Device Synchronization Using an Optimal Linear Filter

Martin Friedmann, Thad Starner and Alex Pentland [†]

Abstract

In order to be convincing and natural, interactive graphics applications must correctly synchronize user motion with rendered graphics and sound output. We present a solution to the synchronization problem that is based on optimal estimation methods and fixed-lag dataflow techniques. A method for discovering and correcting prediction errors using a generalized likelihood approach is also presented. And finally, MusicWorld, a simulated environment employing these ideas is described.

CR Categories and Subject Descriptors : I.3.6 [Computer Graphics]: Methodology and Techniques - *Interaction Techniques*; D.2.2 [Software Engineering]: Tools and Techniques - *User Interfaces*

Additional Keywords: Real-time graphics, artificial reality, interactive graphics, Kalman filtering, device synchronization.

1 Introduction

In order to be convincing and natural, interactive graphics applications must correctly synchronize user motion with rendered graphics and sound output. The exact synchronization of user motion and rendering is critical: lags greater than 100 msec in the rendering of hand motion can cause users to restrict themselves to slow, careful movements while discrepancies between head motion and rendering can cause motion sickness [3; 5]. In systems that generate sound, small delays in sound output can confuse even practiced users. This paper proposes a suite of methods for accurately predicting sensor position in order to more closely synchronize processes in distributed virtual environments.

Problems in synchronization of user motion, rendering, and sound arise from three basic causes. The first cause is noise in the sensor measurements. The second cause is the length of the processing pipeline, that is, the delay introduced by the sensing device, the CPU time required to calculate the proper response, and the time spent rendering output images or generating appropriate sounds. The third cause is unexpected interruptions such as network contention or operating system activity. Because of these factors, using the raw output of position sensors leads to noticeable lags and other discrepancies in output synchronization.

[†] Vision and Modeling Group, The Media Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139. {martin,testarne,sandy}@media-lab.media.mit.edu

Unfortunately, most interactive systems either use raw sensor positions, or they make an ad-hoc attempt to compensate for the fixed delays and noise. A typical method for compensation averages current sensor measurements with previous measurements to obtain a smoothed estimate of position. The smoothed measurements are then differenced for a crude estimate of the user's instantaneous velocity. Finally, the smoothed position and instantaneous velocity estimates are combined to extrapolate the user's position at some fixed interval in the future.

Problems with this approach arise when the user either moves quickly, so that averaging sensor measurements produces a poor estimate of position, or when the user changes velocity, so that the predicted position overshoots or undershoots the user's actual position. As a consequence, users are forced to make only slow, deliberate motions in order to maintain the illusion of reality.

We present a solution to these problems based on the ability to more accurately predict future user positions using an optimal linear estimator and on the use of fixed-lag dataflow techniques that are well-known in hardware and operating system design. The ability to accurately predict future positions eases the need to shorten the processing pipeline because a fixed amount of "lead time" can be allotted to each output process. For example, the positions fed to the rendering process can reflect sensor measurements one frame ahead of time so that when the image is rendered and displayed, the effect of synchrony is achieved. Consequently, unpredictable system and network interruptions are invisible to the user as long as they are shorter than the allotted lead time.

2 Optimal Estimation of Position and Velocity

At the core of our technique is the optimal linear estimation of future user position. To accomplish this it is necessary to consider the *dynamic* properties of the user's motion and of the data measurements. The Kalman filter [4] is the standard technique for obtaining optimal linear estimates of the state vectors of dynamic models and for predicting the state vectors at some later time. Outputs from the Kalman filter are the maximum likelihood estimates for Gaussian noises, and are the optimal (weighted) least-squares estimates for non-Gaussian noises [2].

In our particular application we have found that it is initially sufficient to treat only the translational components (the x, y, and z coordinates) output by the Polhemus sensor, and to assume independent observation and acceleration noise. In this section, therefore, we will develop a Kalman filter that estimates the position and velocity of a Polhemus sensor for this simple noise model. Rotations will be addressed in the following section.

2.1 The Kalman Filter

Let us define a dynamic process

$$\mathbf{X}_{k+1} = \mathbf{f}(\mathbf{X}_k, \Delta t) + \xi(t) \tag{1}$$

where the function $\mathbf{f}$ models the dynamic evolution of state vector $\mathbf{X}_k$ at time k, and let us define an observation process

$$\mathbf{Y}_k = \mathbf{h}(\mathbf{X}_k, \Delta t) + \eta(t) \tag{2}$$

where the sensor observations $\mathbf{Y}$ are a function $\mathbf{h}$ of the state vector and time. Both ξ and η are white noise processes having known spectral density matrices.

In our case the state vector $\mathbf{X}_k$ consists of the true position, velocity, and acceleration of the Polhemus sensor in each of the x, y, and z coordinates, and the observation vector $\mathbf{Y}_k$ consists of the Polhemus position readings for the x, y, and z coordinates. The function $\mathbf{f}$ will describe the dynamics of the user's movements in terms of the state vector, *i.e.* how the future position in x is related to current position, velocity, and acceleration in x, y, and z. The observation function $\mathbf{h}$ describes the Polhemus measurements in terms of the state vector, *i.e.*, how the next Polhemus measurement is related to current position, velocity, and acceleration in x, y, and z.

Using Kalman's result, we can then obtain the optimal linear estimate $\hat{\mathbf{X}}_k$ of the state vector $\mathbf{X}_k$ by use of the following *Kalman filter*:

$$\hat{\mathbf{X}}_k = \mathbf{X}^*_k + \mathbf{K}_k(\mathbf{Y}_k - \mathbf{h}(\mathbf{X}^*_k, t)) \tag{3}$$

provided that the Kalman gain matrix $\mathbf{K}_k$ is chosen correctly [4]. At each time step k, the filter algorithm uses a state prediction $\mathbf{X}^*_k$, an error covariance matrix prediction $\mathbf{P}^*_k$, and a sensor measurement $\mathbf{Y}_k$ to determine an optimal linear state estimate $\hat{\mathbf{X}}_k$, error covariance matrix estimate $\hat{\mathbf{P}}_k$, and predictions $\mathbf{X}^*_{k+1}$, $\mathbf{P}^*_{k+1}$ for the next time step.

The prediction of the state vector $\mathbf{X}^*_{k+1}$ at the next time step is obtained by combining the optimal state estimate $\hat{\mathbf{X}}_k$ and Equation 1:

$$\mathbf{X}^*_{k+1} = \hat{\mathbf{X}}_k + \mathbf{f}(\hat{\mathbf{X}}_k, \Delta t)\Delta t \tag{4}$$

In our graphics application this prediction equation is also used with larger times steps, to predict the user's future position. This prediction allows us to maintain synchrony with the user by giving us the lead time needed to complete rendering, sound generation, and so forth.

2.1.1 Calculating The Kalman Gain Factor

The Kalman gain matrix $\mathbf{K}_k$ minimizes the error covariance matrix $\mathbf{P}_k$ of the error $\mathbf{e}_k = \mathbf{X}_k - \hat{\mathbf{X}}_k$, and is given by

$$\mathbf{K}_k = \mathbf{P}^*_k \mathbf{H}_k{}^T(\mathbf{H}_k \mathbf{P}^*_k \mathbf{H}_k{}^T - \mathcal{R})^{-1} \tag{5}$$

where $\mathcal{R} = \mathbf{E}[\eta(t)\eta(t)^T]$ is the n x n observation noise spectral density matrix, and the matrix $\mathbf{H}_k$ is the local linear approximation to the observation function $\mathbf{h}$,

$$[\mathbf{H}_k]_{ij} = \partial \mathbf{h}_i / \partial x_j \tag{6}$$

evaluated at $\mathbf{X} = \mathbf{X}^*_k$.

Assuming that the noise characteristics are constant, then the optimizing error covariance matrix $\mathbf{P}_k$ is obtained by solving the *Riccati equation*

$$0 = \dot{\mathbf{P}}^*_k = \mathbf{F}_k \mathbf{P}^*_k + \mathbf{P}^*_k \mathbf{F}^T_k - \mathbf{P}^*_k \mathbf{H}^T_k \mathcal{R}^{-1} \mathbf{H}_k \mathbf{P}^*_k + \mathcal{Q} \tag{7}$$

where $\mathcal{Q} = \mathbf{E}[\xi(t)\xi(t)^T]$ is the n x n spectral density matrix of the system excitation noise ξ, and $\mathbf{F}_k$ is the local linear approximation to the state evolution function $\mathbf{f}$,

$$[\mathbf{F}_k]_{ij} = \partial \mathbf{f}_i / \partial x_j \tag{8}$$

evaluated at $\mathbf{X} = \hat{\mathbf{X}}_k$.

More generally, the optimizing error covariance matrix will vary with time, and must also be estimated. The *estimate* covariance is given by

$$\hat{\mathbf{P}}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}^*_k \tag{9}$$

From this the predicted *error* covariance matrix can be obtained

$$\mathbf{P}^*_{k+1} = \mathbf{\Phi}_k \hat{\mathbf{P}}_k \mathbf{\Phi}^T_k + \mathcal{Q} \tag{10}$$

where $\mathbf{\Phi}_k$ is known as the state transition matrix

$$\mathbf{\Phi}_k = (\mathbf{I} + \mathbf{F}_k \Delta t) \tag{11}$$

2.2 Estimation of Displacement and Velocity

In our graphics application we use the Kalman filter described above for the estimation of the displacements P_x, P_y, and P_z, the velocities V_x, V_y, and V_z, and the accelerations A_x, A_y, and A_z of Polhemus sensors. The state vector $\mathbf{X}$ of our dynamic system is therefore $(P_x, V_x, A_x, P_y, V_y, A_y, P_z, V_z, A_z)^T$, and the state evolution function is

$$\mathbf{f}(\mathbf{X}, \Delta t) = \begin{bmatrix} V_x + A_x \frac{\Delta t}{2} \\ A_x \\ 0 \\ V_y + A_y \frac{\Delta t}{2} \\ A_y \\ 0 \\ V_z + A_z \frac{\Delta t}{2} \\ A_z \\ 0 \end{bmatrix} \tag{12}$$

The observation vector $\mathbf{Y}$ will be the positions $\mathbf{Y} = (P'_x, P'_y, P'_z)^T$ that are the output of the Polhemus sensor. Given a state vector $\mathbf{X}$ we predict the measurement using simple second order equations of motion:

$$\mathbf{h}(\mathbf{X}, \Delta t) = \begin{bmatrix} P_x + V_x \Delta t + A_x \frac{\Delta t^2}{2} \\ P_y + V_y \Delta t + A_y \frac{\Delta t^2}{2} \\ P_z + V_z \Delta t + A_z \frac{\Delta t^2}{2} \end{bmatrix} \tag{13}$$

Calculating the partial derivatives of Equations 6 and 8 we obtain

$$\mathbf{F} = \begin{bmatrix} 0 & 1 & \frac{\Delta t}{2} & & & & & & \\ & 0 & 1 & & & & & & \\ & & 0 & & & & & & \\ & & & 0 & 1 & \frac{\Delta t}{2} & & & \\ & & & & 0 & 1 & & & \\ & & & & & 0 & & & \\ & & & & & & 0 & 1 & \frac{\Delta t}{2} \\ & & & & & & & 0 & 1 \\ & & & & & & & & 0 \end{bmatrix} \tag{14}$$

and

$$\mathbf{H} = \begin{bmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} & & & & & & \\ & & & 1 & \Delta t & \frac{\Delta t^2}{2} & & & \\ & & & & & & 1 & \Delta t & \frac{\Delta t^2}{2} \end{bmatrix} \tag{15}$$

Finally, given the state vector $\mathbf{X}_k$ at time k we can predict the Polhemus measurements at time $k + \Delta t$ by

$$\mathbf{Y}_{k+\Delta t} = \mathbf{h}(\mathbf{X}_k, \Delta t) \tag{16}$$

and the predicted state vector at time $k + \Delta t$ is given by

$$\hat{\mathbf{X}}_{k+\Delta t} = \mathbf{X}^*_k + \mathbf{f}(\hat{\mathbf{X}}_k, \Delta t)\Delta t \tag{17}$$

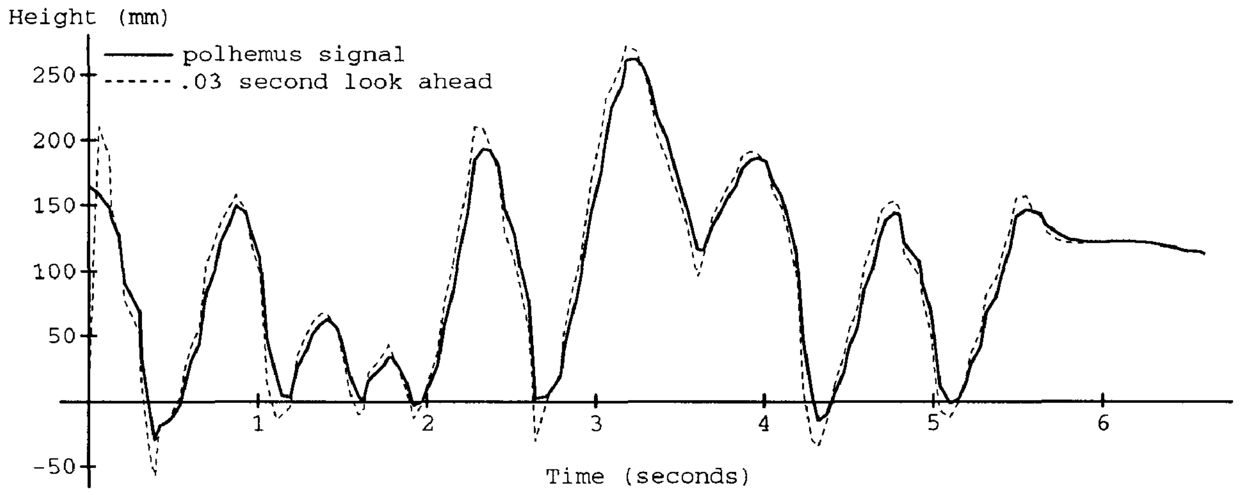

Figure 1: Output of a Polhemus sensor and the Kalman filter prediction of that output for a lead time of 1/30th of a second.

2.2.1 The Noise Model

We have experimentally developed a noise model for user motions. Although our noise model is not verifiably optimal, we find the results to be quite sufficient for a wide variety of head and hand tracking applications. The system excitation noise model ξ is designed to compensate for large velocity and acceleration changes; we have found

$$\xi(t)^T = \begin{bmatrix} 1 & 20 & 63 & 1 & 20 & 63 & 1 & 20 & 63 \end{bmatrix} \quad (18)$$

(where $\mathcal{Q} = \xi(t)\xi(t)^T$) provides a good model. In other words, we expect and allow for positions to have a standard deviation of $1mm$, velocities $20mm/sec$ and accelerations $63mm/sec^2$. The observation noise is expected to be much lower than the system excitation noise. The spectral density matrix for observation noise is $\mathcal{R} = \eta(t)\eta(t)^T$; we have found that

$$\eta(t)^T = \begin{bmatrix} .25 & .25 & .25 \end{bmatrix} \quad (19)$$

provides a good model for the Polhemus sensor.

2.3 Experimental Results and Comparison

Figure 1 shows the raw output of a Polhemus sensor attached to a drumstick playing a musical flourish, together with the output of our Kalman filter predicting the Polhemus's position $1/30th$ of a second in the future.

As can be seen, the prediction is generally quite accurate. At points of high acceleration a certain amount of overshoot occurs; such problems are intrinsic to any prediction method but can be minimized with more complex models of the sensor noise and the dynamics of the user's movements.

Figure 2 shows a higher-resolution version of the same Polhemus signal with the Kalman filter output overlayed. Predictions for 1/30, 1/15, and 1/10 of a second in the future are shown. For comparison, Figure 3 shows the performance of the prediction made from simple smoothed local position and velocity, as described in the introduction. Again, predictions for 1/30, 1/15, and 1/10 of a second in the future are shown. As can be seen, the Kalman filter provides a more reliable predictor of future user position than the commonly used method of simple smoothing plus velocity prediction.

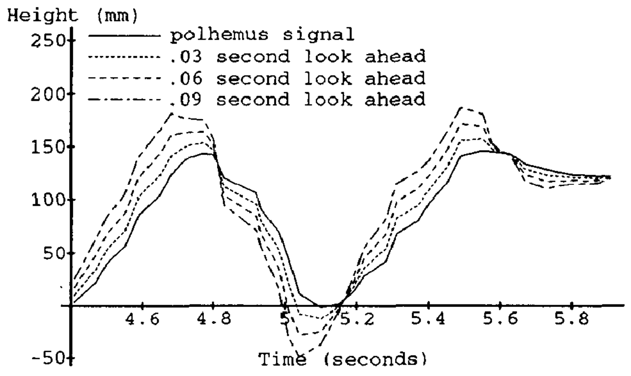

Figure 2: Output of Kalman filter for various lead times

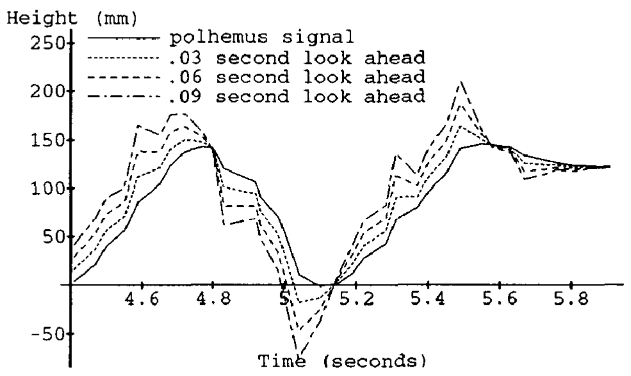

Figure 3: Output of commonly used velocity prediction method.

3 Rotations

With the Polhemus sensor, the above scheme can be directly extended to filter and predict Euler angles as well as translations.

However with some sensors it is only possible to read out instant-by-instant *incremental rotations*. In this case the absolute rotational state must be calculated by integration of these incremental rotations, and the Kalman filter formulation must altered as follows [1]. See also [6].

Let ρ be the incremental rotation vector, and denote the rotational velocity and acceleration by ϑ and α. The rotational acceleration vector α is the derivative of ϑ which is, in turn, the derivative of ρ, but only when two of the components ρ are exactly zero (in some frame to which both ρ and ϑ are referenced). For sufficiently small rotations about at least two axes, ϑ is approximately the time derivative of ρ.

For 3D tracking one cannot generally assume small absolute rotations, so an additional representation of rotation, the unit quaternion $\overset{\circ}{\mathbf{q}}$ and its rotation submatrix $\mathbf{R}$, is employed. Let

$$\overset{\circ}{\mathbf{q}} = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix}, \tag{20}$$

be the unit quaternion. Unit quaternions can be used to describe the rotation of a vector $\mathbf{v}$ through an angle ϕ about an axis $\hat{\mathbf{n}}$, where $\hat{\mathbf{n}}$ is a unit vector. The unit quaternion associated with such a rotation has *scalar part*

$$q_0 = \sin(\phi/2) \tag{21}$$

and vector part

$$\begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} = \hat{\mathbf{n}} \cos(\phi/2). \tag{22}$$

Note that every quaternion defined this way is a unit quaternion.

By convention $\overset{\circ}{\mathbf{q}}$ is used to designate the rotation between the global and local coordinate frames. The definition is such that the orthonormal matrix

$$\mathbf{R} = \tag{23}$$
$$\begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2+q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix}$$

transforms vectors expressed in the local coordinate frame to the corresponding vectors in the global coordinate frame according to

$$\mathbf{v}_{global} = \mathbf{R}\mathbf{v}_{local}. \tag{24}$$

In dealing with incremental rotations, the model typically assumes that accelerations are an unknown "noise" input to the system, and that the time intervals are small so that the accelerations at one time step are close to those at the previous time step. The remaining states result from integrating the accelerations, with corrupting noise in the integration process.

The assumption that accelerations and velocities can be integrated to obtain the global rotational state is valid only when ρ_k is close to zero and ρ_{k+1} remains small. The latter condition is guaranteed with a sufficiently small time step (or sufficiently small rotational velocities). The condition $\rho_k = 0$ is established at each time step by defining ρ to be a correction to a nominal (absolute) rotation, which is maintained externally using a unit quaternion $\overset{\circ}{\mathbf{q}}$ that is updated at each time step.

4 Unpredictable Events

We have tested our Kalman filter synchronization approach using a simulated musical environment (described below) in which we track a drumstick and simulate the sounds of virtual drums. For smooth motions, the drumstick position is accurately predicted, so that sound, sight, and motion are accurately synchronized, and the user experiences a strong sense of reality.

The main difficulties that arise with this approach derive from unexpected large accelerations, which produce overshoots and similar errors. It is important to note, however, that overshoots are *not* a problem as long the drumstick is far from the drum. In these cases the overshoots simply exaggerate the user's motion, and the perception of synchrony persists. In fact, such overshoots seem generally to enhance, not degrade, the user's impression of reality.

The problem occurs when the predicted motion overshoots the true motion when the drumstick is near the drumhead, thus causing a false collision. In this case the system generates a sound when in fact no sound should occur. Such errors detract noticeably from the illusion of reality.

4.1 Correcting Prediction Errors

How can we preserve the impression of reality in the case of an overshoot causing an incorrect response? In the case of simple responses like sound generation, the answer is easy. When we detect that the user has changed direction unexpectedly — that is, that an overshoot has occurred — then we simply send an emergency message aborting the sound generation process. As long as we can detect that an overshoot has occurred before the sound is "released," there will be no error.

This solution can be implemented quite generally, but it depends critically upon two things. The first is that we must be able to very quickly substitute the correct response for the incorrect response. The second is that we must be able to accurately detect that an overshoot has occurred.

In the case of sound generation due to an overshoot, it is easy to substitute the correct response for the incorrect, because the correct response is to do nothing. More generally, however, when we detect that our motion prediction was in error we may have to perform some quite complicated alternative response. To maintain synchronization, therefore, we must be able to detect possible trouble spots beforehand, and begin to compute all of the alternative responses sufficiently far ahead of time that they will be available at the critical instant.

The strategy, therefore, is to predict user motion just as before, but that at critical junctures to compute several alternative responses rather than a single response. When the instant arrives that a response is called for, we can then choose among the available responses.

4.2 Detecting Prediction Errors

Given that we have computed alternative responses ahead of time, and that we can detect that a prediction error has occurred, then we can make the correct response. But how are we to detect which of (possibly many) alternative responses are to be executed?

The key insight to solving this detection problem is that *if* we have the correct dynamic model then we will always have an optimal linear estimate of the drumstick position, and there should be nothing much better that we can to do. The problem, then, is that in some cases our model of the event's dynamics does not match the true dynamics. For instance, we normally expect accelerations to be small and uncorrelated with position. However in some cases (for instance, when sharply changing the pace of a piece of music) a drummer will apply large accelerations that are exactly correlated with position.

The solution is to have *several* models of the drummer's dynamics running in parallel, one for each alternative response. Then at each instant we can observe the drumstick position and velocity, decide which model applies, and then make our response based on that model. This is known as the *multiple model* or *generalized likelihood* approach, and produces a generalized maximum likelihood estimate of the current and future values of the state variables [10]. Moreover, the cost of the Kalman filter calculations is sufficiently small to make the approach quite practical.

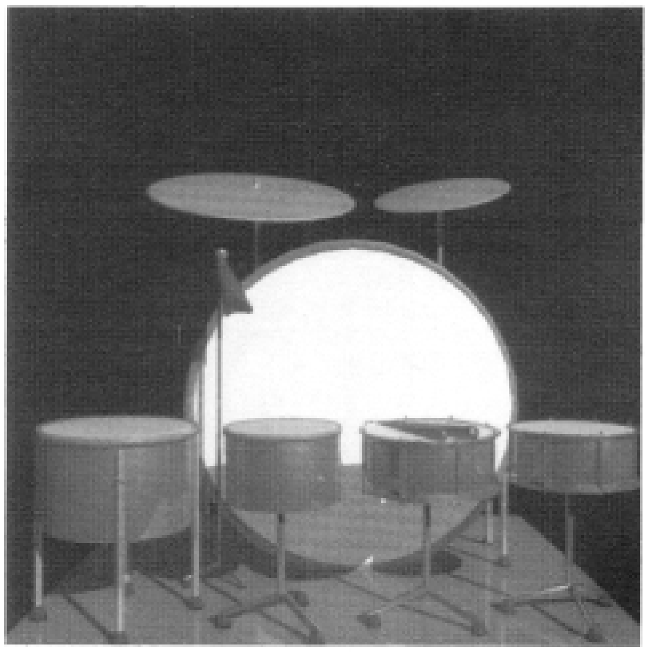

Figure 4: MusicWorld's drum kit.

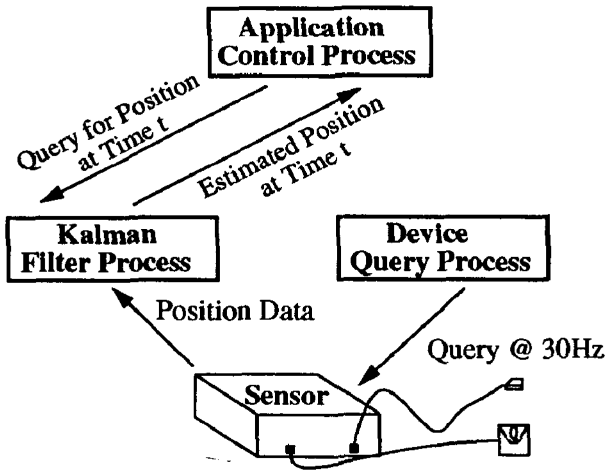
Application Control Process
Query for Position at Time t
Estimated Position at Time t
Kalman Filter Process
Device Query Process
Position Data
Query @ 30Hz
Sensor

Figure 5: Communications used for control and filtering of Polhemus sensor.

Intuitively, this solution breaks the drummer's overall behavior down into several "prototypical" behaviors. For instance, we might have dynamic models corresponding to a relaxed drummer, a very "tight" drummer, and so forth. We then classify the drummer's behavior by determining which model best fits the drummer's observed behavior.

Mathematically, this is accomplished by setting up one Kalman filter for the dynamics of each model:

$$\hat{\mathbf{X}}_k^{(i)} = \mathbf{X}_k^{*(i)} + \mathbf{K}_k^{(i)}(\mathbf{Y}_k - \mathbf{h}^{(i)}(\mathbf{X}_k^{*(i)}, t)) \qquad (25)$$

where the superscript (i) denotes the i^{th} Kalman filter. The *measurement innovations process* for the i^{th} model (and associated Kalman filter) is then

$$\Gamma_k^{(i)} = \mathbf{Y}_k - \mathbf{h}^{(i)}(\mathbf{X}_k^{*(i)}, t) \qquad (26)$$

The measurement innovations process is zero-mean with covariance $\mathcal{R}$.

The i^{th} measurement innovations process is, intuitively, the part of the observation data that is unexplained by the i^{th} model. The model that explains the largest portion of the observations is, of course, the most model likely to be correct. Thus at each time step calculate the probability $P^{(i)}$ of the m-dimensional observations $\mathbf{Y}_k$ given the i^{th} model's dynamics,

$$P^{(i)}(\mathbf{Y}_k) = \frac{1}{(2\pi)^{m/2}\mathrm{Det}(\mathcal{R})^{1/2}} \exp\left(-\frac{1}{2}\Gamma_k^{(i)T}\mathcal{R}^{-1}\Gamma_k^{(i)}\right) \qquad (27)$$

and choose the model with the largest probability. This model is then used to estimate the current value of the state variables, to predict their future values, and to choose among alternative responses.

When optimizing predictions of measurements Δt in the future, equation 26 must be modified slightly to test the predictive accuracy of state estimates from Δt in the past.

$$\Gamma_k^{(i)} = \mathbf{Y}_k - \mathbf{h}^{(i)}(\mathbf{X}_{k-\Delta t}^{*(i)} + \mathbf{f}^{(i)}(\hat{\mathbf{X}}_{k-\Delta t}^{(i)}, \Delta t)\Delta t, t)) \qquad (28)$$

by substituting equation 17.

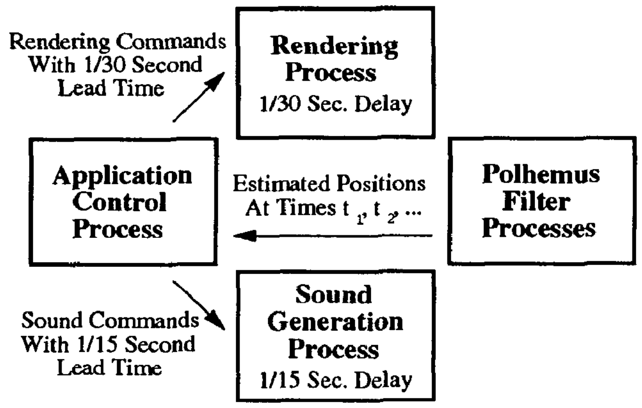
Rendering Commands With 1/30 Second Lead Time
Rendering Process
1/30 Sec. Delay
Application Control Process
Estimated Positions At Times t 1, t 2 ...
Polhemus Filter Processes
Sound Commands With 1/15 Second Lead Time
Sound Generation Process
1/15 Sec. Delay

Figure 6: Communications and lead times for MusicWorld processes.

5 MusicWorld

Our solution is demonstrated in a musical virtual reality, an application requiring synchronization of user, physical simulation, rendering, and computer-generated sound. This system is called *MusicWorld*, and allows users to play a virtual set of drums, bells, or strings with two drumsticks controlled by Polhemus sensors. As the user moves a physical drumstick the corresponding rendered drumstick tracks accordingly. The instant the rendered drumstick strikes a drum surface a sound generator produces the appropriate sound for that drum. The visual appearance of MusicWorld is shown in Figure 4, and a higher quality rendition is included in the color section of these proceedings.

Figure 5 shows the processes and communication paths used to filter and query each Polhemus sensor. Since we cannot insure that the application control process will query the Polhemus devices on a regular basis, and since we do not want the above Kalman loop to enter into the processing pipeline, we spawn two small processes to constantly query and filter the actual device. The application control process then, at any time, has the opportunity to make a fast query to the filter process for the most up to date, filtered, polhemus position. Using shared-memory between these two processes makes the final queries fully optimal.

MusicWorld is built on top of the ThingWorld system [7; 8], which has one process to handle the problems of real-time physical simulation and contact detection and a second process to handle rendering. Sound generation is handled by a third process on a separate host, running CSound [9]. Figure 6 shows the communication network for MusicWorld, and the lead times employed.

The application control process queries the Kalman filter process for the predicted positions of each drumstick at $1/15$ and $1/30$ of a second. Two different predictions are used, one for each output device. The $1/15$ of a second predictions are used for sound and are sent to ThingWorld to detect stick collisions with drums and other sound generating objects. When future collisions are detected, sound commands destined for $1/15$ of a second in the future are sent to CSound. Regardless of collisions and sounds, the scene is always rendered using the positions predicted at $1/30$ of a second in the future, corresponding to the fixed lag in our rendering pipeline. In general, it would be more optimal to constantly check and update the lead times actually needed for each output process, to insure that dynamic changes in network speeds, or in the complexity of the scene (rendering speeds) do not destroy the effects of synchrony.

6 Summary

The unavoidable processing delays in computer systems mean that synchronization of graphics and sound with user motion requires prediction of the user's future position. We have shown how to construct the optimal linear filter for estimating future user position, and demonstrated that it gives better performance than the commonly used technique of position smoothing plus velocity prediction. The ability to produce accurate predictions can be used to minimize unexpected delays by using them in a system of multiple asynchronous processes with known, fixed lead times. Finally, we have shown that the combination of optimal filtering and careful construction of system communications can result in a well-synchronized, multi-modal virtual environment.

7 Acknowledgements

This research was made possible by ARO Grant No. DAAL03-87-K-0005. First thanks go to the ACM for this publication. Thanks are due to Barry Vercoe and Mike Hawley for their help with CSOUND. Thanks! Special mention to Ali Azarbayejani, our newest member. And last but not least, shouts go out to the rest of the real-time programming posse: Irfan Essa, Bradley Horowitz and Stan Sclaroff.

References

[1] Azarbayejani, Ali. *Model-Based Vision Navigation for a Free-Flying Robot.* Masters Thesis, M.I.T. Dept. of Aero. and Astro. (1991).

[2] Friedland, Bernard. *Control System Design.* McGraw-Hill, (1986).

[3] Held, Richard. Correlation and decorrelation between visual displays and motor output. In *Motion sickness, visual displays, and armored vehicle design*, (pp. 64-75). Aberdeen Proving Ground, Maryland: Ballistic Research Laboratory. (1990).

[4] Kalman, R. E. & Bucy, R. S. New results in linear filtering and prediction theory. In *Transaction ASME (Journal of basic engineering)*, 83D, 95-108. (1961).

[5] Oman, Charles M. Motion sickness: a synthesis and evaluation of the sensory conflict theory. In *Canadian Journal of Physiology and Pharmacology*, 68, 264-303. (1990).

[6] Liang, Jiandong. Shaw, Chris & Green, Mark. On temporal-spatial realism in the virtual reality environment. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 19-25, Hilton Head SC. (1991).

[7] Pentland, Alex & Williams, John. Good vibrations: Modal dynamics for graphics and animation. In *Computer Graphics* 23, 4, pp. 215-222. (1989).

[8] Pentland, A., Friedmann, M., Horowitz, B., Sclaroff, S. & Starner, T. The ThingWorld modeling system. In E.F. Deprettere, (Ed.). *Algorithms and parallel VLSI architectures*, Amsterdam : Elsevier. (1990).

[9] Vercoe, Barry & Ellis, Dan. Real-time CSOUND: Software synthesis with sensing and control. In *ICMC Glasgow 1990 Proceedings*, pp. 209-211. (1990).

[10] Willsky, Alan S.. Detection of Abrupt Changes in Dynamic Systems. In M. Basseville and A. Benveniste, (Ed.). *Detection of Abrupt Changes in Signals and Dynamical Systems*, Lecture Notes in Control and Information Sciences, No. 77, pp. 27-49, Springer-Verlag. (1986).

Color Plate 1: Color detail of MusicWorld. "Device Synchronization Using an Optimal Linear Filter" Martin Friedmann, Thad Starner & Alex Pentland.