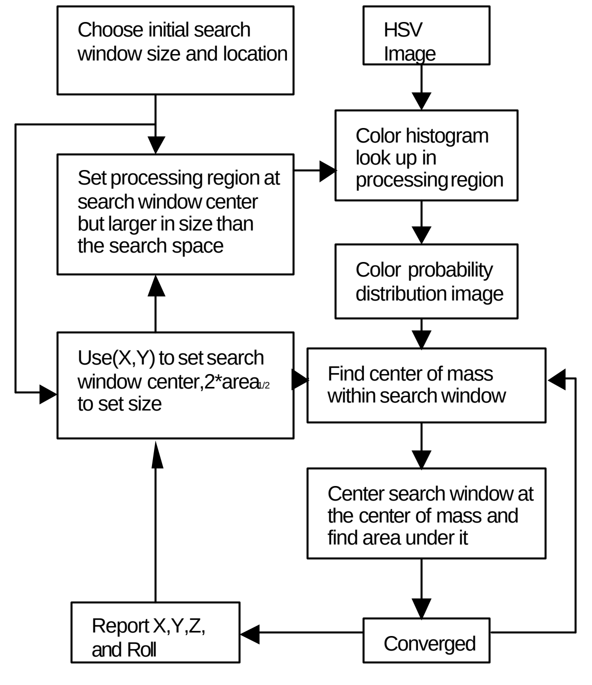
Step1 Choose the initial location of the 2D mean shift search window.

Step 2 Calculate the color probability distribution in the 2D region centered at the search window location in an ROI slightly larger than the mean shift window size.

Step 3 Run mean shift algorithm to find the search window center. Store the zeroth moment (area or size) and center location. The centroid (mean location) within the search window is found as follows:

The zeroth moment:

$$M_{00} = \sum_x \sum_y I(x, y) \qquad (4.2.1)$$



Choose initial search window size and location
HSV Image
Color histogram look up in processing region
Set processing region at search window center but larger in size than the search space
Color probability distribution image
Use(X,Y) to set search window center,2*area1/2 to set size
Find center of mass within search window
Center search window at the center of mass and find area under it
Report X,Y,Z, and Roll
Converged

Figure 2 Camshift Algorithm for finding the hand region

Find the first moment for x and y:

$$M_{10} = \sum_x \sum_y xI(x, y); M_{01} = \sum_x \sum_y yI(x, y) \quad (4.2.2)$$

The centroid then is found by:

$$x_c = \frac{M_{10}}{M_{00}}; y_c = \frac{M_{01}}{M_{00}} \qquad (4.2.3)$$

Where I(x,y) is the pixel (probability) value in the position(x,y) in the image, and x and y range over the search window.

Step 4: For the next video frame, center the search window at the mean location stored in step 3 and set the window's size to a function of the zeroth moment found there.

Step 5 Calculation of 2D Orientation

The 2D orientation of the probability distribution is also easy to obtain by using the second moments in the course of Camshift operation, where(x,y) range over the search window, and I(x,y) is the pixel(probability) value at (x,y). Second moments are:

$$M_{20} = \sum_x \sum_y x^2 I(x, y) \qquad (4.2.4)$$

$$M_{02} = \sum_x \sum_y y \;\; I(x, y) \qquad (4.2.5)$$

Then the object orientation(major axis) is

$$q = \frac{\arctan\left(\frac{2\left(\frac{M_{11}}{M_{00}} - x_c y_c\right)}{\left(\frac{M_{20}}{M_{00}} - x_c^2\right) - \left(\frac{M_{02}}{M_{00}} - y_c^2\right)} \right)}{2} \qquad (4.2.6)$$

The first two eigenvalues(major length and width)of the probability distribution found by Camshift may be calculated in the closed form as follows. Let

$$a = \frac{M_{20}}{M_{00}} - x_c^2, b = 2\left(\frac{M_{11}}{M_{00}} - x_c y_c\right), c = \frac{M_{02}}{M_{00}} - y_c^2 \ . \qquad (4.2.7)$$

Then length *l* and width *w* from the distribution centroid are

$$l = \sqrt{\frac{(a+c) + \sqrt{\left(b^2 + (a-c)^2\right)}}{2}}; w = \sqrt{\frac{(a+c) - \sqrt{\left(b^2 + (a-c)^2\right)}}{2}} \qquad (4.2.8)$$

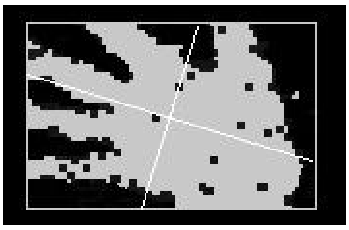


Figure 3 color probability distribution image

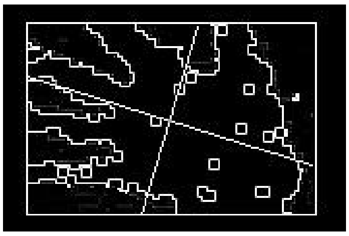


Figure 4 Extracting the contour from figure 3

4.3 Kalman Filter Tracking

Kalman filters provide an efficient, recursive technique to minimize the least-squares error of each prediction where the system model is governed by a linear, stochastic difference equation, therefore,a two-dimensional Kalman filter (stochastic estimator) is used to track the hand region centroid in order to accelerate hand segmentation and choose the correct skin region when multiple image regions are skin coloured. Using a model of constant acceleration motion the filter provides an estimate for hand location, which guides the image search for the hand. The Kalman filter tracks the movement of the hand from frame to frame to provide an accurate starting point for searching a skin colour region which is the closest match to the estimate.

Instead of segmenting the entire input image into multiple skin regions and selecting the region, which is closest to the filter estimate, the filter estimate is used as the starting point for the search for a skin colour region. The measurement vector Z_k consists of the location of the centroid of the hand region. Therefore the filter estimate $H_k \hat{x}_k^-$ is the centre of a distance-based search for a skin coloured pixel. As the hand region may be assumed to have a certain minimum area, a grid of pixel points tested in order of increasing distance from the filter estimate should find the best matching region. The spacing of the grid is determined by the minimum allowable hand area. Upon finding a skin colored pixel, the contour following routine is started to trace the connected skin region around the pixel. If the area of region is below the hand area threshold then the region is discarded and the search is continued with that region excluded from the search grid.

The Kalman filter estimates a process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements. As such, the equations for the Kalman filter fall into two groups: time update equations and measurement update equations. The time update equations can be thought of as predictor equations, which are responsible for projecting forward(in time) the current state $\hat{x}_k$ and error covariance P_k estimates to obtain the a priori estimates $\hat{x}_k^-$ for the next time step.

The time update equations are:

$$\hat{x}_k^- = A_k \hat{x}_k + Bu_k \qquad (4.3.1)$$

$$P_{k+1}^- = A_k P_k A_k^T + Q_k \qquad (4.3.2)$$

The measurement update equations can be thought of as corrector equations, which are responsible for feedback i.e. for incorporating a new measurement Z_k into a priori estimate $\hat{x}_k^-$ to obtain an improved posteriori estimate $\hat{x}_k$. Measurement update equations are:

$$K_k = P_k^- H_k^T \left(H_k P_k^- H_k^T + R_k \right)^{-1} (4.3.3)$$

$$\hat{x}_k = \hat{x}_k^- + K_k \left(z_k - H_k \hat{x}_k^- \right) \qquad (4.4.4)$$

$$P_k = (I - K_k H_k) P_k^- \qquad (4.3.5)$$

Implementation of the filter requires that the error covariance matrices Q_k for process noise and R_k for measurement noise be fixed *a priori*. Process noise represents the accuracy of the model and is usually very hard to determine analytically. In fact it is usually determined empirically. Measurement noise can usually be determined directly from off-line calibration tests. Filter initialization consists of an estimate of $X_k$and P_k. A choice for the state vector is usually obvious from the nature of the system. P_k is hard to choose, but under most conditions its value will soon stabilize through the operation of the filter. Note that if Q_k and R_k are constant then P_k and K_k (Kalman gain) will stabilize and then remain constant. These steady-state matrices can be computed by running the filter and finding the settled values, or by solving the updated equations for steady-state conditions.

The Kalman filter used here is based on a model of two-dimensional constant acceleration motion. The filter may be described in terms of its constituent vectors and matrices. The state space includes the position(s), velocity and acceleration of the hand region centroid in the two dimensions. Since the dimensions are treated independently but identically, the filter will be described for the x axis motion only using s as the coordinate to avoid notation conflict. A state vector $x \in R^3$ is given by:

$$x = \begin{bmatrix} s \\ \dot{s} \\ \ddot{s} \end{bmatrix} \qquad (4.3.6)$$

As only the centroid position is measured, a measurement vector $z \in R$ has the simple form.

$$z = [s] \qquad (4.3.7)$$

Hence the measurement matrix H which relates the measurement to the actual state is trivially given by the 3x1 row vector below.

$$H = [1 \quad 0 \quad 0] \qquad (4.3.8)$$

The constant acceleration model is incorporated in matrix A, the state update matrix

$$A = \begin{bmatrix} 1 & \Delta t & (\Delta t)^2/2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix} \qquad (4.3.9)$$

For a constant sample rate(frame rate) matrix A will clearly remain constant, but under multi-tasking conditions when the system is running with other applications, the sample rate varies according to processor load. Thus Δt and hence a must adjusted at each step accordingly.

Values for the noise covariance matrices were determined empirically and fine tuned for best performance. The values for matrix Q suggest that the model derived position should be considered quite noisy compared to velocity and acceleration. As the measurement vector also provides position information, this has the effect of favouring the measurement in the filter correction phase. Velocity and acceleration are generated entirely by the motion model through Matrix A

$$Q = \begin{bmatrix} 20 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{bmatrix} \qquad (4.3.10)$$

As the noise covariance matrices are constant in the filter formulation, the Kalman gain K and the estimate error covariance matrix P will settle to steady state values.

4.4 Canny Edge Detector

It is a very good method for detecting edges suggested by J.Canny. It takes grayscale image on input and returns bi-level image where non-zero pixels mark dectected edges.The following describes the 4 steps.

Step 1 Image smoothing:The image data is smoothed by a Gaussian function of width specified by the parameter.

Step 2 Differentiation The smoothed image, retrieved at step 1, is differentiated with respect to the directions x and y.From the computed gradient values x and y, the magnitude and the angle of the gradient can be calculated using the hypotenuse and arctangent functions. We can join the smoothing and differentiation in sober operator.

Step 3 Non-Maximum Suppression. After the gradient has been calculated at each point of the image, the edges can be located at the points of local maximum gradient magnitude. It is done via suppression of non-maximums, that is points, whose gradient magnitudes are not local maximums. However, in this cases the non-maximums perpendicular to the edge direction, rather than those in the edge direction, have to be suppressed, since the edge strength is expected to continue along an extended contour. The algorithm starts off by reducing the angle of gradient to one of the four sectors. The algorithm passes the 3x3 neighborhood across the magnitude array. At each point the center element of the neighborhood is compared with its two neighbors along line of the gradient given by the sector value. If the central value is non-maximum, that is, not greater than the neighbors, it is suppressed.

Step 4. Edge Thresholding The Canny operator uses the so-called "hysteresis" thresholding. Most thresholders use a single threshold limit,which means that if the edge values fluctuate above and below this value,the line appears broken.This phenomenon is commonly referred to as "streaking".Hysteresis counters streaking by setting an upper and lower edge value limit.Considering a line segment,if a value lies above the upper threshold it is immediatedly rejected.Points which lie between the two limits are accepted if they are connected to pixels which exhibit strong response.The likelihood of streaking is reduced drastically since the line segment points must fluctuate above the upper limit and below the lower limit for streaking to occur. The reference ratio of high to low limit is in the range of two or three to one based on predicted signal-to-noise ratios.

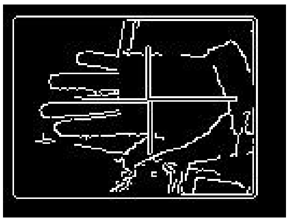


Figure 5 Image only applying Canny edge detector

4.5 Final Output Image Video

Through the system which diagram is figure 1, the output video is shown in Figure 6. It is combination of all of above approaches, which is better than applying each of them respectively. The method is high speed, computationally efficiency, with tracking rate is 30 fps on a standard PC. An example of the video output can be found on *www.csee.uq.edu.au/~iris.*

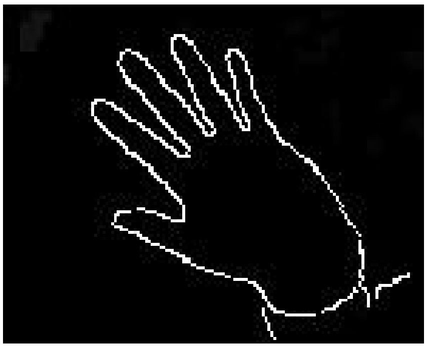


Figure 6 Output image

4. CONCLUSION

The system has been implemented in the Visual C++ environment, running under the Windows 2000 Operating System. It achieves robust real-time performance to track a skin region under very few constraints. Throughout the gesture recognition system, computational efficiency is provided by a compact contour representation and fast contour processing algorithms.

5. REFERENCES

[1] G.A.Agoston, *Color Theory and Its Application in Art and Design*, Springer-Verlag , 1987.

[2] R.A. Bolt, Put-That-Three: Voice and Gesture at the Graphics Interface", Proc SIGGRAPH80,ACM Press,1980.

[3] W.T. Freeman, Hand Gesture Machine Control System", Proc SIGGRAPH80,1980.

[4] V. Pavlovic,"Visual Interpretation of Hand Gestures for Human-Computer Interaction ", Proc. Human Interaction with Complex System,Sept,1995

[5] J. Rehg and T. Kanade, DigitEyes: Vision-Based Human Hand Tracking,Technical Report.CMU-CS93220,CMU, December.1993.

[6] T.Starner and A.Pentland, Real-time American Sign Language Recognition", IEEE Trans, On Pattern Analysis and Machine Intelligence,Vol 20.pp 1371-1375,Dec.1998.

[7] B.-C. Lin and J. Shen. Fast computation of moment invariants. Pattern Recognition, 24(8):807-813,1991.

[8] W.Philips. A new fast algorithm for moment calculation. Pattern Recognition,26(11):1619-1621,1993.

[9] Image Processing Library Reference Manual. developer.intel.com/software/products/perflib/ipl/index.htm.

[10] Intel Open Source Computer Vision Library Reference Manual. www.intel.com/research/mrl/research/opencv

[11] D. Heckenberg and B. C. Lovell, "MIME: A Gesture-Driven Computer Interface", Proceedings of Visual Communications and Image Processing, SPIE, V4067, pp 261-268, Perth, 2000.