Improving Static and Dynamic Registration in an Optical See-through HMD

Ronald Azuma[§] Gary Bishop[†]

Department of Computer Science
University of North Carolina at Chapel Hill

Abstract

In Augmented Reality, see-through HMDs superimpose virtual 3D objects on the real world. This technology has the potential to enhance a user's perception and interaction with the real world. However, many Augmented Reality applications will not be accepted until we can accurately register virtual objects with their real counterparts. In previous systems, such registration was achieved only from a limited range of viewpoints, when the user kept his head still. This paper offers improved registration in two areas. First, our system demonstrates accurate static registration across a wide variety of viewing angles and positions. An optoelectronic tracker provides the required range and accuracy. Three calibration steps determine the viewing parameters. Second, dynamic errors that occur when the user moves his head are reduced by predicting future head locations. Inertial sensors mounted on the HMD aid head-motion prediction. Accurate determination of prediction distances requires low-overhead operating systems and eliminating unpredictable sources of latency. On average, prediction with inertial sensors produces errors 2-3 times lower than prediction without inertial sensors and 5-10 times lower than using no prediction at all. Future steps that may further improve registration are outlined.

CR Categories and Subject Descriptors: I.3.1 [**Computer Graphics**]: Hardware Architecture — *three-dimensional displays*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism — *virtual reality*
Additional Key Words and Phrases: Augmented Reality, registration, calibration

1 Motivation

Head-Mounted Displays (HMDs) and Virtual Environments have been a subject of great interest in the past few years. Less attention has been paid to the related field of Augmented Reality, despite its similar potential. The difference between Virtual Environments and Augmented Reality is in their treatment of the real world. Virtual Environments immerse a user inside a virtual world that completely replaces the real world outside. In contrast, Augmented Reality uses *see-through HMDs* that let the user see the real world around him. See-through HMDs augment the user's view of the real world by overlaying or compositing three-dimensional virtual objects with their real world counterparts. Ideally, it would seem to the user that the virtual and real objects coexisted. Since Augmented Reality supplements, rather than supplants, the real world, it opens up a different class of applications from those explored in Virtual Environments.

Augmented Reality applications attempt to enhance the user's perception and interaction with the real world. Several researchers have begun building prototype applications to explore this potential. A group at Boeing uses a see-through HMD to guide a technician in building a wiring harness that forms part of an airplane's electrical system [6][33]. Currently, technicians use large physical guide boards to construct such harnesses, and Boeing needs several warehouses to store all of these boards. Such space might be emptied for better use if this application proves successful. Other construction and repair jobs might be made easier if instructions were available, not in the form of manuals with text and 2D pictures, but as 3D drawings superimposed upon real objects, showing step-by-step the tasks to be performed. Feiner and his group demonstrated this in a laser printer maintenance application [11]. Feiner's group is also exploring displaying virtual documents in a sphere around the user, providing a much larger workspace than an ordinary workstation monitor. Medical applications might also benefit from Augmented Reality. A group at UNC scanned a fetus inside a womb with an ultrasonic sensor, then displayed a three-dimensional representation of that data in the same physical location as the fetus [4]. The goal is to provide a doctor with "X-ray vision," enabling him to gaze directly into the body. Conceptually, anything not detectable by human senses but detectable by machines might be transducted into something that we can sense and displayed inside a see-through HMD. Robinett speculates that ultimately Augmented Reality is about augmentation of human perception, about making the invisible visible (or hearable, feelable, etc.) [29].

While promising, Augmented Reality is barely at the demonstration phase today, and its full potential will not be realized until several technical challenges are overcome. One of the most basic is the registration problem. The real and virtual objects must be properly

Figure 1: Wooden frame for calibration and registration

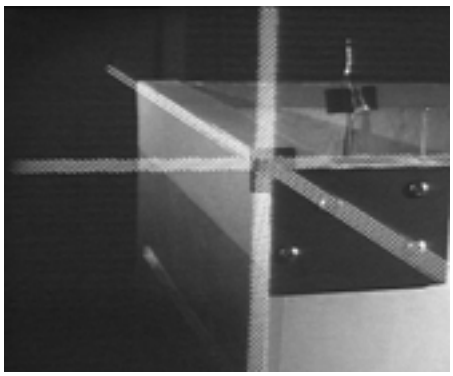


Figure 2: View seen in HMD, virtual axes on real frame

§† CB 3175 Sitterson Hall; UNC; Chapel Hill, NC 27599
§ (919) 962-1848 azuma@cs.unc.edu
† (919) 962-1886 gb@cs.unc.edu

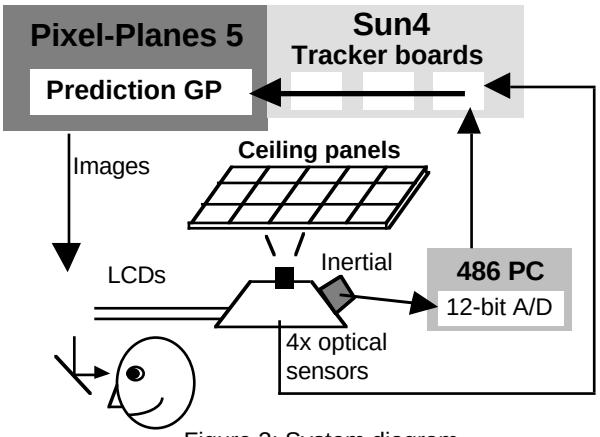


Figure 3: System diagram

aligned with respect to each other, or the illusion that the two coexist will be compromised. This is difficult to do because of the precision required. The human visual system is very good at detecting even small misregistrations, due to the resolution of the fovea and the sensitivity of the visual system to differences. A more tractable bound is provided by the relatively low resolution of the displays in modern HMDs. Errors of just a few pixels are noticeable.

Some applications have strict requirements for accurate registration. For example, imagine a surgeon wearing a see-through HMD displaying virtual objects that identify where and where not to make incisions. Unless the registration errors are kept below a few millimeters, the surgeon is not likely to trust the equipment. Without good registration, Augmented Reality may never be accepted in serious applications.

What causes registration errors? The main sources are:

- Distortion in the HMD optics
- Mechanical misalignments in the HMD
- Errors in the head-tracking system
- Incorrect viewing parameters (field of view, tracker-to-eye position and orientation, interpupillary distance)
- End-to-end system latency

The first four categories may be classified as *static* errors, because they cause registration errors even when the user keeps his head completely still. The 5th category, end-to-end latency, we call a *dynamic* error, because it has no effect until the user moves his head.

No one has achieved perfect registration with a see-through HMD. The current demonstrated state-of-the-art, as reported in the text and pictures of [4][11][18] achieves static errors on the order of 0.5 inches for an object at arm's length away from the HMD, from a small number of viewpoints. Dynamic errors can be much larger. With an end-to-end system latency of 100 ms, and a moderate head rotation rate of 100 degrees/sec, the angular dynamic error will be 10 degrees. At arm's length, this results in registration errors of 5 inches.

2 Contribution

This paper describes a system we built that tackles three of the main sources of registration errors: the head tracker, the determination of viewing parameters, and dynamic errors caused by system latency. We demonstrate improved static and dynamic registration as follows:

Improved static registration: Pictures and videos of existing Augmented Reality systems show registration from only a small number of viewpoints. The user is not allowed to translate or rotate the HMD very far from the initial viewpoint. There are two reasons for this limitation. First, most commercially available head-tracking systems do not provide sufficient accuracy and range to permit such movement without greatly increasing the static registration errors. Second, determining viewing parameters that work from just one viewpoint is much easier than determining parameters that work from many different viewpoints. We show that parameters yielding good registration from one viewpoint may result in static errors of a few inches at another viewpoint.

Our system is capable of robust static registration: keeping a virtual and real object closely aligned across many widely spaced viewpoints and view directions. We use a custom optoelectronic head tracker that provides sufficient range and accuracy. We also developed calibration techniques for determining the viewing parameters. The robust static registration is demonstrated by several still photographs taken from a single video sequence where the user walked 270 degrees around the registration object.

Improved dynamic registration: To reduce dynamic errors caused by the end-to-end latency in the system, we use predictive tracking techniques that guess where the user's head will be in the future. We equipped the see-through HMD with inertial sensors to aid head-motion prediction. A method for autocalibrating the orientation of the sensors on the user's head, along with other parameters, was developed. The reduction in dynamic error for three different motion runs is listed in a table.

The rest of the paper is organized as follows: first we give a brief overview of our system. Next, we describe and evaluate the static registration procedure. Then we do the same for dynamic registration. Each section describes problems encountered and points out limitations. The interested reader will find supplementary materials available in the CD-ROM version of this paper and many more details about our work in a technical report (the first author's dissertation) to be released later in 1994 by UNC Chapel Hill.

The main implications of our work are:

- Robust static registration within a few mm is possible, but it requires trackers with higher accuracy than Virtual Environment applications demand.
- Inertial-aided predictors can greatly reduce dynamic registration errors. On average, inertial-based prediction is 5-10 times more accurate than doing no prediction and 2-3 times more accurate than a representative non-inertial predictor.
- Augmented Reality systems that use predictive tracking require low-overhead operating systems and the elimination of unpredictable sources of latency.

3 System

Our system uses an optical see-through HMD. Its position and orientation are measured by an optoelectronic tracking system, and the images are generated by the Pixel-Planes 5 graphics engine. Readings taken from inertial sensors mounted on the HMD are digitized by an A/D board in a 486 PC. Figure 10 shows the overall setup, and Figure 3 provides a system diagram.

The optical see-through HMD [16] is shown in Figure 8. Optical combiners placed in front of both eyes overlay images on top of the user's view of the real world. The displays are color LCD monitors containing 340x240 pixels each. The field-of-view in each eye is approximately 30 degrees. We chose an optical see-through approach because it does not delay the user's view of the real world (see Section 7).

Tracking is provided by four optical sensors mounted on the back of the HMD, as seen in Figure 8. These are aimed upwards at an array of infrared LEDs mounted in ceiling panels above the user's head. By sighting several LEDs, and given the known geometry of the sensors on the head and the known locations of the beacons in the ceiling, the system is able to compute the position and orientation of the user's head. The data collection and processing are performed by three single-board 68030 and i860-based computers installed in the VME chassis of a Sun4 host [36].

The inertial sensors consist of three Systron Donner QRS-11 angular rate gyroscopes and three Lucas NovaSensor NAS-CO26 linear accelerometers. The gyros measure angular rates within the range of ±300 degrees/second, and the accelerometers detect acceleration within ±2 *g*. A 12-bit A/D board (National Instruments AT-MIO-16D) in a 486 PC digitizes the signals. To minimize noise, we built special power regulation circuits, used shielded twisted-pair wire, differential-mode A/Ds, and analog prefilters. A Bit3 bus extender sends the digitized readings to the optoelectronic tracker boards in the Sun4.

The virtual images are generated by Pixel-Planes 5 (Pxpl5), a highly parallel graphics engine consisting of i860-based Graphics Processors (GPs) to do geometry transformations and Renderer boards that rasterize primitives [14]. One of the GPs is used to run our prediction routine. The computed head positions and orientations from the optoelectronic tracker and the measured inertial signals are fed to this GP, which uses them to estimate future head

locations. The prediction GP also converts the predicted head locations into view matrices that the rest of Pxpl5 uses to generate the graphic images the HMD-wearer sees. Since the normal Pxpl5 software is optimized for maximum throughput and not minimal latency, we use different rendering software written by Mark Olano and Jon Cohen that minimizes Pxpl5 latency [8].

Special care was taken to use fast communication paths and low-overhead operating systems. Interprocessor communication is through shared memory, across Bit3 bus extenders, or through the 640 MByte/sec ring network within Pixel-Planes 5. UNIX is avoided except for initial setup and non-time-critical tasks, like reading buttons. This is discussed more in Section 5.3.

The end-to-end system latency varies from 50-70 ms, with 15-30 ms coming from the tracker, ~12 ms from the predictor, and 16.67 ms from Pxpl5. The rest comes from communication paths and delays caused by the asynchronous nature of our system.

Recording the images that the user sees inside the HMD to create the pictures in this paper was done by mounting a video camera in the right eye of a bust. The HMD was tied to this bust, then carried around.

4 Static registration

4.1 Previous work

Registration of real and virtual objects is not limited to see-through HMDs. Special effects artists seamlessly blend computer-generated images with live footage for films and advertisements. The difference is in time and control. With film, one can spend hours on each frame, adjusting by hand if necessary, and each shot is carefully preplanned. In Augmented Reality we have no such control: the user looks where he wants to look, and the computer must respond within tens of milliseconds (ms).

Deering [10] demonstrated an impressive registration of a real and virtual ruler in a head-tracked stereo system. Registration is a somewhat easier task in head-tracked stereo vs. an HMD-based system because the images do not change nearly as much for the same amount of head translation or rotation [9].

An extensive literature of camera calibration techniques exists in the robotics and photogrammetry communities (see the references in [21] as a start). These techniques digitize one or more pictures of an object of fixed and sometimes unknown geometry, locate features on the object, then use mathematical optimizers to solve for the viewing parameters. However, it is not clear how to directly apply these techniques to an *optical* see-through HMD, where no camera exists. Asking a user to identify the locations of many different points simultaneously while keeping his head still was judged too difficult to be reliable.

We have already mentioned several papers on Augmented Reality, but most focus on applications rather than on the details of calibration and registration. The sole exceptions are [6][18]. We compare our results with theirs in Section 4.4. Methods used to calibrate helmet-mounted sights on helicopter gunships provided the initial inspiration for our approach.

4.2 Problem statement

We reduce the problem to one real object, one set of virtual objects, and a desired registration linking the two. The real object is a wooden frame (Figure 1). The virtual objects are three mutually orthogonal extruded squares that form a coordinate system. The goal is to register the intersection of the three virtual bars with the front left corner of the frame, where the three bars run along the edges that touch the corner (Figures 2 & 14). This task is a good registration test because it's easy to detect small position and orientation errors along the edges of the frame.

Determining parameters that accomplish this task robustly is harder than one might first think. The naive approach, tried by several people in our laboratory, is the following: put a real object at a known or unknown position, wear the see-through HMD, then manually adjust the viewing parameters and the location of the virtual object until the registration "looks right." This rarely yields robust registration, because parameters and locations that work at one viewpoint may generate large registration errors at another. Figure 9 illustrates this. The picture on the left shows good registration at the initial viewpoint. But the same parameters yield a few inches of registration error when used at a different viewpoint, as seen in the picture on the right.

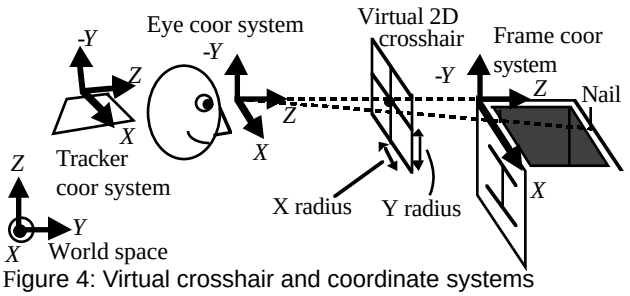


Figure 4: Virtual crosshair and coordinate systems

4.3 Procedure

Since camera calibration techniques seemed too difficult to apply in this domain, we thought of ways to directly measure the viewing parameters, using simple tasks that rely on geometric constraints. If the tasks are sensitive enough and our tracker is accurate enough, then this simple approach might work. We now describe these procedures that systematically determine the viewing parameters. The steps in order are:

- Measure the frame's location
- Determine the apparent center of the virtual image
- Measure the transformation between tracker space and eye space
- Measure the field-of-view (FOV)

We use only the right eye of our HMD, due to mechanical misalignments, color mismatches between the two display systems, and because a monocular display is sufficient to demonstrate registration.

1) Frame measurement: A digitization probe attached to a "hat" with four optical sensors returns the 3D position of the probe tip (Figure 13). We measure eight points on the frame edges where the red and green bars will lie, fit a pair of orthogonal lines through those points, and those determine the axis going down the third edge.

2) Apparent center of virtual image: The center of our 640x512 NTSC frame buffer need not be the center of the virtual image seen by the right eye, requiring off-center projections to properly render the images [12]. Assuming that the frame buffer covers the entire area visible through the optics, we can measure this center by drawing a 2D, non-head-tracked crosshair in the frame buffer (Figure 4). Four numbers specify this crosshair: the *(X,Y)* center coordinate, and the *X* and *Y* radii. The user determines the center by adjusting the *X* center and radius until the left and rightmost lines are equally spaced from the extreme visible edges of the display. This is tested by increasing the radius; both lines should disappear simultaneously or the center is incorrect. A similar procedure determines the *Y* center. Our measured center is (330, 255), which differs from the frame buffer center by about 10 pixels.

3) Eye->Tracker transformation: This is measured by the boresight operation, where a user wearing the HMD looks straight down the left top edge of the frame with his right eye (Figure 4). A 0.25" diameter pipe sticking out along the edge (Figure 1) helps the user line up accurately. Simultaneously, he centers the virtual crosshair with the corner of the frame and aligns the horizontal and vertical crosshair lines with the edges of the frame (Figure 11). Then the Eye coordinate system has the same orientation as the Frame coordinate system, and the *Z* axes coincide.

The boresight establishes the following relationship:

$$Q_{wf} = Q_{we}$$

where we define Q_{wf} to be the quaternion that rotates points and vectors from Frame space to World space, and Q_{we} rotates from Eye space to World space [30]. Then the desired Eye->Tracker orientation Q_{te} is computed by:

$$Q_{te} = Q_{tw} * Q_{we}$$
$$Q_{te} = (Q_{wt})^{-1} * Q_{wf}$$

where Q_{wt} is what the head tracker returns, and Q_{wf} is known from step 1.

The Eye->Tracker position offsets are measured by the boresight and one additional task. The position of the corner of the frame in World space is known, due to step 1. The position of the tracker origin in World space is returned by the head tracker. Therefore, we can draw a vector in World space from the corner of the frame to the

tracker origin. Rotating this vector by $(Q_{te})^{-1} * Q_{tw}$ transforms it to Eye space. Since Eye space and Frame space share the same orientation and their *Z* axes coincide, the *X* and *Y* values of the vector in Eye space are the *X* and *Y* Eye->Tracker offsets, in Eye space. To determine the *Z* offset, we need one more operation. Two nails are on top of the frame, one in front and one in the rear (Figures 1, 4, & 14). While performing the boresight, the user must also position himself so the front nail covers the rear nail. A red LED mounted on the rear nail helps the user detect when this occurs. The known locations of these two nails identify a specific distance along the frame's *Z* axis where the user's eye must be. Subtracting that from the corner->tracker vector in Eye space yields the *Z* component of the Eye->Tracker offset.

The user performs two boresights: one from a few feet away for greater orientation sensitivity, and one less than a foot away (matching the two nails) for greater position sensitivity.

4) FOV measurement: It suffices to measure FOV along the vertical *Y* direction in screen space, since scaling that by the frame buffer's aspect ratio yields the horizontal FOV. The crosshair's *Y* radius is set to 125 pixels so the top and bottom lines are easily visible. The user stands in front of the frame and lines up the top and bottom virtual crosshair lines with corresponding real lines drawn on the frame's front surface (Figures 12, 14). This forces the Eye space *X* axis to be parallel to the Frame's *X* axis. From the information in steps 1, 2 and 3, we can compute the locations of the real lines in Eye space. By intersecting the lines with the *X=0* plane in Eye space, we reduce the geometry to 2D, as shown in Figure 5. We can always get right angles for *y1* and *y2* by using the line *Y=0* as the basis. Then:

$$ß1 = (-1.0) \tan^{-1}(y1/z1)$$
$$ß2 = \tan^{-1}(y2/z2)$$

z1 and *z2* are positive, *y2* is positive and *y1* is negative as drawn. This still works if the user's eye is above or below both of the real lines: the signs of *y1*, *y2*, *ß1* and *ß2* change appropriately. Since the crosshair does not cover the entire frame buffer height (512 pixels), we must scale the result to compute the total FOV ß:

$$ß = (ß1+ß2)(512/(2*125))$$

The parameters measured in steps 1-4 are sufficient to implement registration of the virtual axes. Step 1 tells us where to put the virtual axes. The other parameters tell us how to generate view matrices for the right eye, given reports from the head tracker. The only unusual aspect is the need for an off-center projection.

4.4 Evaluation

These procedures, used with our optoelectronic tracker, generate parameters that work well from many different viewing angles and positions. To demonstrate this, we recorded a video sequence, using only one set of parameters, of a user walking around and looking at the corner of the frame from many different places. If space allows, excerpts from this will be on the CD-ROM. At several places during the run, the HMD was kept still. These viewpoints are identified by the numbered circles in Figure 14, which correspond with the still images in Figures 2 and 15-22. The red and green bars have a 5x5 mm^2 cross-section, while the blue bar is 7x7 mm^2 since it's harder to see at long distances. Note that the corner and edges usually stay within the width of the extruded rectangles at the static viewpoints, which puts the registration within ±4 mm for the red and green bars and ±5 mm for the blue bar.

How do these results compare to the two previous works? Janin [18] takes a very different approach. He directly measures parameters with instruments and runs a camera calibration optimizer that requires the user to identify the location of ~20 object points from several different viewpoints. The best accuracy he achieves is ±12mm. In contrast, step 3 of our procedure is similar to Caudell's [6] registration platform, which computes the Eye->Tracker position and orientation offset by having the user line up two circles and two lines. He does not provide any information on how accurate his static registration is, and his registration procedure lacks equivalents to our steps 2 and 4.

Registration accuracy depends on how successfully the user can complete the registration procedures. Users reported difficulty in keeping their heads still during the boresight and FOV operations, because of the weight of the HMD. To compensate, we use the most recent 60 tracker reports to compute each operation, averaging the

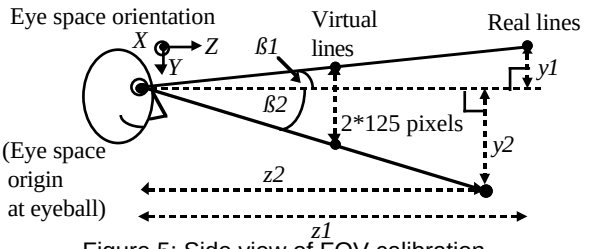


Figure 5: Side view of FOV calibration

results. To measure the remaining variation, we had three users repeat the boresight and FOV steps five times, moving their heads away in between each measurement. The average standard deviations in computed orientation, position, and FOV were 0.32 degrees, 4.8 mm (mostly along the *Z* offset), and 0.1 degrees respectively. While not fatal, this variation does mean users may have to try the procedures more than once to achieve desired results.

5 Dynamic registration

The static registration demonstrated in Figures 15-22 holds only when the user stands still. When the user moves and rotates his head, the virtual objects appear to "swim around" the real objects, because of the system latency. The system requires time to measure the head's location, compute the corresponding images for that location, and display those in the HMD. This delay between measuring the head location and displaying the corresponding images means the images will be incorrect if the user moves his head during this delay. The virtual objects appear to lag behind their real counterparts, causing large dynamic registration errors.

To reduce these dynamic errors, we predict head motion. Instead of using the reported head location to generate the graphic images, we predict where the head will be when the displays get updated. If the prediction is correct, the computed virtual images will match reality at the time they are viewed.

5.1 Previous work

This paper is not the first to do head motion prediction. Two predict head position with head-tracked stereo displays [10][26]. Several HMD systems predict position and/or orientation by extrapolating readings from the head tracker [1][13][23][25][28][34][35]. Two papers [24][37] add angular accelerometers to a head tracker to aid orientation prediction.

How does our system differ from previous work? One difference is the application. We use prediction to aid registration of real and virtual objects in a see-through HMD and evaluate it in that context, something the other systems did not attempt. We also use gyros and accelerometers to aid *both* orientation and position prediction, which no previous system does. Our work also contributes the following:

- Evaluation of how much inertial sensors help
- Measurement and control of prediction distance
- Autocalibration of inertial sensor parameters

5.2 Procedure

We run separate predictors for orientation and position. Each consists of two parts: 1) an estimator that computes our best guess of the current position, velocity, and acceleration, and 2) a predictor that takes those guesses and extrapolates the desired distance into the future. Every time a tracker position and orientation measurement arrives, along with the corresponding inertial measurements, the estimator updates its guesses. Whenever Pxpl5 is ready to compute a new scene, the predictor sends it an output corresponding to the time when that scene will appear.

The estimator used is the Kalman filter [19], which many previous works also use. Space does not permit a detailed description of the filter; please read [22] for that. The Kalman filter is a linear estimator that minimizes the expected mean-square error. For orientation, we use a nonlinear variant called the Extended Kalman Filter (EKF). The filter requires some variables to be estimated, occasional noisy measurements of those variables, and a model of how those variables change with time in the absence of new measurements. It is optimal only if the model is an accurate reflection of reality and if the uncertainty in both the model and the measurements is accurately represented by additive white noise. Even though

these assumptions are usually not met, the Kalman filter is still popular because it tends to perform well even with violated assumptions and its recursive formulation makes it efficient to compute.

Building a Kalman filter is easy. The difficult parts are determining an appropriate model and finding good noise parameters. For the latter task, we collected several runs of tracker and inertial data while the HMD-wearer performed "typical" head motions. Then we ran Powell's method [27] to search for parameters that minimized prediction error at a fixed prediction distance. In practice, this heuristic is able to find a fairly wide range of parameters that meet this goal.

We now outline the orientation estimator and predictor, later describing the translation case by how it differs from orientation.

5.2.1 Orientation

$\mathbf{Q} = [qw\ qx\ qy\ qz]^T, \quad \mathbf{W}=[w0\ w1\ w2]^T$

$\mathbf{X} = \left[qw\ qx\ qy\ qz\ w0\ w1\ w2\ \dot{w0}\ \dot{w1}\ \dot{w2} \right]^T$

where **Q** is a quaternion rotating points and vectors from Tracker space to World space, **W** is omega, the angular rate of rotation in head space, and **X** is the *Nx1* state vector, where *N=10*. **P** is an *NxN* covariance matrix representing the uncertainty in **X**. The initial value of **X** holds the starting quaternion and has zeroes for omega and its derivative. **P** is initially a diagonal matrix with zeroes in all off-diagonal positions, 1 for the first four diagonal terms (quaternion), and 50 for the remaining six diagonal terms (omega and its derivative). The initial covariances are large so the filter will replace the initial **X** with new measurements as they arrive.

X and **P** are maintained at the current time *t*. When each new measurement arrives, say at time *t1*, the filter performs two operations: 1) a time update that advances the state variables to time *t1* based on the model, and 2) a measurement update that blends in the values measured at time *t1*.

1) Time update: A 4th-order Runge-Kutta ODE solver [27] integrates the derivatives of **X** and **P** from time *t* to *t1*. The derivatives are:

$$\dot{\mathbf{P}} = \mathbf{AP} + \mathbf{PA}^T + \mathbf{E}$$
$$\dot{\mathbf{X}} = a(\mathbf{X}, t) \qquad (1)$$

where **E** is an *NxN* matrix representing the noise in the model, and **A** is the *NxN* Jacobian matrix of the nonlinear function *a()* that returns the derivatives of **X** by computing:

$$\dot{\mathbf{Q}} = (0.5)(\mathbf{Q})(\mathbf{W}), \quad \dot{\mathbf{W}} = \left[\dot{w0}\ \dot{w1}\ \dot{w2} \right]^T, \quad \ddot{\mathbf{W}} = 0$$

where for the derivative of **Q**, the multiplications are quaternion multiplications and **W** is written as a quaternion with zero *w* term [7].

2) Measurement update: Measurement **Z** is an *Fx1* matrix, where *F=7*, that holds the measured quaternion $\mathbf{Q_m}$ and omega $\mathbf{W_m}$ reported by our sensors:

$\mathbf{Q_m} = [qw_m\ qx_m\ qy_m\ qz_m]^T, \quad \mathbf{W_m} = [w0_m\ w1_m\ w2_m]^T$

$\mathbf{Z} = [qw_m\ qx_m\ qy_m\ qz_m\ w0_m\ w1_m\ w2_m]^T$

The nonlinear function *h()* generates $\mathbf{Z} = h(\mathbf{X}(t))$ as follows:

$\mathbf{Q_m} = \text{Normalize}(\mathbf{Q}), \quad \mathbf{W_m} = \mathbf{W}$

and the measurement update itself generates a new **X** and **P** given **Z** as follows:

$$\mathbf{K} = \mathbf{PH}^T[\mathbf{HPH}^T + \mathbf{R}]^{-1}$$
$$\mathbf{P} = [\mathbf{I} - \mathbf{KH}]\mathbf{P}$$
$$\mathbf{X} = \mathbf{X} + \mathbf{K}[\mathbf{Z} - h(\mathbf{X})] \qquad (2)$$

where **K** is an *NxF* matrix called the Kalman gain, **H** is the *FxN* Jacobian matrix of the nonlinear function *h()*, and **R** is the *FxF* covariance matrix representing the noise in the measurements. At the end of the measurement update, we explicitly renormalize the quaternion part of **X**. This isn't standard, but without it the quaternion terms quickly become unnormalized.

Noise matrices **E** and **R** are determined during the offline optimization. Both are diagonal matrices with zeroes in all off-diagonal terms. The first six diagonal terms of **E** are set to 0.004452: a tiny amount of noise added to the measured quaternion and omega to help the stability of the EKF. The remaining three diagonal terms are set to 351.0. For **R**, the first four diagonal terms are 0.0001, representing $\mathbf{Q_m}$ noise, and the remaining three diagonal terms are 0.005921, representing $\mathbf{W_m}$ noise.

Predictor: When the scene generator is ready to draw a new image, the predictor bases its extrapolation on the estimated values in **X**. The predictor is the closed-form solution of integrating the quaternion and omega under the assumption that the derivative of omega is constant over the integration interval *t0* to *t*. We define a 4x4 matrix $\mathbf{M}(t)$ as satisfying:

$$\dot{\mathbf{Q}} = (0.5)(\mathbf{Q})(\mathbf{W}) = (\mathbf{M}(t))(\mathbf{Q})$$

where $\mathbf{M}(t)$ essentially rewrites the quaternion multiplication as a matrix multiplication. The solution of this is:

$$\mathbf{Q} = \left[(\mathbf{I})cos(d) + (\mathbf{M})(sin(d) / d) \right] (\mathbf{Q_{t0}})$$

where $\mathbf{Q_{t0}}$ is the original quaternion at time *t0* and:

$$d = \sqrt{a^2 + b^2 + c^2}$$
$$a = (0.5)\left[(t\text{-}t0)w0 + (0.5)(t - t0)^2(\dot{w0}) \right]$$
$$b = (0.5)\left[(t\text{-}t0)w1 + (0.5)(t - t0)^2(\dot{w1}) \right]$$
$$c = (0.5)\left[(t\text{-}t0)w2 + (0.5)(t - t0)^2(\dot{w2}) \right]$$

5.2.2 Position

The position estimation uses three separate linear Kalman filters, one each for X, Y and Z. Since they are identical in form, we look at the Y case only. This section lists the differences from the orientation case:

$\mathbf{X} = \left[\ y \ \ \dot{y} \ \ \ddot{y} \ \right]^T$, where $N = 3$

$$\text{Initial } \mathbf{X} = [y(0)\ \ 0\ \ 0]^T, \quad \mathbf{P} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 500 & 0 \\ 0 & 0 & 500 \end{bmatrix}$$

Time update: Replace (1) with:

$$\dot{\mathbf{X}} = \mathbf{AX}, \text{ where } \mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Measurement update: Replace (2) with:

$$\mathbf{X} = \mathbf{X} + \mathbf{K}[\mathbf{Z} - \mathbf{HX}] \text{ where } \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and Z is a 2x1 matrix (*F=2*) containing the reported Y position and the linear Y acceleration, in World space. Recovering linear acceleration from the accelerometers is complicated because they detect both linear and angular accelerations, plus gravity. Space does not permit an explanation here; please see Appendix A in the CD-ROM version of this paper for details.

$$\mathbf{R} = \begin{bmatrix} .01 & 0 \\ 0 & .05 \end{bmatrix}, \quad \mathbf{E} = \begin{bmatrix} .007 & 0 & 0 \\ 0 & .007 & 0 \\ 0 & 0 & 2000000 \end{bmatrix}$$

Predictor:

$$y(t) = 0.5 * \ddot{y}(t0) * [t - t0]^2 + \dot{y}(t0) * [t - t0] + y(t0)$$

5.2.3 Autocalibration

The inertial outputs must be rotated into the Tracker coordinate system, because the inertial sensor packs are tilted with respect to Tracker space. To perform these rotations, we must know the orientation of each pack on the HMD. While it is possible to mechanically build a mount that holds each pack at a specified orientation, it's easier to mount the packs at some rigid, but unknown, orientations, then measure them. Also, we would like to measure other sensor parameters, like the biases and scales. Autocalibration refers to mathematical methods that determine such constant parameters by applying geometrical constraints to collected data. One demonstration of this measured the locations of the beacons in the panels of our optoelectronic tracker [15]. Two such methods that we use with our inertial sensors are described in Appendix C on the CD-ROM. They are good at determining the orientation and biases, but not the scales.

5.3 Evaluation

From the user's perspective, prediction changes dynamic registration from "swimming around the real object" to "staying close."

	Walkaround			Rotation			Swing		
	Ang	Pos	Screen	Ang	Pos	Screen	Ang	Pos	Screen
No prediction	**1.3**	**14.3**	**9.3**	**2.2**	**6.6**	**33.6**	**2.5**	**17.8**	**37.1**
	4.3	**38.0**	**62.0**	**5.3**	**17.6**	**92.1**	**6.5**	**46.0**	**118.6**
Prediction without Inertial	**0.2**	**2.5**	**4.5**	**0.6**	**3.3**	**13.6**	**0.6**	**5.2**	**16.2**
	0.8	**9.0**	**26.7**	**1.6**	**11.7**	**51.0**	**1.8**	**17.1**	**62.8**
Prediction with Inertial	**0.1**	**1.1**	**2.7**	**0.18**	**1.6**	**5.2**	**0.2**	**2.7**	**7.2**
	0.4	**6.1**	**15.1**	**0.57**	**9.8**	**36.1**	**0.7**	**17.8**	**30.1**

Average error Peak error

Angular error in degrees, Position error in mm, Screen error in pixels
Prediction distance set at 60 ms for all runs

Figure 6: Performance table of predictors on three motion datasets

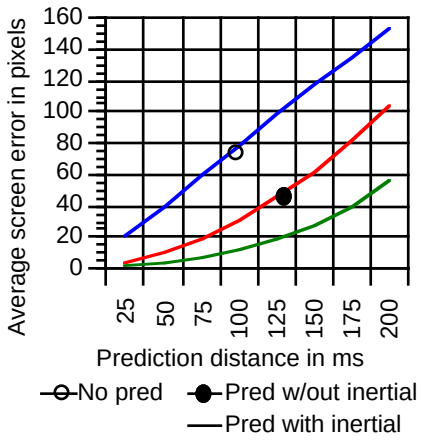


Figure 7: Average error vs. prediction distance

Without prediction, registration errors are large enough to strain the illusion that the real and virtual coexist. With prediction, the real and virtual objects stay close enough that the user perceives them to be together. Although the prediction is not perfect, it demonstrably improves the dynamic registration.

The predictor was run on three recorded motion datasets that are considered representative of this registration task. During each motion sequence, the user keeps the corner of the frame visible in his field-of-view. In the Walkaround dataset, the user walks slowly around the corner of the frame. The Rotation dataset has the user yawing, pitching, and circling his head while standing in place. The Swing dataset combines fast translation and rotation motion.

We compared our inertial-based predictor on these three datasets against doing no prediction and against a predictor that does not use inertial sensors. Directly comparing our predictor against previous work is difficult because our system is unique. Instead, we wrote a Kalman-filter-based predictor that is representative of many previous works that do not use inertial sensors. We ran that on the datasets, keeping all other variables constant. Three error metrics evaluate the accuracy of the predicted outputs vs. the actual tracker measurements. Angular error is computed in degrees as follows:

$$Q_{diff} = (Q_{actual})(Q_{predicted})^{-1}$$
$$angle\_err = (2)\ acos(Q_{diff}[qw])$$

Position error is the distance between the predicted and actual translations. Screen error combines orientation and translation errors by measuring the difference, in pixels, between the 2D projection of the real frame and the 2D point where the three virtual axes intersect on a hypothetical 512x512 screen. That is, it measures error in terms of what the user sees inside an HMD. The peak and average errors are summarized in Figure 6. On average, our inertial-based predictor is 5-10 times more accurate than doing no prediction and 2-3 times more accurate than prediction without inertial sensors.

Appendix B on the CD-ROM provides additional materials that demonstrate the results of prediction. Depending upon the space allocation on the CD-ROM, these may include error graphs, the motion datasets, and a short QuickTime video.

Figure 7 shows how the average screen-based errors for the Rotation run change as the prediction distance is varied from 25-200 ms. Again, inertial sensors clearly help. But what this graph does not show is that at prediction distances of ~100 ms or more, the jitter in the predicted outputs often reaches objectionable levels. In practice, the only solution is to keep system latency at tolerable levels, below ~80 ms. Thus, prediction cannot compensate for arbitrary amounts of latency; to be effective, it must be combined with efforts that minimize system lag. See Appendix D on the CD-ROM.

Because one cannot accurately predict without knowing how *far* to predict, our system requires accurate clocks and control over latency. Our tracker and graphics engine run asynchronously, requiring an estimation of the prediction distance at each iteration. Miscalculating the prediction distance by as little as 10 ms leads to visible registration errors. The clocks in the tracker boards and Pxpl5 are good to under a millisecond. Synchronization occurs through a message that takes less than 1 ms to make a round trip of all the processors. Since clocks that differ by one second every six hours change by 8.3 ms every three minutes, skew rate compensation and occasional resynchronizations are performed. Controlling latency means removing all unpredictable delays, so we have direct communication paths not shared with other users, and we run a low-overhead operating system called VxWorks. Processes running on UNIX can suffer unbounded amounts of delay. Pauses of 60-200 ms are common occurrences on our Sun4 host. Therefore, we avoid UNIX for all time-critical tasks, directly injecting data from the tracker into Pxpl5 without going through the Sun4 host. While these steps reduce flexibility and make it harder to debug the system, they are needed to insure accurate prediction. Appendix E on the CD-ROM demonstrates the accuracy of our prediction distance estimation.

6 Additional lessons

Augmented Reality demands higher accuracy from head trackers than Virtual Environment applications do [3]. The main difficulty in duplicating our demonstration of static registration is in acquiring a head tracker that one can trust at long ranges. Wooden crates are easy to build, and the calibration techniques are straightforward and applicable to any tracking system or see-through HMD. But many commercially available trackers commonly used for Virtual Environments do not provide sufficient performance. For example, in our laboratory the widely used Polhemus magnetic trackers give distorted outputs at long distances because of the metal in the environment. A coworker experimenting with our optoelectronic tracker discovered a distortion that, when the tracker "hat" yaws 360 degrees about its origin, causes the reported positions to trace an ellipse about an inch wide. Since this distortion seems to be systematic, we were able to compensate for it. The fact that this distortion was undetectable in the Virtual Environment applications we run but was quite noticeable in Augmented Reality only serves to underscore the latter's need for accurate, long-range trackers.

The need to accurately measure time and avoid unpredictable sources of latency has serious ramifications on the design of effective Augmented Reality systems. Tracker measurements must be timestamped, a feature not provided by most commercial trackers. Almost all interactive graphics applications in our laboratory use UNIX because of its convenient programming environment. We build applications to the desired complexity, then we extract as much speed as possible. Flight simulators and some other researchers [20] take the opposite approach: set a minimal standard for performance, then see how much complexity can be supported. Since accurate prediction requires guaranteed performance, future Augmented Reality applications may need to take this latter approach.

7 Future work

Much work remains to further improve static registration. We only match one virtual object with one real object, where the real object is the calibration rig itself. Because our optoelectronic tracker loses accuracy when the sensors are not aimed at the ceiling beacons, we cannot move the HMD far away from the wooden frame, nor can we tilt the HMD far from horizontal. Our system is monocular; while our static registration procedure could be applied to

both eyes, stereo displays involve additional issues like convergence that we have not addressed. We have not compensated for the optical distortion in the see-through HMD. Because our HMD has narrow field-of-view displays, this distortion is small and detectable only near the edges of the displays. We can eliminate this error by mapping the distortion, then predistorting the graphic images before displaying them [31].

More sophisticated prediction methods might further reduce dynamic registration errors. Adaptive methods that adjust to varying head motion deserve more exploration. Using a nonadaptive predictor is like trying to race a car at constant speed; slowing down on the curves and speeding up on the straight-aways will improve your time. Analyzing head motion for recognizable patterns or high-level characteristics may aid prediction. Other researchers have begun doing this [32], and Fitts' Law has been shown to apply to head motion [2][17].

Our work has not dealt with video see-through HMDs, where a video camera provides a view of the real world and the graphics are composited with the digitized images of the real world. With this class of see-through HMD, standard camera calibration techniques could determine the viewing parameters. And since the computer has digitized images of what the user sees, it may be possible to use image processing or computer vision techniques to detect features in these images and use them to aid registration. The disadvantage of this technology is that the video camera and digitization hardware impose inherent delays on the user's view of the real world. Therefore, even if the graphics are perfectly registered with the digitized images, a problem remains: the latency in the video stream will cause the user to perceive *both* the real and virtual objects to be delayed in time. While this may not be bothersome for small delays, it is a major problem in the related area of telepresence systems and may not be easy to overcome.

Acknowledgements

This system would not exist without the contributions of many people. We thank Mike Bajura, Suresh Balu, Brad Bennett, Devesh Bhatnagar, Frank Biocca, Fred Brooks, Steve Brumback, Vern Chi, David Ellsworth, Mark Finch, Henry Fuchs, Jack Goldfeather, Stefan Gottschalk, David Harrison, Rich Holloway, John Hughes, Kurtis Keller, Jack Kite, Jonathan Marshall, Carl Mueller, Ulrich Neumann, Mark Olano, Jannick Rolland, Andrei State, Brennan Stephens, Russell Taylor, John Thomas, and Mark Ward for their advice and/or help with this project.

We thank the anonymous reviewers for their helpful comments and constructive criticisms.

Funding was provided by ONR contract N00014-86-K-0680, ARPA contract DABT63-93-C-C048, the NSF/ARPA Science and Technology Center for Computer Graphics and Visualization (NSF prime contract 8920219), and a Pogue Fellowship.

References

[1] Albrecht, R. E. An adaptive digital filter to predict pilot head look direction for helmet-mounted displays. MS Thesis, University of Dayton, Ohio (July 1989).

[2] Andres, Robert O., and Kenny J. Hartung. Prediction of Head Movement Time Using Fitts' Law. *Human Factors 31*, 6 (1989), 703-713.

[3] Azuma, Ronald. Tracking Requirements for Augmented Reality. *CACM 36*, 7 (July 1993), 50-51.

[4] Bajura, Michael, Henry Fuchs, and Ryutarou Ohbuchi. Merging Virtual Objects with the Real World: Seeing Ultrasound Imagery within the Patient. *Proceedings of SIGGRAPH '92* (Chicago, IL, July 26-31, 1992), 203-210.

[5] Beer, Ferdinand P. and E. Russell Johnston, Jr. *Vector Mechanics for Engineers: Statics and Dynamics (5th ed).* McGraw-Hill, 1988.

[6] Caudell, Thomas P. and David W. Mizell. Augmented Reality: An Application of Heads-Up Display Technology to Manual Manufacturing Processes. *Proceedings of Hawaii International Conference on System Sciences* (Jan. 1992), 659-669.

[7] Chou, Jack C.K. Quaternion Kinematic and Dynamic Differential Equations. *IEEE Trans Robotics and Automation 8,* 1 (Feb. 1992), 53-64.

[8] Cohen, Jonathan, and Mark Olano. Low Latency Rendering on Pixel-Planes 5. UNC Chapel Hill Dept. of Computer Science technical report TR94-028 (1994).

[9] Cruz-Neira, Carolina, Daniel Sandin, and Thomas DeFanti. Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. *Proceedings of SIGGRAPH '93* (Anaheim, CA, Aug. 1-6, 1993), 135-142.

[10] Deering, Michael. High Resolution Virtual Reality. *Proceedings of SIGGRAPH '92* (Chicago, IL, July 26-31, 1992), 195-202.

[11] Feiner, Steven, Blair MacIntyre, and Dorée Seligmann. Knowledge-Based Augmented Reality. *CACM 36*, 7 (July 1993), 53-62.

[12] Foley, James D., Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice, 2nd edition.* Addison-Wesley (1990), 238-239.

[13] Friedmann, Martin, Thad Starner, and Alex Pentland. Device Synchronization Using an Optimal Filter. *Proceedings of 1992 Symposium on Interactive 3D Graphics* (Cambridge, MA, 29 March - 1 April 1992), 57-62.

[14] Fuchs, Henry, John Poulton, John Eyles, et al. Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories. *Proceedings of SIGGRAPH '89* (Boston, MA, July 31-Aug 4, 1989), 79-88.

[15] Gottschalk, Stefan and John F. Hughes. Autocalibration for Virtual Environments Tracking Hardware. *Proceedings of SIGGRAPH '93* (Anaheim, CA, Aug 1-6, 1993), 65-72.

[16] Holmgren, Douglas E. Design and Construction of a 30-Degree See-Through Head-Mounted Display. UNC Chapel Hill Dept. of Computer Science technical report TR92-030 (July 1992).

[17] Jagacinski, Richard J., and Donald L. Monk. Fitts' Law in Two Dimensions with Hand and Head Movements. *Journal of Motor Behavior 17*, 1 (1985), 77-95.

[18] Janin, Adam L., David W. Mizell, and Thomas P. Caudell. Calibration of Head-Mounted Displays for Augmented Reality Applications. *Proceedings of IEEE VRAIS '93* (Seattle, WA, Sept. 18-22, 1993), 246-255.

[19] Kalman, R. E., and R. S. Bucy. New Results in Linear Filtering and Prediction Theory. *Trans ASME, J. Basic Eng., Series 83D* (Mar. 1961), 95-108.

[20] Krueger, Myron W. Simulation versus artificial reality. *Proceedings of IMAGE VI Conference* (Scottsdale, AZ, 14-17 July 1992), 147-155.

[21] Lenz, Reimar K. and Roger Y. Tsai. Techniques for Calibration of the Scale Factor and Image Center for High Accuracy 3-D Machine Vision Metrology. *IEEE Transactions on Pattern Analysis and Machine Intelligence 10*, 5 (Sept. 1988), 713-720.

[22] Lewis, Frank L. *Optimal Estimation.* John Wiley & Sons, 1986.

[23] Liang, Jiandong, Chris Shaw, and Mark Green. On Temporal-Spatial Realism in the Virtual Reality Environment. *Proceedings of the 4th annual ACM Symposium on User Interface Software & Technology* (Hilton Head, SC, Nov 11-13, 1991), 19-25.

[24] List, Uwe H. Nonlinear Prediction of Head Movements for Helmet-Mounted Displays. Technical report AFHRL-TP-83-45 [AD-A136590], Williams AFB, AZ: Operations Training Division (1984).

[25] Murray, P.M. and B. Barber. Visual Display Research Tool. *AGARD Conference Proceedings No. 408 Flight Simulation* (Cambridge, UK, 30 Sept. - 3 Oct. 1985).

[26] Paley, W. Bradford. Head-Tracking Stereo Display: Experiments and Applications. *SPIE Vol. 1669 Stereoscopic Displays and Applications III* (San Jose, CA, Feb. 12-13, 1992), 84-89.

[27] Press, William H., et al. *Numerical Recipes in C.* Cambridge University Press, 1988.

[28] Rebo, Robert. A Helmet-Mounted Virtual Environment Display System. MS Thesis, Air Force Institute of Technology (Dec 1988).

[29] Robinett, Warren. Synthetic Experience: A Proposed Taxonomy. *Presence 1*, 2 (Spring 1992), 229-247.

[30] Robinett, Warren, and Richard Holloway. Implementation of Flying, Scaling and Grabbing in Virtual Worlds. *Proceedings of 1992 Symposium on Interactive 3D Graphics* (Cambridge, MA, 29 March - 1 April 1992), 189-192.

[31] Robinett, Warren, and Jannick P. Rolland. A Computational Model for the Stereoscopic Optics of a Head-Mounted Display. *Presence 1*, 1 (Winter 1992), 45-62.

[32] Shaw, Chris and Jiandong Liang. An Experiment to Characterize Head Motion in VR and RR Using MR. *Proceedings of 1992 Western Computer Graphics Symposium* (Banff, Alberta, Canada, April 6-8, 1992), 99-101.

[33] Sims, Dave. New Realities in Aircraft Design and Manufacture. *IEEE CG&A 14*, 2 (March 1994), 91.

[34] Smith Jr., B. R. Digital head tracking and position prediction for helmet mounted visual display systems. *Proceedings of AIAA 22nd Aerospace Sciences Meeting* (Reno, NV, Jan. 9-12, 1984).

[35] So, Richard H. Y. and Michael J. Griffin. Compensating Lags in Head-Coupled Displays Using Head Position Prediction and Image Deflection. *Journal of Aircraft 29*, 6 (Nov-Dec 1992), 1064-1068.

[36] Ward, Mark, Ronald Azuma, Robert Bennett, Stefan Gottschalk, and Henry Fuchs. A Demonstrated Optical Tracker With Scalable Work Area for Head-Mounted Display Systems. *Proceedings of 1992 Symposium on Interactive 3D Graphics* (Cambridge, MA, 29 March - 1 April 1992), 43-52.

[37] Welch, Brian L., Ron Kruk, et al. Flight Simulator Wide Field-of-View Helmet-Mounted Infinity Display System. Technical report AFHRL-TR-85-59, Williams AFB, AZ, Operations Training Division (May 1986).

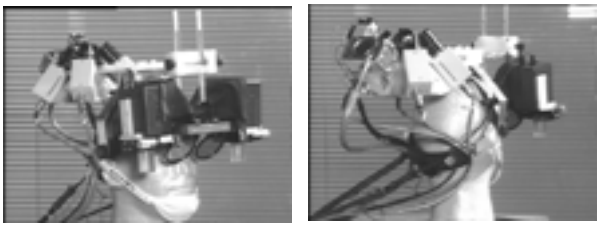


Figure 8: Front and back views of optical see-through HMD

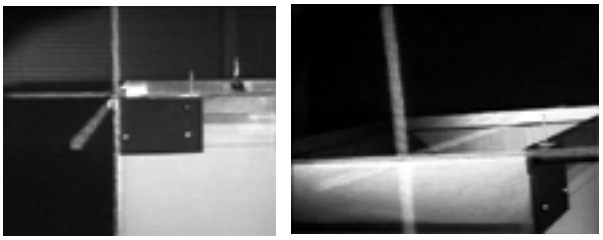


Figure 9: Naive approach yields non-robust registration

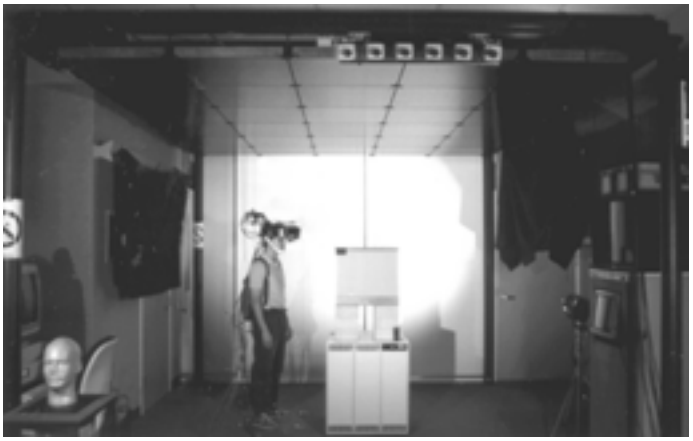


Figure 10: Picture of overall system

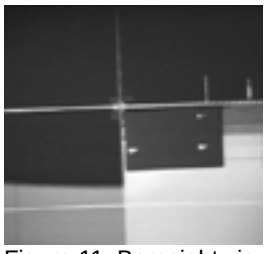


Figure 11: Boresight view

Figure 12: FOV calib

Figure 13: Measuring frame

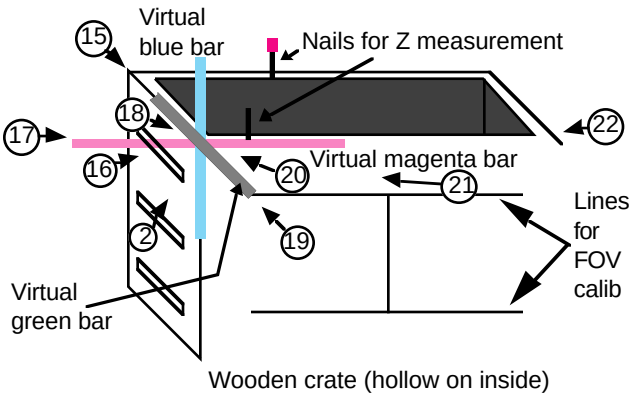


Figure 14: Wooden frame and static registration viewpoints

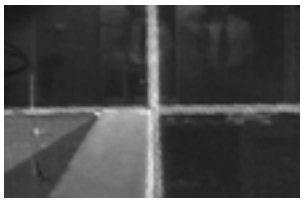


Figure 15

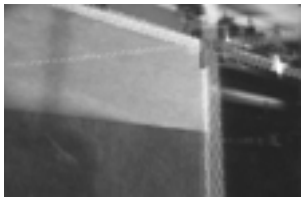


Figure 16

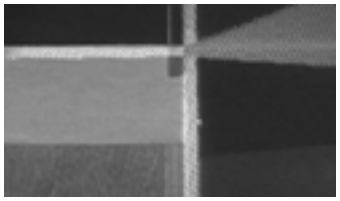


Figure 17

Figure 18

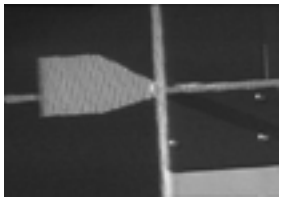


Figure 19

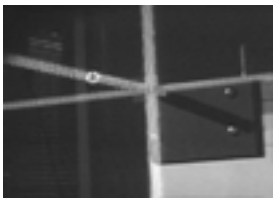


Figure 20

Figure 21

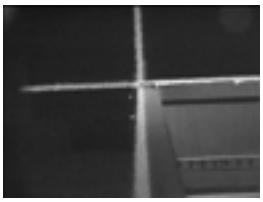


Figure 22

Figures 15-22: Static registration of virtual axes with real frame as seen inside the HMD from viewpoints specified in Figure 14

Introduction to Appendices

These appendices, which are included only in the CD-ROM version of the paper, contain supplementary materials that could not be included with the proceedings version, due to the eight page limit. These appendices are:

- Appendix A: Extracting World space acceleration
- Appendix B: Evaluating dynamic errors
- Appendix C: Autocalibration
- Appendix D: Limits of prediction
- Appendix E: Estimating total prediction distance
- Appendix F: Miscellaneous comments

Please also see the README files on the CD-ROM for a guide to the other available supplementary materials.

A Extracting World space acceleration

Section 5.2.2 describes a filter and estimator that performs position prediction. It requires measurements of the linear acceleration of the tracker, in World space. This information comes from the accelerometers, but calculating it is not as trivial as recovering angular velocity from the rate gyroscopes. Why? The rate gyroscopes only detect angular velocity and are basically unaffected by other types of motion. Therefore, recovering the angular velocity from the gyroscopes is simply a matter of biasing, scaling, and rotating the gyro outputs into Tracker space. In contrast, our linear accelerometers respond to linear acceleration, angular acceleration, and gravity. Their output is a combination of all these inputs. To extract only the linear acceleration from the accelerometers, we must calculate and remove the other two components.

It may help to first describe how the accelerometers work. One can think of an accelerometer as a cantilever beam that extends out over empty space, much like a tiny diving board. Without gravity, this beam extends out straight horizontally. But with gravity, the beam sags down (Figure 23). As the user moves the accelerometer up and down, the beam moves up and down with respect to the rest of the device, due to inertia. The accelerometer electrically measures the height of the beam with respect to the rest of the device, and that voltage is the what the sensor outputs.

We note two important properties from this explanation. First, each accelerometer detects acceleration only along one direction in space. To a first approximation, each accelerometer does not detect any acceleration perpendicular to its sensitive axis. That is why we have three accelerometers, mounted in a mutually orthogonal configuration. The fact that we have three, physically separated, 1-D sensors instead of a single 3-D sensor makes the math slightly more complicated. Second, gravity sets the "bias point" of each accelerometer. That is, the value that the accelerometer reports when it is standing still depends upon its orientation with respect to the gravity vector.

With this background, we now describe how to recover linear acceleration from the three accelerometers. Since this recovery depends on knowledge of angular motion, we must first run all the steps in Section 5.2.1, which provides estimates of angular orientation, velocity, and acceleration. We also need the orientation of the tracker, provided by the head tracker. Figure 24 shows the configuration of the accelerometers in Tracker space.

Step 1 is to compensate for gravity by determining the bias points. The orientation of the accelerometers with respect to Tracker space is known. We can rotate that into World space by using the orientation reported by the head tracker. The gravity vector is assumed to point straight down in World space. Since we know the orientation of the accelerometers in World space, we can calculate what they would output if the accelerometers were kept still at that orientation. These bias values are subtracted from the actual accelerometer outputs, removing the contribution due to gravity. This leaves us with linear and angular acceleration, reported in Accelerometer space.

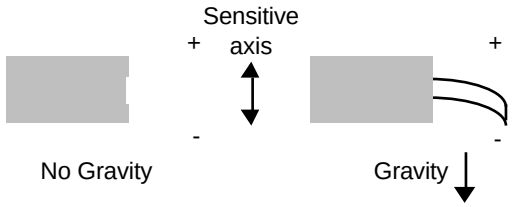


Figure 23: Accelerometers are tiny cantilever beams

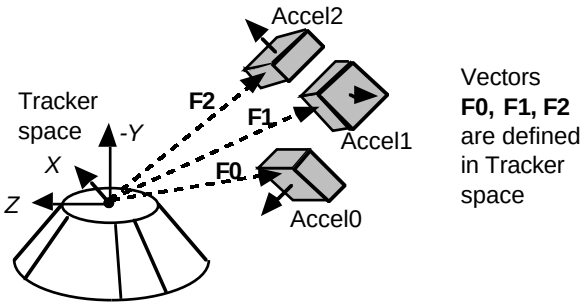


Figure 24: Locations of accelerometers in Tracker space

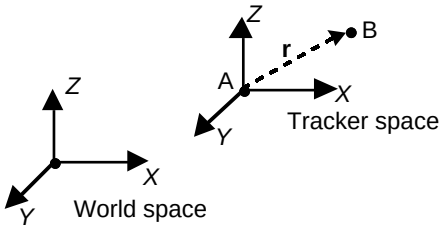


Figure 25: Definitions for rigid body kinematics formula

In Step 2, we change this acceleration into World space. The three accelerometers are mutually orthogonal, so combining their outputs forms a 3-vector in Accelerometer space. We scale the values so they report acceleration as m/s^2, rather than A/D counts. Then we rotate that into World space.

All that's left for Step 3 is to remove the angular acceleration component, leaving the desired linear acceleration. To do that, we use the following formula from the kinematics of rigid bodies [5] (Figure 25):

$$\mathbf{A}_A = \mathbf{A}_B - \dot{\mathbf{W}} \times \mathbf{r} - \mathbf{W} \times (\mathbf{W} \times \mathbf{r})$$

where $\mathbf{A}_A$ is the total acceleration at point A, the origin of Tracker space, $\mathbf{A}_B$ is the total acceleration at point B, the location of one of the accelerometers, $\mathbf{W}$ is the angular velocity of the tracker, the derivative of $\mathbf{W}$ is the angular acceleration, and $\mathbf{r}$ is a vector from point A to point B. Note, however, that this formula assumes that Tracker space shares the same orientation as World space, which is usually not the case. Thus, to use this formula we must have everything in World space. A second problem comes from using three separate 1-D accelerometers. We need three different $\mathbf{r}$ vectors, one for each accelerometer. Each vector results in a different angular acceleration, and each accelerometer detects only the component of angular acceleration that lies along its sensitive axis.

Therefore, we must modify the formula somewhat to deal with these problems. The final procedure that Step 3 uses to remove the angular acceleration component is:

1) Set $\mathbf{A}_B$ to the linear and angular acceleration 3-vector reported by the accelerometers, in World space, which was computed in Step 2.
2) Take **W** and its derivative from the results in Section 5.2.1 and rotate them into World space.
3) For each accelerometer, compute the following: Take vector **F0**, **F1**, or **F2** and rotate it into World space. Call this vector **r**. Compute

$$\mathbf{V} = \dot{\mathbf{W}} \times \mathbf{r} + \mathbf{W} \times (\mathbf{W} \times \mathbf{r})$$

Then take a unit vector along the sensitive axis of the accelerometer, in Accelerometer space, and rotate that into World space. Call this vector **S**. The angular acceleration detected by this accelerometer is the dot product of **V** and **S**. Call vectors **V0** and **S0**, **V1** and **S1**, and **V2** and **S2** as those from accelerometers 0, 1, and 2, respectively.
4) The desired World space linear acceleration at the origin of Tracker space, $\mathbf{A}_A$, is then computed by:

$$\mathbf{A}_A = \mathbf{A}_B - \mathbf{V}0 \cdot \mathbf{S}0 - \mathbf{V}1 \cdot \mathbf{S}1 - \mathbf{V}2 \cdot \mathbf{S}2$$

We can do this because the three accelerometers are mutually orthogonal.

We tested this math on simulated motion datasets before trying it on real data. The simulated datasets were generated by writing explicit equations for the position and orientation of the tracker and its three accelerometers. Differentiating those equations yielded the velocity and acceleration values. By not using the equations of motion in Step 3 to generate the simulated data, we avoid incestual problems that might result in invalid test data.

The need for the derivative of **W** is a potential weakness of our procedure. We do not have a sensor that directly detects angular acceleration, so we must estimate that from the gyro data. Simulation runs indicate that the estimated derivative of **W** lags behind the true derivative by about 20 ms. The errors resulting from this are small, except when the user rotates his head very quickly. One could avoid this problem by adding angular accelerometers.

B Evaluating dynamic errors

This appendix provides a more detailed look at the data summarized in Section 5.3. Nine graphs compare using no prediction, prediction without inertial sensors, and prediction with inertial sensors on the Swing motion dataset.

We should point out that the original positions and quaternions in all three motion datasets were filtered with a noncausal 10 Hz lowpass filter. This lowpass filter does not introduce any phase delay. The filtering is intended to reduce noise in the original positions and quaternions, since we use those as the true values in the angle, position, and screen-based error metrics. Without this filtering, noise in the original signals would show up as additional error. Note that we do *not* lowpass filter the inertial measurements.

The first set of three graphs (Figures 26-28) compares the angle, position, and screen-based errors produced by running no prediction, prediction without inertial sensors, and prediction with inertial sensors on the Swing motion dataset. One-third of the numbers from Figure 6 came from these graphs; they provide some sense of how the numbers in Figure 6 correspond to the actual error graphs.

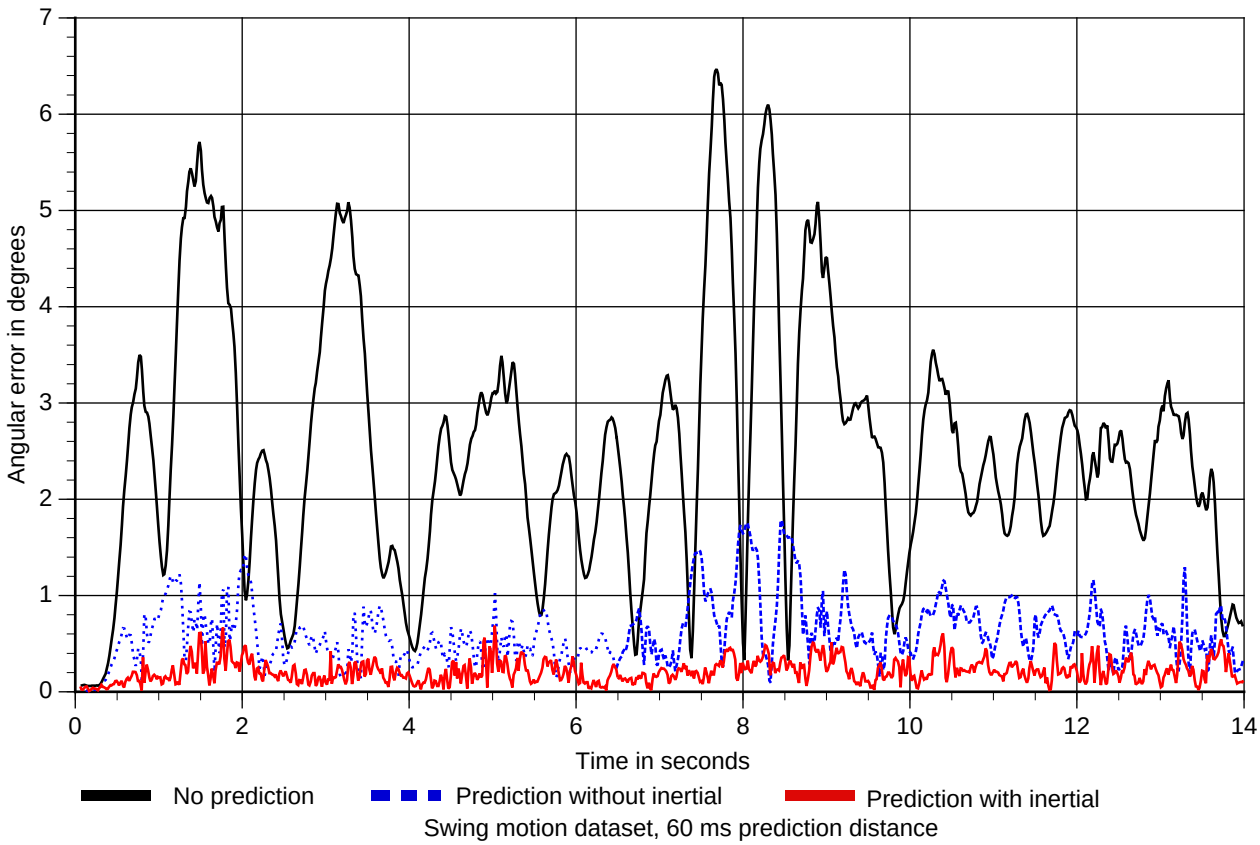


Figure 26: Angular errors generated by no prediction, prediction without inertial, and prediction with inertial

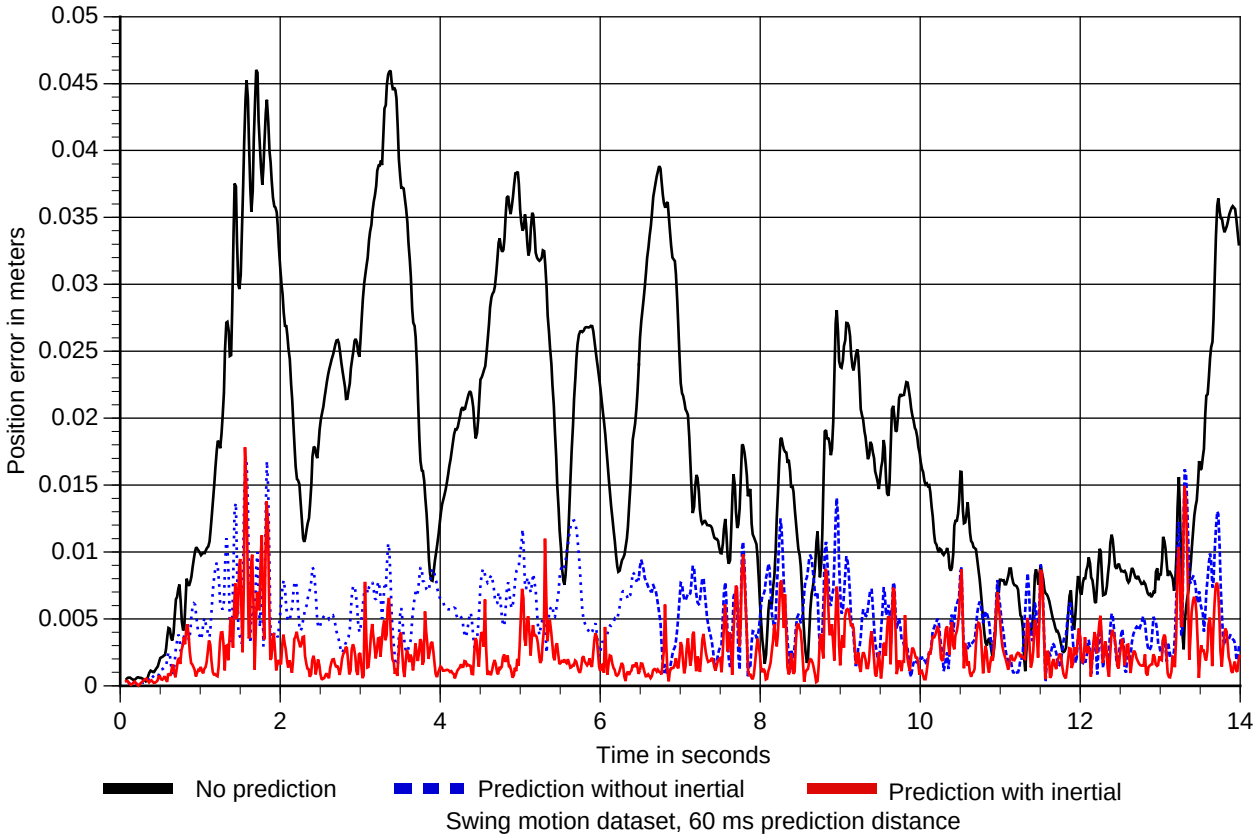


Figure 27: Position errors generated by no prediction, prediction without inertial, and prediction with inertial

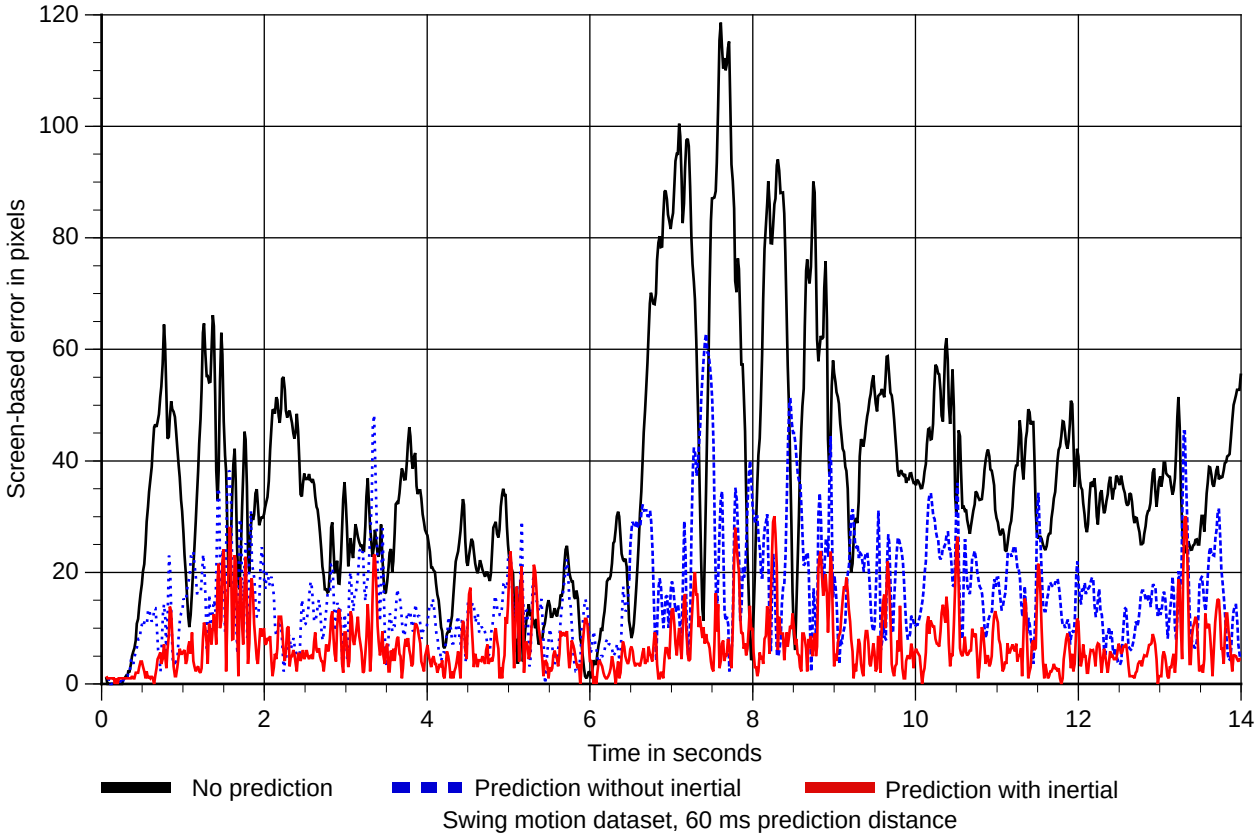


Figure 28: Screen-based errors generated by no prediction, prediction without inertial, and prediction with inertial

The second set of three graphs (Figures 29-31) gives a detailed view of orientation errors. We select a small section of the *qw* orientation curve from the Swing dataset and overlay the predicted *qw* curves that result from no prediction, prediction without inertial sensors, and prediction with inertial.

The third set of three graphs (Figures 32-34) does the same for one of the translation curves: the *Z* position. We select a small section and overlay three predicted curves on that interval.

It is difficult to graph the actual and predicted motion curves because of the difference in scale between the errors and the original motions. Overall views cannot show the errors, while detailed views fail to give a sense of the overall motion. Therefore, we are making the datasets themselves available on the CD-ROM (assuming space is available), so that readers may use their favorite graphing programs to examine any part of the curves they wish. Please see the README files on the CD-ROM for instructions.

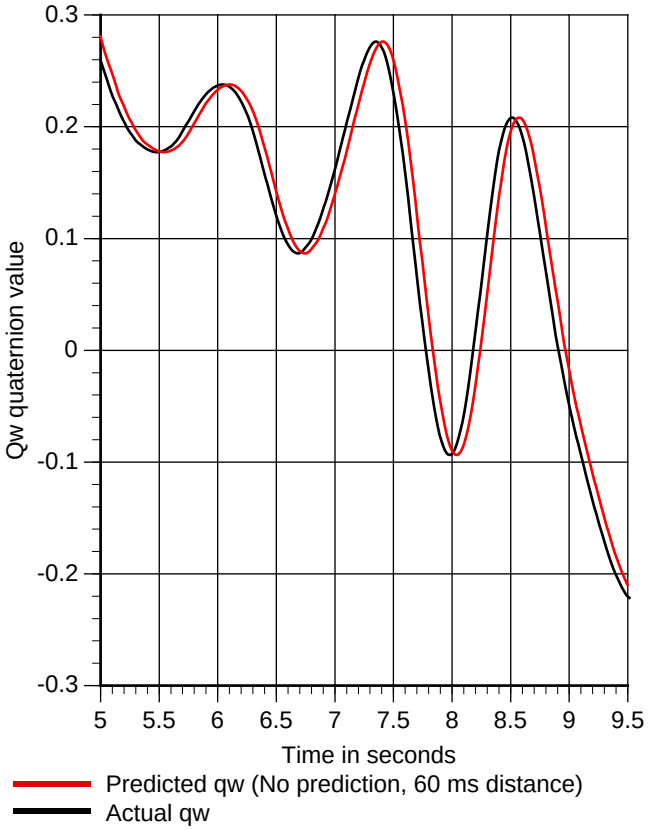


Figure 29: Qw curve with no prediction

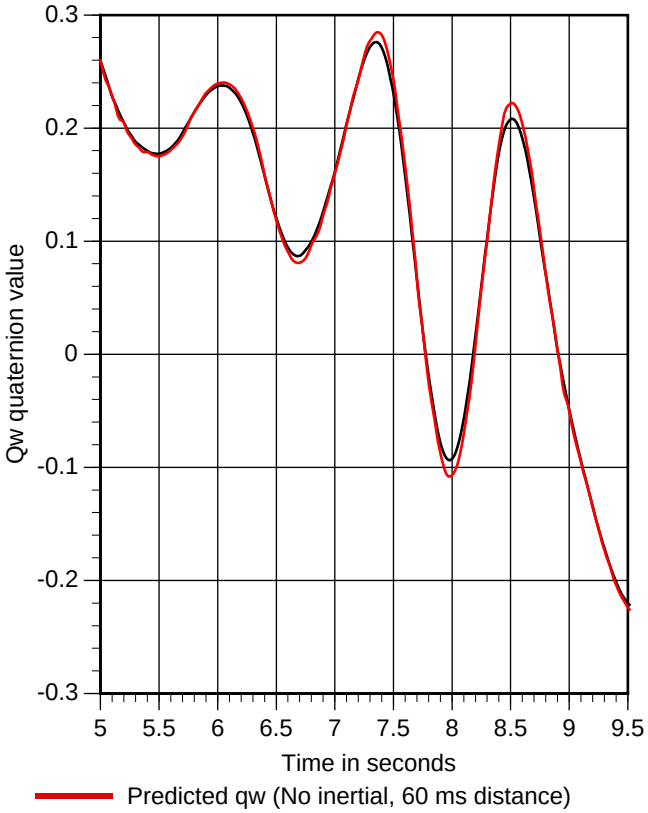


Figure 30: Qw curve with non-inertial-based predictor

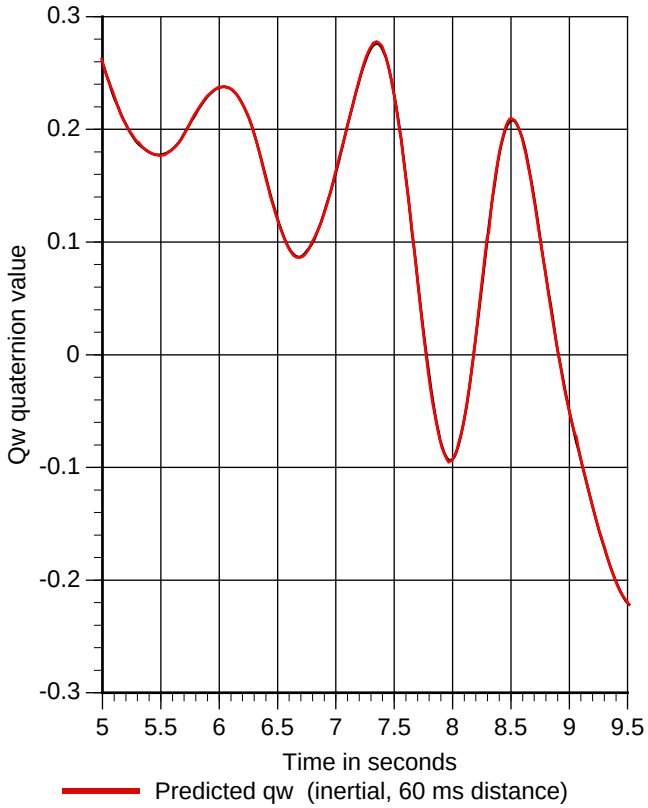


Figure 31: Qw curve with inertial-based predictor

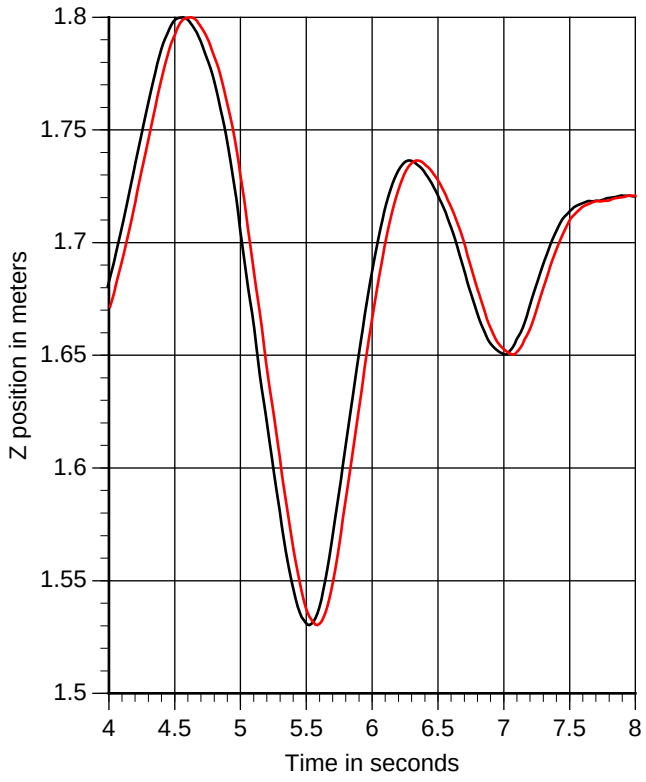


Figure 32: Z curve with no prediction

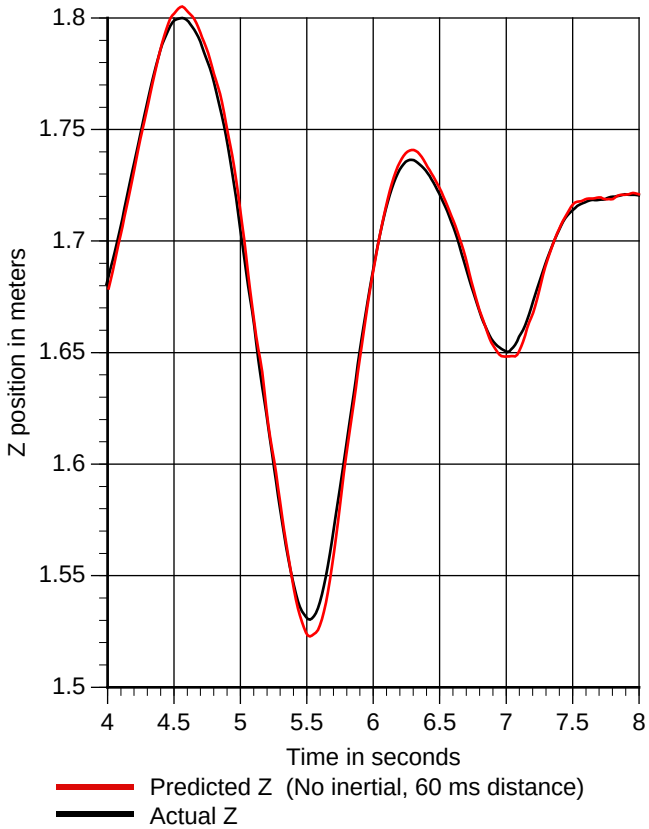


Figure 33: Z curve with non-inertial-based predictor

C Autocalibration

Section 5.2.3 introduced the concept of autocalibration, where optimization routines applied to collected data measure system parameters. In this appendix, we describe and evaluate two approaches for measuring the orientation, biases and scales of our inertial sensors. In both methods, a user wears the HMD and moves around naturally while the inertial and tracker readings are recorded. This dataset is then processed offline to determine the parameters.

Autocalibration requires finding effective bases of comparison. If we can compute the same value in more than one way, by using geometrical relationships or other manipulations, we have a basis for comparing two or more estimates. Since ideally these estimates should be equal, an optimizer varies the system parameters until the best match is achieved between the multiple estimates.

In our case, we must compare the position and orientation readings provided by the head tracker against the velocities and accelerations returned by the gyros and accelerometers. Clearly, we have two basic approaches available. We can either differentiate the tracker positions and orientations and compare those against the velocities and accelerations, or we can integrate the velocities and accelerations and compare those against the tracker positions and orientations.

The first method uses the differentiation approach. It turns out that this approach does not work with the accelerometers. Numerical differentiation is an operation that inherently magnifies noise. Differentiating position twice to get acceleration generates outputs that are too noisy to be useful. However, we have been able to use Kalman filters to differentiate once without adding too much noise, so this approach does work for the gyros. An EKF estimates omega

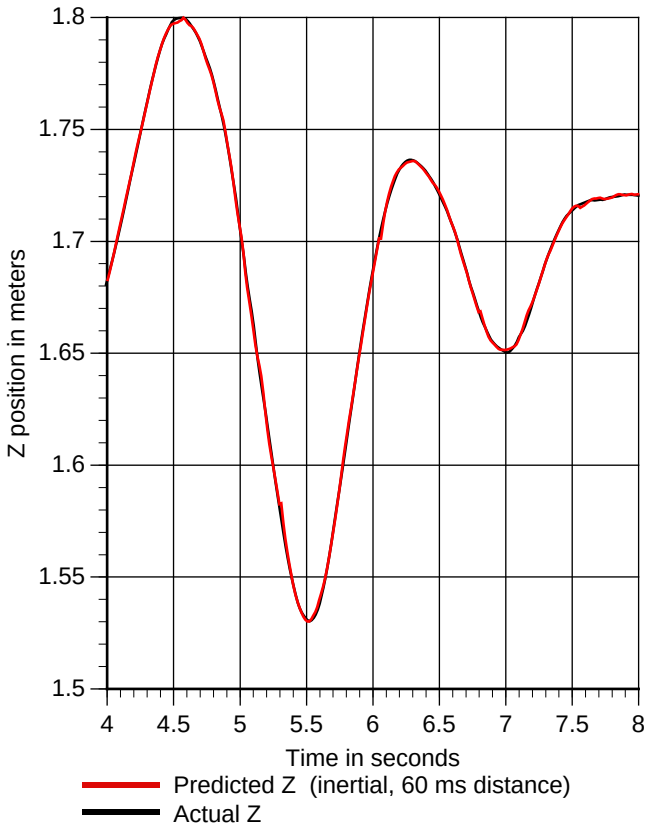


Figure 34: Z curve with inertial-based predictor

from the quaternions reported by the head tracker. It turns out that the estimated omegas are shifted backwards in time, because they are delayed versions of the true values. This timeshift, which is about 60 ms, is one measure of how much inertial sensors help the prediction task. It means that the gyros provide information that would cost 60 ms of lag to generate without gyros. We include a timeshift along with the orientation, biases, and scales as system parameters to be searched for. Powell's method [27] adjusts these parameters until it finds the set that minimizes the mean-square difference between the two estimates of angular velocity.

The second method uses the integration approach. This works for both the gyros and accelerometers. While integration tends to reduce noise, it has a different problem: drift. We cannot integrate the velocities or accelerations for long periods of time without accumulating too much error. This is the same problem that prevents anybody from using a purely inertial tracking system. In practice, we can integrate angular velocities for a few seconds and linear accelerations for a small fraction of a second. The viable time interval for acceleration is small because the double integration needed to recover position generates drift errors that grow quadratically with time. Integration also requires initial estimates for position and velocity. Therefore, we implement the autocalibration routine as an "ideal noncausal predictor." The idea is simple: if one could somehow know the exact future velocities and accelerations of the user's head, integrating these future values should result in nearly perfect prediction. Of course, this is impossible in realtime, but it is possible in simulation on a recorded dataset by using a noncausal approach. Our "predictor" integrates the "future" angular velocities and linear accelerations for 100 ms. This 100 ms "prediction" is repeated at many different starting times along the entire motion dataset. Powell's method searches for the system parameters that result in the best match between the "predicted" positions and orientations and the positions and orientations reported by the tracker.

How well do these autocalibration procedures work? In practice, they provide reasonably consistent outputs for timeshifts, orientation and biases, but they are not robust at measuring scales. For example, on one motion dataset the gyro pack orientations determined by both methods are within 0.5 degrees of each other, and orientations computed from two different collected datasets generated results that differ by 0.6 degrees. Unfortunately, the scales determined by autocalibration are not nearly as consistent. Also, it has proven impossible to use autocalibration to measure the positions of the accelerometers (the **F0**, **F1**, and **F2** vectors in Appendix A), so we simply use a ruler to measure those. We usually get a closer fit with the gyros than with the accelerometers, a result that merits further investigation. The parameters that are most reliably determined appear to be the ones we are most sensitive to, which presumably are also the ones that affect the prediction task the most.

D Limits of prediction

In Section 5.3, we mention that prediction is not effective to arbitrary distances. Part of the reason is shown in Figure 7, which demonstrates how errors in the predicted outputs grow with increasing prediction distances. This should be intuitive; asking a predictor to extrapolate farther into the future makes the task harder and increases the expected error. For long distances, the task is essentially intractable. We cannot predict head motion ten seconds into the future with any real accuracy; in ten seconds the user can be anywhere.

However, accuracy is only half the story. Jitter in the predicted outputs is the other factor. Almost all head-motion energy stays below 2 Hz when looked at in Fourier space, but predicted outputs have additional energy in the 3-12 Hz range. These relatively high frequency energies appear to the user as objectionable oscillations that do not correspond with the user's actual motion; we call this jitter. Jitter is especially noticeable on motion sequences where the user walks around, because the act of walking adds high-frequency wobbles to the pitch and roll rotations and introduces spikes in the accelerometers. Many previous works, including the two that used inertial sensors, focused on sitting users in flight simulators [1][24][25][35][37], so they did not encounter this problem. Smoothing the predicted signals proved ineffective because lowpass filtering adds latency, the very thing we want to remove. The only way we have been able to keep jitter at tolerable levels is by restricting prediction distances to short intervals of around 80 ms or less. Figure 35 shows a small segment of a quaternion curve with an overlayed predicted curve. The predicted curve was computed at 60 ms distance. In comparison, Figure 36 shows the same curve with prediction done at 130 ms distance. Note that not only is the prediction less accurate, but the oscillations are much larger. The same problem occurs with or without the use of inertial sensors. The predicted curve overlayed in Figure 37 was computed at 130 ms distance without inertial sensors.

Prediction is most useful at relatively low prediction distances. Prediction is not a substitute for reducing overall system latency; it must be combined with efforts to reduce system lag. Those who have systems with 200 ms of lag and wish to use head-motion prediction to extrapolate across that distance are likely to be disappointed by the results.

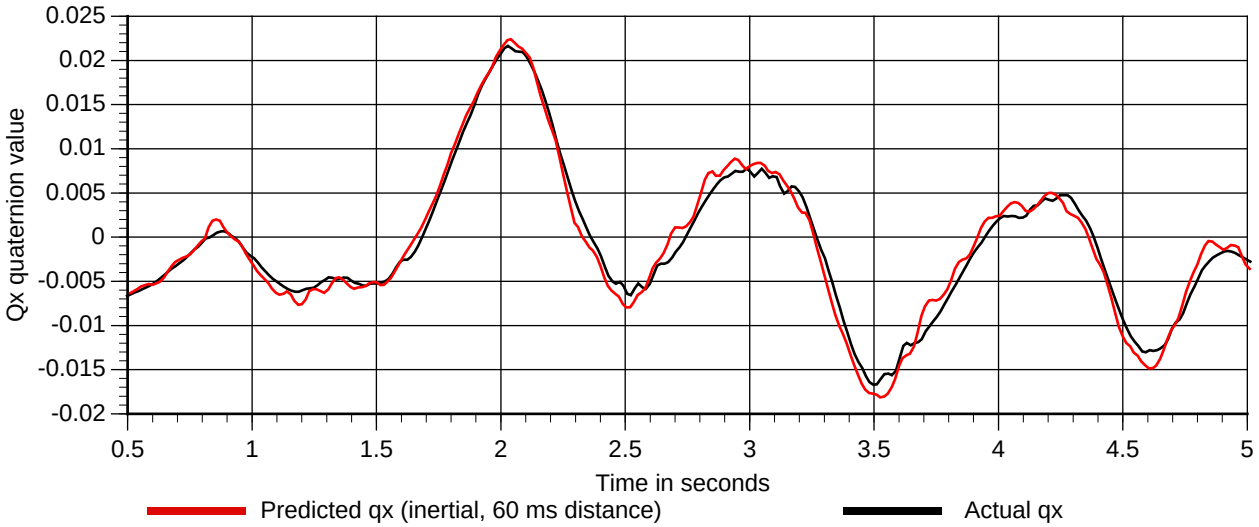


Figure 35: Jitter on qx cuve with inertial-based prediction and a 60 ms prediction distance

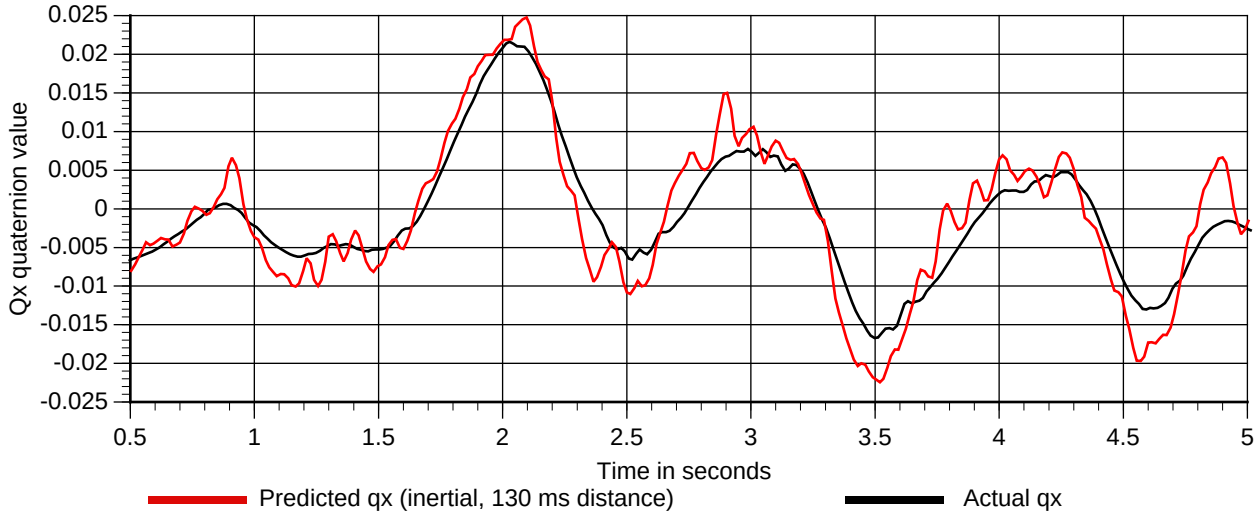


Figure 36: Jitter on qx cuve with inertial-based prediction and a 130 ms prediction distance

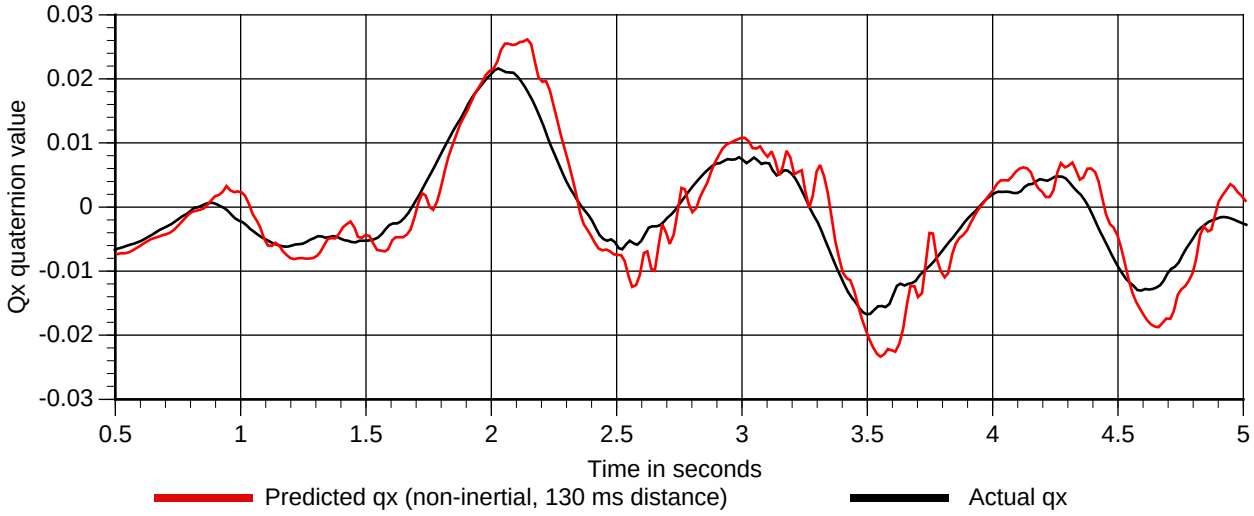


Figure 37: Jitter on qx cuve with non-inertial-based prediction and a 130 ms prediction distance

E Estimating total prediction distance

Accurate prediction requires accurate estimation of how far to predict into the future. When we run predictors in simulation, we use a constant prediction distance. Many previous works do the same, even in runtime. However, using a constant prediction distance is not appropriate in our real system, because our tracker and graphics engine run asynchronously. Unless a system is set up so all components run synchronously, with guaranteed limits on how long each step takes, the prediction distance will vary with time. Therefore, the problem of estimating total prediction distance is likely to occur in most Augmented Reality systems. In Section 5.3 we discussed steps taken to insure accurate timestamps and minimal latency. In this appendix, we describe how we estimate the total prediction distance and evaluate the accuracy of this estimation.

We define the total prediction distance to be the interval from the time when the tracker and inertial measurements are taken to the time when the frame buffer begins scanning out the images corresponding to those measurements (Figure 38). Part of this distance is directly measurable. Our tracker provides timestamps for its measurements, and we can read the clock at the point labeled "Prediction distance must be set here" in Figure 38. However, the rest of the distance must be estimated. Why? The predictor requires a distance to predict into the future. The predictor must be run before viewpoint computation and rendering. Therefore, the prediction distance must be set right before we run the predictor, requiring us to estimate the length of the light-shaded interval in Figure 38.

Figure 39 shows how we generate this estimate. The key observation is that our renderer is synchronized to the vertical retrace signal, which occurs every 16.67 ms. Thus, the renderer is guaranteed to render frames at 60 Hz, whether the tracker can keep up with that or not. The estimated distance is made of three components. Component A, the time needed to run the predictor and compute the view matrices, is basically constant and can be measured during trial runs. We compute component B, the delay until the renderer accepts our new matrices, by finding the next rendering starting point that is greater than Measured + A. The predictor knows where these starting points are because it receives a constant stream of past starting points from the renderer. Since the starting points are separated by a strict 16.67 ms, it's easy to determine all future starting points, given one point in the past. Finally, component C is 16.67 ms: the time it takes the renderer to rasterize its images and copy them to the frame buffer.

Figure 40 demonstrates the effectiveness of the estimation. It plots the estimated total prediction distance vs. the actual total prediction distance for a sequence of frames, from a real motion dataset recorded in realtime. We determined the actual distances by recording the beginning and ending times for all the frames, computing the differences after completing the data collection. Note that the estimation is accurate, so the two curves overlap, except for one frame around #225. When the estimation is wrong, it's wrong by 16.67 ms. Figure 40 also demonstrates how the total prediction distance varies from iteration to iteration.

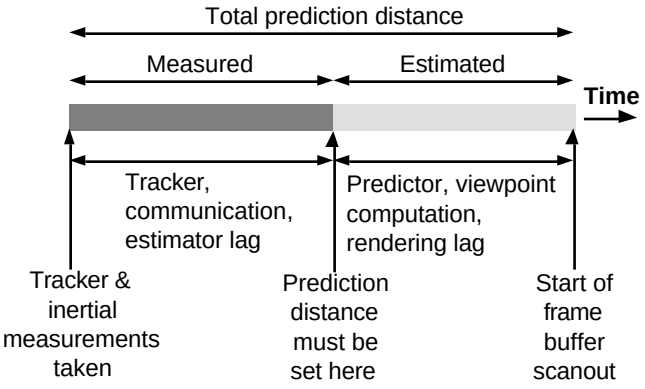
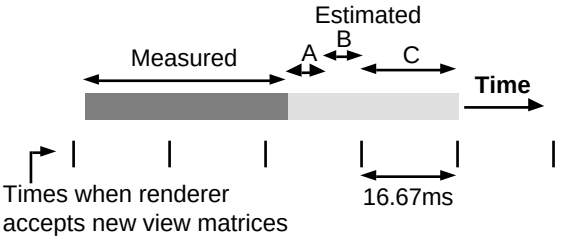


Figure 38: Total prediction distance



A = Time to run predictor and view matrix computation
B = Delay until renderer accepts new matrices
C = Time for renderer to compute new images

Figure 39: Components of estimated distance

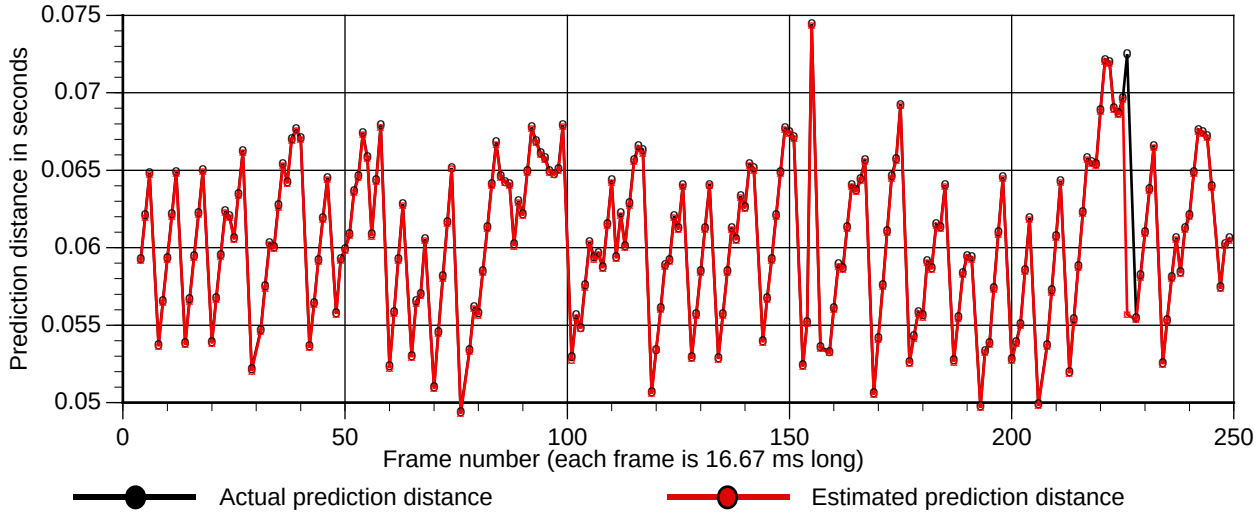


Figure 40: Estimated vs. actual total prediction distances

F Miscellaneous comments

The HMD is not as firmly attached to the user's head as we would like, but that has not turned out to be a major problem. Initially we feared that moving the HMD on the user's head after calibration would ruin the registration, but in practice it seems to have little effect. Even if the HMD slides on the user's head, the HMD itself stays rigid, so the relationship between the right eye display and the head tracker remains constant. The user compensates for the HMD sliding by rotating his eyeball. The net change in the Eye->Tracker transformation is small, causing little difference in apparent registration.