Variability-Aware Latency Amelioration in Distributed Environments

Alexey Tumanov* Robert Allison† Wolfgang Stuerzlinger‡

Dept. of Computer Science & Engineering and the Centre for Vision Research
York University, Toronto, Ontario, Canada.

Abstract

Application designers of collaborative distributed Virtual Environments must account for the influence of the network connection and its detrimental effects on user performance. Based upon analysis and classification of existing latency compensation techniques, this paper introduces a novel approach to latency amelioration in the form of a two-tier predictor-estimator framework. The technique is variability-aware due to its proactive sender-side prediction of a pose a variable time into the future. The prediction interval required is estimated based on current and past network delay characteristics. This latency estimate is subsequently used by a Kalman Filter-based predictor to replace the measurement event with a predicted pose that matches the event's arrival time at the receiving workstation. The compensation technique was evaluated in a simulation through an offline playback of real head motion data and network delay traces collected under a variety of real network conditions. The experimental results indicate that the variability-aware approach significantly outperforms a state-of-the-art one, which assumes a constant system delay.

Keywords: virtual environment, network delay, jitter

1 Introduction

Virtual Environments (VE) allow users to gain a sense of immersion into a synthetic reality that is populated with manipulatable objects and can be navigated, making it possible to experience a modeled world from an egocentric perspective [21]. Distributed Virtual Reality (VR) applications offer the ability for multiple people to collaborate in a virtual environment, effectively providing a shared synthetic reality. Normally, participants are geographically separated and the whole system is therefore called a Distributed Interactive Virtual Environment (DIVE).

Despite substantial advances, the main limiting factors for DIVEs are still the limitations of current network technology. The interactive nature of DIVE applications necessitates maintaining a spatio-temporally consistent state of the shared environment as well as the objects and autonomous or human operated avatars that inhabit it. Non-deterministic state changes must be communicated to other DIVE nodes explicitly or implicitly via commands that bring those changes about.

Ironically, the networks that enable DIVEs are also a source of virtual environment state inconsistencies [10]. More specifically, network propagation delay, defined as the amount of time it takes for a packet dispatched by the sender to reach the receiver's computer at the application layer level, and even more so the *non-deterministic variability* remain to this day the main problem of real-time distributed applications.

*e-mail: atumanov@cse.yorku.ca
†http://www.cse.yorku.ca/∼allison
‡http://www.cse.yorku.ca/∼wolfgang

1.1 Motivation

End-to-end latency leads to a number of serious anomalies. It takes a toll on human performance in cooperative tele-operation tasks, as demonstrated by Park and Kenyon [20]. In their task, one of the users controlled a virtual rod, while the other manipulated a virtual ring. Their common goal was to transfer the ring from one end of the rod to the other with the minimum number of object collisions as quickly as possible. The authors concluded that participants adopt a move-and-wait strategy to synchronize their movements, consequently increasing the total time required to complete the task. This study, as well as other similar research [3] converge on the conclusion that, in addition to increased time-to-completion, task accuracy also suffers in setups with larger latencies.

Furthermore, latency can result in oscillopsia, which is the perception that the visual world appears to swim about or oscillate in space [2]. Such perceptual instability has been identified as a major cause of what has become known as cybersickness [13]. Cybersickness refers to "sensations of nausea, oculomotor disturbances, disorientation, and other adverse effects associated with VE exposure" [21]. Finally, latency has been shown to result in causal anomalies in multi-operator DIVEs, such as multiplayer games, with "dead man shooting" serving as the classic example [18].

While end-to-end delay has been clearly identified as a major impairment to DIVEs, jitter has recently been acknowledged as having an even greater impact [10, 20, 3]. Park and Kenyon [20] conclude that network latency jitter disarms compensatory prediction techniques otherwise available for known constant delay systems. From a human performance perspective, it becomes difficult, if not impossible, to adapt to variable display lag. Such considerations motivate variability-aware latency compensation techniques.

1.2 Our Latency Amelioration Approach

Figure 1 depicts what constitutes a single cycle of a client-server type VE application without prediction. t_s and t_c represent the flow

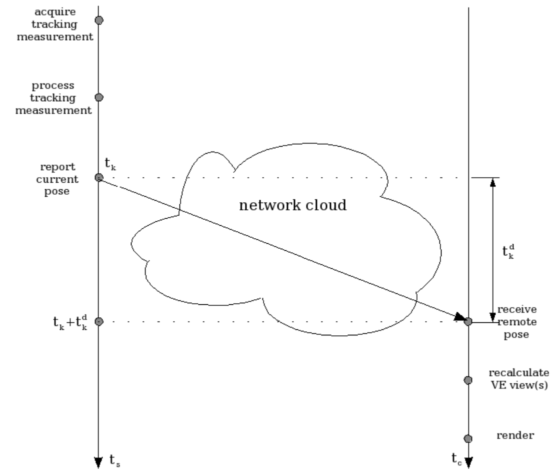

Figure 1: client-server VE application cycle: no prediction

of time, respectively, on the server and client platforms according to a global world clock. Current pose transmission incurs network transport delay t_k^d, where k is the discrete-time independent variable and ^{d} stands for delay. The client receives what is believed to reflect server-controlled entity's pose at time $t_k + t_k^d$, whereas, in fact, the pose is "current" as of $t = t_k$.

Our approach to network delay amelioration rests on the following idea: why broadcast an entity's current pose, if it is already destined to arrive outdated? We leverage a sender's knowledge about its motion profile, event history, and a possible physical model of its motion trajectory. This allows us to address VE state inconsistency by having VE entities exchange their *predicted* pose instead of the currently measured one. This approach departs from the wide-spread method of correcting for the VE state inconsistencies due to network latency *a posteriori*. It conceptually modifies the cycle illustrated above through the introduction of a pose and network delay estimator framework, see fig. 2.

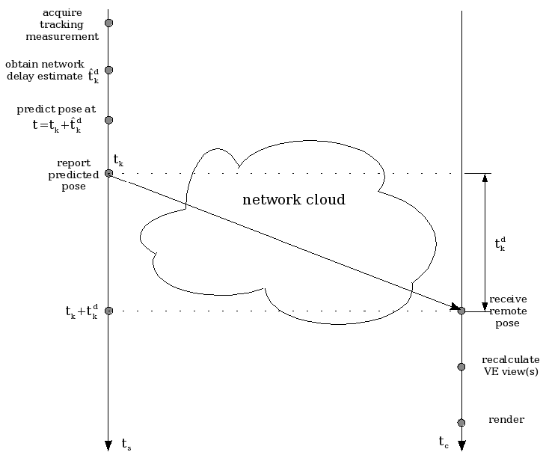

Figure 2: client-server VE application cycle: with prediction

With this modification, clients receive a pose calculated for $t = t_k + \hat{t}_k^d$ at $t = t_k + t_k^d$. Even with a modest performance of network delay estimation, we expect that $\hat{t}_k^d \approx t_k^d$, in which case the predicted pose will closely match the current pose of the server-controlled entity, conditioned upon the pose estimator performance. We accomplish this through the introduction of a variability-aware framework, by coupling a user pose predictor with a network delay estimator to determine the prediction interval required for the former to estimate a future pose at the time of the event's arrival.

Ultimately, our contribution is a two-tier adaptive predictor framework (see fig. 3) which implements a proactive approach to network latency amelioration that is sensitive to the variability in network delays.

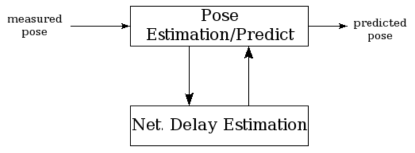

Figure 3: Two-layer latency amelioration framework

1.3 Server-side vs. Client-side Prediction

Alternatively, pose prediction can be carried out at the receiving end - the most widespread approach we classify under reactive compensation in section 2.1. This method would render latency estimation unnecessary, since delay can be measured upon timestamped packet arrival. The primary disadvantage, however, is the need for tight clock synchronization among participating workstations, conditioning the accuracy of delay measurement on that of synchronization over a latency-impaired communication medium. Furthermore, client-side prediction requires smart clients, having access to the motion model & sufficient computational capacity to use it.

Sender-side prediction addresses these shortcomings, providing additional benefits. Security and privacy concerns are eliminated where the knowledge of tracked object's dynamic behavior is valuable intellectual property, and/or certain access privileges (such as security clearance) are required. Clock synchronization becomes unnecessary, as the round-trip time is measured when packet acknowledgments are received. Furthermore, this approach presents clients with a ready to use pose estimate, eliminating the need for smart receivers and additional computational resources. Finally, sender-side latency estimation reduces sensitivity to packet loss. In contrast, client-side prediction may suffer from lack of continuous and prompt pose measurements from remote workstations, whereas senders can rely on access to the complete data stream regardless of network reliability.

2 Delay Amelioration Classification

Network delay has detrimental effects across a whole spectrum of immersive synthetic reality applications. A range of techniques have been promoted to address the problem including attempts to minimize that latency and its variability to active compensation techniques. The classification we introduce separates methods of latency compensation into two major categories — reactive and proactive delay amelioration. The former is characterized by providing algorithmic solutions to a problem once it has already taken place, while the latter acts in anticipation of the expected delay to be incurred in the network. Proactive methods can be further subdivided into jitter-insensitive and variability-aware.

2.1 Reactive Latency Compensation

Prototypical reactive approaches are based on the dead reckoning (DR) algorithm popularized by the well-known IEEE Standard for Distributed Interactive Simulation [11]. Dead reckoning attempts to solve two problems simultaneously: reduce bandwidth consumption and mitigate the effect of network delay. Client-side prediction eliminates the need for a continuous stream of pose updates from the server at the expense of the accuracy of the pose estimated. To impose a cap on the pose misestimation, the same extrapolator is run on the server side to ensure that the discrepancy between its output and the real pose is under a specified threshold. When the threshold is exceeded, the correct pose is dispatched to the client. Dead reckoning is reactive in nature — in essence, it waits for the problem to happen before correcting it. Subsequent smoothing at the client side to blend its inaccurate pose estimate with the newly arrived correct pose only intensifies its reactive nature.

Despite its disadvantages, approaches based on dead reckoning continue to have a strong presence, e.g. [4, 19]. Network games represent an important area of research and they are one of the most wide-spread examples of Distributed Virtual Environments, as many inhabitants share the same virtual world and interact with each other on a regular basis. Delay compensation in these games is typically a combination of DR and client-side interpolation [8].

2.2 Proactive Latency Compensation

In light of the disadvantages of reactive latency amelioration, predictive compensation comes across as "the only viable approach to mitigating the consequences of delay" [14]. Predictive compensation is a proactive approach and can be further differentiated into delay jitter insensitive and variability-aware methods. The former category is the state-of-the-art approach at the time of writing, as proactive delay amelioration methods found in the literature assume, explicitly or implicitly, a constant delay. This manifests itself in ways ranging from experiment setups with simulated constant delay to delay stability assumptions in extrapolation equations.

Wu & Ouhyoung's [24] comparative evaluation of their Grey System method was carried out by fixing the prediction interval to constant values. Akatsuka and Bekey [1] evaluated their method of head-tracking latency compensation assuming a static network delay that depended on the complexity of the rendered scene. A constant prediction interval is adopted, again, in another evaluation of a proposed KF-based head-motion predictor [14, 15]. The consequences of added high-frequency noise and overshoot as a result of predictive compensation were evaluated, with a look-ahead prediction interval of 50ms [14] and for constant latencies ranging from 0 to 100ms [15]. Finally, in the discussion of the suitability of client-side prediction for games [19], Pantel and Wolf evaluated the performance of seven different prediction schemes for simulated delay values of 100 and 200 milliseconds. In doing so, they too adopted the assumption of a constant delay.

Azuma departs from the constant delay assumption [5, 6]. His closed-form expressions for pose predictors parameterize the interval of look-ahead. The testbed implementing his work was not distributed, however, running on the same local platform with tight control on and the knowledge of various parts of the end-to-end system delay. Furthermore, clock synchronization and timestamp acquisition throughout various portions of the tracker-to-the-screen pipeline enabled Azuma to deterministically compute the prediction interval required [6]. The latter along with the distributed nature of DIVEs clearly separates Azuma's work from ours. Public switched networks afford little, if any, control over the propagation delay, with network latency undergoing significantly more variation compared to local processing delays. Finally, the extent of variability itself can easily change throughout a single experiment or a DIVE session.

Azuma later introduced a theoretical framework for head motion predictor analysis, allowing researchers to perform comparisons across several predictive trackers w.r.t. *specified* system parameters, such as a prediction interval [7]. LaViola offers an entire testbed for the empirical evaluation of predictor performance [17]. Both Azuma's theoretical framework and LaViola's testbed enforce a choice of a specific value for the latency and are hence best suited for the evaluation of systems with constant system latency.

A unifying theme in the literature on proactive compensation is the continued influence of the constant delay assumption. The two-tier compensation framework presented in this paper breaks away from this premise and offers a simulation environment where multiple head motion datasets can be tested against multiple network delay traces, each of which exhibits significant latency jitter.

3 Pose Estimation

Pose estimation represents the top layer of the two-tier framework introduced in section 1.2. The need for estimating the pose, defined as a combination of a tracked object's position in the motion space and its orientation, is evident from the dynamic nature of DIVE systems. The physical system consisting of the user's tracked body parts, the muscles setting them in motion, and the tracking device producing measurements of their pose can be modeled as a system with the input in the form of a white noise disturbance function and a pose as its output. Figure 4 illustrates the model employed for the positional component of the pose.

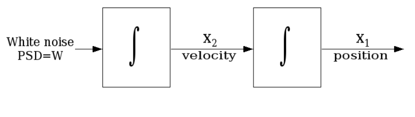

Figure 4: Process model for positional component of the system

3.1 Position

The Kalman Filter recursive framework has been widely used by the VR community for both position and orientation state estimation. For position, we used a per-component state vector $\vec{X} = \begin{bmatrix} p_x \\ \dot{p}_x \end{bmatrix}$, where p_x denotes a positional component and $\dot{p}_x$ its time derivative. Due to the nature of human motion and tracking devices, our system should be regarded as a continuous-time system sampled at discrete points in time. The continuous process' governing equation below, therefore, was chosen to formulate its dynamic behavior:

$$\dot{X}(t) = FX(t) + Gw(t) + Lu(t) \tag{1}$$

X here represents the state vector discussed above, while $\dot{X}$ denotes its time derivative. $w(t)$ is the process noise input into the system, which is assumed to be well-behaved, and $u(t)$ generally signifies a deterministic vector-forcing function. Due to the absence of a deterministic control input, u is set to zero in our case. The square matrix F is of particular importance here and is known as the system dynamics matrix. In general, the coefficients of eq. (1) vary with time, but the time subscript has been dropped for notational convenience. Upon discretization of the process model equation above using the difference equation solution to eq. (1) [9]:

$$x_{k+1} = \Phi(t_{k+1}, t_k)x(t_k) + \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \tau)G(\tau)w(\tau)d\tau \tag{2}$$

we get the more familiar form of the governing equation:

$$x_{k+1} = \Phi_k x_k + w_k$$

Coupling it with the measurement equation linearly relating the observed state of the system to the measurement z_k, we arrive at the system of discrete-time linear equations that are the heart of the KF recursive algorithm:

$$\begin{cases} x_{k+1} & = & \Phi_k x_k + w_k \\ z_k & = & H_k x_k + v_k \end{cases}$$

A constant velocity model was chosen for the positional KF due to our observation that, for adequately small time intervals, linear velocity undergoes insignificant change [23]. This assumption readily leads to the computation of the system dynamics matrix $F = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ and $G = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Fundamental matrix Φ_k is then found by evaluating a Taylor series expansion of the matrix exponential $e^{F\Delta t}$ [9, 25]:

$$\Phi_k = e^{F\Delta t} = I + F\Delta t + \frac{(F\Delta t)^2}{2!} + \ldots = I + F\Delta t = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \tag{3}$$

$H = \begin{bmatrix} 1 & 0 \end{bmatrix}$ is even simpler to find, since measurement z is formed to be a scalar value equaling the corresponding component of a tracked object's position.

With these building blocks in place we can pictorially summarize the recursive KF algorithm in fig. 5, which requires the process noise covariance matrix Q_k to be either computed or analytically derived. We can show that [23] that

$$Q_k = \int_0^{\Delta t} \Phi(\tau) Q \Phi^T(\tau) d\tau \tag{4}$$

is mathematically equivalent to Q_k calculated from its definition (i.e. $E[w_k w_k^T]$), but due to space restrictions present only the final closed-form expression here:

$$Q_k = W \begin{bmatrix} \frac{(\Delta t)^3}{3} & \frac{(\Delta t)^2}{2} \\ \frac{(\Delta t)^2}{2} & \Delta t \end{bmatrix}$$

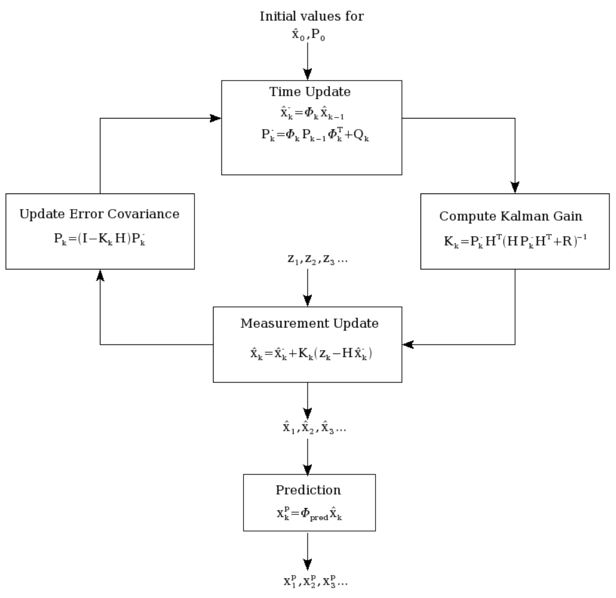
Initial values for
$\hat{x}_0, P_0$
Time Update
$\hat{x}_k^- = \Phi_k \hat{x}_{k-1}$
$P_k^- = \Phi_k P_{k-1} \Phi_k^T + Q_k$
Update Error Covariance
$P_k = (I - K_k H) P_k^-$
Compute Kalman Gain
$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$
$z_1, z_2, z_3 \dots$
Measurement Update
$\hat{x}_k = \hat{x}_k^- + K_k (z_k - H \hat{x}_k^-)$
$\hat{x}_1, \hat{x}_2, \hat{x}_3 \dots$
Prediction
$x_k^p = \Phi_{pred} \hat{x}_k$
$x_1^p, x_2^p, x_3^p \dots$

Figure 5: Recursive KF prediction/correction cycle

where W is the power spectral density of the driving noise.

To summarize, the position is estimated by first projecting the previous estimate of its state forward through the time update and, then, correcting the obtained a priori estimate by incorporating the measurement through the measurement update. The latter is performed by computing the Kalman Gain $K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$ and using it as the weight for the measurement residual $(z_k - H\hat{x}_k^-)$ in the measurement update equation:

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - H\hat{x}_k^-)$$

3.2 Orientation

The orientation component of the pose is represented by a quaternion $q = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$. Together with angular velocity it forms the state vector $X = [q_w\ q_x\ q_y\ q_z\ w_0\ w_1\ w_2]^T$. A similar recursive Kalman Filter approach is applied to the online estimation of orientation. The difference, however, manifests itself through the non-linear nature of the quaternion representation. Eq. (5)

$$\dot{q} = \frac{1}{2}\, q \otimes w \tag{5}$$

is typically used as the basis for the non-linear governing model equation, see [16] for its derivation. The general form of non-linear governing equation in the absence of deterministic control input is:

$$\dot{X} = f(X,t) + w(t)$$

and should be regarded as a generalization of eq. (1). Vector-valued $f : \mathbb{R}^7 \mapsto \mathbb{R}^7$ can be piecewise defined as follows [23]:

$$\begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \\ w_0 \\ w_1 \\ w_2 \end{bmatrix} \mapsto \begin{bmatrix} -\frac{1}{2}w_0 q_x - \frac{1}{2}w_1 q_y - \frac{1}{2}w_2 q_z \\ \frac{1}{2}w_0 q_w + \frac{1}{2}w_2 q_y - \frac{1}{2}w_1 q_z \\ \frac{1}{2}w_1 q_w - \frac{1}{2}w_2 q_x + \frac{1}{2}w_0 q_z \\ \frac{1}{2}w_2 q_w + \frac{1}{2}w_1 q_x - \frac{1}{2}w_0 q_y \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The measurement equation $Z = h(X) + v(t)$ is also non-linear, since function $h : \mathbb{R}^7 \mapsto \mathbb{R}^4$ normalizes the quaternion part of the state:

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix} = Normalize(\begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix}) = \frac{1}{\sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}} \cdot \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix}$$

The time update is performed by employing 4th order Runge-Kutta (RK4) numerical integration for state projection [23], with covariance projection performed similarly to the positional KF case. The fundamental matrix Φ_k for the latter is derived from the Taylor series approximation of exponential $e^{F\Delta t} \approx I + F\Delta t$, see eq. (3). Linearization about the estimated trajectory of motion yields Jacobian matrices $F = \frac{\delta f}{\delta X}\Big|_{X=\hat{X}_k^-}$ and $H = \frac{\delta h}{\delta X}\Big|_{X=\hat{X}_k^-}$ as a result of Taylor series expansion approximation. A closed form expression for Φ_k can now be analytically derived

$$\Phi = \frac{\Delta t}{2} \begin{bmatrix} \frac{2}{\Delta t} & -w_0 & -w_1 & -w_2 & -q_x & -q_y & -q_z \\ w_0 & \frac{2}{\Delta t} & w_2 & -w_1 & q_w & -q_z & q_y \\ w_1 & -w_2 & \frac{2}{\Delta t} & w_0 & q_z & q_w & -q_x \\ w_2 & w_1 & -w_0 & \frac{2}{\Delta t} & -q_y & q_x & q_w \\ 0 & 0 & 0 & 0 & \frac{2}{\Delta t} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{2}{\Delta t} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{2}{\Delta t} \end{bmatrix} \tag{6}$$

Plant equation matrix H surfaces in the measurement update stage of the predictive-corrective KF cycle. The equations for the measurement update are as follows:

$$\begin{aligned} K_k &= P_k^- H^T (H P_k^- H^T + R)^{-1} \\ \hat{X}_k &= \hat{X}_k^- + K_k (Z_k - h(\hat{X}_k^-)) \\ P_k &= (I - K_k H) P_k^- \end{aligned} \tag{7}$$

Special attention must be paid to the proper calculation of the measurement residual, since the predictive estimate of the measurement $h(\hat{X}_k^-)$ does *not* equal to $H\hat{X}_k^-$ in the non-linear EKF case. Using the latter makes the estimate of the measurement degenerate into the zero vector, which results in errorneous filter updates, see [23] for details. The closed-form expression for H follows from taking the Jacobian of function h w.r.t. the a priori estimate of the state. Letting L equal $|q|^2 = q_w^2 + q_x^2 + q_y^2 + q_z^2$, the resulting expression for H is simplified to the following:

$$H = -\frac{1}{L^{3/2}} \begin{bmatrix} q_w^2 - L & q_w q_x & q_w q_y & q_w q_z & 0 & 0 & 0 \\ q_w q_x & q_x^2 - L & q_x q_y & q_x q_z & 0 & 0 & 0 \\ q_w q_y & q_x q_y & q_y^2 - L & q_y q_z & 0 & 0 & 0 \\ q_w q_z & q_x q_z & q_y q_z & q_z^2 - L & 0 & 0 & 0 \end{bmatrix} \tag{8}$$

Finally, Q_k was also analytically derived, using eq. (4) and, similarly to the derivation of Q_k for positional KF, assuming that most noise enters the system through the higher-order derivative terms. The latter translates into constructing the continuous-time process noise covariance matrix Q to equal

$$Q(t) = \begin{bmatrix} 0_{4\times4} & & 0_{4\times3} & \\ & W & 0 & 0 \\ 0_{3\times4} & 0 & W & 0 \\ & 0 & 0 & W \end{bmatrix} = W \begin{bmatrix} 0_{4\times4} & 0_{4\times3} \\ 0_{3\times4} & I_{3\times3} \end{bmatrix}$$

and the resulting $Q_k = \frac{\Delta t^2}{4} \cdot$

$$\begin{bmatrix} \frac{\Delta t}{3}(L - q_w^2) & & & & & & \\ -\frac{\Delta t}{3} q_w q_x & \frac{\Delta t}{3}(L - q_x^2) & & & & & \\ -\frac{\Delta t}{3} q_w q_y & -\frac{\Delta t}{3} q_y q_x & \frac{\Delta t}{3}(L - q_y^2) & & & & \\ -\frac{\Delta t}{3} q_w q_z & -\frac{\Delta t}{3} q_z q_x & -\frac{\Delta t}{3} q_z q_y & \frac{\Delta t}{3}(L - q_z^2) & & & \\ -q_x & q_w & q_z & -q_y & \frac{4}{\Delta t} & & \\ -q_y & -q_z & q_w & q_x & 0 & \frac{4}{\Delta t} & \\ -q_z & q_y & -q_x & q_w & 0 & 0 & \frac{4}{\Delta t} \end{bmatrix}$$

This variance-covariance matrix must be symmetric by definition, hence only the lower triangle part is presented here. To the best of our knowledge, a closed form expression for the constant angular velocity model process noise covariance matrix never appeared in the literature before. This concludes our derivation and description of all construction blocks for the EKF recursive step algorithm.

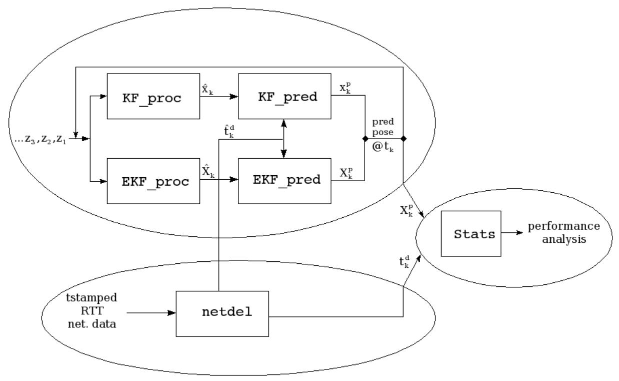

Figure 6: Detailed simulation functionality

4 The Simulator

To evaluate our approach to network delay amelioration, we simulated a unidirectional type client-server DIVE. Fig. 6 provides a functional overview of the simulator. It is composed of 3 principal components: the network, pose, and statistical performance blocks.

4.1 Network delay estimator

Network delay estimator is the bottom layer of our latency amelioration framework. It estimates network propagation delay ($\hat{t}_k^d$), used by the pose estimation & prediction block as T_{pred}. The estimator was modeled after the TCP retransmission timeout — a time-tested auto-regressive (AR) filtering approach introduced by Jacobson as a means to calculate retransmission timeout (RTO) for TCP fragments [22]. TCP RTO was designed for network delay estimation of immediate utility during a single end-to-end connection — an attractive feature for applications with short-term network delay prediction requirements. The retransmission timeout delay estimation algorithm can be summarized as follows:

$$\begin{aligned} delta &= measuredRTT - srtt \\ srtt &\leftarrow srtt + g \times delta \\ rttvar &\leftarrow rttvar + h(|delta| - rttvar) \\ RTO &= srtt + 4 \times rttvar \end{aligned} \tag{9}$$

with weight factors g & h ranging between 0 and 1. This algorithm measures the instantaneous estimation error *delta* and provides filtered versions of the instantaneous network delay *srtt* and its variability *rttvar*. Eq. (9) can be rewritten as a low-pass AR filter:

$$\begin{aligned} srtt_{i+1} &= (1-g) \times srtt_i + g \times rttdel_i \\ rttvar_{i+1} &= (1-h) \times rttvar_i + h \times |rttdel_i - srtt_i| \end{aligned}$$

providing smoothed estimates of round-trip time delay and its deviation from the mean.

4.2 Pose module particulars

The pose module was implemented to play back real head pose data from the motion repository described in [17]. Simulated measurement data was derived from the ground truth by downsampling it & perturbing the result with white noise according to in-house IS900 tracker specs. While positional data perturbation is straightforward, orientation measurement sequence had to be obtained from quaternion sequence perturbation. Our approach to it rests on the geometric interpretation of quaternions [16]. Namely, quaternion perturbation can be achieved by performing quaternion multiplication of the ground truth sequence with noise quaternions generated. The effect of the latter is perturbing the true rotation by a subsequent noise rotation, carried out about a random axis with the angle normally distributed about zero. Details of this approach are omitted due to space constraints and can be found in [23].

Finally, pose prediction was performed using the respective state projection equations. For position, $\Phi_{pred} = \begin{bmatrix} 1 & T_{pred} \\ 0 & 1 \end{bmatrix}$, resulting in $x_p = \Phi_{pred}\hat{x}_k$. For orientation, we formed $q_k = [q_w\ q_x\ q_y\ q_z]^T$ and $w_k = [0\ w_0\ w_1\ w_2]^T$ from $\hat{X}_k$. Then q_p was obtained using RK4 with $\Delta t = T_{pred}$. The resulting quaternion underwent explicit normalization.

5 Results and Discussion

5.1 Experimental Setup

Six head motion datasets were chosen from LaViola's repository [17]. Each dataset features approximately 20 seconds of positional and orientation data captured by an IS900 tracking system. The head motion datasets fall into three major categories and reflect specific motion profiles, as summarized in table 5.1.

name	motion profile
HEAD1	simple head movement where the user is roughly stationary and rotates to view the display screens
HEAD2	more complex head movement where the user is allowed to both walk and look around the CAVE
HEAD3	more complex head movement where the user is examining a fixed virtual object to gain perspective about its structure

Table 1: Motion data summary

Two datasets were selected for each head motion profile and will be referred to by the corresponding name from the table above, augmented with an id $\in \{1,2\}$ for unique identification. The simulation was carried out using Matlab under Linux.

Network delay datasets were collected with two computers, a departmental server for echoing incoming packets and another computer behind a residential ISP with a maximum of 128KB/s bandwidth. To generate significant latency and jitter, we saturated the network link with 50% artificial traffic. We collected 10 network traces with the following characteristics:

trace	min	avg	max	sdev	% var
1	7382	48900	157891	40147	82.1
2	7215	48548	153349	40142	82.6
3	7318	48514	153590	40495	83.4
4	7275	47450	153464	39655	83.5
5	7318	47745	152637	40273	84.3
6	7303	46515	149634	39601	85.1
7	7405	47652	157251	39947	83.8
8	7266	46995	150121	39274	83.5
9	7225	46431	154447	39484	85.0
10	7214	46662	153613	39937	85.5

Table 2: network delay statistics - halfpipe nettrace (μs)

As the granularity of Linux timers is only 10ms (100Hz), we used real-time clock chip interrupts to trigger packet generation at 180Hz, equivalent to the desired sampling frequency of a single-station IS900 tracker [12]. Four different pose predictor conditions were compared in our simulaton:

1. **Const** – network delay estimate was set to a constant value
2. **Runavg** – a running average of network RTT was used as an estimate of current round trip latency
3. **SRTT** – smoothed RTT estimator, as described in section 4.1
4. **Opti** – an omnipotent network delay estimator given perfect knowledge of packet latencies

Opti was included to compare the overall performance of the proactive algorithms with an ideal network delay estimator, which provides the pose predictor with the correct prediction interval every time. Hence, Opti serves as a benchmark we strive to reach through potential improvements to network delay estimation. Constant delay prediction was included for comparison with variability-aware Runavg and SRTT. The value for the constant RTT delay was set to the mean RTT delay for a given network trace.

5.2 Results

Results are presented as root mean square error (RMSE) to measure the extent of the overall framework's accuracy in predicting the pose at the time of events' arrival at the receiving workstation. For position, then, the component-wise difference is determined between the received pose data and interpolated ground truth, resulting in $n \times 3$ *diffpose*, where n is the number of data points. Global position RMSE is calculated as follows:

$$pos\_rmse = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{3} diffpose^2(j,i)}$$

Global orientation RMSE was computed based on the angular error, which provides an intuitive measure of the discrepancy between the ground truth and received orientation in terms of the angular separation between them. The difference quaternion is

$$q_{diff} = q_{real} \otimes q_{pred}^{-1} = q_{real} \otimes q_{pred}^{*}$$

and we can compute the angular error E_α using the q_0 component of this difference quaternion, denoted as $q_{diff}[0]$:

$$E_\alpha = \frac{2 \cdot 180}{\pi} \cos^{-1}(q_{diff}[0])$$

The angular RMSE for the entire motion sequence can then be computed as $ang\_rmse = \sqrt{\sum_{j=1}^{n} \frac{E_\alpha^2(j)}{n}}$. Finally, TImes BETter (TIBET) statistic was used to measure the %-improvement obtained over Const by other predictors on average.

5.2.1 Position

The smoothing performance of the positional Kalman Filter is fairly consistent across all six head motion datasets and is nearly identical for all three components. Figure 7 presents an overlaid plot of

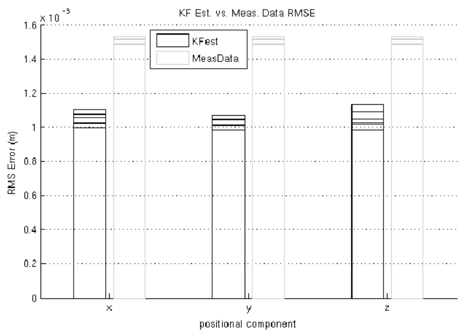

Figure 7: Position KF Estimate vs Measurement Data RMSE

KF estimator RMS error versus using the measurement data. Bars of the same color but varying height represent RMSE information for each of the six motion datasets on a per-component basis. It can be seen that KF smoothing improves the estimate of the state in all cases. Table 3 illustrates the relative performance improvement

pcmp	head11	head12	head21	head22	head31	head32
x	1.4970	1.4949	1.4451	1.4087	1.4367	1.3706
y	1.5082	1.5145	1.4665	1.4546	1.4504	1.4099
z	1.5115	1.5012	1.4446	1.3394	1.4464	1.3852

Table 3: TIBET - KF estimator performance (position)

from Kalman Filter smoothing. A consistent improvement in the neighborhood of 45% is observed. TIBET ratios are slightly higher for HEAD1 than for motion profiles 2 & 3 though, a fact we attribute to small range of translational head motion characteristic of the first motion profile.

Our principal goal is to minimize the gap between the output of the predictor and the ground truth. What follows is, therefore, a look at the aggregate performance of the entire compensation framework. Figure 8 shows the average framework performance

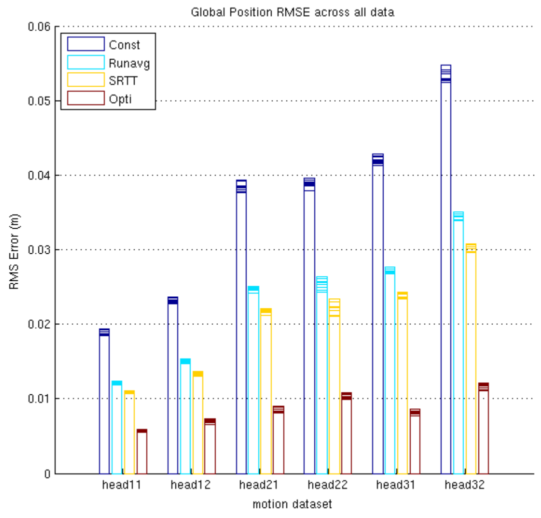

Figure 8: Global Position RMSE across all data

for the four prediction approaches in terms of global head position RMS error across all motion datasets and network traces. The results reveal that both Runavg and SRTT based predictors visibly outperform constant delay prediction, but do not get close to Opti. The apparent consistency of RMSE information across all the network traces, signified here by individual bars of the same color, is noteworthy as well. This motivated us to present the TIBET statistic for a randomly chosen network RTT dataset as a 2D table representation (table 4). Runavg offers 55.5% relative improvement over Const, while SRTT is better by 75.7%.

dataset	Const	Runavg	SRTT	Opti
head11	1.0000	1.5918	1.7773	3.3637
head12	1.0000	1.5427	1.7260	3.2836
head21	1.0000	1.5431	1.7497	4.4087
head22	1.0000	1.5212	1.7213	3.8391
head31	1.0000	1.5510	1.7696	5.2136
head32	1.0000	1.5791	1.7999	4.5335

Table 4: TIBET - overall framework performance (position)

5.2.2 Orientation

Similarly to the positional results presented above, the Extended Kalman Filter based smoother reduces the RMS error w.r.t. raw measurements for each of the quaternion components across all of

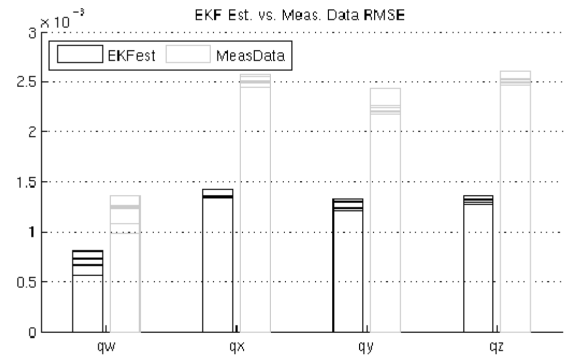

Figure 9: Orientation EKF Estimate vs Measurement Data RMSE

qcmp	head11	head12	head21	head22	head31	head32
q_w	1.7081	1.6396	1.6961	1.7138	1.8428	1.7355
q_x	1.8967	1.8609	1.8177	1.9017	1.8616	1.7555
q_y	1.7063	1.7034	1.7638	1.7025	1.8702	1.8207
q_z	1.8882	1.9546	1.8611	1.9706	1.8748	1.9078

Table 5: TIBET - EKF estimator performance (orientation)

the head motion datasets (fig. 9). According to table 5, the EKF estimator manages to follow the ground truth with approximately 81% less error than the measurement data on its own.

To evaluate the predictor performance, a measure of global orientation error in the form of a rotation angle between predicted and ground truth quaternion was chosen. Figure 10 visualizes the performance of the EKF predictor on average in terms of overlaid RMSE information across all motion datasets and network traces. Table 6 demonstrates the relative performance gain for both Runavg

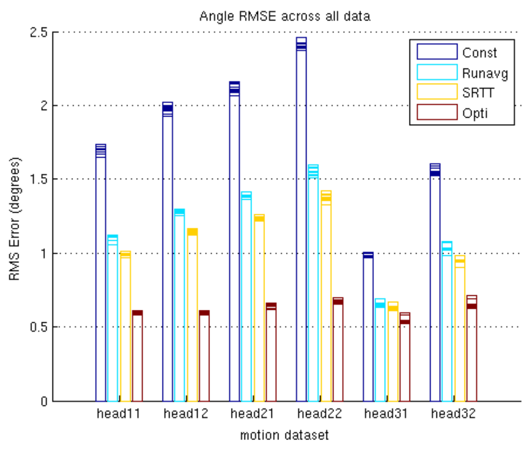

Figure 10: Angle RMSE across all data

and SRTT based EKF predictors over the constant delay prediction. Specifically, Runavg offers 52% relative improvement over Const, while SRTT is better by 67%. The performance improvement from network latency tracking is the least for the third motion profile, consistent with smaller variability in head orientation. Indeed, it's reasonable to hypothesize that the increase in the variability of underlying data will exacerbate the severity of predictor overshoots and undershoots in case of constant prediction time interval. In support of this hypothesis, the ground truth quaternion sequences were converted to the underlying axis-angle representation and the

dataset	Const	Runavg	SRTT	Opti
head11	1.0000	1.5565	1.7049	2.8486
head12	1.0000	1.5293	1.7064	3.2357
head21	1.0000	1.5079	1.6865	3.3344
head22	1.0000	1.5442	1.7402	3.5669
head31	1.0000	1.5050	1.5682	1.8488
head32	1.0000	1.5000	1.6360	2.4663

Table 6: TIBET - overall framework performance (orientation)

standard deviation in both is summarized in table 7. The correlation between the heightened variability in both the axis and angle of rotation and the increase in relative performance improvement is reflected in the smaller values for the third dataset.

dataset	α°	u_x	u_y	u_z
head11	25.6876	0.2943	0.7488	0.0929
head12	25.7972	0.3357	0.8853	0.1670
head21	36.9928	0.3690	0.8785	0.2047
head22	34.1722	0.3175	0.8867	0.2144
head31	20.0750	0.1969	0.0699	0.1900
head32	13.9617	0.3630	0.2316	0.2970

Table 7: Axis-angle standard deviation

5.3 Discussion

The above results demonstrate that variability-aware predictors substantially outperform constant delay based pose prediction. Furthermore, for orientation we pointed out a clear correlation between the amount of variability in motion data and the overall performance of the predictor-estimator framework relative to the Const predictor. More generally, we've established that variability increase along a spatial dimension correlates with relative performance improvement for jitter-sensitive compensation techniques, see also [23]. Orthogonally, we can also show that relatively hightened variability along the time dimension translates into the increase of SRTT's performance as compared to that of Runavg-based predictor! To demonstrate this, a second set of network delay traces was collected by saturating the connection to 100% with artificial traffic, resulting in higher means, see table 8. Remarkably, the standard deviation of the RTT delay was similar to the first experiment.

trace	min	avg	max	sdev	% var
1	24876	225154	293980	39857	17.7
2	12790	225701	290841	43760	19.3
3	23076	227901	290391	41182	18.0
4	14569	224701	280530	39852	17.7
5	14239	226571	282307	42672	18.8
6	14730	220918	277460	41187	18.6
7	13246	218871	283756	44095	20.1
8	14100	223254	278400	38992	17.4
9	14266	221354	282119	42936	19.3
10	14577	220632	279065	38968	17.6

Table 8: network delay statistics - fullpipe nettrace (μs)

Performing the same simulation results in the RMSE data plotted in fig. 11 and summarized in table 9. Clearly, the overall picture has changed compared to fig. 10 and table 6. Specifically, we see much less difference between Runavg and SRTT-based predictor RMS errors.

Comparing data from table 2 and 8, we can say that in the 100% saturated environment there was an average 18% variability, whereas in the 50% saturated environment there was about 83% variability, or about a factor of 4.5 more variability. Correlating this with the performance of the two variability-sensitive

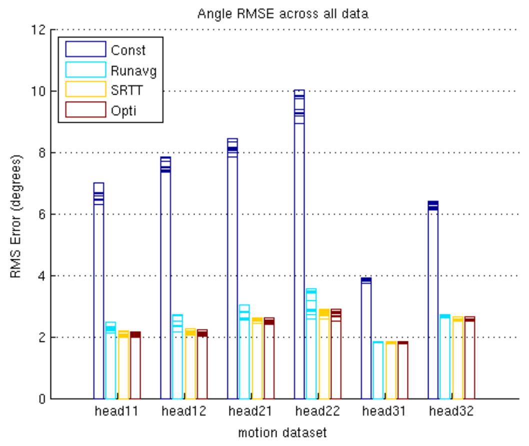

Figure 11: Angle RMSE across all data - fullpipe

dataset	Const	Runavg	SRTT	Opti
head11	1.0000	2.9476	3.1719	3.1955
head12	1.0000	3.4262	3.5847	3.6435
head21	1.0000	3.0347	3.1418	3.1627
head22	1.0000	3.2896	3.2786	3.3148
head31	1.0000	2.0984	2.1514	2.1431
head32	1.0000	2.3523	2.4699	2.4739

Table 9: TIBET - overall framework performance (orientation)

algorithms (tables 6 and 9), we see that in the first case SRTT outperformed runavg by only 4% on average, where as in the second case, SRTT outperformed it clearly by 10%, or about a factor of 2.5 better. SRTT clearly performs better in environments with higher variability. In the interest of space, only the angle RMS errors are compared here, since the position errors follow the same trend.

6 Conclusion

A new variability-aware proactive latency compensation technique was described and evaluated. No assumptions about the dynamics of the network delay to be estimated are either made or enforced. The two-tier predictive framework presented in this paper consists of the pose predictor working in concert with the network delay estimator to perform sender-side prediction of the events. This work represents the only known technique to compensate for latency by performing sender-side prediction a variable time into the future. Evaluation was carried out through an offline playback of real head motion data and network RTT delay traces. This simulator is a conceptual improvement over LaViola's predictive algorithm testbed, where a user was confined to using a constant look-ahead interval. Statistical evaluation of the variability-aware predictive framework shows its substantial improvement over the predictor's assumption of constant latency. Furthermore, the relative performance of variability-aware predictors was shown to increase with increased variability in orientation motion data.

Acknowledgements

The authors wish to thank Ontario Centres of Excellence for funding this research (grant #HP00ELL10). We acknowledge Todd Macuda, Sion Jennings and Carl Swail of NRC Canada and Jonathan Solomon of UNSW, Australia, for their help with network data collection.

References

[1] Y. Akatsuka and G. Bekey. Compensation for end to end delays in a vr system. In *IEEE VR 1998*, pages 156–159, March 1998.
[2] R. S. Allison, L. R. Harris, M. Jenkin, U. Jasiobedzka, and J. E. Zacher. Tolerance of temporal delay in virtual environments. In *IEEE VR '01*, page 247, 2001.
[3] R. S. Allison, J. E. Zacher, D. Wang, and J. Shu. Effects of network delay on a collaborative motor task with telehaptic and televisual feedback. In *VRCAI '04*, pages 375–381, 2004.
[4] J. Aronson. Dead reckoning: Latency hiding for networked games. www.gamasutra.com/features/19970919/aronson_01.htm, 1997.
[5] R. Azuma. *Predictive Tracking for Augmented Reality*. PhD thesis, University of North Carolina at Chapel Hill, February 1995.
[6] R. Azuma and G. Bishop. Improving static and dynamic registration in an optical see-through hmd. In *SIGGRAPH '94*, pages 197–204.
[7] R. Azuma and G. Bishop. A frequency-domain analysis of head-motion prediction. In *SIGGRAPH '95*, pages 401–408, 1995.
[8] Y. W. Bernier. Latency compensation methods in client/server in-game protocol design and optimization. In *Proc. of Game Developers Conference'01*, 2001.
[9] R. G. Brown and P. Y. Hwang. *Introduction to random signals and applied Kalman filtering: with MATLAB exercises and solutions*. John Wiley & Sons, New York, N.Y., 3rd edition, 1997.
[10] D. Delaney, T. Ward, and S. McLoone. On consistency and network latency in distributed interactive applications: a survey - part 1. *Presence*, 15(2):218–234, 2006.
[11] Ieee standard for distributed interactive simulation - application protocols. IEEE Std 1278.1-1995, March 1996.
[12] InterSense, Inc., Burlington, Massachusetts. *InterSense IS-900 Precision Motion Tracker for Virtual Environments: Manual for Models IS-900 VET & IS-900 VWT*, 1.0 edition, 2000.
[13] J. Joseph J. LaViola. A discussion of cybersickness in virtual environments. *SIGCHI Bull.*, 32(1):47–56, 2000.
[14] J. Y. Jung, B. D. Adelstein, and S. R. Ellis. Discriminability of prediction artifacts in a time-delayed virtual environment. In *Proceedings HFES 2000*, volume 1, pages 499–502, 2000.
[15] J. Y. Jung, B. D. Adelstein, and S. R. Ellis. Predictive compensator optimization for head tracking lag in virtual environments. In *Proceedings, IMAGE 2000*, pages 123–132, 2000.
[16] J. B. Kuipers. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*. Princeton University Press, Princeton, New Jersey, 1999.
[17] J. J. LaViola. A testbed for studying and choosing predictive tracking algorithms in virtual environments. In *EGVE '03*, pages 189–198.
[18] M. Mauve. How to keep a dead man from shooting. In *IDMS '00*, pages 199–204, 2000.
[19] L. Pantel and L. C. Wolf. On the suitability of dead reckoning schemes for games. In *NetGames '02*, pages 79–84, 2002.
[20] K. S. Park and R. V. Kenyon. Effects of network characteristics on human performance in a collaborative virtual environment. In *IEEE VR '99*, pages 104–111, 1999.
[21] K. M. Stanney, editor. *Handbook of Virtual Environments: Design, Implementation, and Applications*. Lawrence Erlbaum Associates, Inc., Mahwah, New Jersey, 2002.
[22] W. R. Stevens, B. Fenner, and A. M. Rudoff. *UNIX Network Programming: The Socket Networking API*, volume 1. Addison-Wesley Professional, 3rd edition, 2004.
[23] A. Tumanov. Variability-aware latency amelioration in distributed interactive virtual environments. Master's thesis, York University, 2006.
[24] J.-R. Wu and M. Ouhyoung. A 3d tracking experiment on latency and its compensation methods in virtual environments. In *UIST '95*, pages 41–49, 1995.
[25] P. Zarchan and H. Musoff. *Fundamentals of Kalman filtering: a practical approach*, volume 190 of *Progress in astronautics and aeronautics*. American Institute of Aeronautics and Astronautics, Inc., 2000.