

Synchronization in Virtual Realities

M. Friedmann, T. Starner, A. Pentland

Vision and Modeling Group, The Media Lab, Massachusetts Institute of Technology
Room E15-389, 20 Ames St., Cambridge MA 02139 ¹

ABSTRACT: Interactive graphics, and especially virtual reality systems, require synchronization of sight, sound, and user motion if they are to be convincing and natural. We present a solution to the synchronization problem that is based on optimal estimation methods.

1 Introduction

The core of most virtual reality systems is the ability to closely map sensed user motion to rendering parameters. 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 [2, 4]. In systems that also generate sound, small delays in sound output can confuse even practiced users. This paper proposes a method of 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.

Unfortunately, most virtual reality 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 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 the virtual reality.

We present a solution to these problems based on the ability to more accurately predict future

¹This research was made possible by ARO Grant No. DAAL03-87-K-0005. Great thanks are due Barry Vercoe and Mike Hawley for their help with CSound, as well as to Ali Azabaryejani for bringing this Kalman filter notation to our attention.

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 [3] 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 estimate for Gaussian noises, and are the optimal (weighted) least-squares estimate for non-Gaussian noises [1].

In our system we have found that it is sufficient to treat only the translational components (the x , y , and z coordinates) output by the Polhemus sensor², 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. Generalizations to rotational measurements and more complex models of noise are straightforward, following the framework described in [1], but are quite lengthy and are omitted in the interest of compactness. A detailed treatment of rotation with all relevant equations can be found in [8].

2.1 The Kalman Filter

Let us define a dynamic process

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

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

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

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*:

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

²It should be noted that this assumption is not valid for head rotations, which are far more important than head translations for view changes

provided that the Kalman gain matrix \mathbf{K}_k is chosen correctly [3]. 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:

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

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

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

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

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

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*

$$0 = \dot{\mathbf{P}}_k^* = \mathbf{F}_k \mathbf{P}_k^* + \mathbf{P}_k^* \mathbf{F}_k^T - \mathbf{P}_k^* \mathbf{H}_k^T \mathbf{R}^{-1} \mathbf{H}_k \mathbf{P}_k^* + \mathbf{Q} \quad (7)$$

where $\mathbf{Q} = \mathbf{E}[\xi(t)\xi(t)^T]$ is the $n \times n$ 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} ,

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

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

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

From this the predicted error covariance matrix can be obtained

$$\mathbf{P}_{k+1}^* = \Phi_k \hat{\mathbf{P}}_k \Phi_k^T + \mathbf{Q} \quad (10)$$

where Φ_k is known as the state transition matrix

$$\Phi_k = (\mathbf{I} + \mathbf{F}_k \Delta t) \quad (11)$$

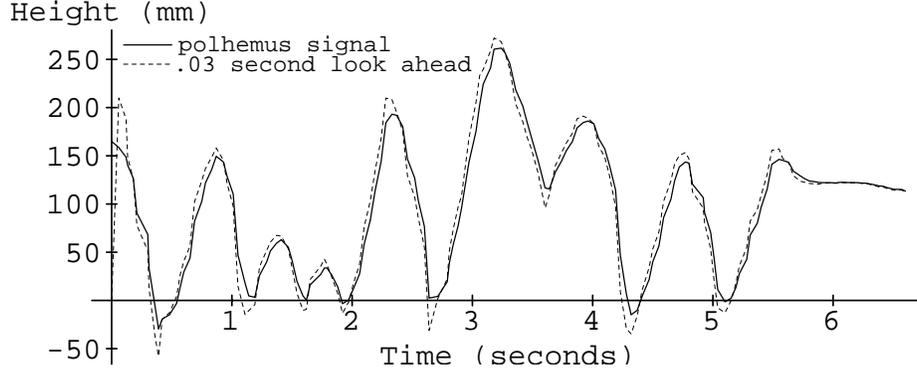


Figure 1: Output of a Polhemus sensor and the Kalman filter prediction of that output for a lead time of 1/30th of a second.

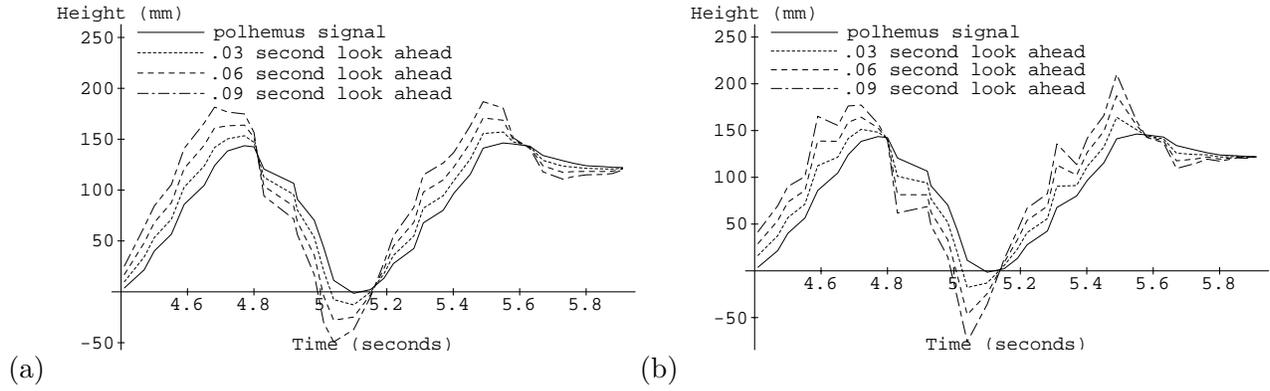


Figure 2: (a) Output of Kalman filter for various lead times, (b) output of commonly used velocity prediction method.

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 hand and head 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 $\mathbf{Q} = \xi(t)\xi(t)^T$) 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 $\mathbf{R} = \eta(t)\eta(t)^T$; we have found that

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

provides a good model for the Polhemus sensor.

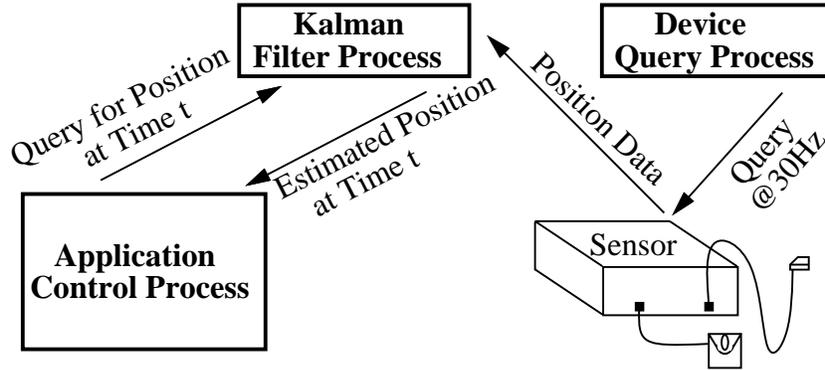


Figure 3: Communications used for control and filtering of Polhemus sensor

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(a) shows a higher-resolution version of the same Polhemus signal with the Kalman filter output overlaid. Predictions for $1/30$, $1/15$, and $1/10$ of a second in the future are shown. For comparison, Figure 2(b) 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.

3 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. When the rendered drumstick strikes a drum surface the sound generator produces the appropriate sound for that drum. The visual appearance of MusicWorld is shown in Figure 4(a).

Figure 3 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

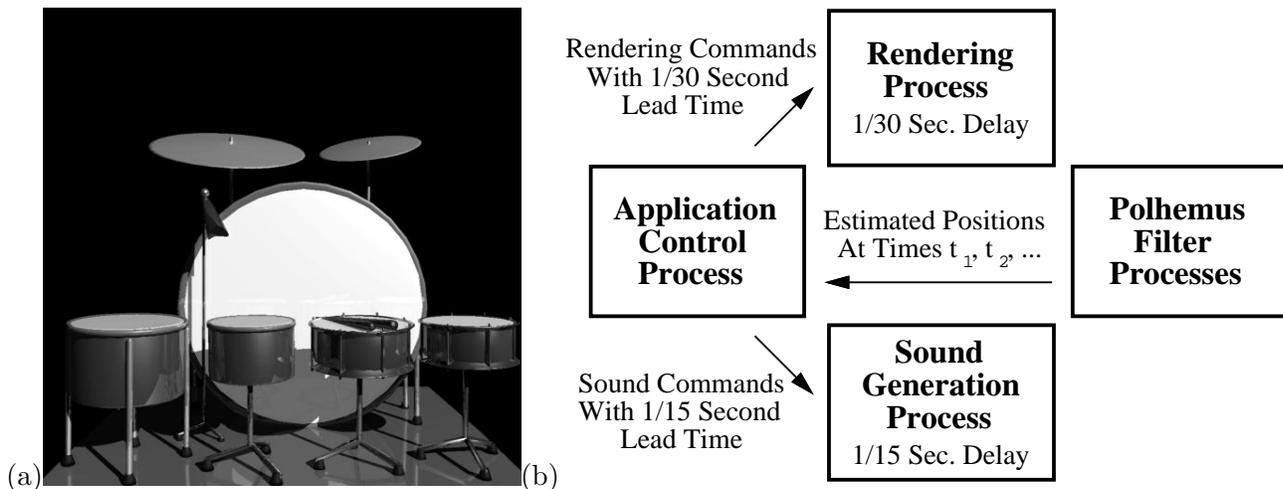


Figure 4: (a) MusicWorld, a musical virtual reality, (b) Communications and lead times for MusicWorld processes.

two processes makes the final queries fully optimal.

MusicWorld is built on top of the ThingWorld system [5, 6], 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 [7]. Figure 4(b) 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.

4 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 estimation of 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.

References

- [1] Friedland, B. (1986). Control system design. McGraw-Hill, 1986.
- [2] Held, R. (1990). Correlation and decorrelation between visual displays and motor output. Motion sickness, visual displays, and armored vehicle design, (pp. 64-75). Aberdeen Proving Ground, Maryland: Ballistic Research Laboratory.
- [3] Kalman, R. E. & Bucy, R. S. (1961). New results in linear filtering and prediction theory. Transaction ASME (Journal of basic engineering), 83D, 95-108.
- [4] Oman, C. M. (1990). Motion sickness: a synthesis and evaluation of the sensory conflict theory. Canadian Journal of Physiology and Pharmacology, 68, 264-303.
- [5] Pentland, A. P. & Williams, J. R. (1989). Good vibrations: Modal dynamics for graphics and animation. ACM Computer Graphics, 23(4), 215-222.
- [6] Pentland, A., Friedmann, M., Horowitz, B., Sclaroff, S. & Starner, T. (1990). The ThingWorld modeling system. In E.F. Deprettere, (Ed.). Algorithms and parallel VLSI architectures, Amsterdam : Elsevier.
- [7] Vercoe, B. & Ellis, D. (1990). Real-time CSOUND: Software synthesis with sensing and control. ICMC Glasgow 1990 Proceedings, 209-211.
- [8] Wu, J., Rink, R., Caelli, T., & Gourishankar, V. (1989). Recovery of the 3-D location and motion of a rigid object through camera image (An extended Kalman approach) Int'l Journal of Computer Vision, 2, 373-394.