Twins: A Practical Vision-Based 3D Mouse

Sifang Li, Wynne Hsu, Pung H. K.
Department of Information Systems and Computer Science
National University of Singapore
Singapore 119260

Abstract
A novel scheme is introduced that uses a mirror and a single camera to restore the 3D position of a finger tip. The camera is positioned in such a way that it captures both the hand as well as its mirror image. The captured images are then processed to extract the contour of the hand. This extraction process is done using a fast algorithm that integrates Bayesian estimation into the traditional chain code generation algorithm. The algorithm works well with complex albeit fixed background. In addition, we have integrated the smoothness assumption into the algorithm so as to obtain a smoother hand contour thus improving the accuracy of our tracking process. Tolerance to noise can be improved by utilizing the constraints imposed by our system setting to eliminate false signals. A prototype system has been implemented and the performance of the 3D mouse before different backgrounds has been analyzed.

1 Introduction

Due to the increased availability of high speed 3D hardware and software on all classes of computer, 3D interaction is becoming more and more popular [1]. 3D applications can be found in domains such as medical and scientific visualization, computer-aided design, computer-aided education and entertainment. Compared with electromagnetic 3D input device such as VPL's DataGlove [2], a 3D vision-based solution has many advantages. The chief advantage is that a vision-based solution frees the user from any form of physical attachment that may constraint his/her movement. For instance, there is no putting on/ taking off of data glove, the user can simply operate with his/her bare hand. Even though some physical devices [3,4] have tried to accomplish free hand human-computer interface, their high cost has inhibited widespread use. Because of this, vision-based approach has attracted many efforts in computer vision fields [5]-[10], but their results are currently too sensitive and computationally too expensive for practical applications. Many of them impose unacceptable constraints on the user, such as smooth or periodic movement [5], static supporting points (shoulder or palm) [6, 7, 9]. Some of them tried to achieve a complex gesture recognition system but finally degraded their system to a 3D mouse [7]-[9] because of the issues of reliability and computation expense. In this paper, instead of try to achieving the complex gesture recognition system, we address some of these issues about a 3D mouse from the practical user interface standpoint. Ideally, a 3D vision-based mouse solution should satisfy the following criteria:

i. Robustness to movement: There should be as few constraints imposed on the user's hand movement as possible. This is to ensure that the ease of usability is maintained.
ii. Fast pick up: The algorithm should be able to pick up the target object any time and anywhere. This allows the user's hand to move in and out of the scene just as s/he tends to pick up a normal 2D mouse now and then.
iii. Low computational cost: Fast response time is a critical factor in the design of a good user interface. Thus, our 3D vision-based solution should not be too computationally expensive.

iv. Robustness to false recognition: Since no tracking algorithm can guarantee a 100% correct recognition rate, it is important for the algorithm to be able to recover from false recognition.
v. Low cost solution: The solution should not be too costly to implement. In other words, we need to cut down on the number of hardware equipment needed to implement the solution.

In this paper, we propose a novel scheme to address these issues. A 3D vision-based mouse is implemented with the help of a mirror and a single camera. Instead of using 2 cameras to provide the two views of an object, we position the mirror and camera in such a way that the reflection from the mirror provides the second image needed to restore the 3D information. This effectively cut down on the number of camera needed thus saving cost (criterion (v)). To satisfy criteria (i) - (iv), we propose an algorithm to detect hand contour before a fixed but complex background. Details of this algorithm is given in Section 3.

2 Related Work

Vision-based 3D mouse was mentioned several times in recent publications [5]-[10]. Many of them rely on strong prior knowledge about the object, such as 2D shape model [5] (Kalman Snakes [11]) or 3D anatomic model [6]-[9] to guide in deriving the measurement of an image. Strong movement models such as smooth, slow, affine or periodic properties are also used in their next model state prediction. Typically, these applications have the following program structure (see Figure 1).

Figure 1 is almost a standard procedure for model-based video analysis. It provides partial solutions to the transient occlusion problem and is able to handle dynamic backgrounds. However, when it comes to applying these models in practical user interface (UI) applications, a number of problems surface. The first problem is the difficulty in initiating the model. For practical UI application, the user's hand should be allowed to re-enter the scene easily. This implies that initiation needs to be done each time the user re-enters the scene. Such initiation is very expensive without introducing additional constraints such as fixed background [6,7,9,10] or arranged initialization area [5,7]. The second problem is the inability to restore from false recognition. In the model-based approach, estimation of the image is directly computed from the previous result. If the result is false due to the inaccuracy of the feature measurement, lock-loss is possible. Once the lock is lost, regaining the lock is difficult because the initiation of the model is expensive. In [5], learning is used to prevent lock losing, but this only applies to those movements which are used to train the system. Given the fact that the false recognition in a vision-based system is inevitable and the user's movement cannot be constrained too much, model-based approach is not appropriate for our application.

3 Twins Architecture

3.1 Gesture

In place of a physical mouse, our system, Twins, allows the use of the bare hand to operate as a 3D mouse. This means that our system must be able to simulate typically mouse operations such as moving the mouse and pushing the mouse button. The gesture used in Twins is described in Figure 2. The index finger is used to specify the 3D position of the mouse while the thumb is used to specify the state of the mouse button. An erected thumb means that the button is not pressed. A bend thumb signifies the pressing of the mouse button. This gesture is similar to the gesture used in [7]. The next question is how we obtain the 3D position of the index finger accurately.

3.2 Twins setup

To accurately determine the 3D position of an object, we need at least two views. Many stereo vision systems use two or more cameras to restore the 3D position. On the other hand, there are also indirect methods to restore 3D position of an object using just one camera. These methods utilize shading, texture or size information [5] or 3D structure [6,7,9] as cues in order to estimate the depth information. However, they tend to be unreliable because the depth information cannot be obtained directly.

In Twins, a novel method using one camera and a mirror is introduced, this is shown in Figure 3. A video camera is placed such that it is projecting at an angle of 45° to the mirror. The coordinate system XYZ is described in Figure 3 with Y axis sticking out from the paper surface. In the image plane, we have two views of an object (the actual object and its image). In general, the camera lens may cause the image formed to be smaller than the projection of the object on the image plane. For this paper, we magnify the image to a particular size and assume the image formed can be considered as the projection of the object on the image plane. This assumption is true so long as the object distance is much further than the focal length of the camera lens. In fact, none of the vision-based user interface application ever takes this into consideration.

In Figure 3, r is the real object and v is the mirror image (virtual object) of r. Their coordinates in the magnified image plane are (x_r, y_r) and (x_v, y_v) respectively. Let us assume that we also know the projection $(x_{vc'}, y_{vc'})$ of the virtual camera VC' which is the mirror image of the camera. We can now decide the 3D position of the real object using these three projections (images) - (x_r, y_r), (x_v, y_v) and $(x_{vc'}, y_{vc'})$. Note that the X and Y positions of the real object are x and y respectively. The Z value is a little more tricky. From the symmetric property of reflection, we have $z = x_{vc'} - x_v$, thus we have equation (1)

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_r \\ y_r \\ x_{vc'} - x_v \end{bmatrix} \tag{1}$$

Using a simple geometry transform, we transform the XYZ coordinate system to a more natural coordinate system UVW with W axis sticking out of the paper (see Figure 3). P is the translation vector.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} -\sin\frac{p}{4} & 0 & \cos\frac{p}{4} \\ -\cos\frac{p}{4} & 0 & -\sin\frac{p}{4} \\ 0 & 1 & 0 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ z \end{bmatrix} + P \tag{2}$$

From (1), (2) can be rewritten as:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} -\sin\frac{p}{4} & 0 & \cos\frac{p}{4} \\ -\cos\frac{p}{4} & 0 & -\sin\frac{p}{4} \\ 0 & 1 & 0 \end{bmatrix} \bullet \begin{bmatrix} x_r \\ y_r \\ x_{vc'} - x_v \end{bmatrix} + P \tag{3}$$

In equation (3), all the variables are known except $x_{vc'}$. This is because the virtual camera, VC', is outside the scope of our camera. Instead, we select the origin, O, of the UVW system such that both O and its virtual object can be seen in the image plane. In other words, we know both (x_{r0}, y_{r0}) and (x_{v0}, y_{v0}). From (3), we have (4).

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -\sin\frac{p}{4} & 0 & \cos\frac{p}{4} \\ -\cos\frac{p}{4} & 0 & -\sin\frac{p}{4} \\ 0 & 1 & 0 \end{bmatrix} \bullet \begin{bmatrix} x_{r0} \\ y_{r0} \\ x_{vc'} - x_{v0} \end{bmatrix} + P \tag{4}$$

Subtracting (4) from (3), we get (5).

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} -\sin\frac{p}{4} & 0 & \cos\frac{p}{4} \\ -\cos\frac{p}{4} & 0 & -\sin\frac{p}{4} \\ 0 & 1 & 0 \end{bmatrix} \bullet \begin{bmatrix} \Delta x_r \\ \Delta y_r \\ -\Delta x_v \end{bmatrix} \tag{5}$$

where

$$\begin{bmatrix} \Delta x_r \\ \Delta y_r \\ -\Delta x_v \end{bmatrix} = \begin{bmatrix} x_r - x_{r0} \\ y_r - y_{r0} \\ -(x_v - x_{v0}) \end{bmatrix}$$

Rewriting (5), we obtain:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} -\sin\frac{p}{4} & 0 & -\cos\frac{p}{4} \\ -\cos\frac{p}{4} & 0 & \sin\frac{p}{4} \\ 0 & 1 & 0 \end{bmatrix} \bullet \begin{bmatrix} \Delta x_r \\ \Delta y_r \\ \Delta x_v \end{bmatrix} \tag{6}$$

Thus, after fixing the origin, O, of the UVW coordinate system, the 3D position of an object can be easily restored using (6) as long as both the real object and its virtual object are visible. This setup helps to save hardware cost as compared to multi-camera systems, and is particularly important for wide user's acceptance as computers with one camera are becoming common nowadays whereas multiple-camera system is still largely a specialized commodity.

3.3 Twins Architecture

3.3.1 Background and Hand Intensity Models

Having described the setup of the Twins system, we now describe the underlying algorithm used by our 3D mouse application. As mentioned in Section 2, model-based approach faces many problems. An alternative approach is to use background information to perform video segmentation. In fact, many practical work such as Pfinder [12] uses background knowledge to perform segmentation. Similarly, in Twins, we assume a fixed background and use prior background and hand intensity knowledge to perform feature measurement. The background and hand intensity knowledge are modeled using simple Gaussian models. To learn the background models, Twins require an initial startup time of n continuous frames to build the Gaussian models:

$$\begin{aligned} m(x,y) &= \frac{1}{n}\sum_{k=0}^{n-1} i_k(x,y), \\ s^2(x,y) &= \frac{1}{n}\sum_{k=0}^{n-1} i_k(x,y)^T i_k(x,y) - m(x,y)^T m(x,y) \end{aligned} \tag{7}$$

where i is the intensity vector expressed in RGB space.

Based on the background Gaussian models, a single Gaussian model describing the hand intensity is built up by placing the hand before the learned background. Hand pixel segmentation is achieved by:

$$f(x,y) = \begin{cases} i(x,y), & if \quad |m(x,y) - i(x,y)| > ts(x,y) \\ 0, & if \quad |m(x,y) - i(x,y)| \le ts(x,y) \end{cases} \tag{8}$$

After the segmentation, the Guassian model for all hand pixels can be built in a similar way:

$$m_h = \frac{1}{n_h} \sum_{f(x,y)\neq 0} f(x,y), \qquad s_h{}^2 = \frac{1}{n_h} \sum_{f(x,y)\neq 0} f(x,y)^T f(x,y) - m_h{}^T m_h \tag{9}$$

where n_h is the total number of non-zero pixels in segmented image.

3.2.2 Chain Code Generation Algorithm Integrating Smoothness Assumption

The next task is to extract the hand contour so as to perform effective tracking. This is achieved by integrating the smoothness assumption to a well-known chain code generation algorithm so as to obtain a smooth hand contour. The proposed algorithm does not require any prior shape knowledge and is thus robust to fast and dramatic changes in shape. Let us first give a brief review of the tradition chain code generation algorithm.

Tradition chain code generation algorithm

The well known chain code generation algorithm [13] for 8-direction chain code is listed as *Algorithm 1*.

Let N and V be two functions where

$N_j(C)$ returns the j^{th} neighbor of vertex C, and

$V(C_i,C_j) = t$, if C_i is the t^{th} neighbor of C_j;

$V(C_i,C_j) = -1$, otherwise;

Given the initial contour queue "C_0C_1," the algorithm is:

Algorithm 1

1. k=1;
2. t=V(C_k, C_{k-1});
3. j=(t+2) mod 8;
4. $C_{k+1} = N_j(C_k)$;
   if (C_{k+1} in the boundary) goto 6;
   else j=(j-1) mod 8;
5. goto 4
6. add C_{k+1} to the contour queue, k=k+1;
   if ($C_k=C_0$) then halt;
7. goto 2

The neighbor function, N, is defined by a grid that is imposed on the image. *Algorithm 1* uses a square grid and hence each node has 8 neighbors. In general, the grid size and its pattern have an effect on the performance of the chain code. Figure 4 shows the hexagonal pattern with the neighbors of grid point A and B marked out. In [14], Scholten et. al proved theoretically that a hexagonal grid can describe a curve more accurately as compared to the triangle or square grid. In Twins, we tried both square and hexagonal grid and found the latter to provide much more accurate result Though the chain coding algorithm on hexagonal grid is slightly more complex in terms of locating a point's neighborhoods, that can be implemented efficiently. Thus, we have chosen the hexagonal grid for our implementation and its diameter is 12 pixels.

The chain coding algorithm on hexagonal grid is described below.

Algorithm 2

1. k=1;
2. t=V(C_k, C_{k-1});
3. j=(t+3) mod 12 ;
4. $C_{k+1} = N_j(C_k)$;
   if (C_{k+1} in the boundary) goto 6;

else j=(j-1) mod 12;
5. goto 4
6. add C_{k+1} to the contour queue, k=k+1;
   if (C_k=C_0) then halt;
7. goto 2

In *Algorithm 2* that the condition in step 4, "C_{k+1} in the boundary" is not yet defined. An obvious approach to test the condition in step 4 is to use the Gaussian models (Section 3.2.1) we have built up to threshold the pixels. In this case, the value used for the threshold process is defined to be:

$$\frac{P(I|H)}{P(I|B)} = \frac{s}{s_h}\exp(\frac{|i-m|^2}{2s^2} - \frac{|i-m_h|^2}{2s_h^2}) \qquad (10)$$

where *I* denotes that the intensity of the pixel is *i*; *H* denotes that the pixel is a hand pixel; and *B* denotes that the pixel is a background pixel.

Figure 5 shows the results when we compute (10) over all image pixels. The intensity of Figure 5 reflects the magnitude of the results. Using (10) as a thresholding condition for step 4 in *Algorithm 2* , we obtain *Algorithm 3* where λ is a pre-determined threshold value.

Algorithm 3
1. k=1;
2. t=V(C_k, C_{k-1});
3. j=(t+3) mod 12 ;
4. C_{k+1}= N_j(C_k);
   if ($\frac{P(I|H)}{P(I|B)} > l$) goto 6;
   else j=(j-1) mod 12;
5. goto 4
6. add C_{k+1} to the contour queue, k=k+1;
   if (C_k=C_0) then halt;
7. goto 2

Thus far, we have assumed equal probability for each pixel to be a hand pixel. Further improvement of the algorithm is possible if we realize that most parts of the hand contour are smooth.

For example, suppose in Figure 6 we have detected the contour points A and B, and we are now trying to determine the next point, C, in the contour. In the hexagonal grid case, C has 10 options. They are distributed in the directions as depicted in Figure 6. *Algorithm 3* will examine X_0, X_1... until it finds X_k which is in the object and X_k is considered as the next boundary pixel C. Given the fact that most part of the contour are smooth, it is reasonable to assume, before we perform the intensity test, that X_3 is the most probable candidate and X_0 and X_9 are least probable candidates. Based on the above assumption, we can assign different weight values $\{w_i\}$ reflecting the relative probabilities of these 10 directions. The criterion in assigning these weight values is to reflect the relative probabilities of being the next contour pixel. In our experiment, we assign the weight values as follows:

$$w_i = \exp(-\frac{(i-m_w)^2}{r}) \qquad (11)$$

where m_w and r are real numbers and they control the two independent properties of such an assignment. The integer that is closest to m_w defines the direction which has the largest weight value – in our case, it must be near to 3. r reflects the relative difference between the weights. If $r \to \infty$, $\{w_j\}$ becomes a constant series. If r is smaller, the difference between adjacent weight values will be larger which means that the contour is expected to be

smoother, for example, a circle. When choosing r, we should consider the smoothness of the contour. In our experiment we choose r as 6.0 and m_w as 3.1. Having assigned these weight values, we are able to refine *Algorithm 3* to obtain a smoother contour. Before we discuss the refinement, let us first define a function *M*.

$$M(C_{k+1}, C_k, C_{k-1}) = (V(C_k, C_{k-1}) - V(C_{k+1}, C_k)) \quad MOD \quad 12 \tag{12}$$

where MOD is slightly different from the standard mod function in that it returns a result from -5 to 6 instead of 0 to 11 in the following way:

$$n \quad MOD \quad m = \begin{cases} n \mod m, & if \quad n \mod m \le \frac{m}{2}; \\ (n \mod m) - m, & otherwise. \end{cases}$$

The function *M* maps the neighboring 3 grid points C_{k-1}, C_k, C_{k+1} in *Algorithm 2* to a signed integer representing the deflection of the contour at point C_k (See Figure 7). From the execution of the algorithm, function *M* only returns value from -3 to 6.

Now let us look at how we can refine *Algorithm 2* to obtain a smoother contour. *Algorithm 2* examines each X_j in turn until it finds the boundary pixel C. Suppose *Algorithm 2* has examined X_0, X_1... X_{k-1} ($0 \le k < 9$) and yet to find the boundary pixel C, we can conclude the following:

i. $X_0, X_1 ... X_{k-1}$ are all background pixels;

ii. C must be one of the pixel in the pixel set $\{ X_k, X_{k+1} ... X_9 \}$.

By normalizing the weight values, $\{w_j\}$, we obtain Equation (7) which gives the probability that X_k is the boundary pixel C. This probability estimate has not taken into consideration the pixel intensity, and it is just a prior guess.

$$P(C = X_k) = \frac{w_{M(X_k, B, A)+3}}{\sum_{j=M(X_k, B, A)+3}^{9} w_j} \tag{13}$$

Now we need to take the intensity of the pixel into consideration. Let's evaluate
$P(C=X_k|$ pixel intensity of X_k is i).
Using Bayesian rule [15], the posterior probabilities can be computed by:

$$P(C = X_k | I) = \frac{P(C = X_k) P(I | C = X_k)}{P(I)} \tag{14}$$

$$P(C \neq X_k | I) = \frac{P(C \neq X_k) P(I | C \neq X_k)}{P(I)} \tag{15}$$

where I denotes the event that "pixel intensity of X_k is i".
Comparing $P(C = X_k | I)$ with $P(C \neq X_k | I)$, we achieve the condition which can be used in step 4 in *Algorithm* 2. The condition becomes (16):

$$\frac{P(I | C = X_k)}{P(I | C \neq X_k)} > \frac{P(C \neq X_k)}{P(C = X_k)} \tag{16}$$

Consider the items in (16):

i. $P(C=X_k)$: This quantity can be calculated from Equation (13). Note that the value changes dynamically as the algorithm proceeds;

ii. $P(C \neq X_k)$: 1 - $P(C=X_k)$;

iii. $P(I|C=X_k)$: If X_k is in the contour, we can use the hand Gaussian model depict by Equation (9) to obtain an estimate. Thus, we have:

$$P(I | C = X_k) = \frac{1}{\sqrt{2p}\, s_h} \exp\left(-\frac{|i - m_h|^2}{2 s_h^2}\right) \Delta i$$

iv. $P(I|C \neq X_k)$: From the nature of chain code generation Algorithm
we know that if C is not in the contour it must be a background pixel. Using Equation (7), we have:

$$P(I | C \neq X_k) = \frac{1}{\sqrt{2p}\, s} \exp\left(-\frac{|i - m|^2}{2 s^2}\right) \Delta i$$

Put all these items in (16), we can rewrite (16) as:

$$\frac{s}{s_h}\exp\left(\frac{|i-m|^2}{2s^2}-\frac{|i-m_h|^2}{2s_h^2}\right) > \frac{1-P(C=X_k)}{P(C=X_k)} \quad (17)$$

Now we can decide whether a pixel is a hand pixel or a background pixel using (17).
Integrating (17) into *Algorithm 2* , we have the following algorithm.

Algorithm 4

1. k=1;
2. t=V(C_k, C_{k-1});
3. j=(t+3) mod 12 ;
4. if $\left(\frac{s}{s_h}\exp\left(\frac{|i-m|^2}{2s^2}-\frac{|i-m_h|^2}{2s_h^2}\right) > \frac{1-P(C=N_j(C_k))}{P(C=N_j(C_k))}\right)$ goto 6;
   else j=(j-1) mod 12;
5. goto 4
6. C_{k+1}= $N_j(C_k)$;
   add C_{k+1} to the contour queue, k=k+1;
   if (C_k=C_0) then halt;
7. goto 2

In addition, we also introduce a simple technique to make sure that the chain code generation algorithm will search the whole image instead of terminating too early. The technique introduces a special line called the "wrist line" to the image. The wrist line is a horizontal line in the image which pass through the wrist position of the hand. Wrist line is an assumed line that needs to be updated constantly while the finger tip moves. The parameter of such a line depends on the previous position of the finger tip. The horizontal wrist line in frame k, y^w_k can be described as:

$$y_k^{\ w} = y^f_{\ k-1} - d_p \quad (18)$$

where $y^f_{\ k-1}$ is the y position of the finger tip in the k-1 frame and d_p is the length between the finger tip and the wrist. The wrist line in Figure 8 is marked as a dotted line in the image. All the pixels above the wrist line are marked as the hand pixels. This allows us to constrain the contour tracking to be carried out from left to right through the entire image each time. After introducing wrist line, the terminating condition in chain code algorithm becomes "if the contour tracing arrives at the right side of the image."

Though *Algorithm 4* works well in most circumstances, it fails in some particular situations. One example is shown in Figure 8.

Initially, the algorithm traced the contour (the solid line) correctly starting from the left and slowly proceeding to the right. However, halfway through the tracing, it entered a loop as shown in Figure 8. The reason for the loop is due to inconsistent pixel classification. Consider Figure 9. Let us assume that the algorithm has detected A then B and is going to determine the next point. The algorithm examines Y first. Suppose at this stage, Y fails to be classified as a hand pixel and so does X. Note that although Y and X fail to be classified as hand pixels this time, they may possibly be classified as hand pixels in the future as the algorithm proceeds. This is because condition (17) depends not only on the intensity of the pixels but also on P(C=X_k) which is dynamic during the execution of the algorithm. Thus, for Figure 9, the algorithm detects C, D, E, ... W and X, where X is classified as hand pixel now – inconsistent classification! After that, C is classified as the hand pixel for the second time and a loop occurs.

The only way to prevent the loop is to avoid such inconsistent classification. If a pixel has been classified before, the previous classification is retained. Condition (17) is used only if the pixel has not been classified before. This modification gives rise to *Algorithm 5* .

Algorithm 5

1. k=1;
2. t=V(C_k, C_{k-1});

3. j=(t+3) mod 12
4. if ($N_j(C_k)$ was ever classified as hand pixel)
   goto 6;
   else if ($N_j(C_k)$ was ever classified as background pixel) {
   j=(j-1) mod 12;
   goto 5;};
   else if $(\frac{s}{s_h}\exp(\frac{|i-m|^2}{2s^2}-\frac{|i-m_h|^2}{2s_h^2}) > \frac{1-P(C=N_j(C_k))}{P(C=N_j(C_k))})$ {
   mark $N_j(C_k)$ in the classification history as hand;
   goto 6; };
   else {
   mark $N_j(C_k)$ in the classification history as background;
   j=(j-1) mod 12; };
5. goto 4
6. $C_{k+1} = N_j(C_k)$;
   add C_{k+1} to the contour queue, k=k+1;
   if (the contour tracing arrives at the right side of the image) then halt;

The history can be maintained in a 2 dimensional array for fast access. If we also record the current frame number as well as the classification result in the array element, we do not need to reinitialize the array each time. Thus the computational complexity is not changed as compared to the traditional chain code generation algorithm.

4 Experimental Results

A number of experiments have been conducted to compare the performance of *Algorithm 3* (without integrating the smoothness assumption) and *Algorithm 5* (integrating the smoothness assumption). In all the experiments, the contour pixel classifications in *Algorithm 3* and *5* are done on a cross mask on the image pixels to enhance the reliability. Of course, the wrist line and the altered termination condition introduced previously are used in both algorithm implementations. Our experimental results indicate that *Algorithm 5* outputs a smoother contour and is also more accurate in extracting the contour in most cases. Figure 10 shows two frames obtained from a single tracking session. The thresholds and configure values used in this tracking session are listed in Table 1.

The thick line is the contour obtained using *Algorithm 5* while the thin line is the contour obtained using *Algorithm 3*. The differences between the contour results are numbered and highlighted using arrows. In the position marked 1, it is obvious that the *Algorithm 5's* performance (thick line) is better. *Algorithm 3* failed here because the threshold, λ, is too small which makes the algorithm too sensitive. To improve the performance of *Algorithm 3* here, we need to increase the value of λ. This is indicated in the first column of Table 2. Similar comparisons are carried out for all the differing points (marked position 2 to marked position 10). The results of analysis are summarized in Table 2.

From the Figure 10 and Table2, we see that:

i. *Algorithm 5* gives a smoother contour result;
ii. Since the hand contour is smooth, *Algorithm 5* has a better performance in terms of accuracy;
iii. Since a smoother contour is usually shorter, *Algorithm 5* saves unnecessary computational costs;
iv. Changing the threshold value in *Algorithm 3* does not improve its performance globally. We can see this from the second row of Table 2, 5 columns suggest decreasing λ, while 2 columns suggest increasing λ, and 3 columns suggest no change.

Figure 11 and 12 show the contour length results of four contour tracking sessions performed on 26 continuous frames captured under normal lighting condition. Out of the four tracking sessions, one session uses *Algorithm 5* while the remaining sessions use *Algorithm 3* with different threshold values. The values used in these sessions are all shown in Table 3.

The results indicate that the performance of *Algorithm 3* is hardly affected in terms of the contour length when we vary λ between 1.25 and 1.75. Although *Algorithm 5* occasionally performs worse than *Algorithm 3*, in most cases, it outperforms *Algorithm* 3. This improvement is not achieved by introducing additional feature measurement or post-processing, so no significant computation cost is added to the algorithm.

5 Contour Analysis

Recall that the two gestures that are needed for our 3D mouse application are the user's index finger tip and the thumb state. Hence, elaborate features recognition is not necessary. For our purpose, we constrain the user's index finger tip to be farthest from the wrist line (Section 3.2.2). In Twins, the two points that are farthest from the wrist line are tagged as the index finger tip positions corresponding to the "real finger" and "finger's mirror image" respectively.

Thumb state detection is done by measuring finger tip features. We select a specific range within the hand contour as likely positions for the thumb finger tip. This is possible because we already know the index finger tip positions in the contour. Within the specific range, we detect whether the contour bend significantly (corresponding to the tip of the thumb). If such bend is found, we have found the thumb. The bending value can be calculated incrementally.

6 Post-processing

After the finger tips positions have been determined, (6) is used to restore the 3D coordinates of the finger tip. Over time, a stream of 3D coordinate signals is generated from the video stream. This stream of 3D coordinate signals are unstable and succeptable to noise, hence, post-processing is needed to make the 3D mouse practical.

6.1 Screening the False Signals

The first post-processing is to screen out the false signals. After the two index finger tips are detected, we apply the law of physics to derive a constraint for screening out false signals.

In Figure 12, the XYZ coordinate system is set up. R and V correspond to the real finger and its mirror image respectively, and they share a single projection (x',y') in the mirror plane (XOY plane). Let L denote the optical center of the len used in the video camera. The coordinates of L is (x_0,y_0,z_0). Let R' and V' be the image of R and V in the image plane respectively. Then, lines VV' and RR' should pass through L. Since the camera is projecting at an angle of 45° to the mirror, the image plane can be simply specified as:

$$x = z \tag{19}$$

Assume T is a point on the line RV and the coordinates of T is (x', y', h). Then, TL can be specified as:

$$\frac{x - x'}{x_0 - x'} = \frac{y - y'}{y_0 - y'} = \frac{z - h}{z_0 - h} \tag{20}$$

From (19) and (20), we can obtain the position of T's image which is the intersection between TL and the image plane.

$$\begin{cases} x = z = \dfrac{x' z_0 - x_0 h}{z_0 - h - x_0 + x'} \\ y = \dfrac{y' z_0 - y' x_0 + x' y_0 - y_0 h}{z_0 - h - x_0 + x'} \end{cases} \tag{21}$$

In (21), as $h \to -\infty$, we have:

$$\begin{cases} x = x_0 \\ y = y_0 \\ z = x_0 \end{cases} \quad (22)$$

Thus, we can conclude that all straight lines that pass through an object and its mirror image must converge to one point in the image plane. We call this point the *RV center* (Figure 13).

The RV center is used to screen out the false contour analysis results. If the straight line, which passes through the 2 index finger tips (real object and its mirror image), does not pass through the RV center, the signal is considered an invalid signal and will be screened out. In our experiment, less than 5% of the raw signals are screened out by this process. In the event that the signal has been screened out, a "hole" is created in the continuous 3D coordinate stream. Such holes are filled by using the predictions of Kalman filters (see below) and a constant signal stream is still maintained.

6.2 Kalman Filtering and Adaptive Filtering

A set of linear Kalman filters [16] are used both to suppress the Gaussian noise in the 3D co-ordinates signal and make predictions for the holes in the signal. This is the second step of our post-processing task. Preliminary experiments show that the Kalman filtering results are not stable. This is partly due to the fact that the noise in the signal is not purely Gaussian and partly due to our desire to maintain short signal delay. To smooth the signals further, we add a set of adaptive filters. The filtering results are shown in Figure 14.

By combining Kalman filtering and adaptive filtering, we smooth the signals while reducing errors. The post-processing is surprisingly successful in the sense that the outputs are fairly smooth and accurate and the delay caused by the filtering is relatively minimum (between 2 - 4 frames that is 0.08 - 0.16 sec at the 25 frame per sec video rate). The thumb state signal (mouse's switch state) is also filtered by an adaptive filter.

The thumb status signal is also processed by an adaptive filter. Because the signal is interpreted as mouse switch thus more critical and the user is less sensitive to the lag of this signal, we configure the adaptive filter in such a way that the delay is longer (<0.3 sec) but the output is more stable.

7 System performance

The time complexity of Algorithm 5 is O(n), where n is the number of points along the contour. Since the bending value can be calculated incrementally, the complexity of the contour analysis process is O(m), where m is the length of the possible range in the contour where the thumb may occur. It is obvious that m<n. In the post-processing stage, the results of Kalman filter and adaptive filter are both recursively calculated from the previous results and their time complexity is O(1). Hence, the time complexity of the entire system is O(n).

A 3D mouse demon (Figure 15) is built to demonstrate the final performance of Twins on SGI Indy. A 25 frames (640X480) per sec rate has been achieved in the 3D mouse demon. (After the post-processing, the system produces 25 stabilized 3D co-ordinates per sec.) A 3D room with several cubes in are drawn on the screen. The user use her/his index finger tip and thumb to operate the 3D mouse in a 3D room to pick up and place the cubes.

We tested our system against different backgrounds to evaluate the system's sensitivity to background properties. The system displays strong adaptability to different backgrounds. Three backgrounds were used in our experiments as shown in Figure 16, 17 and 18 respectively.

Figure 16a shows a colorful and complex background. Figure 17a shows a normal office background. Figure 18a shows a dark background. The test results demonstrate the strong adaptability of our algorithm. In the experiments, we move the finger tip in a circular motion. The circle movement performed in these tests is 1-2 Hz. The periodical outputs of the 3

coordinates UVW of the finger tip movement is shown in Figure 16b, 17b and 18b respectively. The results indicate that our system can follow the finger tip movement very well and the output signals are smooth in general. We also tested the 3D mouse using a pencil held in a human hand. The system is able to track the pencil just as well as it tracks the finger. This is because we do not make any assumption regarding the shape of the object to be tracked. Hence, our system is robust to fast movement and adaptive to shape changes. In addition, the system does not have lock loss problem. In terms of the gesture recognition rate, we found that in a cluttered environment like Figure 16a, the error rate (before the signals are being post-processed) is less than 5% for the bent thumb gesture and less than 10% for the erected thumb gesture. The results are derived when the hand is moving at a fast pace. For critical signals such as mouse switch states, this error rate is not acceptable. Careful examination shows that the errors occur randomly and are sparsely distributed over time. With the use of an adaptive filter in the post-processing stage, we are able to suppress and screen out all such errors from the final output

7 Conclusions

In this work, we propose a novel low cost solution that uses one camera and a mirror to derive accurate 3D position of a finger tip. Bayesian estimation is used in our algorithm to improve the chain code generation. We have also addressed the chain loop problem in this paper. It is shown that our improved algorithm results in a smoother and more accurate contour tracing. In video analysis, it can be used to track fast moving object over a known complex background where shape information is not available. Using the algorithm, a practical 3D mouse is designed and implemented. Post-processing is done to stabilize the outputs. Experimental results show that our 3D mouse satisfies the requirements for a robust and convenient 3D input device.

REFERENCES

1. K. P. Herndon, A. Dam and M. Gleicher, "The Challenges of 3D Interaction," *SIGCHI Bulletin,* Vol 26, No. 4, 1994, pp. 36-43.
2. J. Foley, et. al, *Computer Graphics: Principles and Practice,* Reading MA: Anddison-Wesley Publishing Company, 1990.
3. T. Baudel and M. Beaudouin-Lafan, "Charade, Remote Control of Objects with Freehand Gestures," *Communications of the ACM,* vol. 36, no. 7, 1993, pp. 28-35.
4. D. Sturman, D. Zeltzer and S. Pieper, "Hands-on Interaction with Virtual Environment," *UIST:Proceedings of the ACM SIGGRAPH Symposium on User Interfaces,* Williamsburg, VA, Nov. 1994, pp. 19-24.
5. A. Blake, M. Isard and D. Reynard, "Learning to Track the Visual Motion of Contours," *Artificial Intelligence,* Vol 78, 1995, pp. 101-133.
6. L. Goncalves and E. D. Bernardo, "Monocular tracking of the human arm in 3D," *Proc. Of 15th International Conference on Computer Vision,* 1995, pp 764-770.
7. J. Kuch and T. Huang, "Virtual Gun: A Vision Based Human Computer Interface Using the Human Hand, *"Proc. IAPR Workshop on Machine Vision Application,* Tokyo, 1994, pp. 196-199.
8. J. Kuch and T. Huang, "Human Computer Interaction via the Human Hand: A Hand Model," *Proc. Of the 28th Asilomar Conference on Signals, Systems & Computers,* 1995, pp. 1252-1256.
9. J. Rehg, "DigitEyes: Vision-Based Human Hand Tracking for Human-Computer Interaction," *Proc. of the 1994 IEEE Workshop on Motion of Non-rigid and Articulated Objects.* 1994, pp.16-22.
10. A. Utsumi, "Real-time Hand Gesture Recognition System," *Proc. of ACCV'95,* 1995, pp. 249-253.
11. M. Kass, A. Witkin, and D. Terzopoulos Snakes: Active Contour Models, *Proc. Ist Int. Conf. on Computer Vision,* 1987, pp. 259-268.

12. C. R Wren., A. Azarbayejani, and A. Pentland, "Pfinder: Real-time Tracking of the Human Body," *Proc. of the 2nd International Conference on Automatic Face and Gesture Recognition*, 1996.
13. Freeman, H., "Computer Processing of Line-drawing Data," *Comput. Surveys*, v. 6, Mar. 1974, pp. 57-59.
14. D. K. Scholten and S. G. Wilson, "Chain Coding with Hexagonal Lattice," *IEEE Trans. PAMI*, No. 5, Sept. 1983, pp. 526-533.
15. C. W. Therrien, *Decision Estimation and Classification*, John Wiley & Sons, US, 1989.
16. Chui, C. K. and Chen, G., *Kalman Filtering with Real-Time Applications*, Springer-Verlag, 1987.

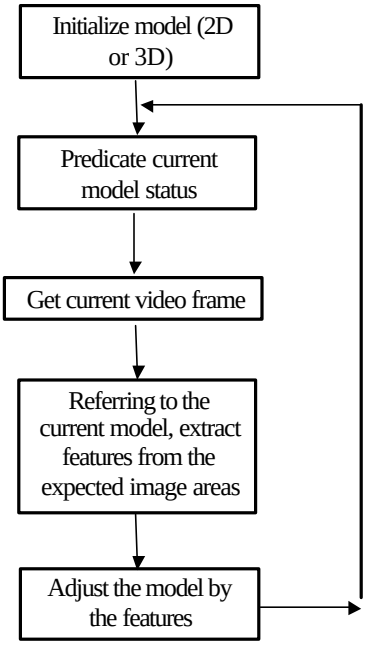

Figure 1. Program structure of model-based approach in video tracking.

Figure 2. Gestures used in the 3D mouse.

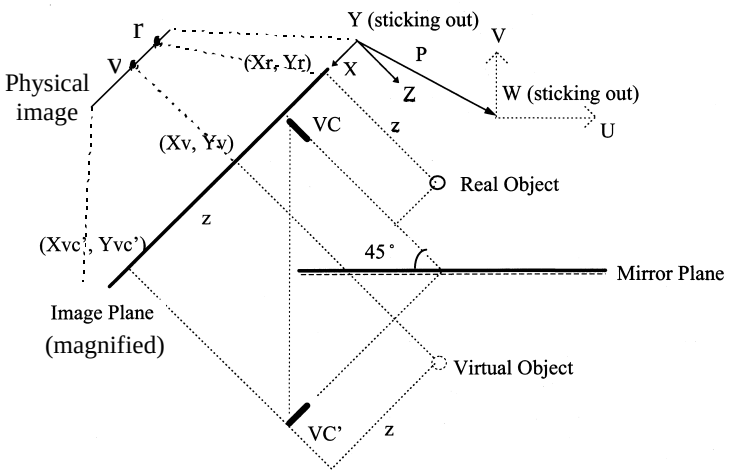

Figure 3. Twins setup arrangement that uses a mirror to provide another view of the object.

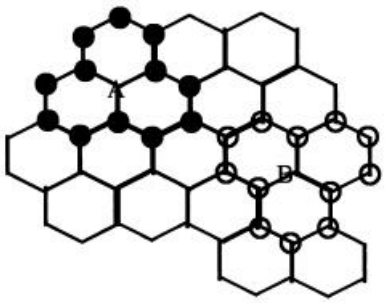

Figure 4. The hexagonal grid is used in the chain code generation.

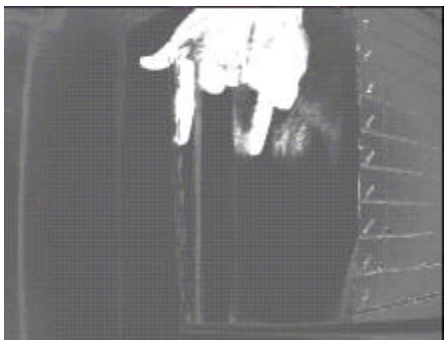

Figure 5. The pixel classification results using hand and background intensity models.

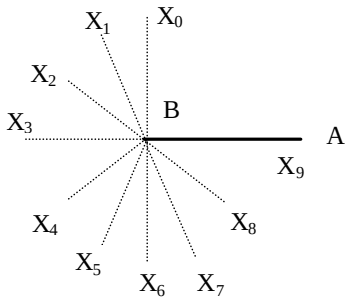

Figure 6. After detecting A and B, there are 10 possible positions for the next point X_0-X_9 in the chain code algorithm and their possibilities are not identical considering the contour should be smooth.

Figure 7. The deflection of the contour at point C_k. C_{k+1} is the 0^{th} neighbor of C_k and C_k is the 11^{th} neighbor of C_{k-1}, thus $M(C_{k+1}, C_k, C_{k-1}) = (0 - 11) \text{ MOD } 12 = -1$. We can see that the vector $C_k\ C_{k+1}$ deflects from the vector $C_k\ C_{k-1}$ with one unit in the clockwise direction.

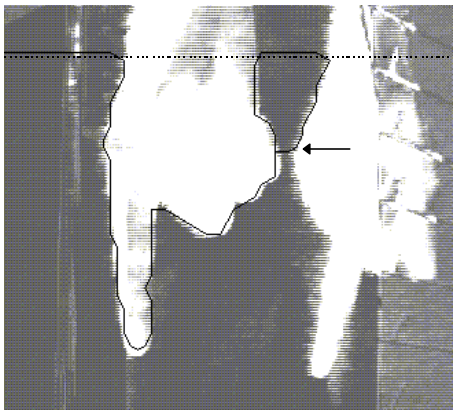

Figure 8. All the pixels above the wrist line (dotted) are classified as hand pixels forcefully. Loop occurs occasionally in the improved algorithm.

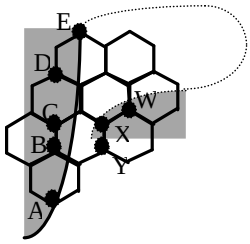

Figure 9. Inconsistent classification of pixel which cause the loop chain code.

Table 1. Values used in the experiment.

λ in *Alg.3*	m_w in *Alg. 5*	ρ in *Alg. 5*
1.5	3.1	6.0

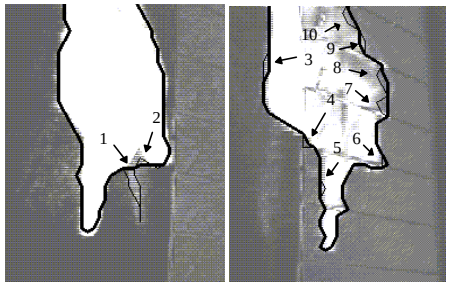

Figure 10. Comparable contour tracking results using traditional chain code generation algorithm (thin line) and our improved algorithm (thick line).

Table 2. The summary of the experiment result.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Contour error*	-	+	0	-	-	-	-	-	+	-
Contour length*	-	-	0	-	-	+	-	-	0	-
l **to be changed****	↑	=	=	↑	↓	↓	↓	↓	=	↓

* +: the thick line has a larger value;
-: the thin line has a larger value;
0: performances are almost equal.

** ↑: λ to be increased;
↓: λ to be decreased;
=: no suggestions to the change.

Table 3 Threshold values used in the tracking sessions

Algorithm 3			*Algorithm 5*	
l **in (1)**	l **in (2)**	l **in (3)**	$\mathbf{m_w}$	r
1.5	1.25	1.75	3.1	6.0

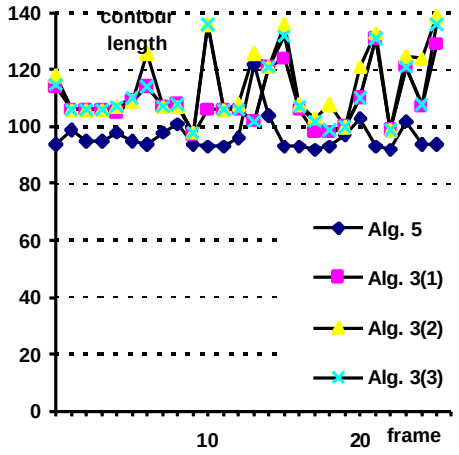

Figure 11. Comparison of the contour length in the number of points in the contour.

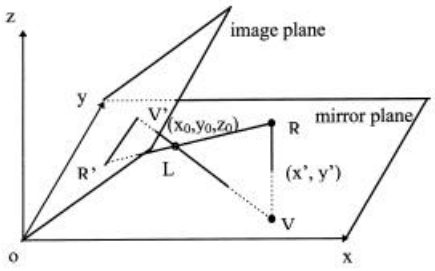

Figure 12. The constraint provided by our equipment setting.

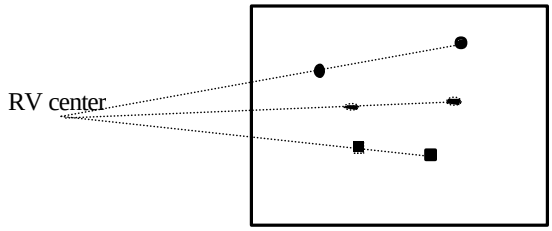

Figure 13. The straight lines passing through a real object and its mirror image converge to one point in the image plane.

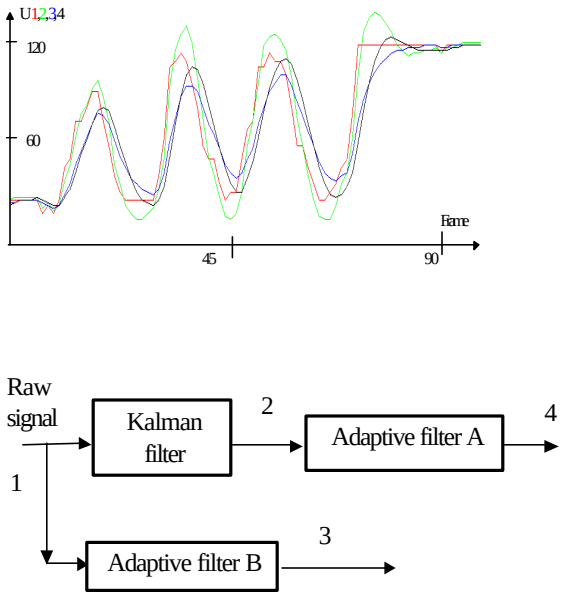

Figure 14. The filtering process and the signals comparison. Kalman filter's output is still unstable while only using adaptive filter causes significant error. Their combination produces better results.

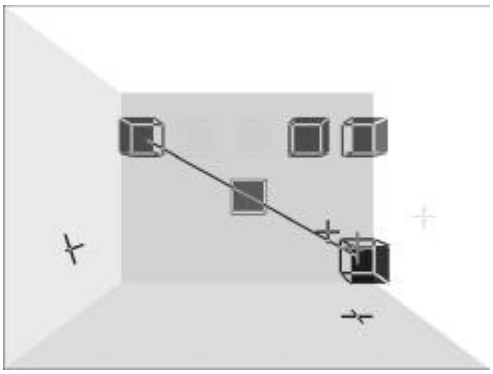

Figure 15. The 3D room demon with cubes in. A user can operate the cross cursor to pick up and place these cubes in the room by his/her index finger tip.

Figure 16a. Background with colorful newspaper.

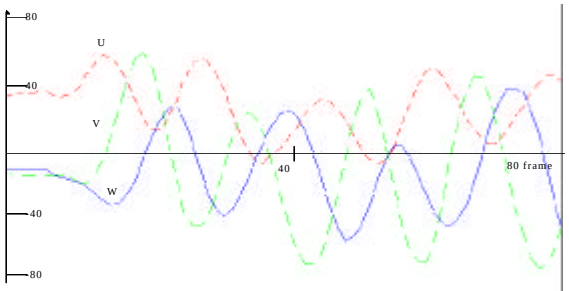

Figure 16b. Output signals of the colorful background experiment.

frame

Figure 17a. Background in an office environment.

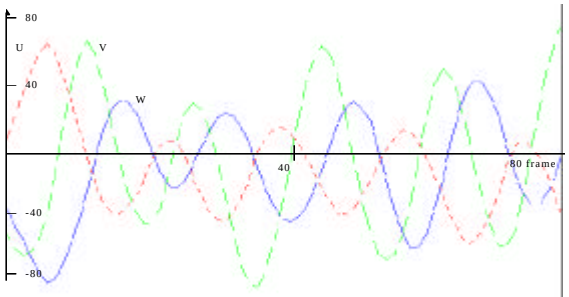

Figure 17b. Output signals of the normal office background experiment.

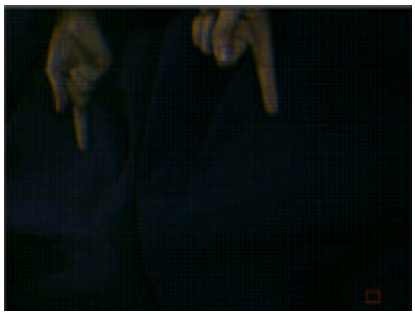

Figure 18a. Dark background.

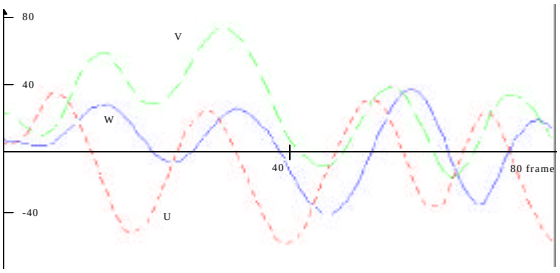

Figure 18b. Output signals of the dark background experiment.