

Sensor Fusion for Augmented Reality

Jurjen Caarls, Pieter Jonker, and Stelian Persa

Delft University of Technology
{jurjen, pieter, stelian}@ph.tn.tudelft.nl

Abstract. In this paper we describe in detail our sensor fusion framework for augmented reality applications. We combine inertia sensors with a compass, DGPS and a camera to determine the position of the user's head. We use two separate extended complementary Kalman filters for orientation and position. The orientation filter uses quaternions for stable representation of the orientation.

1 Introduction

Wireless technologies that enable continuous connectivity for mobile devices will lead to new application domains. An important paradigm for continuously connected mobile users based on laptops, PDAs, or mobile phones, is context-awareness. Context is relevant in a mobile environment, as it is dynamic and the user interacts in a



Fig. 1. Augmented Reality Prototype



Fig. 2. Example of augmentation of the visual reality (1)

different way with an application when the context changes. Context is not limited to the physical world around the user, but also incorporates the user's behavior, his

terminal and the network characteristics. As an example of a high-end context aware application, we are in the development of an Augmented Reality system that can be connected to a roaming PDA.



Fig. 3. Example of augmentation of the visual reality (2)

Our user carries a wearable terminal and a see-through display in which the user can see virtual visual information that augments reality (Figure 1). Augmented Reality differs from Virtual Reality in the sense that the virtual objects are rendered on a see-through headset. As with audio headphones, which make it possible to hear sound in private, partly in overlay with the sounds from the environment, see-through headsets can do that for visual information. The virtual objects are in overlay with the real visual world (Figures 2, 3, and 4). It can also be used to place visual information on otherwise empty places, such as white parts of walls of a museum. The 3D vector of position and orientation is referred to as pose. Knowing the pose of those walls and the pose of a person's head, visual data can be perfectly inlayed on specific spots and kept there while the head is moving.

To lock the virtual objects in the scene, the head-movements must be sampled with such a frequency and spatial accuracy that the rendering of virtual images does not cause motion sickness. Our system can be applied in Tour Guiding, Remote Maintenance, Design Visualization and Games.

The wearable system contains a radio link that connects the user to computing resources and the Internet. For outdoor augmented reality, location determination is based on Differential GPS, while WLAN is used for the connection with backbone services. For communication in places without WLAN access-points near-by, GPRS can be used. For indoor applications, a presence and location server can be based on the signal strength of Bluetooth and/or WLAN access-points [4]. Based on the course position and orientation from the servers, a camera on the AR headset can capture the user's environment, which, fused with data from an inertia system: gyroscopes, accelerometers, and compass, can make the PDA fully aware of the absolute position and orientation of the user. Camera images can be sent to the backbone and matched to a 3D description of the environment to determine the user's position and to answer

questions of the user and his PDA that relate to the environment. This description can be derived from a GIS database, for outdoor, or a CAD database for indoor applications, see [5]. For simple indoor applications tags can be used that e.g. stick on known positions on the walls. To prevent motion sickness, rendering latencies lower than 10 ms are necessary. In conventional systems with a refresh rate of 50Hz it takes 20 ms to display a single frame. The time to render that frame will add to the total latency. It is clear that it is not possible to reach the required latency for augmented reality (<10ms) by sequentially rendering and displaying a frame. Consequently, our system renders only a part of the frame just ahead of the display's raster beam in four slices, and has a combined rendering and display latency of 8 ms [6].



Fig. 4. Example of an overlaid GIS/CAD model of a building.

In this paper we address the problem of the fusion of the sensors that are needed to obtain an accurate, fast and stable rendering system for augmented reality of indoor and outdoor scenes. We fused data of various sensors with different update rates and accuracies, including vision and DGPS, by using extended Kalman Filtering. The novelty in our approach is the use of quaternion descriptions inside the filter.

2 Sensors

To track the pose (position and orientation) of the user's head, we use a combination of sensors, which can be divided into relative sensors (angular velocity and linear acceleration) and absolute sensors (orientation and position). For the relative sensors we used three gyroscopes (Murata) and three accelerometers (ADXL202) combined in one board linked to a LART platform [1] developed at the Delft University of Technology. For the absolute sensors we use a Precision Navigation TCM2 compass, tilt sensor, and a JAI CV-S3300 camera.

The Murata Gyrostar piezoelectric vibrating gyros can measure up to 300 °/s. They are inexpensive but have a large bias that varies with time up to 9 °/s. Consequently, we had to correct for this bias. After our correction, the noise level is around 0.2 °/s

when sampled at 100Hz. The accelerometers (ADXL202) have also a varying offset. This offset can be 0.5 m/s^2 and the residual noise level is around 0.06 m/s^2 when sampled at 100Hz. The maximum acceleration that can be measured is $2g$ in both directions.

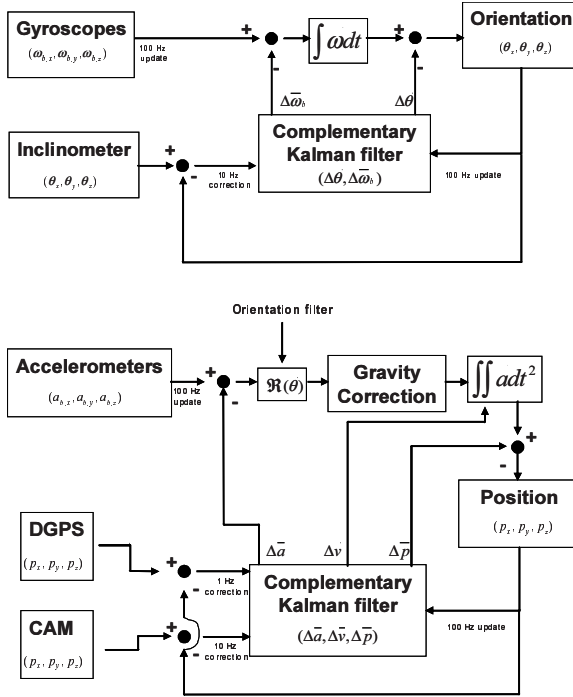


Fig. 5. Fusion of data from the sensors in the pose tracking system. *Top:* orientation. *Bottom:* position

The TCM2-50 liquid inclinometer uses a viscose fluid to measure the inclination with respect to the gravity vector with an accuracy of 0.2° . The heading is calculated using three magnetometers with an accuracy of $0.5\text{-}1.5^\circ$. Because the liquid will slightly slosh when accelerations are applied, we have to cope with an error of about 20° . The update rate is 16 Hz. The JAI CV-S3300 camera with a resolution of 320×240 pixels in grayscale has a wide-angle lens with a 90° opening angle, which introduces spherical distortions. We calibrate the camera using the Zhang algorithm [7]. Images are grabbed at 15 Hz. When the camera sees a specific block pattern in the image it can track the full 6D pose of the camera. The LART platform, that is used for data acquisition and preprocessing, has an 8-channel fast 16-bit AD-converter to acquire synchronous data from the accelerometers, gyros and temperature data. The gyros and the accelerometers are analog devices, which are sampled at 100 Hz by the AD converter. The TCM2 updates at 16 Hz and is read via a serial line. When the TCM2 and the gyros are read out simultaneously, there is an unknown difference in the time of

the physical events. We could compensate the relative latencies by attaching a time stamp to the readouts. Figure 5 shows a diagram of the fusion of data from the sensors in the pose tracking system, which is explained in detail in the next section.

2.1 Fusion Framework

For our fusion framework, we use the following coordinate systems:

- Ψ_b The body frame. It is attached to the body of the headset for which we need the pose.
- Ψ_n The navigation frame. This frame is a rotated body frame. It has the z-axis pointing in the direction of the gravity vector, while the x-axis is pointing to the earth's North Pole.
- Ψ_p The pattern frame. This frame is attached to the pattern used to determine the camera's position.
- Ψ_w The world frame. This frame is like the navigation frame, but has a fixed origin.

Sub indices like $\Psi_{b,1}$ denote a specific instance of a moving frame.

To be able to combine sensors measurements with different update rates and error characteristics, we have chosen a Kalman filter setup [2]. Due to the rotations, the Kalman equations become non-linear, and hence we need to linearise the filter. We have chosen to use the errors in position and orientation as filter states, as then we can update the real states using nonlinear formulas to obtain a better performance. To overcome singularities when representing orientation in Euler angles, we used the quaternion notation. Quaternions can be used to represent orientations in 3D. The advantage over the use of common Euler angles is that the representation of the orientation is continuous, i.e. without jumps from 2π to 0.

2.2 Quaternions

A quaternion has a scalar part and a vector part:

$$\begin{aligned}
 q &= \begin{pmatrix} q_0 \\ \vec{q} \end{pmatrix} \text{ and } q^* = \begin{pmatrix} q_0 \\ -\vec{q} \end{pmatrix} \\
 q_0 &= \cos(\theta/2) \\
 \vec{q} &= \vec{n} \cdot \sin(\theta/2)
 \end{aligned} \tag{1}$$

in which θ is the angle of rotation around the normalized vector \vec{n} , and q^* is the complex conjugate of q . If q represents an orientation, i.e. a unit quaternion, then its inverse becomes:

$$q^{-1} = \frac{q^*}{\|q\|^2} = q^* \tag{2}$$

A quaternion that represents a rotation of a frame Ψ_A expressed in terms of frame Ψ_B is represented by q_A^B , the rotation that rotates frame Ψ_B to frame Ψ_A expressed in Ψ_B . Let the quaternion representation of a vector \vec{v}_a be:

$$q_{\vec{v}_a} = \begin{pmatrix} 0 \\ \vec{v}_a \end{pmatrix} \tag{3}$$

Then the rotation of this vector is obtained by a double quaternion multiplication:

$$q_{\vec{v}_b} = q_A^B \otimes q_{\vec{v}_a} \otimes q_A^{B*} = R_A^{B*} \vec{v}_A \tag{4}$$

in which the operator \otimes is the quaternion multiplication. In matrix form this becomes:

$$\begin{aligned} q_1 \otimes q_2 &= \widetilde{q_1} q_2 = \begin{pmatrix} q_{1,0} & -q_{1,x} & -q_{1,y} & -q_{1,z} \\ q_{1,x} & q_{1,0} & -q_{1,z} & q_{1,y} \\ q_{1,y} & q_{1,z} & q_{1,0} & -q_{1,x} \\ q_{1,z} & -q_{1,y} & q_{1,x} & q_{1,0} \end{pmatrix} \begin{pmatrix} q_{2,0} \\ q_{2,x} \\ q_{2,y} \\ q_{2,z} \end{pmatrix} \\ &= \widehat{q_2} q_1 = \begin{pmatrix} q_{2,0} & -q_{2,x} & -q_{2,y} & -q_{2,z} \\ q_{2,x} & q_{2,0} & q_{2,z} & -q_{2,y} \\ q_{2,y} & -q_{2,z} & q_{2,0} & q_{2,x} \\ q_{2,z} & q_{2,y} & -q_{2,x} & q_{2,0} \end{pmatrix} \begin{pmatrix} q_{1,0} \\ q_{1,x} \\ q_{1,y} \\ q_{1,z} \end{pmatrix} \end{aligned} \tag{5}$$

In which \widetilde{q} is the quaternion matrix and \widehat{q} the transmuted quaternion matrix.

The representation of angular velocity vectors using quaternions is analogue to the case of rotations of position vectors over angles. The angular velocity of Ψ_i with respect to Ψ_j expressed in Ψ_i is given by:

$$\dot{R}_i^j = R_i^j \cdot \widetilde{\omega}_i^{j,j} = R_i^j \cdot \begin{pmatrix} 0 & -\omega_x & \omega_y \\ \omega_x & 0 & -\omega_z \\ -\omega_y & \omega_z & 0 \end{pmatrix} \tag{6}$$

The subscripts i and j in $\tilde{\omega}_i^{i,j}$ are removed for clarity. In quaternion notation this is:

$$\dot{q}_i^j = q_i^j \otimes \frac{1}{2} q_{\tilde{\omega}_i^{i,j}} = \frac{1}{2} \widehat{q_{\tilde{\omega}_i^{i,j}}} \cdot q_i^j \quad (7)$$

As the scalar part of $q_{\tilde{\omega}_i^{i,j}}$ is zero, the solution to equation (7) is:

$$\begin{aligned} q_i^j(t) &= e^{\frac{1}{2} \widehat{q_{\tilde{\omega}_i^{i,j}}} \cdot t} q_i^j(0) \\ &= \left(I \cdot \cos\left(\frac{1}{2} \|\tilde{\omega}_i^{i,j}\| t\right) + \sin\left(\frac{1}{2} \|\tilde{\omega}_i^{i,j}\| t\right) \cdot \frac{\widehat{q_{\tilde{\omega}_i^{i,j}}}}{\|\tilde{\omega}_i^{i,j}\|} \right) q_i^j(0) \end{aligned} \quad (8)$$

or:

$$q_i^j(t) = q_i^j(0) \otimes \begin{pmatrix} \cos\left(\frac{1}{2} \|\tilde{\omega}_i^{i,j}\| t\right) \\ \sin\left(\frac{1}{2} \|\tilde{\omega}_i^{i,j}\| t\right) \frac{\tilde{\omega}_i^{i,j}}{\|\tilde{\omega}_i^{i,j}\|} \end{pmatrix} \quad (9)$$

A more detailed treatment is given in [3]

2.3 Kalman Filters

The time update for a Kalman filter without control inputs is given by:

$$\begin{aligned} \hat{x}_k^- &= \Theta_{t_k, t_{k-1}} \hat{x}_{k-1}^- \\ P_k^- &= \Theta_{t_k, t_{k-1}} P_{k-1} \Theta_{t_k, t_{k-1}}^T + Q(t_k, t_{k-1}) \\ Q(t_k, t_{k-1}) &= \int_{t_{k-1}}^{t_k} \Theta_{t_k, s} \cdot Q(s) \cdot \Theta_{t_k, s}^T ds \\ &\approx \Theta_{t_k, t_{k-1}} \cdot Q(t_{k-1}) \cdot \Theta_{t_k, t_{k-1}}^T \cdot (t_k - t_{k-1}) \\ Q(t) &= \text{cov}(w(t), w(\tau)) \end{aligned} \quad (10)$$

in which \hat{x}_k^- is the current a-priori estimate of the state x_k , Θ is the state transition matrix that projects a state into the future, and $w(t)$ is the white noise process, which models not modeled changes in the state. When an observation is done, the a-posteriori estimate of the state can be calculated from the estimate of the observation:

$$\begin{aligned}
 y_k^- &= H \hat{x}_k^- \\
 \hat{x}_k &= \hat{x}_k^- + K (y_k - y_k^-) \\
 P_k &= (I - KH) P_k^- \\
 K &= P_k^- H (H P_k^- H^T + R)^{-1} \\
 R &= \text{cov}(v(t), v(\tau))
 \end{aligned} \tag{11}$$

in which y_k^- is the predicted observation, y_k is the real observation, H is the output matrix, K is the Kalman gain, and $v(t)$ white noise that models measurement noise in y_k .

In the sequel we use X for the estimate of the real states (a-posteriori and a-priori) and dX for the error states of the Kalman filter. The state-vector X contains orientation, angular velocity, position, linear velocity, linear acceleration, current gyro and current accelerometer bias:

$$X = \left(q_b^n, \vec{\omega}_b^{b,n}, \vec{p}_b, \vec{v}_b^n, \vec{a}_b^b, \vec{b}_{gyro}^b, \vec{b}_{acc}^b \right)^T \tag{12}$$

To simplify the system we use two Kalman filters. One for the orientation, with state:

$$dX_{orient} = \left(dq_b^n, d\vec{b}_{gyro}^b \right)^T \tag{13}$$

being the error in orientation and the drift in the gyroscopes, and one for the position containing the error in position, linear velocity and accelerometer drift:

$$dX_{pos} = \left(d\vec{p}_b^n, d\vec{v}_b^n, d\vec{b}_{acc}^b \right)^T \tag{14}$$

The accelerometer bias and its error state are expressed in body coordinates, and therefore the position filter depends on the orientation. So in this set-up it is essential to have an accurate orientation estimate, as its error will propagate with t^2 into the position.

2.4 Time Update

Each time a measurement is obtained from the inertial sensors, the estimate of the actual state X is updated, using eq.(9) and:

$$\begin{aligned}
 \vec{v}_b^n(\Delta t) &= \vec{v}_b^n(0) + \left(R_b^n(0) \cdot \vec{a}_b^b(0) - \vec{g}^n \right) \Delta t \\
 \vec{p}_b^n(\Delta t) &= \vec{p}_b^n(0) + \frac{1}{2} \left(\vec{v}_b^n(0) + \vec{v}_b^n(\Delta t) \right) \Delta t
 \end{aligned} \tag{15}$$

Note, that we do not take the rotational speed into account, which results in an error in $\vec{a}_b^b(t)$ of about $3 \cdot 10^{-3} \text{ m/s}^2$ when rotating at 200 deg/s and $\Delta t = 1/100\text{s}$, in a direction perpendicular to the gravity vector.

The state update for the position Kalman filter is based on the formulas (10) and (15), and we neglect rotations errors in R_b^n . The change in rotation should be small due to the high update rate, and the estimate in orientation is rather accurate. The state transition is now given by:

$$dX_{pos}(t_k) = \Phi dX_{pos}(t_{k-1}) \Rightarrow \begin{pmatrix} d\vec{p}_b^n \\ d\vec{v}_b^n \\ d\vec{b}_{acc}^b \end{pmatrix}(t_k) = \begin{pmatrix} I & I \cdot \Delta t & \frac{1}{2} R_b^n(t_k) \cdot \Delta t^2 \\ 0 & I & R_b^n(t_k) \cdot \Delta t \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} d\vec{p}_b^n \\ d\vec{v}_b^n \\ d\vec{b}_{acc}^b \end{pmatrix}(t_{k-1}) \quad (16)$$

The state update for the orientation Kalman filter is more complicated, because we use the orientation difference in quaternion notation:

$$q_b^n = q_b^{n-} \otimes dq_b^n \Leftrightarrow dq_b^n = q_b^{n-*} \otimes q_b^n \quad (17)$$

in which q_b^{n-} is the estimated orientation by integration using eq. (9) and q_b^n is the real state. Using:

$$\begin{aligned} q_b^{n-*} \otimes q_b^{n-} = 0 &\Rightarrow q_b^{n-*} \otimes q_b^{n-} + q_b^{n-*} \otimes q_b^{n-} = 0 \Rightarrow \\ q_b^{n-*} &= -q_b^{n-*} \otimes q_b^{n-} \otimes q_b^{n-*} \end{aligned} \quad (18)$$

the time derivate of the estimate of dq_b^n becomes:

$$\begin{aligned} \dot{dq}_b^n &= q_b^{n-*} \otimes \dot{q}_b^n + q_b^{n-*} \otimes \dot{q}_b^n \\ \dot{dq}_b^n &= -q_b^{n-*} \otimes \dot{q}_b^{n-} \otimes q_b^{n-*} \otimes q_b^n + \frac{1}{2} q_b^{n-*} \otimes (q_b^n \otimes q_{\omega_b^{b,n}}) \\ \dot{dq}_b^n &= -q_b^{n-*} \otimes \left(\frac{1}{2} q_b^{n-} \otimes q_{\omega_b^{b,n-}} \right) \otimes dq_b^n + \frac{1}{2} dq_b^n \otimes q_{\omega_b^{b,n}} \\ \dot{dq}_b^n &= -\frac{1}{2} q_{\omega_b^{b,n-}} \otimes dq_b^n + \frac{1}{2} dq_b^n \otimes q_{\omega_b^{b,n}} \\ \dot{dq}_b^n &= -\frac{1}{2} \widetilde{q_{\omega_b^{b,n-}}} \cdot dq_b^n + \frac{1}{2} \widetilde{q_{\omega_b^{b,n}}} \cdot dq_b^n \\ \dot{dq}_b^n &= \frac{1}{2} \left(\widetilde{q_{\omega_b^{b,n}}} - \widetilde{q_{\omega_b^{b,n-}}} \right) \cdot dq_b^n \end{aligned} \quad (19)$$

We can calculate the true $\omega_b^{b,n}$ from its estimate $\omega_b^{b,n-}$ with:

$$\omega_b^{b,n} = \omega_b^{b,n-} - d\vec{b}_{gyro}^b \tag{20}$$

Then equation (19) becomes after some elaboration:

$$\dot{dq}_b^n = \frac{1}{2}(\widetilde{q}_\omega - \widetilde{q}_{\omega^-}) \cdot dq = \frac{1}{2} \begin{pmatrix} -d\vec{b} \cdot d\vec{q} \\ q_0 \cdot d\vec{b} - d\vec{b} \times d\vec{q} + 2\widetilde{w} \times d\vec{q} \end{pmatrix} \tag{21}$$

Now this can be linearized around the state dX_{orient} at time $t = 0$ (the previous estimate). In the indirect filter setup, the error states are reset after every observation update. This means that $d\vec{b}$ will be assumed constant and 0, $d\vec{q}$ will be small, and dq_0 will be approximately 1. Our linearization of the time derivative becomes:

$$\begin{pmatrix} \dot{dq}_0 \\ \dot{d\vec{q}} \\ \dot{d\vec{b}} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \widetilde{w} & \frac{1}{2} \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} dq_0 \\ d\vec{q} \\ d\vec{b} \end{pmatrix} \tag{22}$$

In the case \widetilde{w} is assumed constant (zero order hold), we can find the state transition to be:

$$dX_{orient}(t_k) = \Phi dX_{orient}(t_{k-1}) \Leftrightarrow \begin{pmatrix} dq_0 \\ d\vec{q} \\ d\vec{b} \end{pmatrix}_{(t_k)} = e^{\begin{pmatrix} 0 & 0 & 0 \\ 0 & \widetilde{w}(t_k) & \frac{1}{2} \\ 0 & 0 & 0 \end{pmatrix} \Delta t} \begin{pmatrix} dq_0 \\ d\vec{q} \\ d\vec{b} \end{pmatrix}_{(t_{k-1})} \tag{23}$$

Of course, after this update, dq should be normalized to unity again. The algebraic solution to this exponent of a matrix was found using a mathematical software package.

2.5 Observation Update

When an inclinometer measurement becomes available, it is converted to quaternion notation. Now it becomes easy to determine the observation estimate of the error in orientation:

$$dq_{b,obs_est}^n = q_b^{n-*} \otimes q_{b,inclino}^n \tag{24}$$

This observation estimate replaces y_k in eq. (11)

Because both estimates are in quaternions, the output matrix H becomes:

$$y_k^- = H \hat{x}_k^- = \begin{pmatrix} I & 0 \end{pmatrix} \begin{pmatrix} dq \\ d\vec{b} \end{pmatrix} \quad (25)$$

A problem now is the Kalman measurement noise \vec{v} , which is dependent on both the measurement and the estimated real state. We only take into account the measurement error, as it will be larger than the estimate error, which is the error in the integration of the gyro measurements.

To find the covariance matrix R , we determined the linearised matrix that relates the deviation in dq_{b,obs_est}^n to the inclinometer measurement deviation $\delta\vec{\theta}$:

$$\begin{aligned} dq_{b,obs}^n(\vec{\theta}_{inclino} + \delta\vec{\theta}) &\approx dq_{b,obs}^n(\vec{\theta}_{inclino}) + \frac{\partial dq_{b,obs}^n(\vec{\theta}_{inclino})}{\partial \vec{\theta}} \delta\vec{\theta} \\ \delta dq_{b,obs}^n &= \frac{\partial dq_{b,obs}^n(\vec{\theta}_{inclino})}{\partial \vec{\theta}} \delta\vec{\theta} = G \cdot \delta\vec{\theta} \\ \vec{v} &= G \cdot \vec{w} \\ R &= \text{cov}(\vec{v}(t), \vec{v}(\tau)) = G \cdot \text{cov}(\vec{u}(t), \vec{u}(\tau)) \cdot G^T \end{aligned} \quad (26)$$

in which $\vec{u}(t)$ is the presumed white noise in the inclinometer measurement.

In the position update, the estimated position error is:

$$d\vec{p}_{b,obs_est}^n = \vec{p}_b^{n-} - \vec{p}_{b,observation}^n \quad (27)$$

and using y_k^- for $d\vec{p}_{b,obs_est}^n$ we get for formula (11):

$$y_k^- = H \hat{x}_k^- = \begin{pmatrix} I & 0 & 0 \end{pmatrix} \begin{pmatrix} d\vec{p} \\ d\vec{v} \\ d\vec{b} \end{pmatrix} \quad (28)$$

Matrix G that relates the error in $d\vec{p}_{b,obs_est}^n$ to the error in $\vec{p}_{b,observation}^n$ is the negation of the identity matrix.

2.6 Coping with Lag

The inclinometer output is delayed with about 0.375s (or 6 samples @ 16Hz). When we ignore this delay, the orientation Kalman filter will assume an error in orientation and will adjust the current error and bias estimate. After the rotation, the filter needs some time to recover. A bigger problem is that the position filter uses the orientation, and this delay will hence introduce a non-existent acceleration.

In our method we store all the observations of the sensors, as well as the Kalman states and matrixes, at every step and keep a history of 30 steps. When an inclinometer measurement finally arrives, we step back to the position in time of that measure-

ment, and do the filtering in the Kalman state that belongs to that point in time. From here on all the other measurements (such as gyro, camera, GPS) are processed again up to the current time. In this way the best estimate at the current time is achieved.

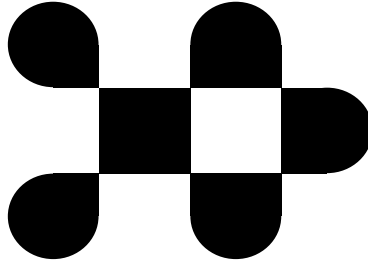


Fig. 6. Pattern with 6 saddle points ordered in two 3-point rows.

3 Camera Positioning

3.1 Finding Markers

For camera positioning we use a pattern like in Fig. 6. This pattern consists of 6 saddle points, which are easy to detect using the determinant of the Hessian. The filter output $g(\vec{p})$ of a 2D image $f(\vec{p})$ is given by:

$$g(\vec{p}) = -\det[H[f(\vec{p})]] = -\begin{vmatrix} \partial_{xx} & \partial_{xy} \\ \partial_{yx} & \partial_{yy} \end{vmatrix} f(\vec{p}) \quad (29)$$

The derivatives ∂ are implemented using the derivative of a Gaussian with $\sigma = 2.0$. To find the saddle points, we threshold the output with value:

$$th_{\text{detector}} = \frac{1}{2} \max_{\vec{p}} [g(\vec{p})] \quad (30)$$

and apply a peak detection filter in a 3×3 neighborhood S . This leaves us with a set of saddle points:

$$sp = \left\{ \vec{p} \mid g(\vec{p}) = \max_{\vec{q} \in S} [g(\vec{p} + \vec{q})] \wedge g(\vec{p}) > th_{\text{detector}} \right\} \quad (31)$$

To obtain sub pixel accuracy at each point a paraboloid is fit. This is possible as the filter has a parabolic shape at a saddle point. The true saddle point is located at an offset with respect to a point in sp . The model is:

$$g(x, y) = d + a((x - x_m)^2 + (y - y_m)^2) \text{ or}$$

$$\mathbf{y} = A\mathbf{x} = \begin{pmatrix} d + a(x_m^2 + y_m^2) \\ -2ax_m \\ -2ay_m \\ a \end{pmatrix}^T \begin{pmatrix} 1 \\ x \\ y \\ x^2 + y^2 \end{pmatrix} \quad (32)$$

With the standard least squares method applied to the 3x3 neighborhood of each saddle point we find for A :

$$A = (\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \mathbf{y} \quad (33)$$

Note, that if we translate the coordinate system such that the origin is the position of the estimated saddle point, we can pre-calculate $(\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T$ to speed up processing.

From A the sub-pixel position of the saddle point can be calculated. For convenience we define:

$$\begin{aligned} \vec{p} &\in sp \\ \vec{p} &\text{ with subpixel accuracy} \\ g(\vec{p}) &\text{ is still the original filter output} \end{aligned}$$

To find all markers (as in Figure 6) we match groups of 6 points. When only marker saddle points are detected and the markers are not too close, we can find the pattern by looking at the 6 nearest neighbors at every point. We use the 10 nearest neighbors to be robust against non-marker saddle points. Observing Figure 6 one can see that the pattern consists of two 3-point lines. These lines will stay perfectly linear under all circumstances when viewed with a calibrated camera.

Consequently, for every point we find all possible 3-points long lines, with that point in the middle. The distance between the point and the line between the two other points should not be greater than some threshold ε :

$$L_{p_2} = \{\vec{p}_1, \vec{p}_2, \vec{p}_3 \mid \min_{s \in [0,1]} [\|\vec{p}_2 - (\vec{p}_1 + s(\vec{p}_3 - \vec{p}_1))\|] < \varepsilon, p_i \in sp\} \quad (34)$$

To find the pattern we consider each pair of two lines that do not have points in common. We find the set of candidate markers that consist of two lines with 6 unique points:

$$C = \{L_1 \in L_{p_1}, L_2 \in L_{p_2} \mid L_1 \cap L_2 = \emptyset\} \quad (35)$$

3.2 Determining the Pose from a Markers Feature Points

For each candidate marker a fitness value is calculated. First the position of the camera is estimated using the 6 points and part of the Zhang calibration algorithm [7] as described below. Then the fitness value is calculated as the mean square error of the back-projected marker points. We use a calibrated camera, so we can correct for lens distortion, skewing, scaling, and offset of the center point in the image. For each point we calculate the position on a fictive plane at distance 1 in camera coordinates:

$$P^C = \begin{pmatrix} s_x & 0 & x_{offset} \\ s_{sk} & s_y & y_{offset} \end{pmatrix} \begin{pmatrix} p^i \\ 1 \end{pmatrix} = AP^i \tag{36}$$

An estimate of the position of the camera can be found using the relation between the 6 points in camera coordinates, and the 6 points of the model (a 3D homography):

$$sP^C = s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = (\mathbf{R} \quad \mathbf{T}) \begin{pmatrix} p_x^P \\ p_y^P \\ p_z^P \\ 1 \end{pmatrix} \tag{37}$$

in which P^C is the homogeneous image-plane position, p_i^P are the components of the homogeneous position in the pattern-frame (the model), and \mathbf{R} and \mathbf{T} are the rotation and translation from the pattern frame, to the camera frame. This formula can be simplified by the fact that we defined $p_z^P = 0$. When we remove the z-coordinate from formula (37) we obtain:

$$sP^C = s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} r_1 & r_2 & \mathbf{T} \end{pmatrix} \begin{pmatrix} p_x^P \\ p_y^P \\ 1 \end{pmatrix} = \mathbf{H}P^P \tag{38}$$

This can be rewritten as:

$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} H_1 \\ H_2 \\ H_3 \end{pmatrix} P^P \tag{39}$$

This set of equations can be reordered to:

$$\begin{aligned} s &= H_3 P^P \\ H_1 P^P - u H_3 P^P &= 0 \\ H_2 P^P - v H_3 P^P &= 0, \text{ or} \end{aligned} \tag{40}$$

$$\begin{pmatrix} P^{PT} & 0 & -uP^{PT} \\ & P^{PT} & -vP^{PT} \end{pmatrix} \begin{pmatrix} H_1^T \\ H_2^T \\ H_3^T \end{pmatrix} = L\mathbf{x} = 0$$

in which P^{PT} is the transpose of P^P . The matrix L can be extended downwards for all 6 points, and the solution for \mathbf{x} is the right singular vector of L , associated with the smallest singular value. To get L numerically well conditioned, data normalization can be used. From \mathbf{x} we can reconstruct r_1 , r_2 and \mathbf{T} . To complete the rotation matrix \mathbf{R} we can use:

$$r_3 = r_1 \times r_2 \tag{41}$$

Because R is estimated, the matrix is not orthonormal. We can find the best orthonormal matrix using singular value decomposition:

$$R_{\text{estimate}} = UDV^T \Rightarrow R_{\text{orthonormal}} = UIV^T \tag{42}$$

We found that the resulting camera pose is not very stable, so we applied a Levenberg-Marquardt algorithm that optimizes the pose by minimizing our fitness criterion. The final step is to threshold the fitness value to get rid of candidate markers that do not match well enough. Note that the pose(s) found are in pattern coordinates, for each pattern. This means that we still have to perform a coordinate transformation from pattern coordinates to world coordinates.

3.3 Converting Pattern to World Coordinates

There are two difficulties in converting the position of the camera expressed in the pattern position frame to one expressed in the world position frame. One is, that we might not know the pose of the pattern in the world, and the other is, that the coordinate system of the (pinhole) camera is rotated with respect to the coordinate system of the inclinometer (body frame).

We can find the pattern's orientation expressed in the world frame by measuring three vectors. Each vector can be measured in pattern coordinates and in world coordinates. If the directions of the vectors span all three dimensions, we can find the coordinate transformation matrix:

$$\begin{aligned} \begin{pmatrix} v_1^W & v_1^W & v_1^W \end{pmatrix} &= R_p^W \begin{pmatrix} v_1^P & v_1^P & v_1^P \end{pmatrix} \\ R_p^W &= \begin{pmatrix} v_1^P & v_1^P & v_1^P \end{pmatrix}^{-1} \begin{pmatrix} v_1^W & v_1^W & v_1^W \end{pmatrix} \end{aligned} \tag{43}$$

These vectors can be found by moving the camera in these three directions, but we do not have an absolute positioning system for measuring the world coordinates. Another method is to turn the device around each of the vectors. The rotation axes can be found by looking at the change in orientation in both coordinate systems. The world origin can be chosen freely, and for now we use the startup position as the

world origin. Using that assumption, we can determine the position of the pattern in the world.

This translation is combined with the rotation matrix in a homogeneous matrix $H = \begin{pmatrix} R & T \end{pmatrix}$. We define:

H_A^B The transformation that brings frame Ψ_B to frame Ψ_A expressed in Ψ_B using homogenous coordinates

The pose of the camera – or body frame - is given by:

$$H_b^W = H_p^W H_b^P \quad (44)$$

Unfortunately, we only have the pose H_{CP}^P , of which Ψ_{CP} is the frame at the focal point of the camera, and orientated with the z-axis in the direction of the optical axis. This means that we have to find the unknown relation between Ψ_{CP} and Ψ_b , before we can determine the camera pose by:

$$H_b^W = H_p^W H_{CP}^P H_b^{CP} \quad (45)$$

The transformation H_b^{CP} is constant, and therefore we can just measure the pose of the camera expressed in world coordinates as well as the pose expressed in pattern coordinates. The required transformation is then given by:

$$H_b^{CP} = H_p^{CP} H_W^P H_b^W \quad (46)$$

measured at any one instant.

4 Results and Conclusion

Due to our set-up with a sensor cube with inertia sensors and the TCM2 mounted on top of the camera, the axes of the sensors will not be aligned either. After calibration however, we found that the misalignment in the sensor cube is minimal. Results showed that the orientation Kalman filter converged quickly. The stability of the gyros provided very accurate orientation values during motion, but due to the resolution of the inclinometer (0.2 degrees) the overall accuracy cannot be higher than 0.2 degrees, even when the 0.5 degrees accuracy in the heading has no systematic error.

To verify the camera positioning we tracked an A4 pattern (Figure 6) perpendicular to the optical axis, with a distance from 30-190 cm at 5 cm intervals. The pattern was kept roughly in the middle of the picture. The calibration was done, using the Zhang method. We measured the mean error and RMS value, σ , of the distance and the roll angle in pattern coordinates, by processing 100 images, real-time at 10 fps, at each distance. This was done on an AMD Athlon XP1700+, and the algorithm used about 40 ms per image. Some results are shown in Figure 7.

Looking at the left figure, one can observe that we have 1 cm accuracy up to a distance of about 1.2m. The feature distance is then 11 pixels. This means that if the markers are further away, the marker should increase in size, or multiple markers should be used. In the right figure one can see that the error in the angle is large at distances greater than 70 cm. This has also a negative impact on the accuracy in x and y, so it will be wise to use the orientation output of the Kalman filter to only optimize the position of the camera during the Levenberg-Marquardt step of the algorithm.

Experiments still have to be done to find accuracy measures for general camera positions, and for the position/orientation Kalman filter output during motion.

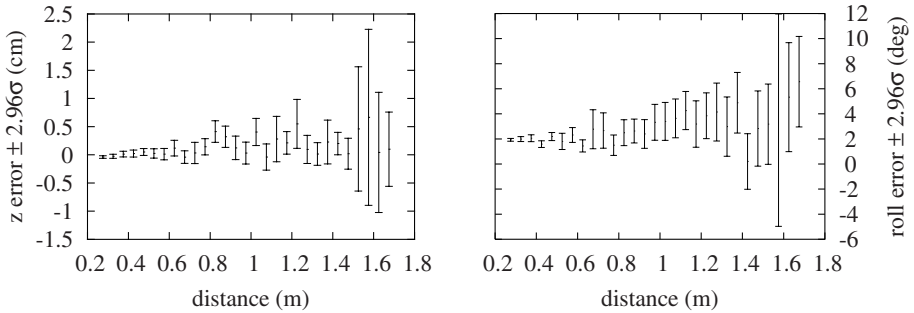


Fig. 7. Experiment in which the camera was moved perpendicular to the pattern. *Left:* error in pattern's z axis with 2.96σ error bars. *Right:* error in roll angle with 2.96σ error bars. Note that this σ is the RMS of the 100 values (see text).

References

1. Bakker, J.D., Mouw, E., Pouwelse, J., The LART Pages. Delft University of Technology, Faculty of Information Technology and Systems (2000). Available at <http://www.lart.tudelft.nl>
2. Brookner, E., Tracking and Kalman Filtering Made Easy, John Wiley & Sons Inc. (1998)
3. Caarls, J., Geometric Algebra with Quaternions, Technical Report (2003) <http://www.ph.tn.tudelft.nl/Publications/phreports>
4. Jonker, P.P., Caarls, J., Eijk, R. van, Peddemors, A., Heer, J. de, Salden, A., Määtä, P., Haataja, V. Augmented Reality Implemented on a Mobile Context Aware Application Framework, submitted to IEEE Computer (2003)
5. Jonker, P.P., Persa, S., Caarls, J., Jong, F. de, Lagendijk, I. Philosophies and Technologies for Ambient Aware Devices in Wearable Computing Grids, Computer Communications Journal, Volume 26, Issue 11, 1 July 2003, Pages 1145–1158. <http://www.sciencedirect.com/science/journal/01403664>
6. Paman, W., Schaaf, A. van der, Lagendijk, R. L., & Jansen, F. W. (1999). Accurate overlaying for mobile augmented reality. Computers & Graphics, 23 (6), 875–881. <http://www.cg.its.tudelft.nl/~wouter>
7. Zhang, Zhengyou. A Flexible New Technique for Camera Calibration. <http://www.research.microsoft.com/~zhang/calib>