7. International Immersive Projection Technologies Workshop
9. Eurographics Workshop on Virtual Environments (2003)
J. Deisinger, A. Kunz (Editors)

A Testbed for Studying and Choosing Predictive Tracking Algorithms in Virtual Environments

Joseph J. LaViola Jr.

Brown University Technology Center
for Advanced Scientific Computing and Visualization
PO Box 1910, Providence, RI, 02912, USA
jjl@cs.brown.edu

Abstract

We present a testbed for comparing predictive tracking algorithms that allows virtual environment system developers to make better choices about which predictors to use in their environments and aids researchers in determining how predictors work across various virtual environment configurations. Our testbed saves the virtual environment developer and researcher both time and effort with the important task of reducing dynamic tracking error and masking latency. The testbed consists of three components: a prediction algorithm library, a motion data repository, and a graphical testing application which provides users with the ability to test different predictive tracking algorithms across a variety of user motion sequences. The testbed provides enough generality for testing across different algorithmic and system parameters such as sampling rate, prediction time, and noise variance. The paper describes the contents of the predictor library and how to extend it, the types of motion data sets collected thus far, the motion data preparation methodology, and the graphical testing application's functionality and architecture. A simple testing scenario showing output from the testbed is also presented.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three-Dimensional Graphics and Realism]: Virtual Reality I.6.7 [Simulation Support Systems]: Environments

1. Introduction

Choosing prediction algorithms for virtual environment(VE) tracking is a challenging task. Certain algorithms may perform better or worse depending on the underlying tracking system's sampling rate, noise variance and tracking technology (e.g., magnetic, acoustic, inertial, hybrid). The types of user motion, including head and hand, play a significant role in determining what prediction algorithms to use. Another critical factor in prediction algorithm determination is the prediction time (i.e., how far one has to predict). Some prediction algorithms may be more or less robust as prediction time increases. Almost all prediction algorithms contain one or more parameters that are used for tuning to optimize performance. Therefore, a significant aspect in determining what prediction algorithms to use is in adjusting an algorithm's parameter values. These adjustments are nontrivial in the sense that an optimal parameter setting for one type of user motion may not be optimal for another.

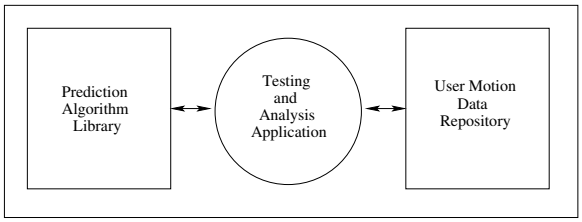

Figure 1: *The three components of our predictive tracking algorithm testbed.*

Currently, there is not a common framework for performing predictor comparisons that address all of these issues. As a result, most predictive tracking work deals with only subsets of the space of issues that arise with developing and utilizing predictive tracking. To address this problem, we developed a testbed for comparing different predictors that

allows VE system developers to make better choices about which predictive tracking algorithms to use in specific environments and aids researchers in determining how prediction algorithms work across various VE configurations. With such a testbed, we provide a set of tools in one central location that will save developers and researchers valuable time and effort in the task of improving the fidelity of their VE systems and applications.

We concluded that a common testbed for studying these algorithms should have three essential components (see Figure 1). These components should work together as a single unit to provide a robust tool set. The first component is an easily extensible prediction algorithm library containing a wide variety of different predictors. The second component is the motion data repository and its associated data preparation methodology. Currently, the repository contains head and hand motion datasets from a Cave-like environment encompassing a variety of motion styles, plus the publicly available head motion datasets from Azuma and Bishop's 1994 Siggraph paper[2]. The data preparation methodology provides an automatic way of adding additional datasets to the repository which is an important part of the testbed, since we want developers and researchers to add datasets from their own VR systems. Finally, the third component is a testing and analysis application which effectively connects the prediction algorithm library and the dataset repository together and provides a number of different testing strategies and error metrics. These three components, working together, save the VE system developer and researcher time and effort by providing them predictor implementations, many different motion datasets, and an application for running experimental studies.

In the next section, we present some related work on the study of predictive tracking algorithm performance. Section 3 presents the details of the prediction algorithm testbed by describing the predictors currently in the library and how to add new ones, the types of motion sequences currently available and the procedure for pre-processing the data, and the testing application's features and architecture. Section 4 shows an example scenario of how the testbed might be used by examining the performance of a least squares predictor. Section 5 discusses future work and Section 6 concludes the paper and provides a URL for downloading the testbed.

2. Related Work

A significant body of work exists on predictive tracking algorithms in both augmented and virtual reality. However, there has been little work on comparing different predictive tracking algorithms head-to-head on the same data and hardly any work on developing an easily accessible testbed to facilitate such studies. Azuma and Bishop compared Kalman filter-based predictors for a see-through HMD[2] and performed tests with and without inertial sensors on three head motion datasets. Although their system was unique and they showed good performance, a testbed with a robust prediction algorithm library would have enabled them to try out a number of other prediction algorithms that may have yielded better performance. Wu and Ouhyoung[19] presented experiments comparing a Kalman filter-based predictor, simple extrapolation, and Grey System prediction[20], however they were limited to only testing with head motion sequences based on two applications. Having a testbed with a variety of motion datasets from different application types and body parts would have enabled them to easily conduct a wider range of experiments.

Besides using the testbed as a tool for making better choices about what prediction algorithms to use for specific VR systems, it also can be used to study predictors more generally. There has been little work done in this area. For example, Azuma and Bishop's study[2] did not take differing sampling rates, prediction times, and sensor noise variances into account. Friedmann et.al. used Kalman filter-based predictors for hand motion, but their studies were limited (i.e., drumming) and were not thoroughly tested across different system parameters[7]. This is one of the few cases where a predictive tracking algorithm was tested with motion coming from a body part other than the user's head. Other Kalman filter-based predictors have been developed and studied with similar restrictions on system parameters and user motion[11, 15, 16]. In addition, most of these prediction algorithm's parameters were tuned using a limited amount of motion datasets, optimizing their performance to motion data with similar characteristics. However, if the predictors need to be applied to other types of motion, these tuned parameters may not yield accurate results. A testbed with a wide variety of motion data sequences makes algorithm tuning easier to perform and could lead to the development of predictors that adapt their parameters to the types of user motion under consideration.

Finally, Azuma and Bishop's frequency-domain analysis of predictive tracking algorithms is similar to our work in that they developed a theoretical "testbed" for characterizing predictor behavior based, in part, on prediction time and the motion signal's power spectrum[3]. With their work, extensive comparative studies are not as important since they can obtain "closed form solutions" which provide a powerful measure of predictor accuracy. However, their theoretical framework does have limitations in that they do not take into account sampling rate or static sensor error. In addition, they were not developed with hand motion in mind and do not take nonlinear or adaptive predictors into account. Although our testbed is designed for mostly empirical testing, we believe that it can act as a complement to their formalisms and also provide the ability for more general and robust predictor analysis.

3. Predictive Tracking Algorithm Testbed

The testbed contains three essential components including am extensible prediction algorithm library, a motion data repository with an associated data preparation methodology, and a testing application which provides a number of useful features for experimenting with prediction algorithms under various circumstances. The following sections describe the testbed in more detail.

3.1. Prediction Algorithm Library

The prediction algorithm library provides an easily accessible collection of predictive tracking algorithms that VR system developers and researchers can use in their own work. The library contains a variety of different predictors, written in C++, that have been presented in previous predictive tracking papers and taken from other disciplines such as economic time series forecasting and control theory. It is by no means all inclusive as it is still expanding (see Section 5 for a list of other predictors we plan to incorporate into the library).

3.1.1. Current Predictors

The library, in its current form, is divided into four categories: simple extrapolation routines, integerized predictors, filter-based approaches, and multiple model adaptive estimation. We chose these predictors as our initial set for two main reasons. First, they are not as training intensive as some other types of predictors such as neural networks. Second, we wanted to use the library's development as a chance to try some new predictors that have not been applied to the VE tracking domain. To describe all the algorithmic details and tradeoffs of each predictor is beyond the scope of this paper. Therefore, we only briefly discuss them here.

There are four simple extrapolation routines available in the predictor library based on Lagrangian polynomials, cubic splines, least square approximation using orthogonal polynomials[8], and 2nd order Taylor expansions. A common theme with the simple extrapolation routines is that they all are based on approximating a function with polynomials using the previous n user poses from an evolving motion sequence, and then using that approximating function to extrapolate the user's future location. The different predictors in this category all use different function approximation schemes.

The "intergerized" predictors include exponential smoothing[4] and Grey system prediction[19]. These predictors are called "intergerized" because future poses are forecast as integer multiples of the time between samples, Δt. Such a prediction scheme presents no difficulties when predicting $i\Delta t$ steps into the future. However, modifications to the algorithms are required if i is not an integer. For general applicability, the predictors in this category are modified to handle any prediction time i by predicting $\lfloor i \rfloor \Delta t$ and $\lceil i \rceil \Delta t$ steps into the future and then interpolating to predict the user pose at the exact time.

The filter-based category includes four different types of predictor/corrector filtering techniques including Kalman filtering[18] and extended Kalman filtering[18], the most common predictive tracking algorithms used today, plus unscented Kalman filtering[17] and Covariance Intersection[9]. In general, these filters use an underlying process model to make an estimate of the current state of the system and then correct the estimate using any tracker measurements available. Then, after the correction is made, the process model is used to make a prediction.

The last class of predictors in the library utilize Multiple Model Adaptive Estimation (MMAE)[12]. MMAE-based prediction assumes that either a single process model or a single parameter setting cannot adequately cover the wide range of user motions that are possible in a virtual environment. Therefore, a bank of different Kalman filters (or extended Kalman filters for orientation), work in parallel to better predict user motion by adapting to changing dynamics.

3.1.2. Extending the Library

Although the library contains the majority of the predictive tracking algorithms used today in both VR research and practical applications, it is important that the library is extensible so new predictors can be easily added when they become available. To add a predictor to the library, the developer must make a subclass off of the `Predictor` abstract base class which contains the virtual `prepare` and `predict` functions as well as the `ParameterBundle` object. This object holds all algorithmic parameter variables that the predictors might need. These parameters can be set either through the testing application (see Section 3.3) or in the context of a real predictive tracking application. The `prepare` and `predict` routines must be redefined for a given prediction algorithm with the `prepare` routine performing necessary initializations and the `predict` routine taking the current user pose and prediction time and returning a predicted pose. All predictors are stored in a list in the `PredictorHolder` object. This object creates the predictor objects and allows for easy accessibility using an enumerated type which indexes into the predictor list. Using this approach, we have been able to easily and quickly add predictors to the library.

3.2. User Motion Data Repository

The second major component in our testbed is the user motion data repository. The main goal of this repository is to have a set of user motion datasets from a variety of different motion styles and tracking systems in one central location, thus making it easier to test and analyze prediction algorithms.

3.2.1. Motion Datasets

Currently, the datasets in the repository are from two types of tracking systems. The first is an Intersense IS900 used in our Cave, and the second is Azuma and Bishop's custom tracking system used in an AR setup[2]. Each collected dataset contains pose records consisting of a timestamp, a position vector and a unit length quaternion.

With the first tracking system, we collected both head and hand user data representing typical motions found in a variety of our Cave applications and interaction techniques[1, 10, 13, 14, 21, 22]. All of these datasets are approximately 20 seconds in length and sampled between 210 and 217 Hz.

Head motions can be divided into three categories. The first category is simple head movement where the user stands roughly in place and rotates either the head or body to view the different Cave display screens[1]. The second category is head movement from the user both walking and looking around in the Cave. This type of motion is a common occurrence when using the StepWIM navigation technique[13]. The third category is head motion from the user examining a fixed object by moving up and down and side to side in order to gain perspective about its structure[22].

The hand motion sequences are also be divided into three categories. The first category is hand motion used in navigating through a VE such as the type of motion found using the LaserGrab technique [21]. The second category is hand motion used in object selection, manipulation, and placement (i.e., grabbing and manipulating a streamline in a scientific visualization application[14]). The third category is freeform hand motion from users working in CavePainting[10], a tool for creating 3D artistic scenes.

In addition to these Cave-based datasets, we also have incorporated the Swing, Walkaround, and Rotation head motion datasets from Azuma and Bishop's custom AR tracking system[2]. These datasets give the repository more variety in the types of head motion dynamics available for prediction algorithm testing (see Section 5 for plans for expanding the repository further).

3.2.2. Data Preprocessing

In order to determine how well a given prediction algorithm is performing, we need comparison data. Additionally, we want the test data to be in a format which handles the variety of different prediction algorithms as well as different testing procedures. Comparing predicted output with reported user poses is problematic since these records have noise and small distortions associated with them. Thus, any comparison with the recorded data would count tracking error with the prediction error. Therefore, we require a set of "ground truth" datasets which are assumed to be the user's actual motion. Reducing these small distortions is a difficult problem, however, we can reduce the static tracker noise[2]. Although these "ground truth" datasets may not be the exact motion of a given user, we can consider them to be close to what the user has done and this gives us appropriate data to test against.

To develop the "ground truth" datasets, each dataset was resampled at a higher sampling rate to make each motion sequence's sampling rate a little over 1kHz using lowpass interpolation. The resampling makes testing over different sample rates much easier to perform. The resampled motion sequences have noise associated with them from both the lowpass interpolation and the sensors themselves. To create truth signals, we examined the power spectrum of each to determine the lowpass and highpass parameters to a zero-phase shift filter used to remove high frequency noise. Note that once the signals are filtered, the quaternions are renormalized to ensure they are of unit length.

Many predictors that achieve high levels of accuracy utilize velocity and/or acceleration information from other sensors like gyroscopes and accelerometers[2]. Therefore, it is important to have the flexibility to examine these predictors. Since we do not have access to these types of sensors, we do the next best thing by simulating them. Using the clean motion sequences, first and second derivatives (i.e., velocity and acceleration) are calculated using 4th order finite central differencing schemes. Because the derivatives were calculated on clean signals with high sampling rates, the 4th order differencing schemes provide a nice approximation to what one would get with today's gyroscopes and accelerometers. With the derivative information for both position and orientation, each pose record contains a timestamp, a position, velocity, and acceleration vector and an orientation quaternion, an angular velocity pure vector quaternion, and an angular acceleration pure vector quaternion. These motion sequences are then ready for use in prediction algorithm testing.

Although we plan to continually add more motion data sequences to the repository, we realize that most VE developers and researchers will want to use their own motion datasets in addition to already existing repository datasets. Therefore, to make it easier to add new datasets to the repository, we have automated our data preparation process. Users can simply record motion sequences from their own applications and run our data preprocessing routine to add new datasets to the repository.

3.3. Testing and Analysis Application

The testing and analysis application (see Figure 2) connects the motion data repository and the prediction algorithm library. Having a testing application helps alleviate the extra effort required to take full advantage of the testbed. It also makes it easier to study prediction algorithms. However, the first two testbed components can be used independently with the VE system developer or researcher developing a testing application of their own.

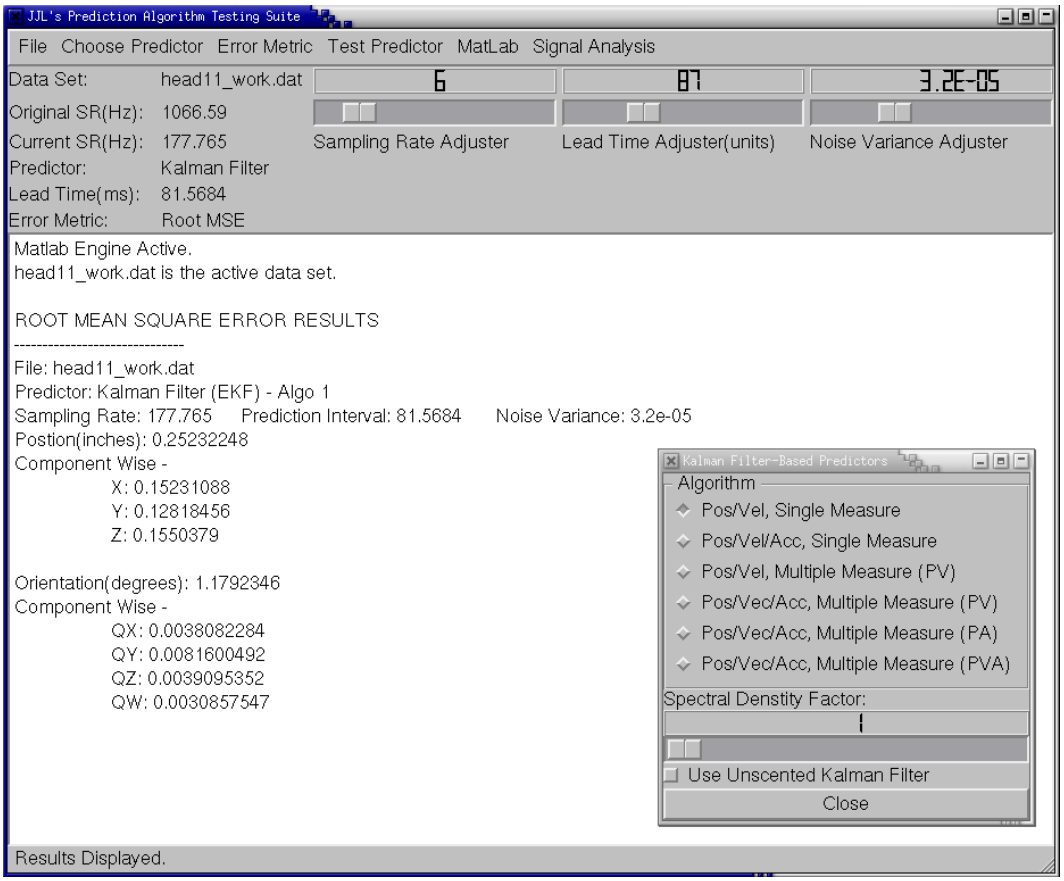

Figure 2: *A screenshot of the testing and analysis application. The window in the lower right is used for choosing various Kalman, extended Kalman, and unscented Kalman filter-based predictors.*

3.3.1. Functionality

The testing application provides a number of useful features. By setting specific parameters (i.e., sampling rate, prediction time, noise variance, and algorithmic parameters), users can run a single test which provides global and component wise error results for a number of different error metrics. The application also provides testing procedures for examining predictors across different sampling rates, prediction times, error variances, and algorithmic parameters (depending on the particular predictor). In addition, the application provides a full parameter test where sampling rate, prediction time, error variance, and algorithmic parameters are all variables. This test can be used for collecting data points across the space of parameters.

In general, determining the best way to compare different prediction algorithms is a difficult problem. There are a variety of error metrics that could be used, each one providing distinct insights into predictor performance. We chose to use three of the most common error metrics including root mean square (RMSE), max norm (also known as the L_∞ norm), and percent better. All of these metrics are computed for global position and orientation as well as the individual components of the position vector and quaternion. With the RMSE metric, we can see how a predictor is performing on average, and with the max norm, we can see a predictor's performance in the worst case. The percent better error metric determines how predictors are performing relative to doing no prediction at all (i.e, using the previous pose from the tracking system as the predicted pose) by counting the number of times the predicted pose is closer to the true pose than the previous pose and dividing by the total number of predictions made. Although the supported error metrics provide only a small sample of the possible ways of analyzing predictors both numerically and perceptually, they are a good starting point and provide a significant amount of information about predictor accuracy.

VE system developers and researcher might be interesting in a better understanding of the quantitative aspects of a motion dataset. Therefore, the testing application also has a signal analysis module which provides a variety of tools

for examining motion signals. These tools give the user the ability to easily examine power spectrums and sample autocorrelation functions as well as variability in the signals, total and average power, and absolute speeds and velocity variances. These tools can offer more insight into a predictor's performance by providing a better understanding of a motion signal's characteristics.

3.3.2. Software Architecture

The testing application's software architecture, shown in Figure 3, has four main components. The first, the user interface component, was developed with the Qt interface toolkit and provides the user with control panels for loading datasets, setting system parameters, setting prediction algorithm parameters, and running different experiments. In addition, it allows the user to start a MATLAB engine used for producing graphical error plots from the various test types.

The UI component has a two way connection to the second component, the Dispenser. The Dispenser acts as a link between the UI component and the prediction algorithm library. It takes all the required system, algorithmic, and test parameter information plus the active dataset from the UI component and simulates a tracking system by providing pose records to the prediction algorithm library's active predictor. Note that the library's only connection to the testing application is through the Dispenser because we want it to be as independent as possible. This independence makes porting the predictors to real-time VR and AR applications much easier. The Dispenser is also in charge of adding noise to the clean motion sequences according to the noise variance parameter (We assume zero mean Gaussian white noise).

The Dispenser's other job is to send the predicted as well as the clean and corrupted motion sequences to the Results and Error Analysis component. This component performs the error computations as well as prepares the results for both tabular and graphical display according to the particular test run. Finally, the fourth component is the Signal Analysis component. This component is also connected to the UI component and provides routines for understanding motion signal characteristics. The Signal Analysis component also connects to the MATLAB engine so it can utilize MATLAB's signal processing toolbox.

4. A Testbed Scenario

We present a simple example scenario illustrating how the testbed might be used to understand a prediction algorithm's performance. The example scenario also helps to show some of the output that the testing application produces. In this particular example, we examine the performance of the least squares orthogonal polynomial (LSOP) predictor[8] using a head dataset taken from the StepWim navigation technique[13]. We only consider position prediction. Examining user orientation prediction would be conducted in a similar manner.

We first run an algorithmic parameter test to find appropriate values for the LSOP predictor's two parameters, the window size and the polynomial degree. In this example, our criteria for choosing the best parameter values is the percent better metric since we want to maximize the times in which LSOP predictor is performing better than no prediction at all. With the tracker's sampling rate at approximately 215 Hz, the noise variance set to 2e-05 (determined from a static tracker), and the prediction time set to 75 ms (assuming around 20 fps) the output of the test is shown in Figure 4. The output from the test shows that the window size should roughly be 12 and the polynomial degree should be set to one.

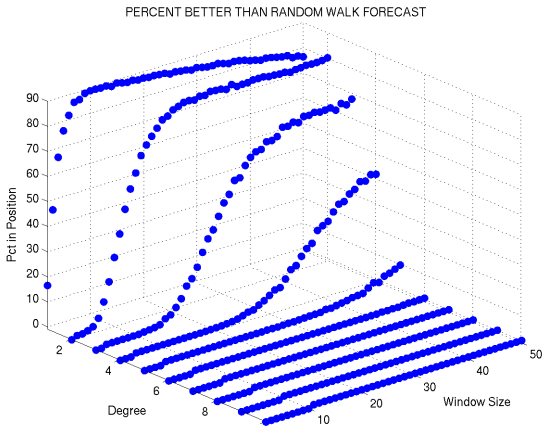

Figure 4: *Results from an algorithm parameter test for the LSOP predictor of position data.*

Using this information, we can run a simple test to get a better idea of the predictor's performance under the given conditions. This time we use the RMS error metric as our testing criteria to see how the predictor performs in the RMS sense. The test produces both textual (see Figure 5) and graphical (see Figures 6 and 7) output. The output shows that the predictor has an overall RMS error of 0.36 inches and slightly better performance for the individual position components. From Figure 6, we can see a close up view of a segment of X position data. Even though the prediction results look poor, the Y-axis values are in inches showing that the error is small. Note that since all the graphs are generated with MATLAB, users can utilize the built in zooming and 3D rotation features to more closely examine the output.

In most VR applications, the framerate rarely stays constant. Therefore, it would be nice to see how the LSOP predictor performs across different prediction times. We keep all other parameters constant and once again use the RMS error metric. Additionally, we superimpose the output of a test using no prediction at all to show the predictor's performance relative to simply using the previous pose as the predicted pose. The output of this test is shown in Figure

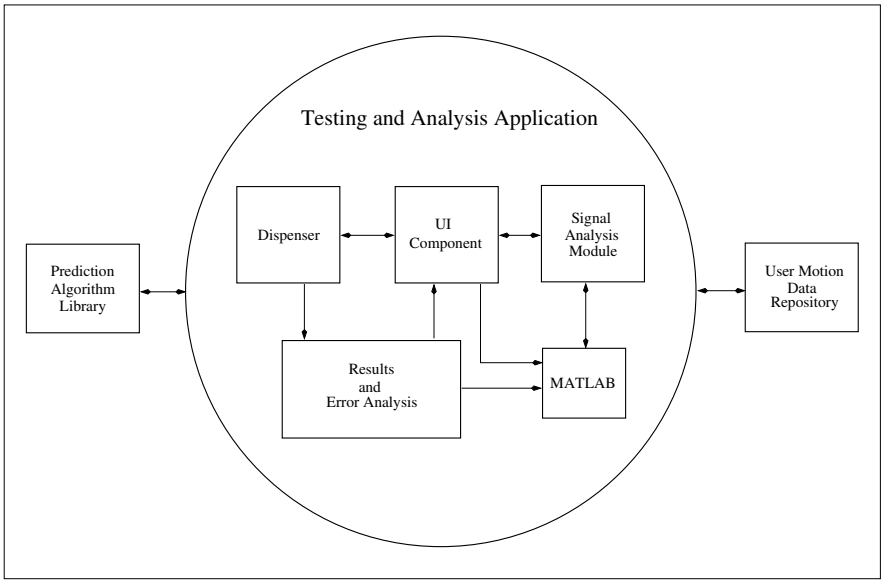

Figure 3: *A software architecture diagram of the predictive tracking algorithm testing and analysis application.*

```
head21_work.dat is the active data set.
Matlab Engine Active.

ROOT MEAN SQUARE ERROR RESULTS
------------------------------
File: head21_work.dat
Predictor: Local Polynomial - Algo 2
Window Size: 12 Degree: 1
Sampling Rate: 213.463
Prediction Interval: 75.8914
Noise Variance: 2e-05

Position(inches): 0.3623756
Component Wise -
                X: 0.22594799
                Y: 0.17811517
                Z: 0.22031471
```

Figure 5: *Textual output from a simple predictor test. These results were copied from the testing application's text window.*

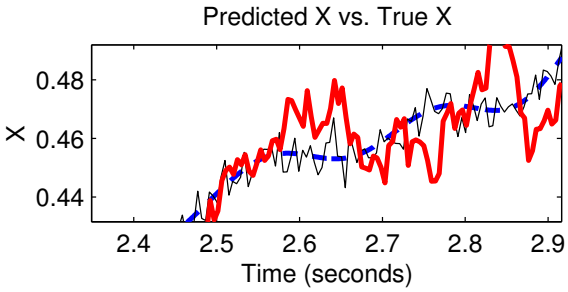

Figure 6: *A small segment of X position (in inches) data from the LSOP predictor test run. The dotted line represents the "truth" data, the thin solid line is the noise corrupted signal, and the thick solid line is the predicted signal.*

8 and shows that the LSOP predictor performs much better in the RMS sense over different prediction times compared with no prediction at all. The testbed supports the ability to superimpose any combination of predictor results onto the same graph so they can easily be compared. These initial results indicate that the LSOP predictor might be a good choice for our VR configuration although more tests on different datasets and comparisons with other predictors would still be needed.

5. Future Work

Although the testbed is quite robust, it is still evolving and there are a number of areas for future work. We wish to continue adding prediction algorithms to the library component such as neural network-based techniques, ARIMA forecasters[5], and support vector machine regressors[6]. More datasets are required from other VR and AR applications and tracking systems so more thorough and general analyses of prediction performance can be made. Therefore, we plan to continue to record new datasets and hope to solicit other researchers and VR system developers to record motions sequences to add to the repository. Finally, we wish to provide more functionality to our testing and analysis application by adding more error metrics, such as a screen error

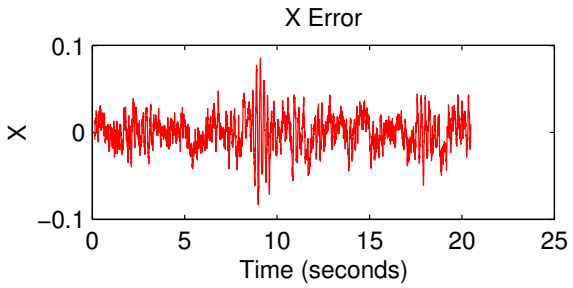

Figure 7: *The prediction error associated with the X position. These values represent the predicted values minus the "truth" values.*

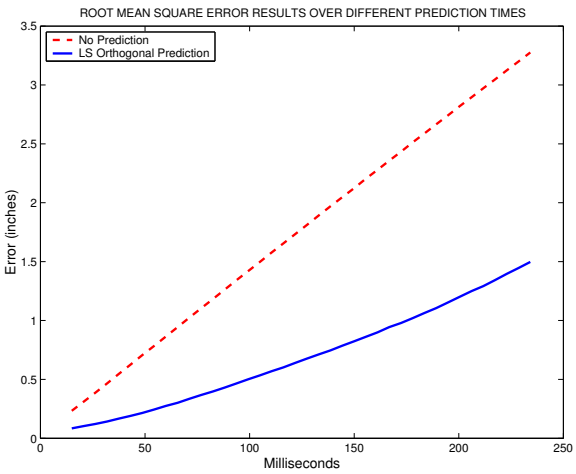

Figure 8: *Results from a prediction time test using the LSOP predictor and no prediction at all. The graph shows that the LSOP predictors performs much better in the RMS sense over different prediction times.*

metric[2], more signal analysis tools for quantifying motion sequences, and more sophisticated modules for understanding tracker noise characteristics and real time algorithmic parameter tuning.

6. Conclusion

We presented a testbed for studying predictive tracking algorithms which is comprised of a prediction algorithm library, a user motion data repository, and a testing and analysis application. The testbed is publicly available and can be downloaded at **`www.cs.brown.edu/people/jjl/ptracking/ptracking.html`**. The testbed makes it easier for VR system developers and researchers to study prediction algorithms and provides the foundation and necessary infrastructure for running a variety of experiments across many different conditions. Having this machinery in one location will ultimately save developers and researchers valuable time and effort which could otherwise be used for the task of masking latency, reducing dynamic tracking error, and improving the fidelity of VE applications.

Acknowledgments

Special thanks to Gary Bishop, Greg Welch, John Hughes, and Andy van Dam for valuable guidance and discussion. This work is supported in part by the NSF Graphics and Visualization Center, IBM, the Department of Energy, Alias/Wavefront, Microsoft, Sun Microsystems, and TACO.

References

1. Acevedo, Daniel, Eileen Vote, David H. Laidlaw, and Martha S. Joukowsky. Case Study: Archaeological Data Visualization in VR – Analysis of Lamp Finds at the Great Temple of Petra. In *Proceedings of IEEE Visualization 2001*, 493-496, 2001.

2. Azuma, Ronald and Gary Bishop. Improving Static and Dynamic Registration in a See-Through HMD. In *Proceedings of SIGGRAPH'94*, 197-204, 1994.

3. Azuma, Ronald and Gary Bishop. A Frequency Domain Analysis of Head Motion Prediction. In *Proceedings of SIGGRAPH'95*, 401-408, 1995.

4. Bowerman, Bruce J. and Richard T. O'Connell. *Forecasting and Time Series: An Applied Approach*. Duxbury Thomson Learning, 1993.

5. Chatfield, Chris. *Time-Series Forecasting*. Chapman and Hall/CRC, 2001.

6. Cristianini, Nello and John Shawe-Taylor. *An Introduction to Support Vector Machines: An Other Kernel-Based Learning Methods*, Cambridge University Press, 2000.

7. Friedmann, Martin, Thad Starner, and Alex Pentland. Device Synchronization Using and Optimal Linear Filter. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, 57-62, 1992.

8. Golub, Gene and James M. Ortega. *Scientific Computing: An Introduction with Parallel Computing*, Academic Press, 1993.

9. Julier, Simon J. and Jeffrey K. Uhlmann. General Decentralized Data Fusion With Covariance Intersection(CI). In *Handbook of Multisensor Data Fusion*, David Hall and James Llinas (eds.), CRC Press, 2001.

10. Keefe, D., Acevedo, D., Moscovich, T., Laidlaw, D., and LaViola, J. CavePainting: A Fully Immersive 3D Artistic Medium and Interactive Experience. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, 85-93, 2001.

11. Kiruluta, Andrew, Moshe Eizenman, and Subbarayan Pasupathy. Predictive Head Movement Tracking Using a Kalman Filter. In *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 27(2):326-331, April 1997.

12. Kyger, David W. and Peter S. Maybeck. Reducing Lag in Virtual Displays Using Multiple Model Adaptive Estimation. In *IEEE Transactions on Aerospace and Electronic Systems*, 34(4):1237-1247, 1998.

13. LaViola, Joseph, Daniel Acevedo, Daniel Keefe, and Robert Zeleznik. Hands-Free Multi-Scale Navigation in Virtual Environments. In the *Proceedings of the 2001 Symposium on Interactive 3D Graphics*,ACM Press, 9-15, 2001.

14. LaViola, Joseph. MSVT: A Virtual Reality-Based Multimodal Scientific Visualization Tool, In *Proceedings of the Third IASTED International Conference on Computer Graphics and Imaging*, 1-7, 2000.

15. Liang, Jiandong, Chris Shaw, and Mark Green. On Temporal-Spatial Realism in the Virtual Reality Environment. In *Proceedings of UIST'91*, 19-25, 1991.

16. Mazuryk, Tomasz and Michael Gervautz. Two-Step Prediction and Image Deflection for Exact Head Tracking in Virtual Environments. In *EUROGRAPHICS'95*, 29-41, 1995.

17. Wan, E. A., and R. van der Merwe. The Unscented Kalman Filter, In *Kalman Filtering and Neural Networks*, S. Haykin (ed.), Wiley Publishing, 2001.

18. Welch, Greg and Gary Bishop. An Introduction to the Kalman Filter. Technical Report TR 95-041, Department of Computer Science, University of North Carolina at Chapel Hill, 1995.

19. Wu, Jiann-Rong and Ming Ouhyoung. On Latency Compensation and Its Effects on Head-Motion Trajectories in Virtual Environments. *The Visual Computer* 16(2): 79-90, 2000.

20. Wu, Jiann-Rong and Ming Ouhyoung. A 3D Tracking Experiment on Latency and its Compensation Methods in Virtual Environments. In *Proceedings of UIST'95*, 41-49, 1995.

21. Zeleznik, Robert, Joseph LaViola, Daniel Acevedo, and Daniel Keefe. Pop Through Buttons for Virtual Environment Navigation and Interaction. In *Proceedings of Virtual Reality 2002*, 127-134, 2002.

22. Zhang, Song, C. Demiralp, Daniel Keefe, et al. Case Study: An Immersive Virtual Environment for DT-MRI Volume Visualization Applications. In *Proceedings of IEEE Visualization 2001*, 437-440, 2001.

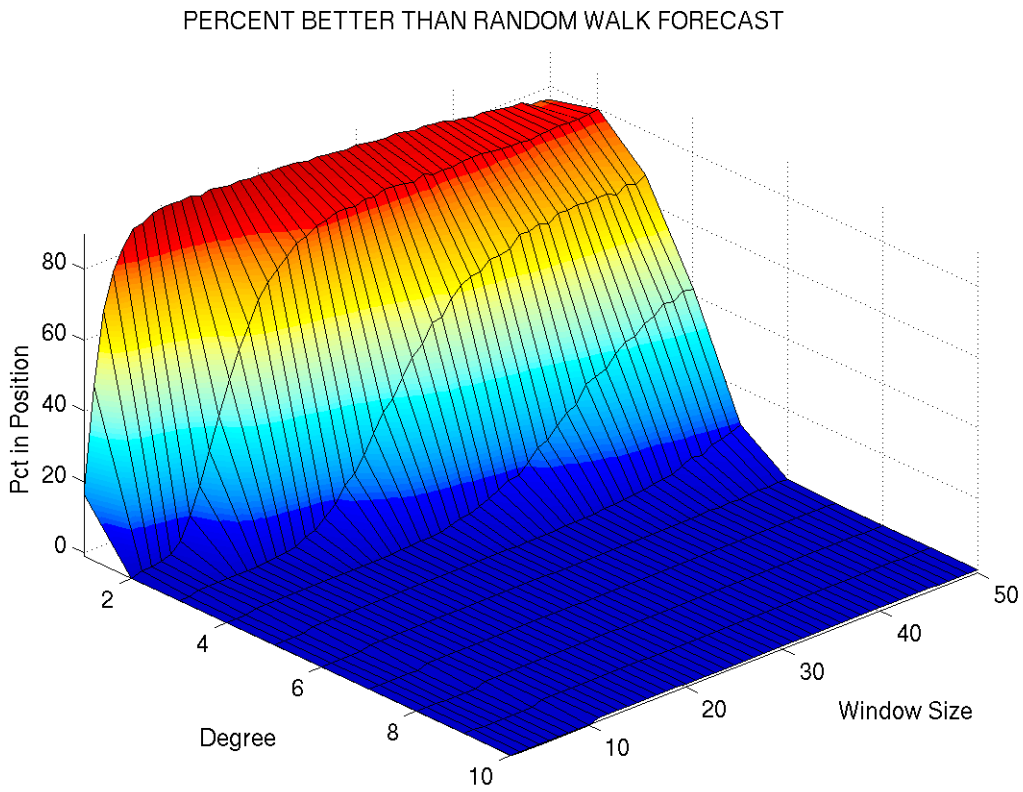

Figure 9: *An alternate graphical representation of the results from the algorithm parameter test (shown in Figure 4) for the LSOP predictor of position data.*