VARIABILITY-AWARE LATENCY AMELIORATION IN DISTRIBUTED INTERACTIVE VIRTUAL ENVIRONMENTS

Alexey Tumanov

A thesis submitted to the Faculty of Graduate Studies
in partial fulfilment of the requirements
for the degree of

Master of Science

Graduate Programme in Computer Science and Engineering
York University
Toronto, Ontario
April, 2006

Abstract

Designers of applications of collaborative distributed Virtual Environments must account for the impairment of the network connecting them and its detrimental effects on user performance. Based upon analysis and classification of existing latency compensation techniques, this thesis introduces a novel amelioration method 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 online 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 the one based on state-of-the-art assumption of a constant system delay.

Acknowledgements

I'm thankful to all the people who, in one way or another, helped me throughout the years of my Master's Thesis research.

First and foremost, successful completion of this research project would never be possible without the continued support, guidance, and valuable advice of my primary research supervisor, Dr. Robert Allison. I'm deeply grateful for his openness to discussion and availability, even during his first sabbatical. I would also like to thank my co-supervisor, Dr. Wolfgang Stürzlinger, for his research insights, editorial help, and supportive collaboration.

Special thanks goes to Olena, my love, for being there for me when it was most needed, for her emotional support and undying pride in my work as well as her interest in the area of my research in general.

The unconditional support of my parents, Nadezhda and Alexander Tumanov, my sister Svetlana, and my friends is also immensely appreciated and will never be forgotten. Many thanks to Peter Olsar, my friend and the ACM programming contest teammate, for many insightful and deeply theoretical discussions ranging from game engine improvement ideas to the meaning of life.

Finally, I'd like to extend my gratitude to everybody who helped me with network data collection by hosting my computers, providing me with a user account on their systems, and opening up their firewalls.

Table of Contents

Abstract **iv**

Acknowledgements **v**

Table of Contents **vi**

1 Introduction **1**

1.1 Research Issues 2

1.2 Motivation 5

1.3 Our Approach 8

1.4 Thesis Outline 12

2 Previous Work **14**

2.1 Reactive Latency Compensation 15

2.2 Proactive Latency Compensation 18

3 Pose Estimation - Position **25**

3.1 Background and Intuition 25

3.2 Model Formulation 28

3.3 Process Noise Covariance Matrix 32

3.3.1 Proof of Equivalence 33

3.3.2 Q_k derivation 35

3.4 Positional KF Algorithm 36

4 Pose Estimation - Orientation **39**

4.1 Model Formulation 40

4.2 Measurement Update 45

4.3 Time Update 48

4.3.1 State Projection 48

4.3.2 Covariance Projection 50

4.4 Prediction 53

4.5 EKF algorithm summary 55

5 Network Delay Estimation **58**

5.1 Introduction 58

5.2 Motivation 60

5.2.1 Predetermined Network Delay 61
5.2.2 Client-Side Measurement Approach 62
5.2.3 Sender-side Delay Estimation 64
5.3 Methods of Estimation 65
5.4 The Chosen Method 70

6 The Simulator **73**
6.1 Network Delay Estimator 75
6.2 Pose Estimation and Prediction 76
6.2.1 Positional KF Initialization 77
6.2.2 Orientation EKF Initialization 77
6.3 Statistical Analysis 85

7 Results and Discussion **89**
7.1 Experimental Setup 89
7.1.1 Head Motion 89
7.1.2 Network Setup 90
7.1.3 Predictor Conditions 93
7.2 Results and Discussion 94
7.2.1 Position 94

7.2.2 Orientation 99

7.3 Concluding Remarks 105

8 Conclusions and Future Work **108**

8.1 Conclusion 108

8.2 Implications and Future Work 109

A Process Noise Autocorrelation **112**

B Quaternion Sequence Perturbation **114**

C Quat. Sequence Interpolation Algorithm **117**

Bibliography **119**

1 Introduction

Virtual Environments (VE) allow users to gain a sense of immersion into synthetic reality, populated with manipulatable objects and traversable environments, making it possible to experience a modeled world from an egocentric perspective [Sta02]. Certain applications leverage the ability to collaborate in a virtual environment (CVE), whereby the synthetic reality is shared by more than one individual. In the common case, when collaborating parties brought together by a CVE are, in reality, geographically separated, the virtual environment is said to be distributed and takes upon the name of a Distributed Interactive Virtual Environment (DIVE).

DIVE technology has matured over the years from a science fiction inspired idea to research prototypes confined to a few well-financed institutions, to broader application in the industry, to, finally, the mass market. With recent exponential advances in CPU and graphics processing unit (GPU) performance as well

as memory throughput, Virtual Reality (VR) simulations have begun infiltrating our everyday lives. There have been successful applications of Collaborative Virtual Environment technology in E-commerce [MH99], tele-learning, video-conferencing, industrial engineering training, pilot virtual flight practice, game industry, tele-surgery operation and a variety of other applications. Yet, with the ever-evolving graphical appeal of synthetically generated environments, photorealism of their virtual inhabitants, and realistic physics simulation engines, a whole array of open research problems inherent to the distributed nature of DIVEs persists.

1.1 Research Issues

Capps and Stotts identify three major problem areas [CS97]:

a. content generation and management

b. architecture

c. network delivery

The issue of network delivery is fundamental for any distributed simulation allowing geographically separated operators to share a common environment and, particularly, interact and cooperate in it. The interactive component of DIVEs

calls for ways of maintaining a consistent state of the shared environment as well as the objects and autonomous or human-operated avatars that inhabit it. Non-deterministic or unknown movements or other state changes must be communicated to other DIVE nodes explicitly or implicitly (e.g. via commands that bring those changes about). The communication requirement for state consistency maintenance unveils three facets of the network delivery issue.

First, delivery of network-routed messages is clearly subject to the reliability of the underlying network medium. It may or may not be a critical issue, depending on the nature and the type of messages lost. Naturally, some research has been done to introduce frameworks that differentiate between important messages and those, whose loss the intended recipient is less sensitive to. The delivery of the former can be guaranteed by the use of connection-oriented protocols such as TCP, while the latter can be sent via lower-overhead connectionless channels, such as offered by the light-weight UDP protocol.

Second, depending on the number of participants and the level of collaboration, network bandwidth may be a limiting factor as well. An increase in the number of remote users translates into a rise in the amount of information transmission required to maintain some global state, not to mention a consistent one. It leads directly to the network bandwidth consumption increase that may quite

easily surpass levels physically allowed by dial-up modems, for instance.

Finally, 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, is, perhaps, the most persistent research issue of all. Indeed, according to Patterson, latency lags bandwidth and "improves by no more than a factor of 1.2 to 1.4" for a double growth in bandwidth and exhibits a tendency to continue this trend in the foreseeable future [Pat04].

What makes matters worse is that we can propose solutions to the first two network delivery concerns, which may be at the expense of an increase in delay. We can introduce reliability to network communication, for example, by using TCP, but it is well-understood that connection-oriented protocols operate at the expense of network delay. We may improve bandwidth by introducing end-point queuing buffers [Pat04], but that too would introduce additional queuing latency. Ultimately, there's no hope to completely eliminate network delay due to the speed of light limiting signal propagation through physical media. Reportedly, it amounts to about $0.1s$ between Europe and Australia [PW02b].

For these reasons, combating network delay in DIVEs is an open research area that became the focus of this thesis as well.

1.2 Motivation

The motivation to focus our research on combating the network latency is manifold. First, it's a major contributor to end-to-end system delay in networked virtual environments. End-to-end latency leads to a number of serious anomalies, especially when it comes to intimate collaboration or interaction of geographically distributed participants.

It takes a toll on human performance in cooperative tele-operation tasks, for instance. Park and Kenyon [PK99] looked at how the performance in cooperative manipulation of objects degrades as a function of increased network propagation delay. In their task, one of the users controlled a virtual rod, while the other manipulated a virtual ring. Their common goal was defined to transfer the ring from one end of the rod to the other with the minimum number of object collisions and 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. Park and Kenyon [PK99] as well as other similar research studies [AZWS04] converge on a consistent conclusion that, in addition to increased time-to-completion, task accuracy also suffers the most in longer latency experiment setups.

Furthermore, perceptual instability of visual worlds as a result of end-to-end

system latency has been shown to be the major cause of what has become known as cybersickness [JJL00]. Cybersickness is a term that has been defined in a variety of ways, but in general, it refers to "sensations of nausea, oculomotor disturbances, disorientation, and other adverse effects associated with VE exposure" [Sta02]. It is a very undesirable effect, especially in networked VE systems designed for flight simulation and combat training, which inherently require prolonged exposure to DIVEs. In fact, a pilot grounding policy is in effect at many air force bases to prevent VE exposure aftereffects from having an adverse and, possibly, fatal effect on pilot's performance in control of a real aircraft [JJL00]. It is, therefore, difficult to underestimate the importance of eliminating or ameliorating the effect of primary causes of cybersickness. Network delay reduction and compensation techniques play a vital role in improving VE perceptual stability.

Finally, end-to-end latency has been shown to result in causal anomalies in multi-operator DIVEs, such as multiplayer games. A classic example is a "dead man shooting" [Mau00]. In a three-person scenario with players A, B, and C, player A can shoot B, but before B's workstation receives this launched projectile information, player B may successfully terminate C. Close range combat significantly exacerbates the problem, as the time of projectile flight is decreased, making it more probable for network propagation delay to exceed it. Even such

attempts to compensate as event prediction based on an avatar's posture or limb/body movement, for instance, can be completely ineffective due to the non-determinism in the avatar's behavior. Consider a sniper simply looking through the scope without taking a shot. He has every indication of the intention to generate the bullet shot event, but it is completely unpredictable when, if at all, it will actually occur.

In summary, end-to-end latency significantly degrades human performance and serves as a major cause of oscillopsia, defined by Allison et al as referring to the perception that the visual world appears to swim about or oscillate in space [AHJ+01]. The impairment due to latency, further, contributes to the sensation of cybersickness and causal anomalies in multi-operator DIVEs as well as has a deleterious effect on the perceived consistency of the virtual world. However, in addition to exhibiting significant latency, networks used for DIVEs often feature significant variation in that latency as well. Yet, it is our belief that network delay compensation has been approached with a certain disregard to latency variation (a.k.a. jitter), despite existing evidence of its importance [PK99], [AZWS04].

Park and Kenyon [PK99] conclude that network latency jitter disarms compensatory prediction techniques otherwise available for known constant delay systems. From human performance perspective, it becomes difficult, if not im-

possible, to adapt to display lag. Intermediate to advanced operators are usually able to adapt to constant delay and use prediction to excel in target-tracking tasks despite latency. Its variability, however, renders operators' implicit prediction significantly less accurate, naturally causing their extrapolation heuristic to overshoot or undershoot along the target's actual trajectory — all of this due to the continuous change in the prediction interval. Such evidence builds up intuition for variability-aware latency compensation techniques.

1.3 Our Approach

I have argued that participating dynamic entities need to exchange their movement or state change information with other relevant participating nodes. It is worth noting that a dynamic entity's state is a fairly general concept and may extend beyond an avatar's kinematic characteristics (such as position, linear and angular velocity or acceleration) to dynamic or, perhaps, even thermodynamic properties. Examples of the former could be an amount of force exerted and its direction that would be most useful for tele-haptic DIVEs, whereas the latter could be the atmospheric temperature and pressure in a flight simulator. Within the scope of this thesis, however, we will constrain ourselves to a discussion of kinematic information, commonly referred to as a pose.

A worthwhile detour into the status quo of current compensation techniques will illustrate a number of advantages we offer. Figure 1.1 depicts what con-

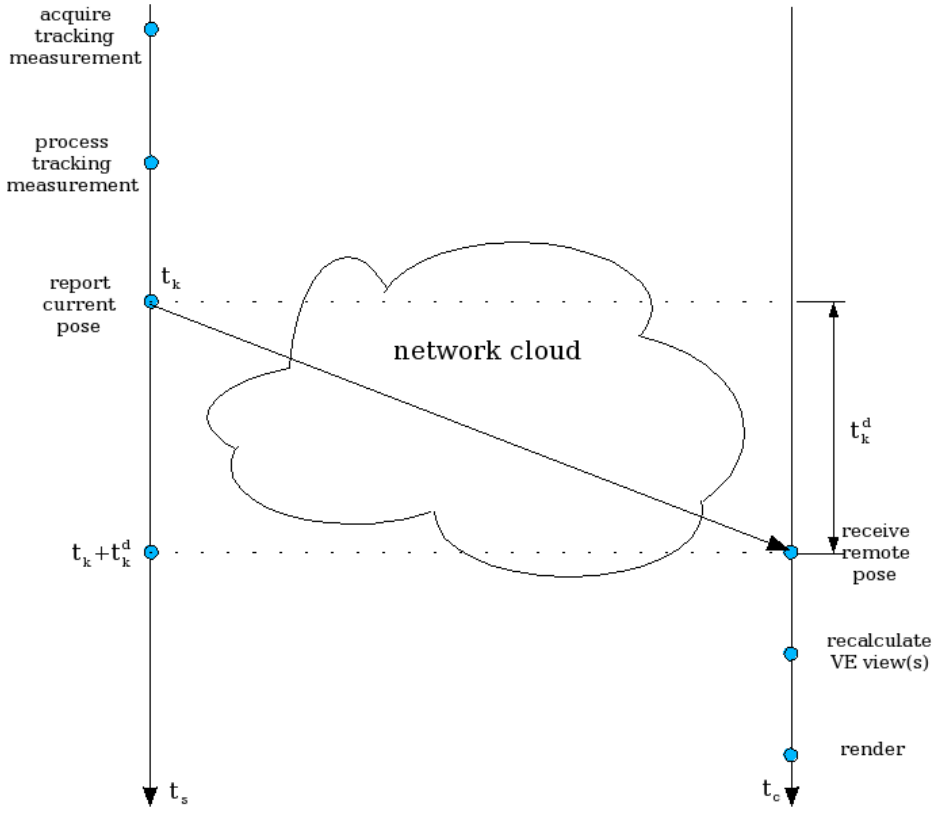

Figure 1.1: client-server VE application cycle: no prediction

stitutes a single cycle of a client-server type VE application without prediction. t_s and t_c represent the flow 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, and 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 a relatively intuitive idea: why broadcast an entity's current pose if it is destined to arrive outdated? We leverage participating entities' exclusive self-knowledge and access to their motion profile, event history, and a possible physical model of their motion trajectory. This allows us to prevent VE state inconsistency by having VE entities exchange their predicted pose instead of the currently measured one. Such approach departs from currently wide-spread way of correcting for the VE state inconsistencies due to network latency *a posteriori*. It conceptually modifies the cycle illustrated above through the introduction of pose and network delay estimator framework, as can be seen from figure 1.2.

Upon such modifications, client receives 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 $\hat{t_k^d} \approx t_k^d$, in which case the predicted pose will most closely match the current pose of the server-controlled entity, conditioned upon the pose estimator performance. It is accomplished through the introduction of a variability-aware framework, coupling a user pose predictor with network delay estimator to determine the prediction interval required for the former in order to produce the estimate of the

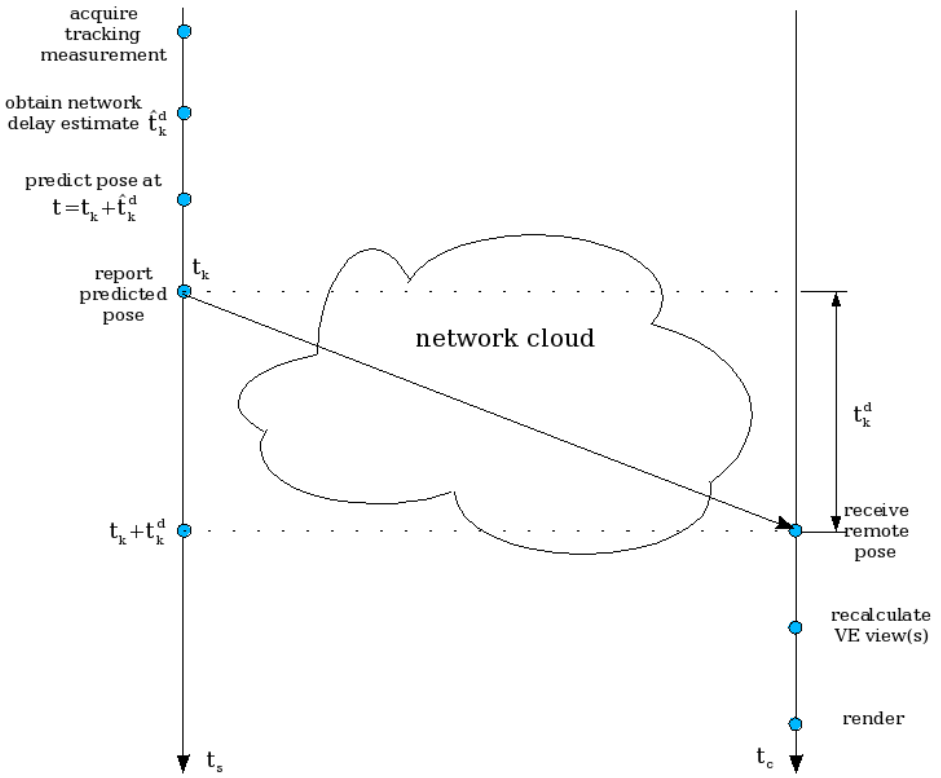

Figure 1.2: client-server VE application cycle: with prediction

future pose at the time of arrival.

Ultimately, our contribution is formed by a two-tier adaptive predictor framework offering the only proactive approach to the network latency amelioration sensitive to the variability in the network delay. Furthermore, we decouple the implementations of user pose and network delay predictors. That ensures modularity of our developmental efforts and, thus, enables the scientific community to plug and test numerous different prediction techniques for both estimators within our framework.

1.4 Thesis Outline

The remainder of the thesis is organized as follows. Chapter 2 presents an overview of research work related to this thesis. Chapters 3 and 4 provide a detailed treatment of the framework's pose estimation layer, focusing on position smoothing and prediction followed by the discussion of estimation and prediction for head orientation. The network delay estimation layer is detailed in chapter 5, while chapter 6 describes the simulation put together for the purposes of evaluating our approach to network delay amelioration. This chapter highlights the principal components of the simulator and their functionality. The results of the comparative evaluation and their discussion are presented in chapter 7. Finally,

chapter 8 concludes with a summary of contributions, their implications, and possible directions of future work.

2 Previous Work

Network delay has detrimental effects across a whole spectrum of immersive synthetic reality applications, ranging from tele-haptics, where a sense of touching remote objects is provided, to tele-operated robotics, where a robotic platform is remotely operated by a user, to collaborative VR. Therefore, mitigating or masking system delay effects is an active research topic. The research work relevant to the issue of delay amelioration ranges from providing evidence for the detrimental effects of system latency, to attempts to minimize that latency and its variability, to, finally, investigating various compensation techniques. The classification I 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 definition for the proactive methods can be borrowed from Merriam-Webster Online as "acting in anticipation of future problems, needs, or changes."

2.1 Reactive Latency Compensation

The most commonly used reactive compensation techniques derive from dead reckoning (DR) algorithms popularized by the well-known IEEE Standard for Distributed Interactive Simulation ([DIS96]). Dead reckoning attempts to solve two problems simultaneously: reduce bandwidth consumption and mitigate the effect of network delay by client-side prediction. The latter 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 in order to correct it. Subsequent smoothing at the client side to correct its pose estimate with the newly arrived correct pose only intensifies the reactive nature of this network delay compensation technique.

Furthermore, DR requires a smart client with computational abilities to perform pose prediction for all remotely operated entities, such as soldier avatars or simulated aircraft. Access to the knowledge of adequate motion models that describe the dynamic behavior of entities whose pose is to be extrapolated becomes

necessary as well. A need for such access, depending on the application, may raise some concerns of privacy or access to intellectual property. Finally, security clearance may also be required to have access to the code that programmatically reveals the movement characteristics of the aircraft or other combat vehicles. The knowledge of differential equations that describe their motion may reveal certain intricacies of their control systems, whose details are typically safely guarded.

Despite its disadvantages, approaches based on dead reckoning continue to have a strong presence in the gaming industry (e.g. [Aro97], [PW02b]). Network games represent an important area of research, since they are one of the most wide-spread examples of a Distributed Virtual Environment, where many inhabitants share the same virtual world and interact with each other on the regular basis. Furthermore, in the game world, masking the effects of latency is especially important due to the variety in the computer hardware and network connection speeds that the game engine is expected to accommodate. Consequently, some prominent work has been done in this area, particularly, by the developers at Valve Software, known for its Half-Life and Half-Life 2 game engines ([Ber01]). The approach taken to latency amelioration is basically a combination of dead reckoning and client-side interpolation. In addition to the reactive nature of the former, the latter actually increases the overall system latency, which Bernier duly

acknowledges in pointing out that this is a tradeoff between the visual smoothness of the simulation and the magnitude of delay.

The article also mentions the adoption of dead reckoning smoothing technique, referred to as time-warping in the game development realm, in order to reactively compensate for the differences in client-side prediction and the authoritative server pose update.

Aggarwal et al [ABK+04] were among a number of researchers found to question the suitability of dead reckoning for latency compensation. The authors conducted experiments to evaluate traditional dead reckoning approaches to sharing participants' states. Even relatively small network delays had substantial objective and subjective effects on the quality of the gaming experience that was attributed to perceptible inaccuracies in the real and rendered trajectories of user avatars in a shared VE. A simple extension to the traditional DR proposed was demonstrated to reduce the discrepancy at the expense of the need to synchronize local clocks for network delay calculation. The performance evaluation of both the traditional and timestamp-augmented DR-based compensation was carried out with an assumption of fixed amounts of delay.

Hikichi et al [HMA+02] addressed the effect of dead reckoning compensation on shared virtual environments augmented with a sense of touch. According to

[AZWS04], touch may be the most affected sensory modality when it comes to the "mismatch between motor action and simulated sensory feedback." Due to the tight coupling between input and output and the high update rate required, compensating for the latency in tele-haptic DVEs is particularly important. Hikichi et al, however, illustrated the unsuitability of dead reckoning for tele-haptic applications, especially, in the presence of latency jitter. The authors hypothesized a direct relationship between a relative increase in the number of threshold violations and the quality of DR compensation. The number of times a correct pose had to be dispatched due to the violation of the threshold was shown to rise when using dead reckoning under variable network conditions.

2.2 Proactive Latency Compensation

In light of the disadvantages presented for the reactive methods of latency amelioration, predictive compensation comes across as "the only viable approach to mitigating the consequences of delay" ([JAE00a]). Predictive compensation shifts focus to the class of proactive compensation techniques, which I further differentiate into the category of delay jitter insensitive and variability-aware methods of latency amelioration. The former category is the state-of-the-art approach at the time of writing, as proactive delay amelioration methods found in the liter-

ature assume, explicitly or implicitly, a constant delay. This manifests itself in a variety of ways, from setting up experiments with simulated constant delay to merely assuming stability in the delay in the extrapolation equations.

Wu & Ouhyoung, for instance, have done extensive work on predictive head tracking in [WO94], [WO00], and [WO95], where they compared relative performance improvements of the Grey System (GS) approach over Kalman Filter based prediction and simple linear extrapolation. The authors claim improvements in both the running time and prediction accuracy using the GS-based compensation method. The comparative evaluation, however, is carried out by fixing the prediction interval to constant values in [WO95], while [WO94] reports performing latency measurements for virtual objects of various complexity, which the authors later used as a lookup table for their grey system theory based head motion predictor.

Akatsuka and Bekey proposed a method of latency compensation with an Euler angle based head orientation model ([AB98]). In evaluation of their technique, the authors go as far as to derive a linear relationship between the end-to-end system delay and the complexity of a rendered scene based on three virtual models used. It is apparent, however, that, for a given scene or a scene of a given complexity, the delay is assumed to be constant. Specifically, the proposed com-

pensation technique was evaluated for prediction intervals set to 75 and 150ms.

Rather than measuring objective compensation performance, Jae Jung et al ([JAE00b]) focus on subjective perceptual performance evaluation. The goal in their work is to keep both the effect of latency and prediction artifacts under the threshold of the perceptible. The authors propose their own implementation of a Kalman Filter (KF) based head motion predictor with redefined cost functions for optimization of KF noise parameters. A constant prediction interval is adopted again, however, in the evaluation of their compensation technique's effect on human perception. 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 in [JAE00a] and for constant latencies ranging from 0 to 100ms [JAE00b].

Finally, in the discussion of the suitability of client-side prediction for games ([PW02b]), 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 in an attempt to ameliorate it.

Azuma's work is most notable for its apparent departure from the constant delay assumption through the derivation of closed form predictors that accept

the interval of prediction (or, equivalently, a timestamp in the near future) as an argument (see [Azu95], [AB94]). In his technique, head position and orientation were predicted at the time the renderer rasterizes the image to be displayed. Both the head tracker and the graphics engine were part of the same local platform with tight control on and the knowledge of various parts of the end-to-end system delay. Nevertheless, Azuma's latency compensation technique can be classified as variability aware, due to the ability of the predictor to generate a head pose an arbitrary amount of time into the future. The implementation of a clock synchronization mechanism and timestamp acquisition throughout the various portions of the tracker-to-the-screen pipeline enabled Azuma to deterministically compute the prediction interval required ([AB94]).

The variability-aware latency compensation framework offered in this thesis deviates from the work of Azuma and Bishop on three principal points. First, my framework operates in the context of distributed applications, inherently involving more than one workstation networked together. Secondly, due to the nature of public switched networks, they yield little, if any, control over the propagation delay incurred when traversing them. Finally, the network latency is significantly more variable compared to local processing delays, and the extent of variability can easily change within the running time of a single experiment or user session.

Handshake Interactive (HIT) has commercialized a similar compensation technique for tele-haptics ([AZWS04]). Both Azuma and HIT propose for the tracked user information to be substituted with its prediction just before the event takes effect. In Azuma's system it happens just prior to its use by the renderer, while in the HIT system, the prediction occurs immediately before the event takes effect on a remote force-feedback device. Furthermore, the interval of prediction is deterministically computed in the context of tight control over various parts of the black box the event has to traverse. Indeed, HIT determines the interval by measuring the network delay incurred by the packet at the receiving end. This requires tight synchronization of the event initiator and receiver, such as clock synchronization as well as client-side knowledge of the prediction model. The framework proposed in this thesis renders such requirements unnecessary by performing prediction at the sending end.

Azuma's later work on the frequency-domain analysis of head-motion prediction ([AB95]) highlighted the sensitivity of prediction to the magnitude of the prediction interval. The study theoretically evaluated two classes of head-motion prediction, specifically exploring their performance changes with variation of system parameters. One such parameter of interest is the interval of prediction. In fact, the error in the predicted signal was derived as a function of the latter,

allowing researchers to compare various predictors w.r.t. their error characteristics for *specified intervals of time*. The utility of Azuma's theoretical framework, however, is inherently limited to researchers who use prediction in either constant system delay environments or with the assumption of a constant delay, which is the de facto approach to proactive latency masking at the time of writing.

LaViola offers an entire testbed for the empirical evaluation of predictor performance (e.g. [LaV03b], [LaV03a], [LaV03d]). Both Azuma's theoretical framework and the predictive tracking algorithm testbed force a choice of a specific value for the latency for the evaluation to take place. This is especially true in the latter case, where the graphical user interface restricts the choice to a constant prediction interval.

A unifying theme bringing the state-of-the-art research work of Azuma, HIT, and LaViola on proactive delay compensation together is the continued influence of constant delay prediction assumption. The two-tier compensation framework presented in this thesis 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 latency jitter.

Finally, latency reduction is not to be overlooked. Minimization of the end-to-end system delay has been identified as an obvious first step by many authors,

most notably Azuma & Bishop ([AB94]), Wu & Ouhyoung ([WO94]), and Jae Jung et al ([JAE00b]). Azuma's work featured using a low-overhead operating system called VxWorks as well as direct communication paths between the tracker and the graphics engine. Wu & Ouhyoung point out that the implementation of a predictor itself incurs latency and, thus, in their work on the grey system theory based prediction, focus both on the minimization of predictor's running time and compensation for the remaining delay. Similarly to Azuma, Jung et al reduce latency through the provision of a dedicated communication path between the sensor and a stand-alone software process responsible for further dispatching of sensor events to other simulation processes. Their choice of the C programming language is also undoubtedly motivated by the optimization considerations leading to end-to-end system delay reduction.

3 Pose Estimation - Position

3.1 Background and Intuition

Generally, virtual environments contain objects of two major categories: entities that have no direct relationship to the objects in the real world and, thus, are synthetically generated, and those that do. In order to properly render the virtual representations of the latter, it is necessary to track the pose of the real-world objects they represent. Even when the entire VE is synthetic and exhibits no dynamic behavior, tracking a user's pose may still be desirable to ensure a certain level of interactivity with the VE, suitable for such applications as architectural walkthroughs and scientific visualization, to name a few.

The pose is typically defined as a combination of a tracked object's position in the motion space (two- and three-dimensional Eucledian spaces being most widespread) and its orientation. Pose measurements are subject to a variety of problems of both static and dynamic nature, such as high-frequency noise, mis-

calibration of a tracking device, and system delay. The measurements obtained, therefore, cannot be used directly to influence the recomputation of the virtual embodiment of a tracked object (usually referred to as an avatar) or the view of a VE an operator gets as a result of her translational and rotational motion. Hence the need for pose estimation.

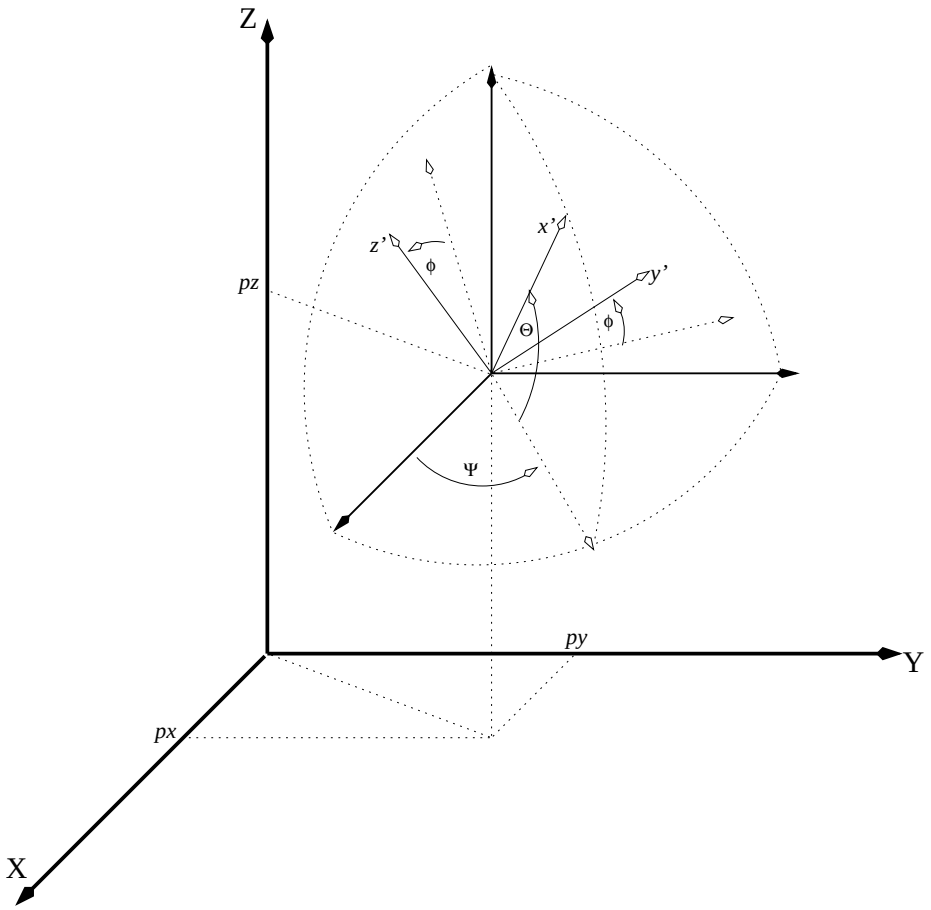

Figure 3.1: Position and orientation in Eucledian 3D space

Regarding pose representation, a rigid untethered object in 3D Eucledian space has six degrees of freedom, making it possible to translate along each of the 3 axes (X, Y, Z) as well as to rotate about them. Cartesian coordinates in a frame of reference defined by independent orthogonal axes are typically used to represent position. Orientation is more complicated and several candidate representations are common depending on application and mathematical apparatus chosen.

State vector notation is typically adopted to keep track of the pose. In addition to position and orientation concatenated together to form this vector, object's velocity and acceleration may also be of interest for dynamic modeling of the system. The state vector may, therefore, grow to as many as 18 elements (e.g. [Wel96]). For computational convenience, the dynamic information, therefore, was broken down into three component-wise positional state vectors and one for the orientation.

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 3.2 illustrates the model employed for the positional component of the pose. It reveals the separate treatment of individual positional components, yielding a state vector of only

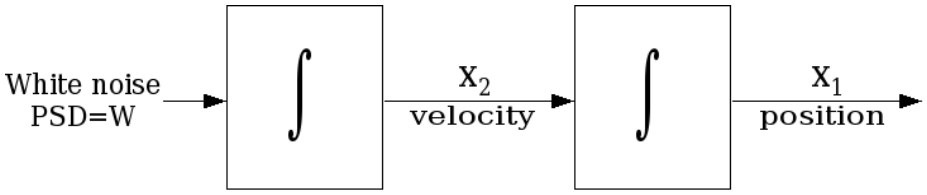

Figure 3.2: Process model for positional component of the system

two elements $\vec{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$. A tracked object is modeled as a system that accepts white noise as input, performing integration on it to obtain linear velocity. Position is then derived by integrating the velocity component.

Such model calls for a system of differential equations (DE) to mathematically formulate the system's dynamic behavior. Due to the inertia of tracked body parts, such as a user's head, an assumption can be introduced that linear acceleration and even velocity will not significantly change between two consecutive measurements. Provided that the sampling frequency of a tracking device is adequately large, this is a reasonable assumption to make.

3.2 Model Formulation

Due to the nature of human motion and devices typically employed to track it, 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:

$$\dot{X}(t) = FX(t) + Gw(t) + Lu(t) \tag{3.1}$$

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. Square matrix F is of particular importance here and is known as the system dynamics matrix. In general, the coefficients of eq. (3.1) may vary with time, but the time subscript has been dropped for notational convenience.

General Kalman Filter formulation is completed with a measurement equation that linearly relates the state X with a tracking measurement z through a transformation matrix H as follows:

$$z = HX(t) + v$$

where v refers to the measurement noise.

Since the behavior of the continuous-time system modeled is observed only at discrete points in time, we proceed to discretize the process model equation by borrowing the difference equation solution to (3.1) from Brown and Hwang

[BH97a]:

$$x_{k+1} = \Phi(t_{k+1}, t_k)x(t_k) + \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \tau)G(\tau)w(\tau)d\tau \tag{3.2}$$

The short form notation of the above is the familiar governing equation for discrete processes:

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

where Φ_k clearly serves as a transition matrix between consecutive states in time. Intrinsic to the above transformation is an assumption that observations of system's state will occur at regular intervals in time:

$$t_{k+1} - t_k = \Delta t \approx const \quad \Rightarrow \quad \Phi(t_{k+1}, t_k) = \Phi_k(\Delta t)$$

The plant equation introduced in its general continuous-time form is also rewritten to reflect the discrete nature of the tracking device operation, yielding a system of discrete-time linear equations as follows:

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

It can be solved resulting in a well-known recursive-step algorithm summarized in the diagram 3.3 (see [Gel74], [BH97a], [DJ00], [WB95] for derivation).

Due to a consistent observation that, for adequately small time intervals, linear velocity undergoes insignificant change, a constant velocity model was

adopted. The assumption of constant velocity translates into $\dot{p_x} = 0 \Rightarrow p_x = const$ and can be written as a system of linear homogeneous differential equations:

$$\begin{aligned} \dot{p_x} &= 0 \cdot p_x + \dot{p_x} \\ \ddot{p_x} &= 0 \cdot p_x + 0 \cdot \dot{p_x} \end{aligned} \Leftrightarrow \begin{bmatrix} \dot{p_x} \\ \ddot{p_x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ \dot{p_x} \end{bmatrix} = Fx = \dot{x}$$

which, in fact, yields a system dynamics equation in the absence of noise. To account for white noise input into the system, the above transforms as follows:

$$\dot{X}(t) = \begin{bmatrix} \dot{p_x} \\ \ddot{p_x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ \dot{p_x} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} w(t) \tag{3.3}$$

Further, it can be shown that fundamental matrix Φ_k can be found by evaluating a Taylor series expansion of matrix exponential $e^{F\Delta t}$ ([BH97a], [ZM00]):

$$\Phi_k = e^{F\Delta t} = I + F\Delta t + \frac{(F\Delta t)^2}{2!} + \frac{(F\Delta t)^3}{3!} + \ldots = I + F\Delta t = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \tag{3.4}$$

since, $\forall n \geq 2$, $\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}^n = 0$

H 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. Matrix $H_{1\times 2} = [1 \quad 0]$ then satisfies the following equality:

$$p_x = H \begin{bmatrix} p_x \\ \dot{p_x} \end{bmatrix} = [1 \quad 0] \begin{bmatrix} p_x \\ \dot{p_x} \end{bmatrix}$$

3.3 Process Noise Covariance Matrix

The previous section has crystallized the building blocks of the system dynamics and the measurement equations, deriving the fundamental matrix Φ and H. Process noise covariance matrix Q_k remains to be formulated and is defined as a variance-covariance of process noise w_k. Hence, Q_k equals $E[w_k w_k^T]$ by definition and encapsulates the degree of confidence we have in our model's proper reflection of the true underlying system behavior. There are two principal approaches to the analytical derivation of Q_k — from the definition presented or following Zarchan's approach [ZM00] and obtaining Q_k from continuous-time matrix $Q(t)$ as follows:

$$Q_k = \int_0^{\Delta t} \Phi(\tau) Q \Phi^T(\tau) d\tau \tag{3.5}$$

Continuous-time process noise covariance matrix Q is set by Zarchan and, subsequently, by LaViola in [LaV03a] to $W \begin{bmatrix} 0 & 0 \\ \\ 0 & 1 \end{bmatrix}$, reflecting the widely accepted assumption that most process noise enters the system at its highest derivative. It may not be immediately clear how this assumption translates into the mentioned expression for Q, nor whether the two approaches to Q_k derivation are equivalent. The purpose of this section is to prove their equivalence and analytically arrive at the closed-form expression for Q_k.

3.3.1 Proof of Equivalence

Starting with the definition of $Q_k = E[w_k w_k^T]$, we substitute in the value of $w_k = \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \tau) G(\tau) w(\tau) d\tau$ based on eq. (3.2). It follows then that

$$Q_k = E\left\{ \left[\int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \xi) G(\xi) w(\xi) d\xi \right] \left[\int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \eta) G(\eta) w(\eta) d\eta \right]^T \right\}$$

Since the transpose of an integral is equal to the integral of the transpose, Q_k can be rewritten as follows:

$$Q_k = E\left\{ \int_{t_k}^{t_{k+1}} \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \xi) G(\xi) w(\xi) \left[\Phi(t_{k+1}, \eta) G(\eta) w(\eta) \right]^T d\xi d\eta \right\}$$

$$Q_k = E\left\{ \int_{t_k}^{t_{k+1}} \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \xi) G(\xi) w(\xi) w^T(\eta) G^T(\eta) \Phi^T(t_{k+1}, \eta) d\xi d\eta \right\}$$

Keeping in mind that the expected value of a deterministic component is that deterministic component, we proceed to rewrite the above as

$$Q_k = \int_{t_k}^{t_{k+1}} \int_{t_k}^{t_{k+1}} E\left\{ \Phi(t_{k+1}, \xi) G(\xi) w(\xi) w^T(\eta) G^T(\eta) \Phi^T(t_{k+1}, \eta) \right\} d\xi d\eta$$

$$\Rightarrow Q_k = \int_{t_k}^{t_{k+1}} \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \xi) G(\xi) E\left\{ w(\xi) w^T(\eta) \right\} G^T(\eta) \Phi^T(t_{k+1}, \eta) d\xi d\eta \quad (3.6)$$

At this point, the reader is referred to appendix A to see that the inner expected value term actually equals to $W\delta(\xi - \eta)$, where W is the power spectral density of the white noise process. Performing this substitution, we get

$$Q_k = \int_{t_k}^{t_{k+1}} \left[\int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \xi) G(\xi) W \delta(\xi - \eta) d\xi \right] G^T(\eta) \Phi^T(t_{k+1}, \eta) d\eta$$

The inner integral above readily yields itself to the sifting property of the Dirac delta function, becoming

$$\int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \xi) G(\xi) W \delta(\xi - \eta) d\xi = \Phi(t_{k+1}, \eta) G(\eta) W$$

making a mental note that

$$\xi, \eta \in [t_k; t_{k+1}] \Rightarrow \exists\ \xi \in [t_k; t_{k+1}] : (\xi - \eta) = 0$$

This simplification allows us to rewrite Q_k as follows:

$$Q_k = \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \eta) G(\eta) W G^T(\eta) \Phi^T(t_{k+1}, \eta) d\eta$$

Substituting $G(\eta)$ with its value from eq. (3.3), the product of three innermost terms becomes

$$G(\eta) W G^T(\eta) = W \begin{bmatrix} 0 \\ \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix} = W \begin{bmatrix} 0 & 0 \\ \\ 0 & 1 \end{bmatrix} = Q$$

which is the continuous-time process noise covariance matrix used by Zarchan and LaViola.

$$\Rightarrow Q_k = \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \eta) Q \Phi^T(t_{k+1}, \eta) d\eta$$

At this point recall that for our system, the fundamental matrix $\Phi(t_{k+1}, t_k)$ was derived as a function of the difference $\Delta t = t_{k+1} - t_k$ under the assumption

that $\Delta t \approx const$. Therefore, we rewrite the expression for Q_k above to reflect that:

$$\Rightarrow Q_k = \int_{t_k}^{t_{k+1}} \Phi(t_{k+1} - \eta) Q \Phi^T(t_{k+1} - \eta) d\eta$$

Performing substitution $\left| \begin{array}{rcl} \tau & = & t_{k+1} - \eta \\ \eta & = & t_{k+1} - \tau \end{array} \right|$, we finally get

$$Q_k = \int_{t_{k+1}-t_k}^{0} \Phi(\tau) Q \Phi^T(\tau)(-1) d\tau = \int_0^{\Delta t} \Phi(\tau) Q \Phi^T(\tau) d\tau$$

■

3.3.2 Q_k derivation

Having established the equivalence of the two mentioned methods for Q_k derivation, either can be chosen for that purpose. We'll use eq. (3.5), substituting $\Phi(\tau)$ with the expression derived for the fundamental matrix in eq. (3.4):

$$Q_k = \int_0^{\Delta t} \Phi(\tau) Q \Phi^T(\tau) d\tau = W \int_0^{\Delta t} \begin{bmatrix} 1 & \tau \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \tau & 1 \end{bmatrix} d\tau =$$

$$= W \int_0^{\Delta t} \begin{bmatrix} \tau^2 & \tau \\ \tau & 1 \end{bmatrix} d\tau = W \begin{bmatrix} \frac{(\Delta t)^3}{3} & \frac{(\Delta t)^2}{2} \\ \frac{(\Delta t)^2}{2} & \Delta t \end{bmatrix}$$

This completes the formulation of the principal blocks for the positional Kalman Filter.

3.4 Positional KF Algorithm

Positional Kalman Filter recursive-step algorithm can now be summarized as follows:

I. Initialization:

1) set state $\hat{x}_0 = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \text{first measured position} \\ 0 \end{bmatrix}$

2) set covariance matrix $P_0 = \begin{Bmatrix} 0, & \forall\, i \neq j \\ 100^{1}, & \forall\, i = j \end{Bmatrix}$

II. KF procedure

For each received k^{th} measurement do:

1) Time update

a. compute fundamental matrix: $\Phi_k = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$

b. compute process noise covariance matrix:

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

[1]Large numbers are used to trust the first couple of measurements more than the model. E.g. see [LaV03a] for KF initialization details.

c. obtain *a priori* estimates of state x and covariance P

$$\begin{aligned} \hat{x}_k^- &= \Phi_k \hat{x}_{k-1} \\ P_k^- &= \Phi_k P_{k-1} \Phi_k^T + Q_k \end{aligned}$$

2) Measurement update

a. compute Kalman Gain

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$$

b. update the estimate of state

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

c. update covariance P: $P_k = (I - K_k H) P_k^-$

III. KF prediction

1) set $\Phi_{pred} = \begin{bmatrix} 1 & T_{pred} \\ 0 & 1 \end{bmatrix}$

2) $x_p = \Phi_{pred} \hat{x}_k$

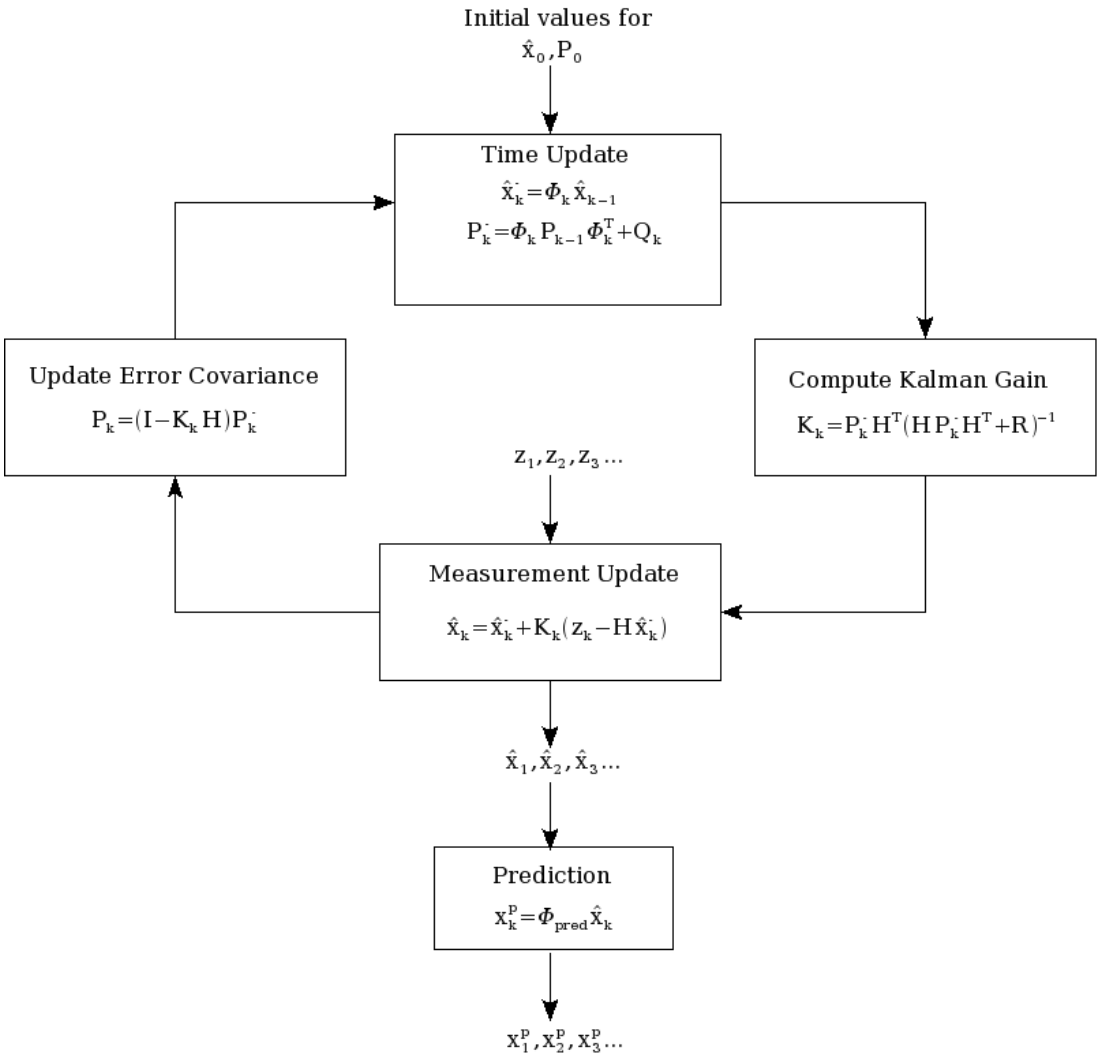

Figure 3.3: Recursive KF prediction/correction cycle

4 Pose Estimation - Orientation

Head orientation yields itself to an even greater variety of representation than position and subsequently dictated mathematical apparati to model it. Johnson has a fairly recent treatment of common techniques for mathematical modeling of rigid body orientation in CG animation [Joh03], including the coordinate matrix, axis-angle, Euler angles, and the quaternion approach (see also [DKL98]). Rodriguez, modified Rodriguez parameters, and direction cosine representations are closely related to Euler angles (see [CM96] for their treatment), but are seldom used in Virtual Reality and tracking applications. In this thesis, I use the 4D quaternion representation approach to avoid singularity problems inherent to 3D parameterizations. Quaternions are defined as hyper-complex numbers of the form

$$q = q_0 + \mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3 = q_0 + \vec{q}$$

where q_0 is a scalar part of the quaternion and $\vec{q}$ — its vector part, and the following relationship between the imaginary numbers holds:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$$

Quaternions become increasingly common in tracking applications as they form "the minimal non-singular attitude parameterization" [CBIO04]. They offer an intuitive geometric interpretation of rotational motion, a solution to the gimbal lock singularity problem, and elegant interpolation mechanisms that prove especially useful in our statistical analysis of Extended Kalman Filter (EKF) performance. Furthermore, they have successfully been used by Azuma [Azu95] for the same purpose and yield a relatively simple governing differential equation.

4.1 Model Formulation

As with the positional Kalman Filter described in the previous chapter, a constant velocity model is adopted, approximating a head as a rigid body with inertia sufficient to suppress most of the high-frequency content in its motion profile. Furthermore, it is reasonable to assume that angular velocity will remain relatively constant during short intervals of time between tracker samples. Finally, higher order models, such as constant acceleration, are not used, as the

tracking device does not furnish measurement data for the derivative elements of the state.

Their advantages notwithstanding, the use of quaternions introduces non-linearity into the model equation. A general form of non-linear governing equation in the absence of deterministic control input is as follows:

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

and should be regarded as a generalization of eq. (3.1).

In our model, the state vector X will contain parameters responsible for keeping track of orientation and angular velocity and, therefore, it is commonly referred to as the orientation-velocity (OV) model. The orientation state is maintained by storing all four components of the orientation quaternion $q = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$. Angular velocity is appended to the state vector on a per-component basis, resulting in a state vector $X = [q_w\ q_x\ q_y\ q_z\ w_0\ w_1\ w_2]^T$.

Kuipers ([Kui99]) provides an excellent and intuitive derivation of a quaternion derivative, giving rise to what has become the most commonly used governing differential equation for head orientation:

$$\dot{q} = \frac{1}{2}\, q \otimes w \tag{4.1}$$

where $\otimes$ represents quaternion multiplication. For quaternions $p = (p_0, \vec{p})$ and

$q = (q_0, \vec{q})$, written as a combination of their scalar and vector parts, their product can be shown to equal to the following:

$$p \otimes q = (p_0 q_0 - \vec{p} \cdot \vec{q}, \quad p_0 \vec{q} + q_0 \vec{p} + \vec{p} \times \vec{q})$$

The quaternion product can also be rewritten as a matrix-vector multiplication, if quaternions themselves are represented in the form of a 4D vector $[q_0 \ q_1 \ q_2 \ q_3]^T \in \mathbb{R}^4$, where $[q_1 \ q_2 \ q_3]^T = \vec{q}$:

$$p \otimes q = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

Eq. (4.1) can, therefore, be rewritten as

$$\dot{q} = \frac{1}{2} \begin{bmatrix} 0 & -w_0 & -w_1 & -w_2 \\ w_0 & 0 & w_2 & -w_1 \\ w_1 & -w_2 & 0 & w_0 \\ w_2 & w_1 & -w_0 & 0 \end{bmatrix} \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} = M \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} \tag{4.2}$$

noting that w for this differential equation is represented as a pure quaternion $w = [0 \ w_0 \ w_1 \ w_2]^T$. This may give a false feeling of the possibility of a homogeneous linear ODE for our model. Indeed, rewriting the above, now with the state vector

X instead of q, we get

$$\dot{X} = \begin{bmatrix} M & 0_{4\times3} \\ 0_{3\times4} & 0_{3\times3} \end{bmatrix} X \tag{4.3}$$

Eq. (4.3) is a governing equation for the OV model in the absence of noise. The last three rows of zeros are due to the model's assumption of angular velocity $\vec{w} = \begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix}^T = const \Rightarrow \vec{w}' = \vec{0}$. The non-linearity becomes apparent noting that matrix M is actually a function of angular velocity state vector components, which cannot be factored out. With the addition of process noise to eq. (4.3), we get the complete governing model equation:

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

where vector-valued $f : \mathbb{R}^7 \mapsto \mathbb{R}^7$ is piecewise defined as follows:

$$\begin{cases} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} \otimes \begin{bmatrix} 0 \\ w_0 \\ w_1 \\ w_2 \end{bmatrix} \\ \begin{bmatrix} f_5 \\ f_6 \\ f_7 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \end{cases} \iff f : \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \\ w_0 \\ w_1 \\ w_2 \end{bmatrix} \mapsto \begin{bmatrix} -\frac{1}{2}w_0q_x - \frac{1}{2}w_1q_y - \frac{1}{2}w_2q_z \\ \frac{1}{2}w_0q_w + \frac{1}{2}w_2q_y - \frac{1}{2}w_1q_z \\ \frac{1}{2}w_1q_w - \frac{1}{2}w_2q_x + \frac{1}{2}w_0q_z \\ \frac{1}{2}w_2q_w + \frac{1}{2}w_1q_x - \frac{1}{2}w_0q_y \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Non-linearity is introduced into the measurement equation as well. Here,

though, it's due to the normalization of the quaternion part of the state by function $h : \mathbb{R}^7 \mapsto \mathbb{R}^4$, piecewise defined as follows:

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

h establishes the relationship between the state and the tracker measurements obtained and allows for meaningful calculation of a measurement residual through the use of explicitly normalized unit quaternions. With the introduction of possible measurement noise into the system, the plant equation then becomes

$$Z = h(X) + v(t)$$

We end up with a system with both nonlinear dynamics and nonlinearity in the measurement equation and need to linearize it in order to be able to use the Kalman filtering framework introduced in the positional Kalman Filter chapter. Various methods of linearization exist, most notably, linearization about the known nominal trajectory, as would be the case for satellites, and the trajectory estimated online [BH97a]. The latter has come to be known as the Extended Kalman Filter and is most applicable for head motion tracking, where it is impossible to predict or otherwise precompute the trajectory of its motion.

Linearization rests on the assumption that the nominal or estimated trajectory closely approximates the real one.

4.2 Measurement Update

Brown and Hwang offer one such method of linearization by formulating the extended Kalman Filter as an extension to the linearized KF [BH97b]. Linearization about the estimated trajectory of motion gives rise to the use of 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.

This section derives a closed-form expression for the plant equation matrix H used in the measurement update stage of the predictive-corrective Kalman Filter cycle. This stage calls for special emphasis in the EKF case, as it introduces an important change in the calculation of the measurement residual, defined as the difference between the actual measurement Z_k and its predictive estimate. The latter is computed as follows:

$$\hat{Z}_k^- = H\hat{X}_k^- = \frac{\delta h}{\delta X} \cdot \hat{X}_k^- = h(\hat{X}_k^-)$$

in the linear KF case, whereas the equality $\frac{\delta h}{\delta X} \cdot \hat{X}_k^- = h(\hat{X}_k^-)$ in general breaks down as soon as the function h becomes non-linear. Therefore, it is important to

emphasize that employing the KF measurement update equations verbatim for the non-linear case will result in an erroneous update of the state. Hence, the measurement update equations appropriate for the Extended Kalman Filter are included here for completeness:

$$\begin{aligned} K_k &= P_k^- H^T (H P_k^- H^T + R)^{-1} \\ \hat{X}_k &= \hat{X}_k^- + K_k (Z_k - h(\hat{X}_k^-)) \\ P_k &= (I - K_k H) P_k^- \end{aligned} \tag{4.4}$$

This thesis is believed to be the first to expose this measurement residual computation discrepancy in LaViola's recent testbed implementation for empirical comparison of predictive tracking algorithms [LaV03b].

H in the measurement update equations (4.4) is calculated as previously mentioned to be

$$H = \frac{\delta h}{\delta X} = \begin{bmatrix} \frac{\delta h_1}{\delta q_w} & \frac{\delta h_1}{\delta q_x} & \frac{\delta h_1}{\delta q_y} & \frac{\delta h_1}{\delta q_z} & \frac{\delta h_1}{\delta w_0} & \frac{\delta h_1}{\delta w_1} & \frac{\delta h_1}{\delta w_2} \\ \vdots & & & & & & \vdots \\ \frac{\delta h_4}{\delta q_w} & & & \cdots & & & \frac{\delta h_4}{\delta w_2} \end{bmatrix}$$

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:

$$H = -\frac{1}{L^{3/2}} \begin{bmatrix} q_w^2 - L & q_w q_x & q_w q_y & q_w q_z & 0 & 0 & 0 \\ q_w q_x & q_x^2 - L & q_x q_y & q_x q_z & 0 & 0 & 0 \\ q_w q_y & q_x q_y & q_y^2 - L & q_y q_z & 0 & 0 & 0 \\ q_w q_z & q_x q_z & q_y q_z & q_z^2 - L & 0 & 0 & 0 \end{bmatrix} \tag{4.5}$$

A closed-form analytical expression for H puts us in a position to investigate the severity of improper measurement residual calculation effects and, in fact, to show that, for our model, $H \cdot X$ actually equals to a zero vector:

$$H \cdot X = \vec{0}$$

To take it one step further, I prove that

$$\left.\frac{\delta h}{\delta X}\right|_{X=\hat{X}_k^-} \cdot \hat{X}_k^- = \vec{0} \tag{4.6}$$

even if the ordering of quaternion components is altered, as happens to be the case in LaViola's testbed implementation. Specifically, the scalar and vector components are swapped, resulting in $quat = [q_x \ q_y \ q_z \ q_w]^T$. Just a few lines of Matlab code will convince the inquisitive reader that eq. (4.6) holds true:

```
syms w0 w1 w2 qw qx qy qz real
q=[qx qy qz qw];
state=[q w0 w1 w2];
h=q/sqrt(dot(q, q)); %normalization
H=jacobian(h', state);
simplify(H*state')
```

Performing the above symbolic computation results in a zero vector. The consequences of this flaw are significant, as it follows that the state update step will correct the a priori estimate not by the weighted measurement residual, but by the weighted measurement, which is meaningless. It casts a shadow of doubt on a whole list of publications that build on the performance of EKF as reported by LaViola's predictive tracking algorithm testing suite: [JL04], [LaV03c], [LaV03a], [LaV03d], [LaV02].

4.3 Time Update

4.3.1 State Projection

Time update equations, specifically, the a priori state estimate, require a solution to non-linear differential equation (4.3) that can be obtained numerically employing 4^{th} order Runge-Kutta (RK4) numerical integration [Zil97].

For the quaternion component, the right hand side of eq. (4.1) will have to be evaluated at four different points, with the results forming a weighted additive term as follows:

$$q_k = q_{k-1} + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

To simplify notation, we form two quaternions from the components of $\hat{X}_{k-1} =$

$[q_w\ q_x\ q_y\ q_z\ w_0\ w_1\ w_2]_{k-1}^T$ to be $q = [q_w\ q_x\ q_y\ q_z]^T$ and $w = [0\ w_0\ w_1\ w_2]^T$. Any manipulation of q and w is strictly in accordance with quaternion algebra. Then the above $k_i, i = 1..4$ are computed as follows:

$$\begin{aligned} k_1 &= \tfrac{\Delta t}{2} \cdot q \otimes w \\ 2k_2 &= \Delta t \cdot \left[q + \tfrac{1}{2}k_1\right] \otimes w &= k_2' \\ 2k_3 &= \Delta t \cdot \left[q + \tfrac{1}{2}k_2\right] \otimes w &= \Delta t \cdot \left[q + \tfrac{1}{4}k_2'\right] \otimes w = k_3' \\ k_4 &= \Delta t \cdot \tfrac{1}{2}\left[q + k_3\right] \otimes w &= \tfrac{\Delta t}{2} \cdot \left[q + \tfrac{k_3'}{2}\right] \otimes w \end{aligned} \tag{4.7}$$

and $q_k = q + \frac{1}{6}(k_1 + k_2' + k_3' + k_4)$. Δt is the step size over which the integration takes place.

The angular velocity portion of the a priori state estimate is set to its value on the previous step, consistent with the OV model:

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}_k = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}_{k-1}$$

The state projection step has, thus, been established:

$$\hat{X}_k^- = \left\{ \begin{array}{l} \text{Numeric solution to} \\ \text{IVP (4.3) at } t = t_k \text{ subject} \\ \text{to } X = \hat{X}_{k-1} \text{ at } t = t_{k-1} \end{array} \right\} \tag{4.8}$$

4.3.2 Covariance Projection

A priori estimate of covariance matrix P_k^- is found similarly to the positional KF and is obtained as follows:

$$P_k^- = \Phi_k P_{k-1} \Phi_k^T + Q_k$$

This covariance projection step necessitates the knowledge of a fundamental matrix Φ_k as well as the process noise covariance matrix Q_k. It is possible to obtain a closed form expression for Φ_k for some low-order dynamic models through an inverse Laplace transform $\mathcal{L}^{-1}\left[(sI - F)^{-1}\right]$ ([BH97a], [ZM00]). I, however, remained consistent in resorting to the same Taylor series approximation of $\Phi_k = e^{F\Delta t} \approx I + F\Delta t$ (see eq. (3.4)).

For head orientation EKF

$$F = \frac{\delta f}{\delta X} = \begin{bmatrix} \frac{\delta f_1}{\delta q_w} & \frac{\delta f_1}{\delta q_x} & \cdots & \frac{\delta f_1}{\delta w_2} \\ \vdots & & & \vdots \\ \frac{\delta f_7}{\delta q_w} & \frac{\delta f_7}{\delta q_x} & \cdots & \frac{\delta f_7}{\delta w_2} \end{bmatrix}$$

and the resulting expression for Φ_k has been derived to equal to the following:

$$\Phi = \begin{bmatrix} 1 & -\frac{1}{2}\Delta t w_0 & -\frac{1}{2}\Delta t w_1 & -\frac{1}{2}\Delta t w_2 & -\frac{1}{2}\Delta t q_x & -\frac{1}{2}\Delta t q_y & -\frac{1}{2}\Delta t q_z \\ \frac{1}{2}\Delta t w_0 & 1 & \frac{1}{2}\Delta t w_2 & -\frac{1}{2}\Delta t w_1 & \frac{1}{2}\Delta t q_w & -\frac{1}{2}\Delta t q_z & \frac{1}{2}\Delta t q_y \\ \frac{1}{2}\Delta t w_1 & -\frac{1}{2}\Delta t w_2 & 1 & \frac{1}{2}\Delta t w_0 & \frac{1}{2}\Delta t q_z & \frac{1}{2}\Delta t q_w & -\frac{1}{2}\Delta t q_x \\ \frac{1}{2}\Delta t w_2 & \frac{1}{2}\Delta t w_1 & -\frac{1}{2}\Delta t w_0 & 1 & -\frac{1}{2}\Delta t q_y & \frac{1}{2}\Delta t q_x & \frac{1}{2}\Delta t q_w \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.9}$$

k^{th} subscript has been dropped for notational convenience. Φ_k or, rather, F_k, is evaluated at $X = \hat{X}_k^-$ obtained from (4.8) since we linearize about the estimated motion trajectory as discussed in the previous section.

The process noise covariance matrix derivation is driven by the same assumption as for the positional KF, namely, that most process noise enters the model at its highest derivative. Zarchan's approach to Q_k derivation has been shown to follow directly from the definition of $Q_k = E[w_k w_k^T]$ in section 3.3, and, therefore, will be used for the EKF as well. Given a seven element state vector $X = [q_w\ q_x\ q_y\ q_z\ w_0\ w_1\ w_2]^T$, with $[w_0\ w_1\ w_2]^T = w$ denoting the highest derivative components, the continuous-time process noise covariance matrix is

constructed as follows:

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

where W signifies the spectral density of white noise entering the system. Further, following eq. (3.5), Q_k is derived as follows:

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

As a result of this integration, Q_k takes shape of the following 7x7 matrix:

$$Q_k = \frac{\Delta t^2}{4} \cdot \tag{4.10}$$

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

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 OV model process noise covariance matrix has never appeared in the literature. This contribution concludes the derivation and description of all the necessary construction blocks for the EKF recursive step algorithm.

4.4 Prediction

As developed in the previous section, the EKF provides a means to estimate orientation through a recursive predictor-corrector cycle working with the OV model and a sequence of measurements made available to it at discrete, relatively evenly spaced points in time. The result is the estimator's best guess of the current quaternion attitude. The two-layer predictor-estimator framework proposed, however, necessitates predicting the user pose a variable time into the future. For linear models, such as the positional constant velocity model, this prediction step is carried out identically to the projection of the state during the time update, namely $x_p = \Phi_{pred}\hat{x}_k$, where Φ_{pred} is set in accordance with the prediction interval T_{pred} (see section 3.4).

For orientation prediction, Azuma offers an elegant closed form solution to eq. (4.2) (e.g. [Azu95], [AB94]):

$$Q_p = \left[I \cos d + \frac{M(t_k)}{d} \sin d \right] Q_{t_k}$$

where d is a substitution variable calculated as a second norm of the vector $\frac{1}{2}\int_0^{T_{pred}} w\mathrm{d}t$. For the orientation-velocity model,

$$w = const \Rightarrow \frac{1}{2}\int_0^{T_{pred}} w\mathrm{d}t = \frac{1}{2}T_{pred}\cdot w = \frac{1}{2}T_{pred}\cdot \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

Then d becomes $\frac{1}{2}T_{pred}\sqrt{w_0^2 + w_1^2 + w_2^2} = \frac{1}{2}T_{pred}\cdot \mathsf{norm}(\mathsf{w}, 2)$.

$M(t_k)$ is constructed according to equation (4.2) with w_0, w_1, w_2 taken from the current state estimate $\hat{X}_k$. The predicted quaternion is then explicitly normalized. With only a few lines of Matlab code to implement it, Azuma's solution comes across as an attractive option. The results obtained using this approach in terms of the RMS error were rather disappointing, however. I hypothesize that one of the reasons could be the division by d, which takes upon values very close to zero. For motion datasets I worked with, angular velocity components happen to oscillate about zero, in some cases not exceeding the range of ± 0.6. The norm of such a near-zero vector is then further multiplied by $T_{pred} \ll 1$.

Therefore, RK4 was favoured over Azuma's closed-form solution in the computation of the state's predictive estimate. In doing so, I maintained consistency with positional KF algorithm in a sense of using state propagation step for prediction as well. Naturally, while employing (4.7) to generate a future pose, T_{pred}

was used in lieu of Δt.

4.5 EKF algorithm summary

EKF based head orientation predictor algorithm can now be summarized as follows:

I. Initialization

1) set state $\hat{X}_0 = [q_w \; q_x \; q_y \; q_z \; w_0 \; w_1 \; w_2]^T = \begin{bmatrix} \text{first measured orientation} \\ 0 \\ 0 \\ 0 \end{bmatrix}$

2) set covariance matrix $P_0 = \begin{bmatrix} I_{4\times4} & 0_{4\times3} \\ 0_{3\times4} & 100 \cdot I_{3\times3} \end{bmatrix}$[1]

3) compute measurement noise covariance matrix R based on available offline data

II. EKF Procedure

For each received k^{th} measurement do:

1) Time Update

[1]Similarly to the positional KF algorithm in section 3.4, large numbers are used to trust the first few tracker measurements over the model.

a. advance the state using RK4 (see (4.7)):

$$\hat{X}_k^- = \left\{ \begin{array}{l} \text{Numeric solution to} \\ \text{IVP (4.3) at } t = t_k \text{ subject} \\ \text{to } X = \hat{X}_{k-1} \text{ at } t = t_{k-1} \end{array} \right\}$$

b. compute fundamental matrix Φ_k according to (4.9)

c. compute process noise covariance matrix Q_k according to (4.10)

d. obtain a priori estimate of the EKF convariance matrix

$$P_k^- = \Phi_k P_{k-1} \Phi_k^T + Q_k$$

2) Measurement Update

a. compute H according to (4.5)

b. compute Kalman Gain

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$$

c. form a $Z_{4\times 1}$ measurement vector

d. update the state

$$\hat{X}_k = \hat{X}_k^- + K_k(Z_k - h(\hat{X}_k^-))$$

e. update covariance P: $P_k = (I - K_k H) P_k^-$

3) Normalize the quaternion part of the state explicitly

III. Prediction

1) form $q_k = [q_w \ q_x \ q_y \ q_z]^T$ and $w_k = [0 \ w_0 \ w_1 \ w_2]^T$ from $\hat{X}_k$

2) set $\Delta t = T_{pred}$

3) obtain q_p using RK4

4) explicitly normalize q_p

5 Network Delay Estimation

5.1 Introduction

Network delay estimation is the lower of the two layers of our latency amelioration framework (see figure 5.1). Its primary purpose is to maintain an estimate

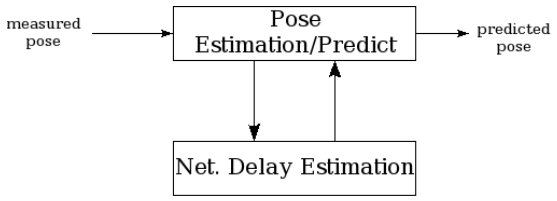

Figure 5.1: Two-layer latency amelioration framework

of network propagation delay between communicating entities sharing a virtual environment. As user poses change, the difference in pose, user commands bringing them about, or the current pose itself are encapsulated into datagrams and shared over the network with other participants. Such information exchange makes network propagation delay an inevitable part of DVE applications.

Distribution of a VE state over a packet switched network suffers from a host of inherent problems, such as apparent lag, variability in that lag, packet loss, out-of-order packet arrival at the receiving end, and variability in the packet interarrival time. Depending on the nature of an over-the-net application and its quality of service (QoS) requirements, the relative influence and detrimental effects of the above mentioned factors vary. Network latency and its variability have been shown to rank among the most influential network aspects for the realm of distributed interactive virtual environments (e.g. see [PK99], [PW02a]). Figure 5.2 illustrates these concepts, highlighting the subtle difference between latency and jitter.

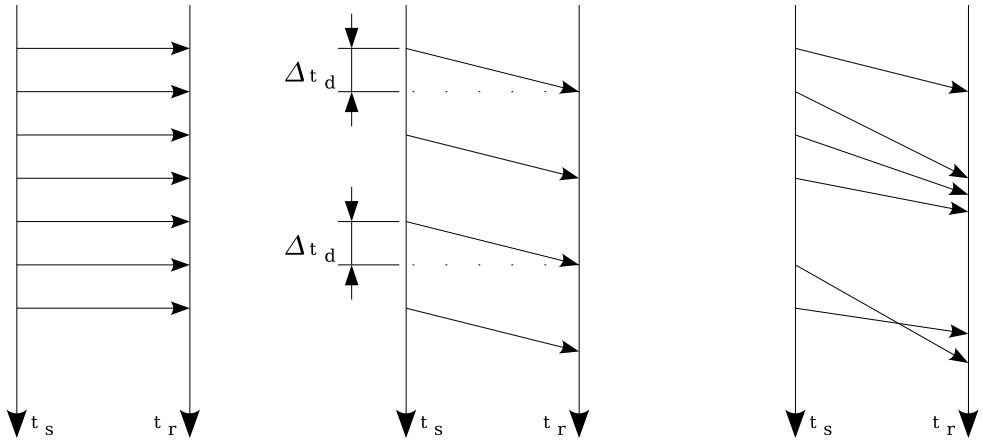

Figure 5.2: Message passing under a)no latency b)constant latency c)latency with jitter

The first of the three timelines illustrates a physically impossible scenario, where a network system exhibits no latency. The time delay between a dispatched event and its effect in such case is essentially zero. A real system may, however, be classified as zero-latency if the time-delay happens to be under the threshold of the perceptible. The second subfigure demonstrates a system with a fixed delay ($\Delta t_d = const$), while in the third one, the time it takes for an event to take effect varies. Networked systems are the most vivid examples of variable time-delay due to the variability in propagation time, queuing delays, and end-point processing typically incurred by transmitted packets. As previously mentioned in the motivation section, latency and jitter, characteristic of the last two types of systems, are detrimental to CVE operator performance. Additionally, they negatively influence the enjoyment and tolerance of shared VEs, forming the foundation for our quest toward a variability-aware latency compensation technique.

5.2 Motivation

Three radically different approaches to incorporating the knowledge of network delay into a distributed simulation naturally yield themselves:

1. obtain a priori network delay information and use it for subsequent simulation runs

2. measure network delay upon each packet's arrival at the receiving end

3. estimate network delay before each packet's departure from the sender

5.2.1 Predetermined Network Delay

The advantages of the first approach are apparent from its simplicity. Indeed, network delay data can be collected over the course of a representative period of time and further processed to calculate mean network delays for different times of day, for instance. It would yield an a priori estimate of delay to use with subsequent runs of a distributed simulation — an estimate typically held constant for the entire duration of the application run. Such approach would provide the pose predictor with a required prediction interval to use, avoiding the necessity for online network delay estimation. Furthermore, given an analytically derived expression for pose prediction as a function of a prediction interval, it would be possible to precompute it, substituting in the value of the latter, which saves valuable processing time at runtime.

It must be noted, however, that prior computation of network delay may work well only for systems with constant or near-constant network latency, as illustrated by figure 5.2b). Stable network delay on publicly accessible non-dedicated networks is highly unlikely, where the traffic is expected to "vary widely and dy-

namically over the course of a [single] connection" [WS95]. Therefore, even more sophisticated examples of the first approach, such as precomputation of the entire sequence of round-trip time (RTT) delays based on past network performance, are not expected to succeed. Yet, the assumption of constant network delay is prevalent in DIVE implementations today and, therefore, was included in our comparison study.

5.2.2 Client-Side Measurement Approach

If network latencies cannot be determined a priori or in light of the disadvantages of this approach, they must be dealt with at run-time. Measuring network propagation time is a distinct possibility, since it can be done after the packet has already traversed the network. The measurement could then be used as T_{pred} in the estimation of the current sender's pose (see sections 3.4 and 4.5) based on the belated pose measurement most recently received. Although not identical, this approach closely resembles the ubiquitous dead reckoning technique that too performs prediction on the client side, taking the last known or computed pose as an initial value in its predictive estimate of remotely controlled entity's current pose.

Among the forefront disadvantages of client-side delay measurement approach

is its apparent need for tight clock synchronization among participating workstations, limiting the range of applications that can benefit from compensation techniques based on this approach. The accuracy of network delay measurement is, naturally, conditioned on the accuracy of the clock synchronization mechanism. For geographically distributed DIVEs, synchronization would require periodic exchange of information over the network (as is the case with the well-known Network Time Protocol), which itself suffers from network propagation latency.

Not the least of the concerns about receiver-side latency measurement approach arise from the fact that it requires a smart client, having access to the model and capacity to use it in order to obtain an estimate of the current pose. The implications of a smart client are twofold. First, it has to be informed, i.e. knowledge of the model for a trajectory of motion or a dynamic behavior of a remote entity is a necessary condition for the client to perform its pose estimation and prediction. Second, the client must be powerful, i.e. adequate computational resources are expected to be present to handle pose estimation and prediction for *all* simulated entities sharing a Virtual Environment at any given time. The former implication, raises intellectual property and privacy concerns, while the latter translates into a financial burden for simulation participants, which can be avoided.

5.2.3 Sender-side Delay Estimation

Network delay estimation at the sending end satisfactorily addresses all of the above disadvantages, providing additional benefits beyond that. Security and privacy concerns are eliminated for distributed applications where the knowledge of tracked object is considered valuable intellectual property, and/or certain privileges (such as security clearance) are required to have access to it. This approach also does not require clock synchronization, as the round-trip time is measured when the packet acknowledgments are received. Using RTT measurements does assume either a symmetry of the connection or a certain known relationship between one-way transit times (OTT) incurred on a forward and reverse paths, which is usually a reasonable assumption.

Furthermore, in this approach, a sender presents clients with a ready to use pose estimate, eliminating the need for smart receivers and adequate computational resources at their disposal. Finally, sender-side latency estimation offers an added bonus of reduced sensitivity to packet loss. In contrast, pose prediction at the receiving end will suffer from lack of continuous and prompt pose measurements from remote workstations, whereas senders can rely on access to the complete data record for pose estimation and prediction regardless of network reliability.

5.3 Methods of Estimation

Its advantages notwithstanding, sender-side network delay estimation does, however, entail meeting several technical challenges. First, there appear to be no good models for network delay useful for its short-term prediction. Numerous studies attempted to model and predict network traffic characteristics, such as network latency, packet interarrival time, and bandwidth consumption. In his seminal paper on TCP extensions, Jacobson [JBB92] referred to network delay estimation as a signal processing problem, and a large variety of signal processing approaches have been proposed. A sample of the literature surveyed on this subject suggested algorithms as diverse as the following:

- economic forecasting theory [MCML05], [WO94]
- machine learning and neural nets [Cro05]
- autoregressive filters
- network weather service
- multimedia playout delay prediction

with many possible variations thereof. Autoregressive filtering approach was found particularly attractive for the reasons we hope to unveil through a brief

discussion of other candidate techniques.

Network Weather Service (NWS) focuses on network performance forecasting specifically in the context of predictive methodologies [Wol98], [WSH99], which places it among the most relevant network delay estimation approaches. Furthermore, NWS adaptively selects the best network forecaster from an expandable pool of predictors, running them all in parallel. It relies on a set of servers taking turns to obtain a distributed sequence of network measurements, and, therefore, a tight clock synchronization mechanism must be in place. This constitutes the first undesirable feature of NWS, as previously discussed in section 5.2.2.

Secondly, both NWS and other techniques, such as ARIMA (see [BMK96], [GP94]), that have been shown to work well for aggregate internet traffic, exhibit apparent unsuitability for applications that demand short-term network latency estimates. The results for NWS performance analysis were presented on the time scale of 24 hours, and ARIMA explicitly targeted long-term network prediction [GP94].

Estimation of network characteristics other than latency may be of interest because they provide indirect knowledge of the end-to-end delay itself. For the example of packet interarrival time (IAT) prediction, while it is uninformative of the overall delay, it can be used to estimate its variation if the frequency of

packet generation by the sender is known at the receiving end. The intuition is that IAT should be precisely equal to the inverse of packet sending frequency in the absence of delay jitter, as illustrated by figure 5.2b). Such network systems are rare, and variability in delay is significantly more common and troublesome. As illustrated by fig. 5.3, the presence of jitter can be symbolically written as:

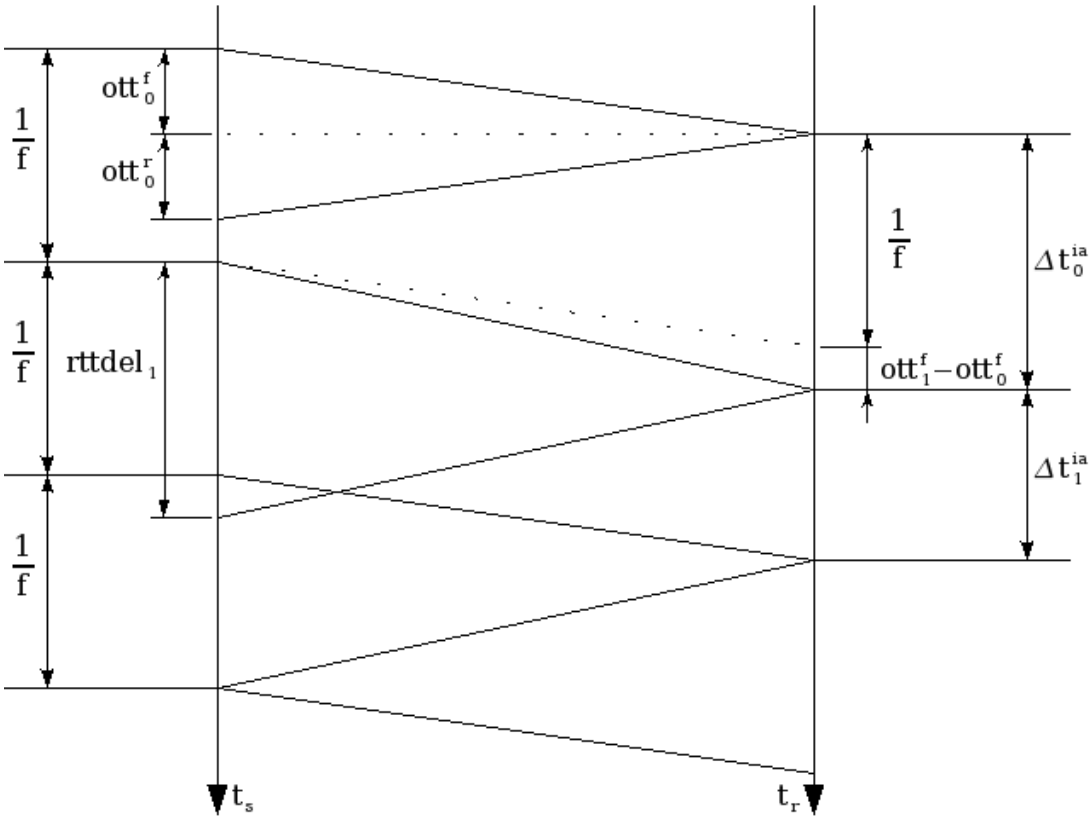

Figure 5.3: Relationship between IAT and RTT delay

$$ott_i^f \neq ott_{i+1}^f$$

where *ott* stands for one-way transit time, with f and r superscripts representing the forward and reverse paths. Δt^{ia} denotes a measurement of packet inter-arrival time(IAT) at the receiving end and is, therefore, non-constant in this case.

A quantitative relationship can be derived that ties together packet IAT, OTT, and the frequency of its generation:

$$\Delta\text{latency} = ott^f_{i+1} - ott^f_i = \Delta t^{ia}_i - \frac{1}{f} \tag{5.1}$$

In this model, changes in Δt^{ia}_i must be due to latency variation. In fact, the right hand side can be used as a gauge of whether the last packet arrived sooner than expected ($\Delta t^{ia}_i - \frac{1}{f} < 0$), later than expected ($\Delta t^{ia}_i - \frac{1}{f} > 0$), or there was no delay jitter whatsoever ($\Delta t^{ia}_i - \frac{1}{f} = 0$).

Eq. (5.1) hints at the possibility of a powerful recurrence relation that allows calculation of most recent one-way transit time based on packet's previous OTT and IAT measurements. Indeed, introducing the assumption that $ott^f_i = ott^r_i = \frac{rttdel_i}{2}$, we can rewrite the above as follows:

$$\frac{rttdel_{i+1}}{2} = \frac{rttdel_i}{2} + (\Delta t^{ia}_i - \frac{1}{f})$$

Ergo, it has been shown that existing work on modeling interarrival packet time distribution can be used for RTT delay estimation. Furthermore, given the recursive nature of the above discrete-time relationship, we even have a poten-

tial for an EKF implementation of the network delay estimator as well. However, despite the apparent promise this approach holds, the research literature surveyed appears to be of highly empirical nature ([Bor00], [FÖ2], [ZA05]) and rather focuses on fitting various distributions to the data collected offline than on formulating the models for prediction/estimation at runtime.

Finally, multimedia playout delay prediction (e.g. [YLLG03], [DS99]) addresses the problem of network delay impairment for a different class of applications. Suitability of compensation techniques borrowed from real-time domains, such as video and voice over IP streaming, for network delay amelioration in DIVEs is questionable for a variety of reasons.

First of all, the focus in the former is shifted more towards reliability (lack of packet loss) and playback smoothness (lack of jitter) than latency reduction or amelioration. Indeed, when using Skype, for instance, lost syllables or choppy playback is significantly more detrimental to usability and quality of experience than a mere lag, which primarily impacts turn-taking. In case of DIVEs we can afford to lose and even deliberately drop pose datagrams, especially in situations where more than two of them are received within a single frame.

Secondly, in case of DIVEs, we don't have to worry as much about out-of-order packets. Whenever outdated information is received - it simply bears no

effect on the simulation, since it's already been updated with a more recently generated pose.

It has brought us to the conclusion that most of the existing approaches to network delay estimation are not suitable for reasons ranging from mismatch between application domains, to the emphasis on offline model fitting to aggregate network traffic, to a clear focus on network delay prediction on a much coarser time scale than required.

5.4 The Chosen Method

The lack of adequate network delay models ultimately brings us to a time-tested auto-regressive (AR) filtering approach introduced by Jacobson as a means to calculate retransmission timeout (RTO) for TCP fragments. 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 algorithm can be summarized as follows [SFR04]:

$$\begin{aligned} delta &= measuredRTT - srtt \\ srtt &\leftarrow srtt + g \times delta \\ rttvar &\leftarrow rttvar + h(|delta| - rttvar) \\ RTO &= srtt + 4 \times rttvar \end{aligned} \tag{5.2}$$

with weight factors g & h ranging between 0 and 1. Jacobson suggests keeping track of a smoothed round-trip time estimate ($srtt$) by weighting the estimation error $delta$ with gain g. $rttvar$ is a similarly smoothed estimator of mean deviation calculated as the weighed average of previous estimate and RTT estimation error $delta$. Indeed, eq. (5.2) can be rewritten as an 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.

The SRTT network delay estimator, as we've denoted it, compares rather well with approaches of similar complexity, such as running average and moving window average. Running average requires infinite memory and assigns ever-decreasing weight to subsequent RTT delay measurements. Using running average for network delay estimation is also undesirable in cases where there are

noticeable steps in measured network delay, or an application runs for an extended period of time. In case of noticeable steps in the network delay, SRTT is expected to converge to a new mean significantly faster.

Moving window average, in its turn, compares rather closely to the autoregressive filter due to a similar dynamics. In contrast to SRTT, however, it requires additional storage and translates into more computational resources in terms of both memory and CPU time consumed by network delay estimation algorithm.

Finally, it is acknowledged that the improvements to SRTT in particular and our approach to network delay estimation in general are possible. One of the most obvious ones would be to tune its weight factors during run-time according to the increase or decrease in $rttvar$. However, experimental results presented in chapter 7 dissuaded the author from doing so due to the fact that possible improvements do not necessarily translate to improved overall network delay amelioration framework performance.

6 The Simulator

For the purposes of evaluating our approach to network delay amelioration, a unidirectional type client-server DIVE illustrated in figure 6.1 was simulated. Our virtual testbed consists of a tracking and rendering workstation, separated

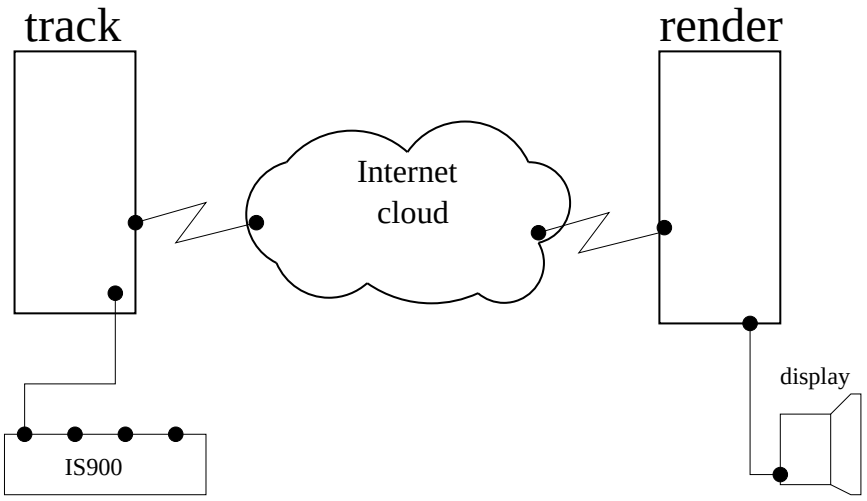

Figure 6.1: Simulated experimental setup

by a network. The tracking workstation provides sampled head position and orientation, while the renderer must properly adjust its display as the user pose

changes over time. The Internet cloud introduces propagation delay we attempt to ameliorate. Figure 6.2 gives a very high-level overview of the simulator's basic functionality. The software system is composed of three principal components: the network, pose, and statistical performance blocks.

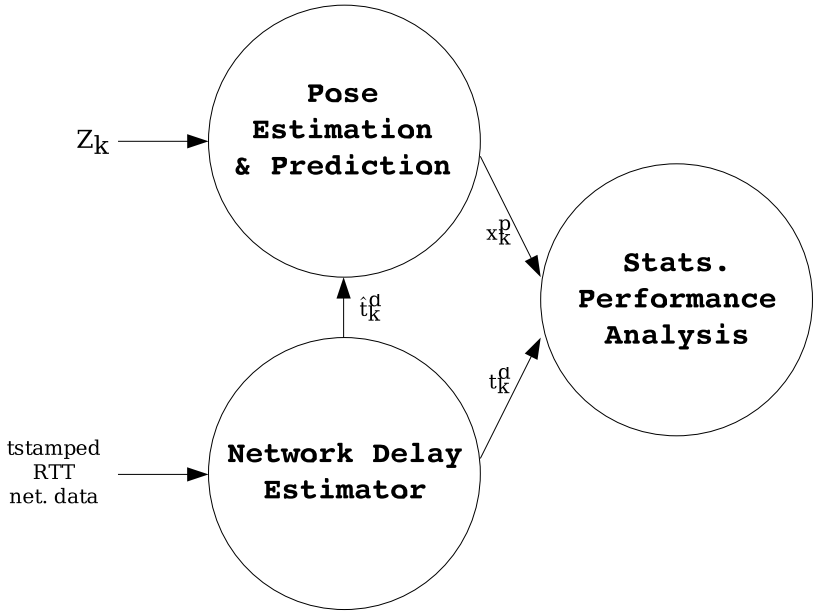

Figure 6.2: Simulation functionality overview

The network delay estimator (chapter 5) is the lower of the two layers of our latency amelioration framework. It performs estimation of network propagation delay $(\hat{t}_k^d)$[1] based on real timestamped roundtrip time measurements of network data. This estimate is used as prediction interval T_{pred} by the pose estimation

[1] k is the discrete-time independent variable and superscript d stands for delay

and prediction block, which maintains an estimate of delay-corrected head pose x_k^p, where superscript p identifies prediction. Based on ground truth data for both head motion and network RTT delay t_k^d, statistical analysis is finally carried out for the overall performance of the dual-estimator system.

6.1 Network Delay Estimator

The network delay estimator introduced in chapter 5 was implemented in C and extended to provide ground truth RTT data to the analysis module in addition to the estimate of network delay.

As part of RTT data collection by this module, the generation of packets at the sending end was driven by interrupts received from a real-time clock (RTC) chip to ensure consistency in the frequency, selected to reflect that of a single-station IS900 head tracker. To minimize the possibility of a local queuing delay, I/O multiplexing was performed on the sending and listening sockets through the use of select() system call.

In addition to the SRTT estimate (see section 5.4), a running average was also computed at each packet interval. Thus, for each session a combined table of ground truth data and two separate estimates: SRTT and runavg were obtained.

6.2 Pose Estimation and Prediction

Details of the functionality of the pose estimation and prediction block is shown in figure 6.3. For scalability and experimental flexibility, there is a clear logical

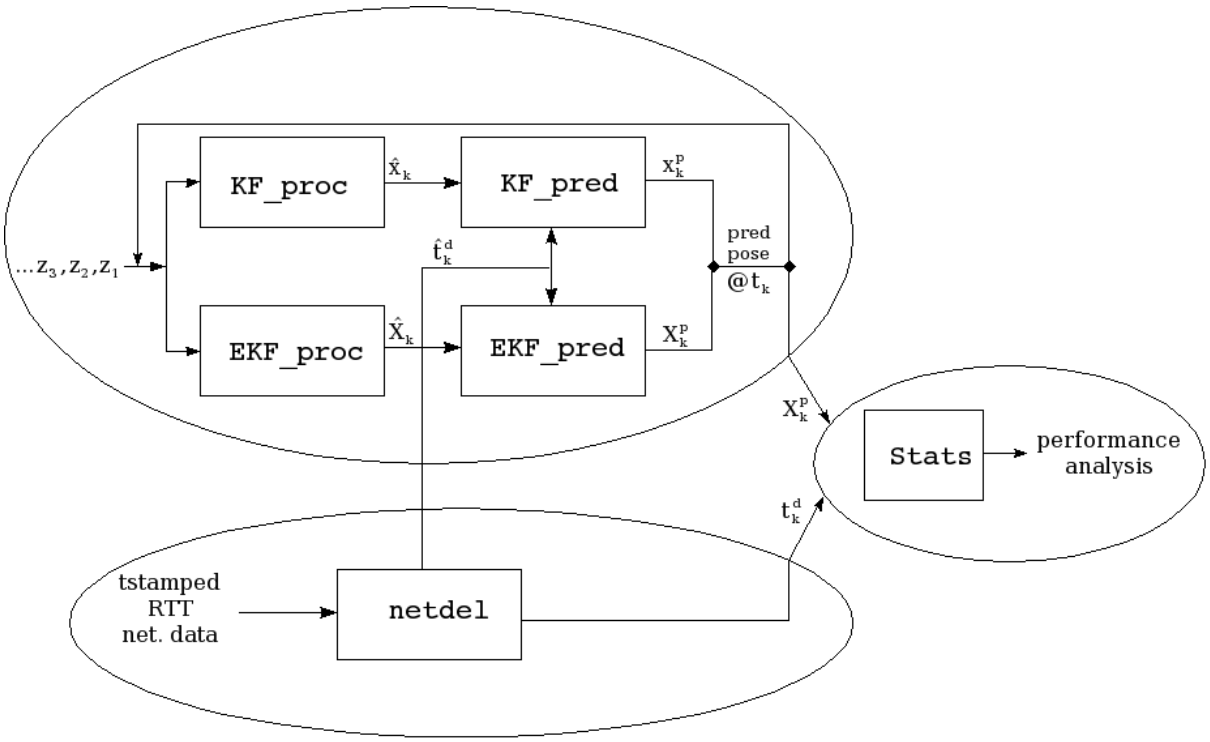

Figure 6.3: Detailed simulation functionality

and implementational separation between positional and orientation modules in addition to the separation of their functional components of estimation (KF_proc, EKF_proc) and prediction (KF_pred, EKF_pred). The third functional component of the pose block is initialization.

6.2.1 Positional KF Initialization

The implementation of the pose block and subsequent evaluation was based upon real head pose data obtained from a motion repository initiated by Azuma & Bishop and later augmented and postprocessed by LaViola [LaV03b]. This repository provided ground truth position and quaternion orientation data collected for different head movement profiles.

Given the specifications of an in-house IS900 tracker, a simulated measurement data sequence was derived from the ground truth dataset by downsampling it to the frequency of single-station IS900 (180Hz) and subsequently perturbing the result with white noise. For positional Kalman Filter, the white noise sequence is simply obtained by generating a vector of normally distributed values with the mean of zero and standard deviation equaling the specified positional RMS error for the simulated IS900 sensor.

6.2.2 Orientation EKF Initialization

Similarly to the positional case, the measurement sequence for orientation was derived from the quaternion ground truth. A naive approach to doing so would be to treat quaternions as a sequence of 4D vectors and generate white noise for each of the vector components, similarly to positional data perturbation. Such

approach is considered by the author inadequate for a variety of reasons:

- It is unclear what standard deviation must be taken for the generation of normally distributed noise sequences.
- It is preferable to leverage the elegant geometrical interpretation of these hyper-complex numbers, offering a meaningful explanation for quaternion sequence perturbation.
- For the simulated IS900 sensor, orientation and precision is specified in terms of angular RMS error, and its relation to *all* of the components of a quaternion, treated as a 4D vector, is not readily apparent.

Orientation measurement derivation, therefore, was based on quaternion sequence perturbation, for which no known widely accepted methods were found at the time of writing. The method used can most easily be understood in terms of the geometric interpretation of quaternions (see Kuiper's [Kui99] for more detail).

A unit quaternion can be written in the following form:

$$q = q_0 + \vec{q} = \cos\frac{\alpha}{2} + \vec{u}\sin\frac{\alpha}{2}$$

where $\vec{u}$ and α signify the axis and angle of rotation, respectively, represented by q. By applying a quaternion rotation operator $L_q(v) = q^* v q$ to any 3D vector

$\vec{v}$, written as a pure quaternion $v = [0\ \vec{v}]$, we rotate $\vec{v}$ about $\vec{u}$ by α. Therefore, through the introduction of operator L_q, quaternion q has taken on a geometric meaning of carrying the axis and angle information.

Furthermore, subsequent application of operator L_p to the result of L_q (see figure 6.4) yields the following:

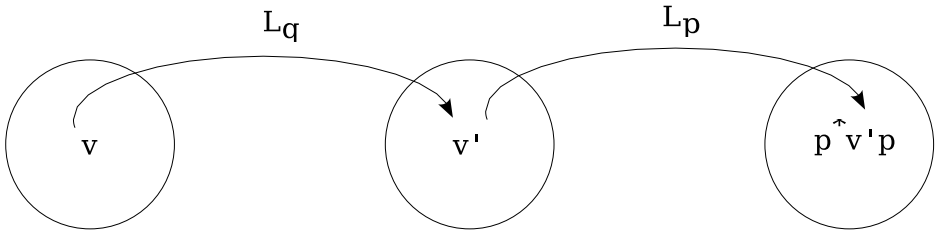

Figure 6.4: Quaternion rotation operator combination

$$L_p(v') = p^* v' p = p^* q^* v q p = (qp)^* v (qp) = L_{qp}(v) \tag{6.1}$$

Ergo, the product of two unit quaternions signifies the combined effect of consecutively carrying out the respective rotations. Clearly from eq. (6.1), the effect of $L_r(v)$, where $r = q \otimes p$, is identical to a sequence of rotations $L_q(v)$ followed by $L_p(v')$. This geometrical insight arms us with necessary understanding to proceed with quaternion perturbation in a more principled manner. Indeed, performing quaternion multiplication of the ground truth sequence with noise quaternions generated, has the desirable effect of perturbing the true rotation

by a subsequent noise rotation, carried out about a random axis with the angle normally distributed about zero. Appendix B will help the reader visualize the result of the above operation.

We implement this approach with the following algorithm:

I. Preparation

1) Set angular and positional standard deviations σ_α & σ_{pos}

2) Set $\sigma_u = \sigma_{pos}\sqrt{0.1}$

where σ_u^2 is the variance for the distribution of the remaining (axial) noise quaternion components. LaViola found that the variance for noise quaternion components is smaller than positional noise variance (σ_{pos}^2) by a factor of 10 for the dataset we are using.

II. Generate the q_0 (angular) component of the noise quaternion sequence

1) Generate normally distributed sequence nth of n values, such that $nth(i) \sim N(0, \sigma_\alpha^2)$

2) Cap nth at some reasonable value, say $\pm 10 \cdot \sigma_\alpha$

3) Perform trigonometric conversion to obtain the q_0 component sequence

$$nq_0 = \cos\left(\frac{nth * \pi}{2 * 180}\right)$$

note that the resulting nq_0 is equivalent in dimension to nth and is an $n \times 1$ vector in our case.

III. Generate the $\vec{q}$ (axial) components of the noise quaternion sequence

1) Construct matrix nq of 3 column vectors, say u_1, u_2, & u_3, such that $u_k(i) \sim N(0, \sigma_u^2),\ k = 1..3\ \wedge\ i = 1..n$:

$$nq = \begin{bmatrix} | & | & | \\ u_1 & u_2 & u_3 \\ | & | & | \end{bmatrix}$$

2) Perform row-wise normalization to obtain a sequence of 3D unit vectors $u(i)$:

$$nq = \begin{bmatrix} - & u(1) & - \\ - & u(2) & - \\ - & u(3) & - \end{bmatrix}$$

3) Scale u_1, u_2, & u_3 by $n \times 1$ vector $\sin \frac{nth*\pi}{2*180}$ element-wise to obtain $\vec{nq}$:

```
for j=1..3
  nq(:,j) = u_j .* sin(nth*pi/360)
end
```

where $a(:, j)$ represents the j^{th} column vector and .* denotes element-wise multiplication (following Matlab convention).

IV. Combine nq_0 and $n\vec{q}$ to construct $nq = [nq_0\ n\vec{q}]$

V. Perturb ground truth

1) Perform perturbation by element-wise multiplication of ground truth sequence with nq:

```
measdata = QuatSeqMult(truedata, nq);
```

Quaternion measurement sequence *measdata* is thus derived from the down-sampled ground truth data. The attentive reader will note, however, that the EKF measurement update (see section 4.5) is completely oblivious to the special algebraic properties of quaternions. In fact, a 4×1 vector is formed from quaternion components and all subsequent algebraic operations are carried out as if dealing with typical real numbers. Furthermore, the underlying assumption is that the measurement noise in each component of the measurement vector is white. Therefore, we must ascertain that the result of quaternion sequence perturbation algorithm does not violate that assumption. A simple check is thus performed by taking a difference (*qdiff*) between obtained noisy quaternion sequence *measdata*, now treated merely as an $n \times 4$ matrix, and the true data matrix, and ensuring that *qdiff* consists of column vectors of normally distributed about zero noise values. Bar plots were constructed to allow for visual inspection of

q_w, q_x, q_y, and q_z distributions and the results - convincingly supportive of the author's hypothesis that the assumption would be upheld (see figures 6.5 and 6.6).

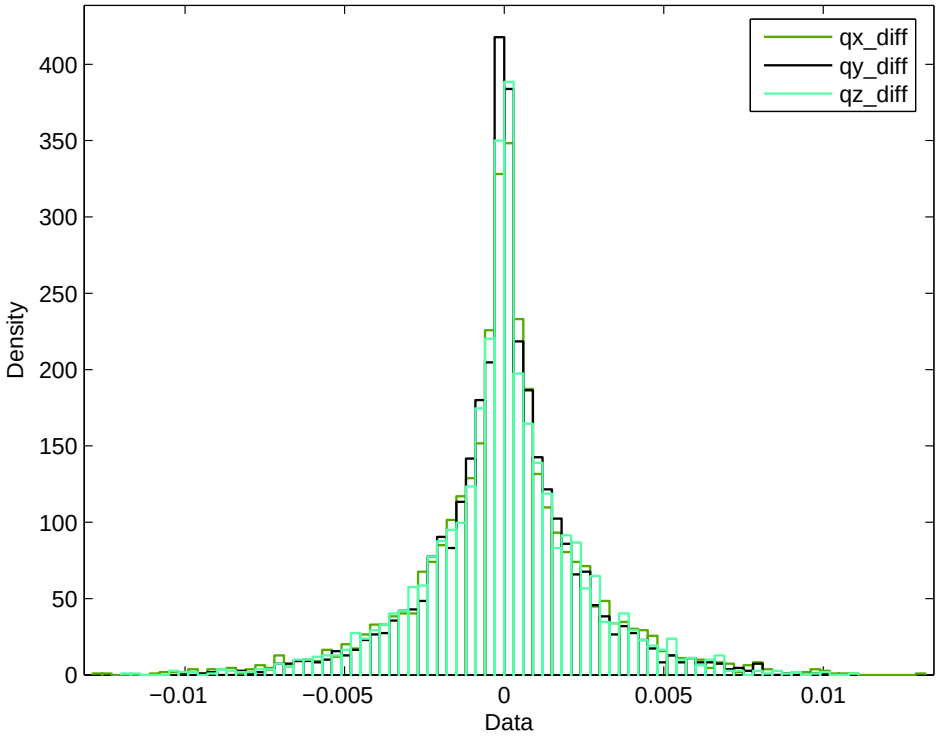

Figure 6.5: Distribution of *qdiff* components

As a reminder, notations $q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$ and $q = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$ have been used interchangeably throughout the research project.

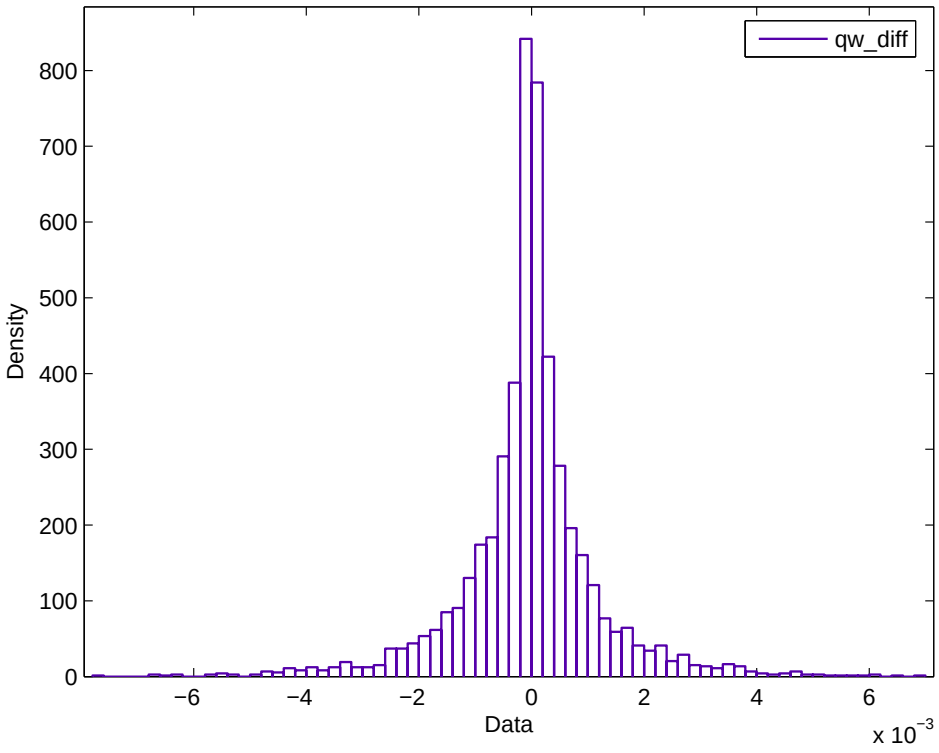

Figure 6.6: Distribution of *qdiff* components

6.3 Statistical Analysis

The purpose of the Analysis block is twofold:

a) match each received pose with a data point in the ground truth sequence according to their respective timestamps

b) perform statistical analysis of overall framework performance

This allows us to make both visual and quantitative judgements of how well the dual-estimator compensation framework was able to match the ground truth, since, ideally, we'd like the rendering workstation to have exactly the same pose information as available locally at the sender at exactly the same time. Graphical visualization of our comparison is rather easy to perform using plotting capabilities of Matlab. To arrive at the quantitative comparison, we have to find the corresponding data points in the ground truth set for each record in the received sequence. It might seem sufficient at first to simply choose the closest matching data point in the denser (by a factor of 6 in our case) ground truth sequence. Keeping in mind the sampling frequency f of the latter, we incur a mismatch error of up to $\frac{1}{2f}$ in doing so. To obtain higher effective precision, interpolation was used.

For position it suffices to employ linear interpolation:

```
intpose = (1-factor).*apose + factor.*bpose
```

where all of the above are column vectors of time sequences, and the multiplication is element-wise.

For orientational component interpolation we used the spherical linear quaternion interpolation algorithm, perhaps better known as the Slerp operator ([Sho85], [Wik06]):

$$Slerp(p_0, p_1, t) = \frac{\sin(1-t)\Omega}{\sin\Omega} p_0 + \frac{\sin t\Omega}{\sin\Omega} p_1 \tag{6.2}$$

where Ω is the angle between unit vectors p_0 & p_1, found by taking their dot product:

$$\cos\Omega = p_0 \cdot p_1$$

and t is the interpolation factor that grows linearly from 0 to 1 as we go from p_0 to p_1 along the arc.

The above is actually completely detached from a quaternion interpretation of end points p_0 and p_1 as well as from the dimension of the Eucledian space in which the sphere is embedded. In the context of quaternions, we have to deal with the duality of orientation representation – a matter of ensuring that p_0 & p_1 are in the same hyper-hemisphere. Hence the prior check of the dot product sign and corresponding manipulation of one of the quaternions in the vectorized algorithm presented in appendix C. Finally, obtaining the interpolated sequence

of ground truth poses puts us in a position to perform statistical comparison of the result with the received sequence, as our timestamps now have been aligned.

Global pose and component-wise root-mean-square (RMS) error metrics were chosen, as is typical of head movement prediction performance analysis in the VR community ([AB94], [Azu95], [LaV03b], [KEP97], [WO95]). For component-wise RMS, the difference between the interpolated pose and the received sequences is calculated first:

$$diffpose = \text{intpose} - \text{recvseq}(:, \text{pose})$$

followed by column-wise computation of

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

Global position RMSE is calculated as follows:

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

where $i \in \{1, 2, 3\}$ represents x, y, and z components respectively.

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. Based on the geometrical properties of quaternions briefly discussed in the previous section,

angular error E_α is extracted from the q_0 component of the difference quaternion

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

denoted as $q_{diff}[0]$ as follows:

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

Angular RMSE for the entire motion sequence can then be computed as

$$ang\_rmse = \sqrt{\sum_{j=1}^{n} \frac{E_\alpha^2(j)}{n}}$$

7 Results and Discussion

7.1 Experimental Setup

7.1.1 Head Motion

Six head motion datasets were chosen from the repository partially collected and preprocessed by LaViola (see [LaV04],[LaV03b],[LaV03a], and [LaV03d]). Each dataset features approximately 20 seconds of positional and orientation data initially captured by an IS900 tracking system. The head motion datasets fall into three major categories and reflect specific motion profiles summarized as follows:

Two datasets were selected for each head motion profile and will be referred to for the rest of the chapter by the corresponding name from the table above, augmented with an id $\in \{1, 2\}$ for unique identification. The simulation was carried out using Linux Matlab version 7.1.0.183 (R14) Service Pack 3 on a time-

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

shared quad Xeon 3.2GHz server with a total of 4GB of RAM.

7.1.2 Network Setup

Network delay datasets (also referred to here as traces) were collected in-house. Due to a relatively low-latency network infrastructure at York University, four separate geographically distributed sites were used to collect each single RTT trace. They basically consisted of an initiator at York (CS), responder at the University of Waterloo, and two relay workstations in between — at NRC Canada location in Ottawa and the York University residential complex (YR). Figure 7.1 illustrates the flow of information from the initiator to responder. To accomplish this, I implemented bidirectional network address translation (NAT) using ipta-

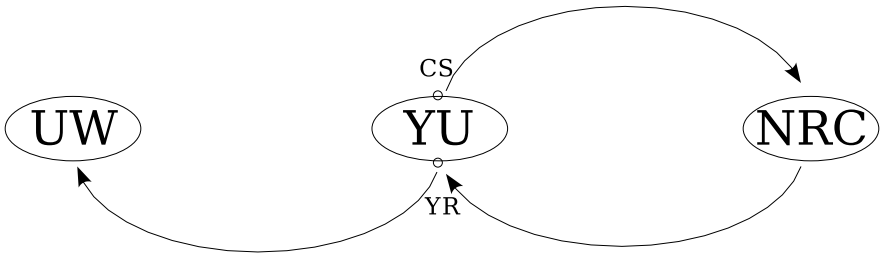

Figure 7.1: Network RTT delay data collection setup

bles [IPt] on the relay workstations. This requires root priviledges significantly limiting my configuration options.

As already mentioned in the Simulation chapter, the client-server network trace collection application was written in C and utilized real-time clock chip interrupts to trigger packet generation by the initiator at the time intervals less than the Linux default time resolution of 10 ms. Without RTC, the frequency of packet generation would be at most 100 Hz (less in practice) - significantly less than the desired 180 Hz sampling frequency of a single-station IS900 tracker ([IS900]).

Nine network delay traces were collected with their basic statistical characteristics summarized in table 7.1 in us units. Typical packets traversed 32 hops in one direction or 64 hops for RTT data used for the simulation. The significance of the large number of hops is in added overall delay jitter due to the variability

trace	min	avg	max	sdev
1	17369	17882.704	32183	543.435
2	17427	17986.952	54363	1256.779
3	17409	17880.449	29508	495.901
4	17392	17937.744	30851	824.272
5	17388	17922.094	41664	946.994
6	17379	17912.935	30595	635.553
7	17450	17955.423	34664	616.575
8	17413	17939.845	30093	570.636
9	17491	17906.292	35111	653.548

Table 7.1: Network RTT trace statistics

in queuing delay at each.

7.1.3 Predictor Conditions

To recapitulate, head motion data is used in concert with estimates of network delay to predict the user pose a certain time into the future. Four different conditions were explored for the pose predictor:

a) **Const** – network delay estimate was set to a constant value

b) **Runavg** – a running average of network RTT was used as an estimate of current round trip latency

c) **SRTT** – smoothed RTT estimator described in section 5.4

d) **Opti** – an omnipotent network delay estimator given the perfect knowledge of the latency for each packet

Runavg and SRTT were introduced and discussed as viable candidates in chapter 5. Opti was included to compare the overall performance of the realistic framework based on SRTT with that based on the ideal network delay estimator, providing the pose predictor with a correct prediction interval every time. Therefore, Opti effectively serves as a benchmark we strive to reach through potential improvements to network delay estimation.

Finally, constant delay prediction is the most widespread approach to delay compensation at the time of writing. The value for the constant RTT delay was chosen to equal to the mean RTT delay for a given network trace — the best possible constant estimate of network delay the framework can have.

7.2 Results and Discussion

Three metrics will be used for performance evaluation: root mean square error (RMSE) – providing the estimate of average performance, absolute maximum error (MAXE) – descriptive of the performance in the worst case, and what I call TIBET. TIBET is an acronym for TImes BETter statistic, where the ratio of RMSE for the specific condition to RMSE of the chosen base is computed as an index of relative performance.

7.2.1 Position

7.2.1.1 Smoothing Performance

The performance of the positional Kalman Filter is fairly consistent across all six head motion datasets and is nearly identical for all three components. Figure 7.2 presents an overlaid plot of KF estimator RMS error versus using the

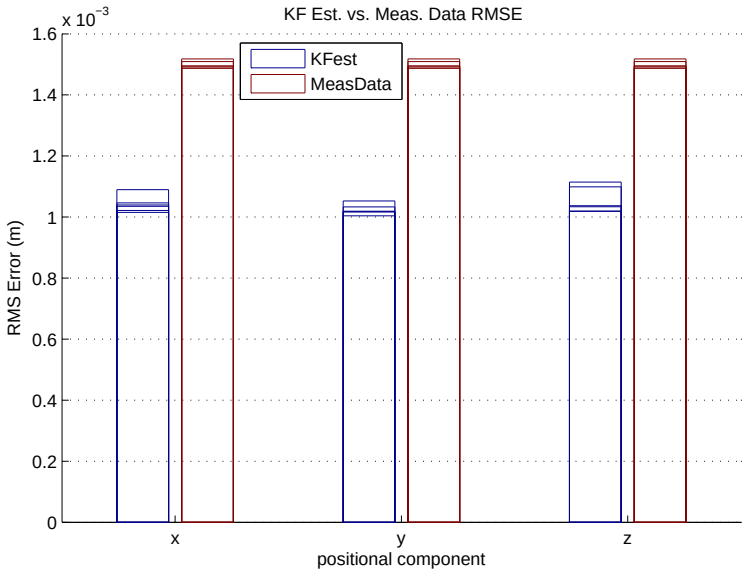

Figure 7.2: KF Estimate vs Measurement Data RMSE

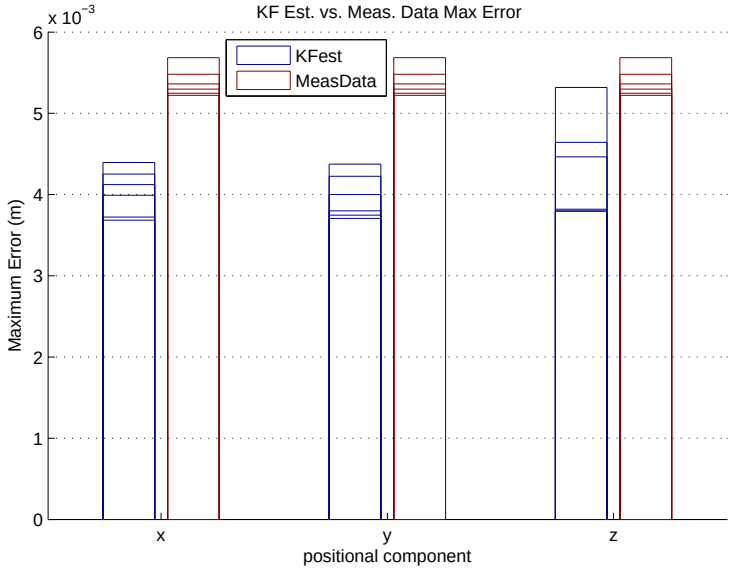

Figure 7.3: KF Estimate vs Measurement Data Maximum Error

measurement data directly. 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 7.2 illustrates the relative performance improvement resulting from

pcmp	head11	head12	head21	head22	head31	head32
x	1.4775	1.4721	1.4506	1.4309	1.4443	1.3662
y	1.4840	1.4884	1.4694	1.4597	1.4697	1.4141
z	1.4821	1.4655	1.4682	1.3350	1.4414	1.3546

Table 7.2: TIBET for KF estimator performance

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.

The KF smoothing also significantly reduces the worst case error as can be seen in figure 7.3.

7.2.1.2 Predictor Performance

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 7.4 shows the average framework

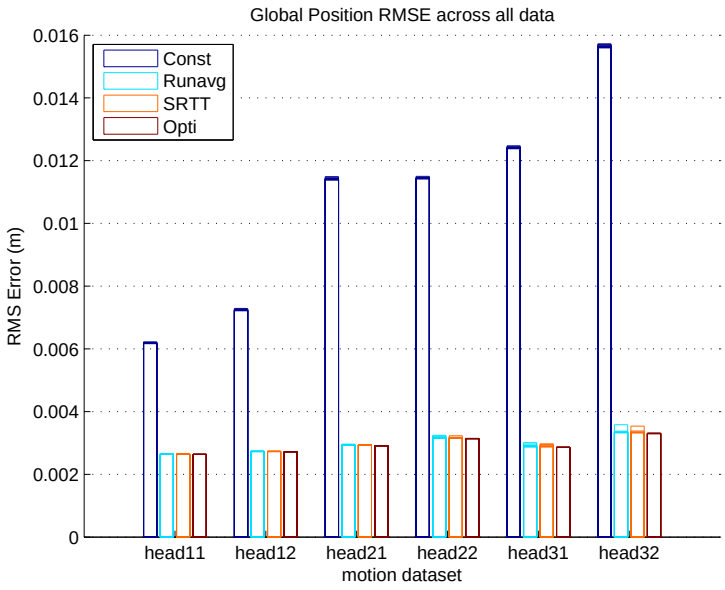

Figure 7.4: Global Position RMSE across all data

performance 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. Furthermore, the improvement gained from the perfect knowledge of the network delay in the case of Opti does not translate into a noticeable performance increase. The apparent consistency of RMSE information across all the network traces, signified here by individual bars of the same color, is noteworthy as well. It motivated us to present the TIBET statistic for a randomly chosen network

RTT dataset to allow for a 2D table representation (Table 7.3). Performance of

dataset	Const	Runavg	SRTT	Opti
head11	1.0000	2.3421	2.3479	2.3527
head12	1.0000	2.6451	2.6553	2.6666
head21	1.0000	3.8868	3.9055	3.9377
head22	1.0000	3.5576	3.6015	3.6441
head31	1.0000	4.2791	4.3098	4.3495
head32	1.0000	4.6968	4.7196	4.7526

Table 7.3: TIBET for overall framework performance for all head motion datasets

the predictors relative to a constant delay prediction increases from the first motion profile to the third. A clear correlation is observed from fig. 7.5 between the extent of variability in the global position and the increase in the framework performance relative to constant delay prediction. The figure illustrates the spread of the distance from a global position mean for each head motion dataset. The arithmetic means were also included to emphasize the increase in variability from motion profile 1 to 2 & 3.

Finally, a maximum error bar plot is presented in figure 7.6 for the same network trace. Consistent with the RMSE data, runavg and SRTT based predictors

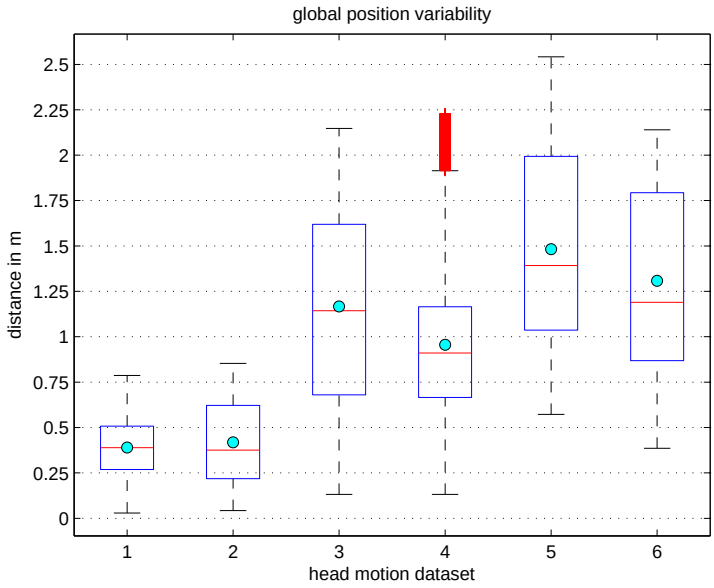

Figure 7.5: global position variability

still provide better performance than constant delay prediction, and the overall trend of a steady increase in TIBET from left to right is clear from mere visual inspection.

7.2.2 Orientation

7.2.2.1 Smoothing Performance

Similarly to the positional KF case above, the Extended Kalman Filter based smoother reduces the RMS error w.r.t. raw measurements for each of the quater-

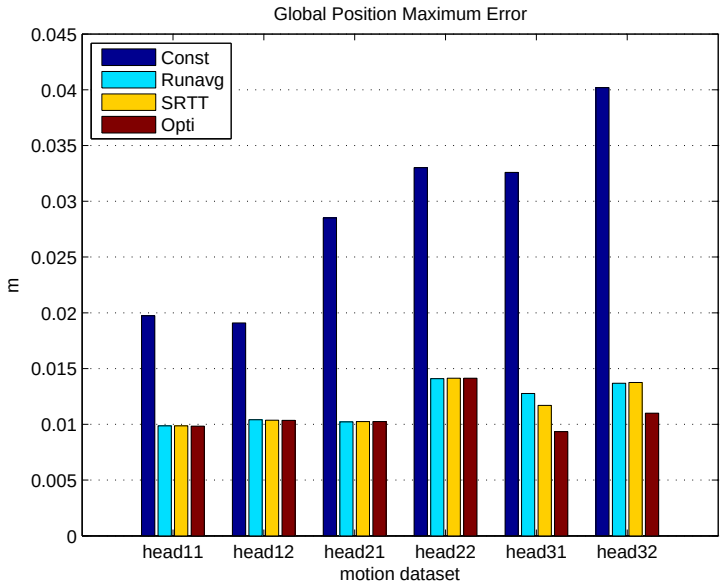

Figure 7.6: Global Position Maximum Error

nion components across all of the head motion datasets (Fig. 7.7). For a given component, bars of the same color signify RMSE readings for individual motion sequences. According to table 7.4, the EKF estimator manages to follow the ground truth with approximately 90% less error than the measurement data on its own with no apparent trends in the variation of relative smoothing performance across the motion profiles. Figure 7.7 however does show a subtle difference in consistency with which q_x & q_z components are estimated compared to q_w & q_y. As in the positional KF case it appears that the lack of variability in the ground truth data directly translates into the quality of its estimation.

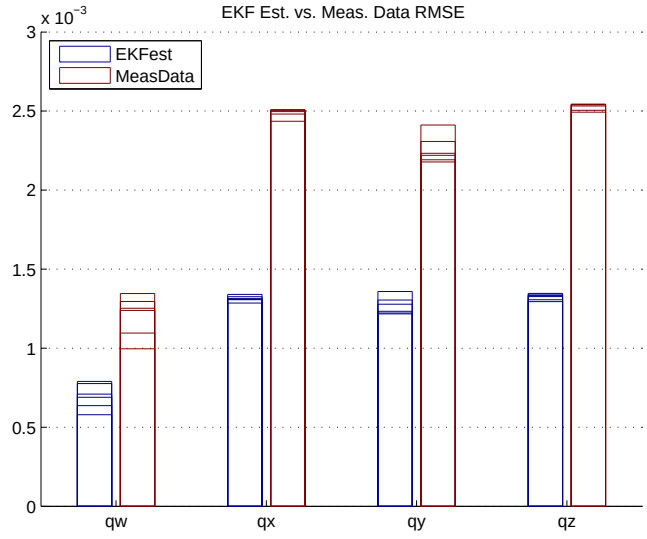

Figure 7.7: EKF Estimate vs Measurement Data RMSE

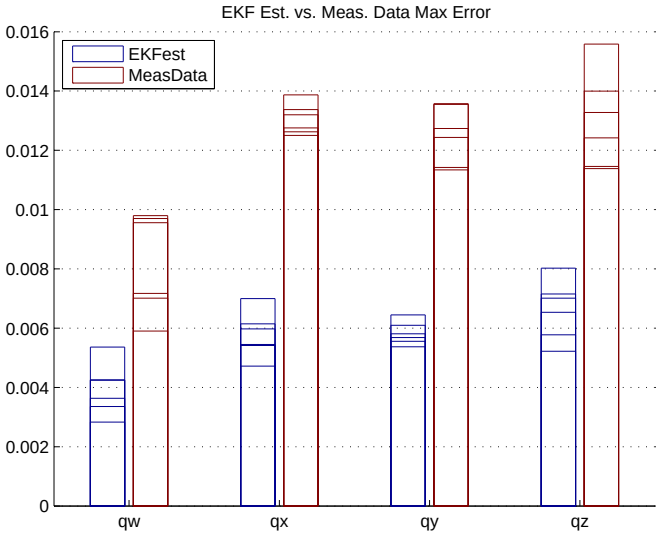

Figure 7.8: EKF Estimate vs Measurement Data Maximum Error

qcmp	head11	head12	head21	head22	head31	head32
q_w	1.7969	1.7204	1.7341	1.6413	1.7649	1.7207
q_x	1.9100	1.9507	1.8175	1.9156	1.9095	1.8715
q_y	1.8248	1.8055	1.7656	1.7113	1.7931	1.7759
q_z	1.8795	1.9260	1.8919	1.9389	1.9153	1.8894

Table 7.4: TIBET for EKF estimator performance

Indeed, with the exception of the last motion dataset, all others exhibit greater standard deviation in q_w & q_y than q_x & q_z as can be seen from table 7.5.

dataset	q_w	q_x	q_y	q_z
head11	0.0998	0.0542	0.4063	0.0367
head12	0.0903	0.0654	0.4191	0.0570
head21	0.1859	0.1662	0.5037	0.0955
head22	0.1528	0.0842	0.4689	0.0535
head31	0.0804	0.0823	0.1627	0.0717
head32	0.0446	0.1059	0.1344	0.1194

Table 7.5: Standard Deviation for orientation

Finally, the EKF estimator does surprisingly well in the worst case as well (Fig. 7.8). The variability in the maximum error is much greater, as expected,

but the overall performance improvement relative to the measurement error is reminiscent of that for estimator performance on average.

7.2.2.2 Predictor Performance

A measure of global orientation error in the form of a rotation angle between predicted and ground truth quaternion is chosen for the evaluation of the framework performance. Figure 7.9 captures the performance of EKF predictor on average in terms of overlaid RMSE information across all motion datasets and network traces. For each given head motion dataset, the RMS error is so consistent across all network RTT runs, that the variation in the height of the bars is barely visible. Table 7.6 demonstrates the relative performance gain for both Runavg and SRTT based EKF predictors over the constant delay prediction.

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 standard deviation in both is summarized in table 7.7.

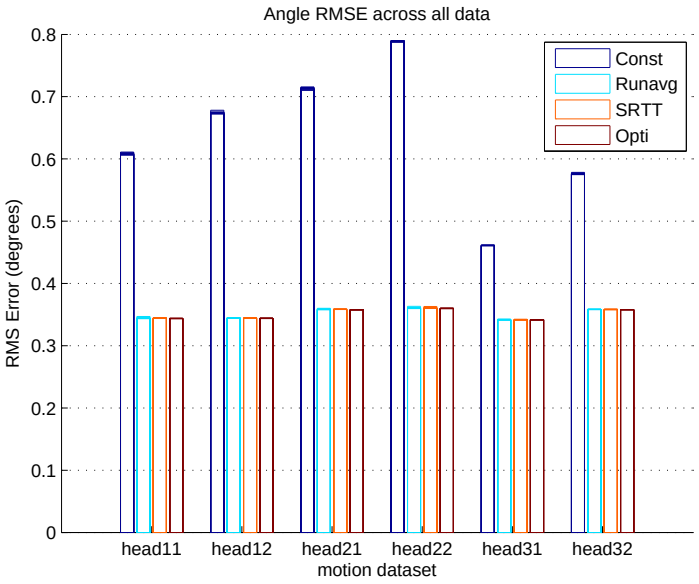

Figure 7.9: Angle RMSE across all data

dataset	Const	Runavg	SRTT	Opti
head11	1.0000	1.7702	1.7724	1.7747
head12	1.0000	1.9577	1.9604	1.9632
head21	1.0000	1.9899	1.9951	1.9976
head22	1.0000	2.1941	2.1931	2.1936
head31	1.0000	1.3482	1.3501	1.3504
head32	1.0000	1.6105	1.6117	1.6116

Table 7.6: TIBET for overall framework performance

The correlation between the heightened variability in both the axis and angle of

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 7.7: Axis-angle standard deviation

rotation and the increase in relative performance improvement becomes apparent.

Finally, the performance of the framework w.r.t. its maximum error in orientation prediction is exposed in figure 7.10. The variation in the height of overlaid bars is more pronounced for the maximum errors, as expected, but the overall benefit of variability-aware prediction is clear, echoing that for the RMSE.

7.3 Concluding Remarks

In addition to offering performance advantages in both the average case and maximum error scenarios, variability-aware predictors also exhibit comparable

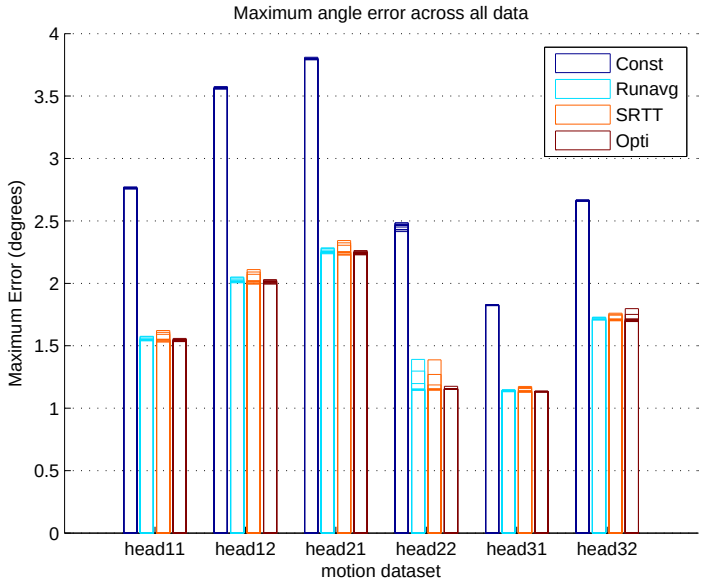

Figure 7.10: Maximum angle error across all data

performance to Opti based compensation. This phenomenon suggests that the prediction error due to the inadequacy of the model overshadows the error contributed by misestimation of the prediction interval itself. This is expected to change, however, as the mean network RTT increases, and the misestimation of current delay becomes more pronounced in the absolute sense. The take-home point, therefore, is that little if any improvement to the network delay estimator is necessary for DVE applications running on networks of comparable latency characteristics, and the focus should fall primarily on the selection of more accurate motion models. As the mean and variability in the latency elevate, the

gap between the SRTT and Opti based framework performance may indicate the need to upgrade the network delay estimator as well.

8 Conclusions and Future Work

8.1 Conclusion

A variability-aware proactive alternative to traditional latency compensation techniques was described and evaluated. No assumptions of any kind are either made or enforced about the dynamics of the network delay to be estimated. The two-tier predictive framework offered 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.

This approach was evaluated in a simulation through an offline playback of real head motion data and network RTT delay traces. The simulation 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 substantial improvements both on

average and in the worst case over the assumption of constant network delay.

8.2 Implications and Future Work

The framework introduced in this thesis is directly applicable to any form of distributed VE application that involves time-sensitive collaboration or interaction of geographically separated operators. Primary examples of such applications would be military combat and flight simulations and network games. Industrial tele-conferencing applications, where remote engineers collaborate on the design or development of complicated machinery may also benefit from proactive latency compensation. The implications of my research extend beyond the area of virtual reality, however. It is envisioned to have positive impact on distributed shared VEs of any kind, including multi-operator multi-robot (MOMR) and tele-haptic applications in particular. The latter has a direct application in tele-surgery, where a correctly simulated sense of touch is critical to the success of the operation [ZSG04].

My research on variability-aware latency compensation is by no means exhaustive, however. The framework itself can be improved through a better head motion model than a traditional kinematic approximation. The non-linearities of quaternions employed to represent head orientation may also be used directly

through the implementation of the Unscented Kalman Filter [JUDW95], avoiding the approximation errors due to Taylor series linearization of the model. Finally, the network delay estimation can be improved as well, but is subject to further research on network latency modeling adequate for short term RTT delay prediction.

The most immediate extension of this research work is envisioned in the form of a real testbed. The simulation presented in this thesis was deliberately designed to be more of a playback mechanism with statistical analysis capabilities in order for its results to be directly transferable to real DVEs. The testbed could take advantage of the comprehensive in-house suite of VE libraries (VElib) created for platform-independent virtual reality application development. A specific provision of the VElib is the filter chain, allowing a programmer to dynamically insert custom tracking event filters. My simulation was designed to resemble the overall VElib filter structure for easier portability to C and real applications in the future.

Finally, less immediate work of interest includes the implementation of the compensation framework for a prototype network game. The benefits of sender-side latency amelioration would then be studied in comparison to performing commonly accepted compensation techniques in the game industry, such as client-

side interpolation.

More sophisticated research studies could be conducted on adaptive approaches to network delay compensation, where a combination of proactive latency amelioration and dead reckoning is implemented. Our framework may be used as an enhancement or an extension to the traditional compensation algorithms, not necessarily replacing them altogether.

A Process Noise Autocorrelation

Process noise covariance matrix expression in eq. (3.6) resulted in the expected value term $E\left\{w(\xi)w^T(\eta)\right\}$. Noting that $w(t)$ is a scalar process, the transpose can be dropped, resulting in

$$E\left\{w(\xi)w^T(\eta)\right\} = E\left\{w(\xi)w(\eta)\right\} = R_w(\xi,\eta) = \begin{cases} 0, & \xi \neq \eta \\ E\left\{w^2(\xi)\right\}, & \xi = \eta \end{cases}$$

where $R_w(\xi,\eta)$ is the autocorrelation function of a stationary process $w(t)$. Recalling that for random variable $X \sim N(\mu,\sigma^2)$, $E\left\{X^2\right\} = \sigma^2 + \mu^2$, which for zero mean equals $\sigma^2 + 0$, the expected value term then becomes

$$E\left\{w(\xi)w^T(\eta)\right\} = R_w(\xi,\eta) = \begin{cases} 0, & \xi \neq \eta \\ \sigma^2, & \xi = \eta \end{cases} = \sigma^2\delta(\xi-\eta)$$

where σ^2 is the variance of the white process noise $w(t)$.

Finally, there's a known relationship between the variance of the white noise stationary process and its power spectral density $S_w(s) = W$. It too can be

derived from the definition of the latter:

$$S_w(s) = \int_{-\infty}^{\infty} R_w(\tau) e^{-jw\tau} d\tau$$

where $R_w(\tau)$ is the same autocorrelation function as the above $R_w(\xi, \eta)$, which can be rewritten as a function of the time difference $\xi - \eta = \tau$ for stationary processes.

Substituting the expression obtained for $R_w(\xi, \eta)$ into the definition of $S_w(s)$, the desired relationship becomes apparent:

$$S_w(s) = \int_{-\infty}^{\infty} \sigma^2 \delta(\tau) e^{-jw\tau} d\tau = \sigma^2 e^{-jw \cdot 0} = \sigma^2 = W$$

$$\Rightarrow E\left\{w(\xi) w^T(\eta)\right\} = W\delta(\xi - \eta)$$

■

B Quaternion Sequence Perturbation

Section 6.2.2 established the rationale behind approaching orientation data perturbation armed with the geometrical interpretation of its quaternion representation. The multiplication of two quaternions was shown to have an effect of consecutively carrying out the rotations represented by each. Therefore, quaternion measurement sequence derivation was accomplished by generation of a noise quaternion sequence followed by quaternion multiplication of the ground truth data by the resulting noise sequence.

The purpose of this appendix is to offer a geometric insight into the operation of adding white noise to the ground truth data. The result of quaternion data perturbation can be envisioned as the spherical cone of possibilities for the resulting orientation of an arbitrary rotated vector $\vec{v}$ (see figure B.1). Any line segment connecting the origin and a point on the spherical portion signifies a possible result of rotating vector $\vec{v}$ by a noisy quaternion. The angle of rotation

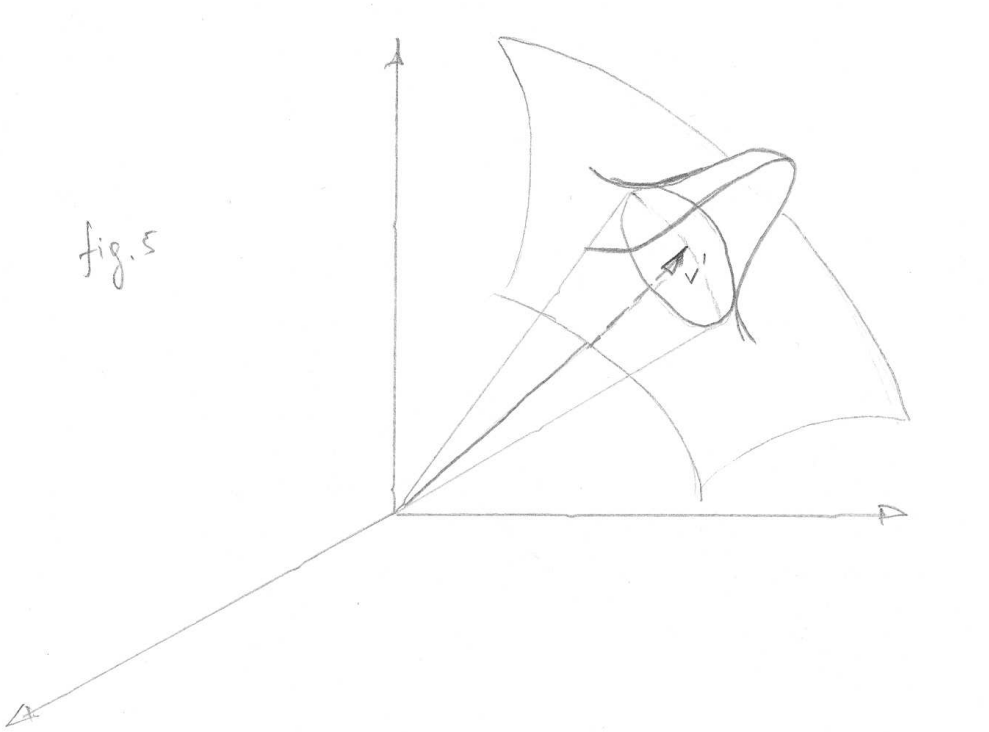

Figure B.1: Effects of quaternion noise on rotated vectors

transforming $\vec{v'}$, as obtained through a rotation of $\vec{v}$ by ground truth quaternion, into the mentioned directed line segment is normally distributed with variance taken from in-house equipment angular RMS specifications. Such geometrical interpretation of quaternion data perturbation could not be found in the literature and is considered a novel contribution.

C Quat. Sequence Interpolation Algorithm

Input: p_0, p_1 — $n \times 4$ arrays of n quaternions

t — $n \times 1$ column vector of interpolation factors

Output: iq — $n \times 4$ array of interpolated quaternions

1. Take a row-wise product of p_0 & p_1 to obtain an $n \times 1$ vector of cosines:

   $ct = dot(p_0, p_1, 2)$

2. Find all negative values in ct and store their indices in ind.

3. For each $i \in ind$ do

   a. reverse the sign of corresponding element in ct: $ct(i) = -ct(i)$

   b. negate the corresponding quaternion in p_1: $p_1(i) = -p_1(i)$

4. $theta = \cos^{-1}(ct)$

5. $st = \sin(theta)$

6. Precompute the multiplicative weights for eq. (6.2):

$$\begin{aligned} f1 &= \sin((1-t) \mathbin{.*} theta) \mathbin{./} st \\ f2 &= \sin(t \mathbin{.*} theta) \mathbin{./} st \end{aligned}$$

where typical operators preceeded by a period are element-wise.

7. for each column $i = 1..4$

$$iq(:,i) = f1 \mathbin{.*} p_0(:,i) + f2 \mathbin{.*} p_1(:,i)$$

The inquisitive reader is directed to more detailed sources of information on matters related to quaternion interpolation and the duality of quaternion orientation representation: [Wik06], [Sho85], [Joh03], [Kir92].

Bibliography

[AB94] Ronald Azuma and Gary Bishop. Improving static and dynamic registration in an optical see-through hmd. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 197–204, New York, NY, USA, 1994. ACM Press.

[AB95] Ronald Azuma and Gary Bishop. A frequency-domain analysis of head-motion prediction. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 401–408, New York, NY, USA, 1995. ACM Press.

[AB98] Y. Akatsuka and G.A. Bekey. Compensation for end to end delays in a vr system. In *Virtual Reality Annual International Symposium, 1998. Proceedings IEEE 1998*, pages 156–159, March 1998.

[ABK+04] Sudhir Aggarwal, Hemant Banavar, Amit Khandelwal, Sarit Mukherjee, and Sampath Rangarajan. Accuracy in dead-reckoning based distributed multi-player games. In *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 161–165, New York, NY, USA, 2004. ACM Press.

[AHJ+01] Robert S. Allison, Laurence R. Harris, Michael Jenkin, Urszula Jasiobedzka, and James E. Zacher. Tolerance of temporal delay in virtual environments. In *VR '01: Proceedings of the Virtual Reality 2001 Conference (VR'01)*, page 247, Washington, DC, USA, 2001. IEEE Computer Society.

[Aro97] Jesse Aronson. Dead reckoning: Latency hiding for networked games. http://www.gamasutra.com/features/19970919/aronson_01.htm, September 1997.

[Azu95] Ronald Azuma. *Predictive Tracking for Augmented Reality*. PhD thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, February 1995.

[AZWS04] Robert S. Allison, James E. Zacher, David Wang, and Joseph Shu. Effects of network delay on a collaborative motor task with telehaptic and televisual feedback. In *VRCAI '04: Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, pages 375–381, New York, NY, USA, 2004. ACM Press.

[Ber01] Y. W. Bernier. Latency compensation methods in client/server in-game protocol design and optimization. In *Proc. of Game Developers Conference'01*, 2001.

[BH97a] Robert G. Brown and Patrick Y.C. Hwang. *Introduction to random signals and applied Kalman filtering: with MATLAB exercises and solutions*. John Wiley & Sons, New York, N.Y., 3rd edition, 1997.

[BH97b] Robert G. Brown and Patrick Y.C. Hwang. *Introduction to random signals and applied Kalman filtering: with MATLAB exercises and solutions*, chapter 9. John Wiley & Sons, New York, N.Y., 3rd edition, 1997.

[BMK96] S. Basu, A. Mukherjee, and S. Klivansky. Time series models for internet traffic. In *INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, volume 2, pages 611–620, March 1996.

[Bor00] Michael S. Borella. Source models of network game traffic. *Computer Communications*, 23(4):403–410, February 2000.

[CBIO04] D. Choukroun, I.Y. Bar-Itzhack, and Y. Oshman. A novel quaternion kalman filter. Accepted by IEEE Transactions on Aerospace and Electronic Systems, January 2004.

[CM96] J.L. Crassidis and F.L. Markley. Attitude estimation using modified rodrigues parameters. In *Flight Mechanics/Estimation Theory Symposium*, pages 71–83, Greenbelt, MD, May 1996. NASA-Goddard Space Flight Center.

[Cro05] S.F. Crone. Stepwise selection of artificial neural network models for time series prediction. *Journal of Intelligent Systems*, 14(2-3):99–121, 2005.

[CS97] Michael V. Capps and P. David Stotts. Research issues in developing networked virtual realities: Working group report on distributed system aspects of sharing a virtual reality. In *WET-ICE '97: Proceedings of the 6th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises*, pages 205–212, Washington, DC, USA, 1997. IEEE Computer Society.

[DIS96] Ieee standard for distributed interactive simulation - application protocols. IEEE Std 1278.1-1995, March 1996.

[DJ00] Gregory Dudek and Michael Jenkin. *Computational Principles of Mobile Robotics*. Cambridge University Press, Cambridge, UK, 2000.

[DKL98] Erik B. Dam, Martin Koch, and Martin Lillholm. Quaternions, interpolation and animation. Technical Report DIKU-TR-98/5, University of Copenhagen, July 1998.

[DS99] P. DeLeon and C.J. Sreenan. An adaptive predictor for media playout buffering. In *Acoustics, Speech, and Signal Processing, 1999. ICASSP '99. Proceedings.*, volume 6, pages 3097–3100, March 1999.

[FÖ2] Johannes Färber. Network game traffic modelling. In *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*, pages 53–57, New York, NY, USA, 2002. ACM Press.

[Gel74] Arthur Gelb, editor. *Applied optimal estimation*. M.I.T. Press, Cambridge, Mass., 1974.

[GP94] N. Groschwitz and G. Plyzos. A time series model of long-term traffic on the nsfnet backbone. In *Proceedings of the IEEE International Conference on Communications (ICC'94)*, May 1994.

[HMA+02] K. Hikichi, H. Morino, I. Arimoto, K. Sezaki, and Y. Yasuda. The evaluation of delay jitter for haptics collaboration over the internet. In *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, volume 2, pages 1492–1496, November 2002.

[IPt] Netfilter: firewalling, nat, and packet mangling for linux. http://www.netfilter.org/projects/iptables/.

[IS900] 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.

[JAE00a] J. Y. Jung, B. D. Adelstein, and S. R. Ellis. Discriminability of prediction artifacts in a time-delayed virtual environment. In *Proceedings, 44th Annual Meeting of the Human Factors and Ergonomics Society (IEA 2000/HFES 2000)*, volume 1, pages 499–502, Santa Monica CA, 2000.

[JAE00b] J. Y. Jung, B. D. Adelstein, and S. R. Ellis. Predictive compensator optimization for head tracking lag in virtual environments. In *Proceedings, IMAGE (Innovative Modeling and Advanced Generation of Environments) 2000*, pages 123–132, Scottsdale AZ, 2000.

[JBB92] V. Jacobson, R. Braden, and D. Borman. Rfc1323: Tcp extensions for high performance, May 1992.

[JJL00] Jr. Joseph J. LaViola. A discussion of cybersickness in virtual environments. *SIGCHI Bull.*, 32(1):47–56, 2000.

[JL04] S.J. Julier and Joseph J. LaViola. An empirical study into the robustness of split covariance addition (sca) for human motion tracking. In *American Control Conference, 2004. Proceedings of the 2004*, volume 3, pages 2190 – 2195. IEEE Press, June 2004.

[Joh03] Michael Patrick Johnson. *Exploiting Quaternions to Support Expressive Interactive Character Motion*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, February 2003.

[JUDW95] S.J. Julier, J.K. Uhlmann, and H.F. Durrant-Whyte. A new approach for filtering nonlinear systems. In *American Control Conference, 1995. Proceedings of the*, volume 3, pages 1628–1632, June 1995.

[KEP97] A. Kiruluta, M. Eizenman, and S. Pasupathy. Predictive head movement tracking using a kalman filter. *Systems, Man and Cybernetics, Part B, IEEE Transactions on*, 27(2):326–331, April 1997.

[Kir92] David Kirk, editor. *Graphics Gems III*. The Graphics Gems Series. Academic Press, Inc., Cambridge, MA, USA, 1992.

[Kui99] Jack B. Kuipers. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*. Princeton University Press, Princeton, New Jersey, 1999.

[LaV02] Joseph J. LaViola. The predictive tracking algorithm testing suite: A tool for developing and analyzing predictive tracking algorithms. Technical Report CS-02-07, Brown University, Department of Computer Science, Providence, RI, July 2002.

[LaV03a] Joseph J. LaViola. Double exponential smoothing: an alternative to kalman filter-based predictive tracking. In *EGVE '03: Proceedings of the workshop on Virtual environments 2003*, pages 199–206, New York, NY, USA, 2003. ACM Press.

[LaV03b] Joseph J. LaViola. A testbed for studying and choosing predictive tracking algorithms in virtual environments. In *EGVE '03: Proceedings of the workshop on Virtual environments 2003*, pages 189–198, New York, NY, USA, 2003. ACM Press.

[LaV03c] Jr. LaViola, J.J. A comparison of unscented and extended kalman filtering for estimating quaternion motion. In *American Control Conference, 2003. Proceedings of the 2003*, volume 3, pages 2435 – 2440. IEEE Press, June 2003.

[LaV03d] Jr. LaViola, J.J. An experiment comparing double exponential smoothing and kalman filter-based predictive tracking algorithms. In *Virtual Reality, 2003. Proceedings. IEEE*, pages 283 – 284, March 2003.

[LaV04] Joseph J. LaViola. Understanding predictive tracking algorithms in virtual environments. http://www.cs.brown.edu/ jjl/ptracking/ptracking.html, 2004.

[Mau00] Martin Mauve. How to keep a dead man from shooting. In *IDMS '00: Proceedings of the 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*, pages 199–204, London, UK, 2000. Springer-Verlag.

[MCML05] P. Melin, O. Castillo, A. Mancilla, and M. Lopez. Simulation and forecasting complex economic time series using neural network models. *Journal of Intelligent Systems*, 14(2-3):193–212, 2005.

[MH99] Yosi Mass and Amir Herzberg. Vrcommerce - electronic commerce in virtual reality. In *EC '99: Proceedings of the 1st ACM conference on Electronic commerce*, pages 103–109, New York, NY, USA, 1999. ACM Press.

[Pat04] David A. Patterson. Latency lags bandwith. *Commun. ACM*, 47(10):71–75, 2004.

[PK99] Kyoung Shin Park and Robert V. Kenyon. Effects of network characteristics on human performance in a collaborative virtual environment. In *VR '99: Proceedings of the IEEE Virtual Reality*, page 104, Washington, DC, USA, 1999. IEEE Computer Society.

[PW02a] Lothar Pantel and Lars C. Wolf. On the impact of delay on real-time multiplayer games. In *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 23–29, New York, NY, USA, 2002. ACM Press.

[PW02b] Lothar Pantel and Lars C. Wolf. On the suitability of dead reckoning schemes for games. In *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*, pages 79–84, New York, NY, USA, 2002. ACM Press.

[SFR04] W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff. *UNIX Network Programming: The Socket Networking API*, volume 1. Addison-Wesley Professional, 3rd edition, 2004.

[Sho85] Ken Shoemake. Animating rotation with quaternion curves. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254, New York, NY, USA, 1985. ACM Press.

[Sta02] Kay M. Stanney, editor. *Handbook of Virtual Environments: Design, Implementation, and Applications*. Lawrence Erlbaum Associates, Inc., Mahwah, New Jersey, 2002.

[WB95] Greg Welch and Gary Bishop. An introduction to the kalman filter. Technical Report TR95-041, University of North Carolina at Chapel Hill, 1995.

[Wel96] Greg Welch. *SCAAT: Incremental Tracking with Incomplete Information*. PhD thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1996.

[Wik06] Slerp. http://en.wikipedia.org/wiki/Slerp, February 2006.

[WO94] Jiann-Rong Wu and Ming Ouhyoung. Reducing the latency in head-mounted displays by a novel prediction method using grey system theory. *Computer Graphics Forum*, 13(3):C/503–C/512, 1994.

[WO95] Jiann-Rong Wu and Ming Ouhyoung. A 3d tracking experiment on latency and its compensation methods in virtual environments. In *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 41–49, New York, NY, USA, 1995. ACM Press.

[WO00] Jiann-Rong Wu and Ming Ouhyoung. On latency compensation and its effects on head-motion trajectories in virtual environments. *The Visual Computer*, 16(2):79–90, March 2000.

[Wol98] Rich Wolski. Dynamically forecasting network performance using the network weather service. Technical Report TR-CS96-494, University of California, San Diego, La Jolla, CA, January 1998.

[WS95] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated: The Implementation*, volume 2. Addison-Wesley Publishing Company, Reading, Massachusetts, USA, 1995.

[WSH99] Rich Wolski, Neil T. Spring, and Jim Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5-6):757–768, October 1999.

[YLLG03] T. Yensen, J.P. Lariviere, I. Lambadaris, and R.A. Goubran. Hmm delay prediction technique for voip. *Multimedia, IEEE Transactions on*, 5(3):444– 457, September 2003.

[ZA05] Sebastian Zander and Grenville Armitage. A traffic model for the xbox game halo 2. In *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pages 13–18, New York, NY, USA, 2005. ACM Press.

[Zil97] Dennis G. Zill. *A First Course in Differential Equations with Modeling Applications*. Brooks/Cole Publishing Company, Pacific Grove, CA, USA, 6th edition, 1997.

[ZM00] Paul Zarchan and Howard Musoff. *Fundamentals of Kalman filtering: a practical approach*, volume 190 of *Progress in astronautics and aeronautics*. American Institute of Aeronautics and Astronautics, Inc., Reston, Virginia, 2000.

[ZSG04] J. Zhou, X. Shen, and N.D. Georganas. Haptic tele-surgery simulation. In *Haptic, Audio and Visual Environments and Their Applications, 2004. HAVE 2004. Proceedings. The 3rd IEEE International Workshop on*, pages 99–104, October 2004.