Inertial Measurement Units II

Gordon Wetzstein
Stanford University
EE 267 Virtual Reality
Lecture 10

stanford.edu/class/ee267/
Lecture Overview

• short review of coordinate systems, tracking in flatland, and accelerometer-only tracking

• rotations: Euler angles, axis & angle, gimbal lock

• rotations with quaternions

• 6-DOF IMU sensor fusion with quaternions
• primary goal: track orientation of head or device

• inertial sensors required pitch, yaw, and roll to be determined
from lecture 2:

vertex in clip space

\[ v_{clip} = M_{proj} \cdot M_{view} \cdot M_{model} \cdot v \]
from lecture 2:

\[ v_{\text{clip}} = M_{\text{proj}} \cdot M_{\text{view}} \cdot M_{\text{model}} \cdot v \]

vertex in clip space

projection matrix  view matrix  model matrix

rotation  translation

\[ M_{\text{view}} = R \cdot T(-\text{eye}) \]
Euler angles

\[ M_{\text{view}} = R \cdot T(-\text{eye}) \]

\[ R = R_z(-\theta_z) \cdot R_x(-\theta_x) \cdot R_y(-\theta_y) \]

- roll
- pitch
- yaw
2 important coordinate systems:

\[
M_{\text{view}} = R \cdot T(-\text{eye})
\]

\[
R = R_z(-\theta_z) \cdot R_x(-\theta_x) \cdot R_y(-\theta_y)
\]

Euler angles

\(
\theta_y, \theta_z, \theta_x
\)

\(\text{roll}, \text{pitch}, \text{yaw}\)
Gyro Integration aka *Dead Reckoning*

- from gyro measurements to orientation – use Taylor expansion

\[
\theta(t + \Delta t) \approx \theta(t) + \frac{\partial \theta(t)}{\partial t} \Delta t + \varepsilon, \quad \varepsilon \sim O(\Delta t^2)
\]

- have: angle at last time step
- have: time step
- want: angle at current time step
- have: gyro measurement (angular velocity)
- approximation error!
Orientation Tracking in *Flatland*

- problem: track 1 angle in 2D space
- sensors: 1 gyro, 2-axis accelerometer
- sensor fusion with complementary filter, i.e. linear interpolation:

\[
\theta^{(t)} = \alpha \left( \theta^{(t-1)} + \tilde{\omega} \Delta t \right) + (1 - \alpha) \text{atan2} \left( \tilde{a}_x, \tilde{a}_y \right)
\]

- no drift, no noise!
Tilt from Accelerometer

- assuming acceleration points up (i.e. no external forces), we can compute the tilt (i.e. pitch and roll) from a 3-axis accelerometer

\[
\hat{\alpha} = \frac{\tilde{\alpha}}{||\tilde{\alpha}||} = R \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = R_z (-\theta_z) \cdot R_x (-\theta_x) \cdot R_y (-\theta_y) \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}
\]

\[
\begin{pmatrix}
-\cos(-\theta_x)\sin(-\theta_z) \\
\cos(-\theta_x)\cos(-\theta_z) \\
\sin(-\theta_x)
\end{pmatrix}
\]

\[
\theta_x = -\text{atan2}(\hat{\alpha}_z, \text{sign}(\hat{\alpha}_y) \cdot \sqrt{\hat{\alpha}_x^2 + \hat{\alpha}_y^2})
\]

\[
\theta_z = -\text{atan2}(-\hat{\alpha}_x, \hat{\alpha}_y) \text{ both in rad}
\]
Euler Angles and Gimbal Lock

• so far we have represented head rotations with Euler angles: 3 rotation angles around the axis applied in a specific sequence

• problematic when interpolating between rotations in keyframes (in computer animation) or integration $\rightarrow$ singularities
Gimbal Lock

The Guerrilla CG Project, The Euler (gimbal lock) Explained – see: https://www.youtube.com/watch?v=zC8b2Jo7mno
Rotations with Axis-Angle Representation and Quaternions
Rotations with Axis and Angle Representation

• solution to gimbal lock: use axis and angle representation for rotation!

• simultaneous rotation around a normalized vector $\mathbf{v}$ by angle $\theta$

• no “order” of rotation, all at once around that vector
Quaternions

- think about quaternions as an extension of complex numbers to having 3 (different) imaginary numbers or fundamental quaternion units $i, j, k$

\[
q = q_w + iq_x + jq_y + kq_z
\]

\[
i^2 = j^2 = k^2 = ijk = -1 \quad \text{and} \quad ij = -ji = k \quad \text{and} \quad ki = -ik = j \quad \text{and} \quad jk = -kj = i
\]

\[
i \neq j \neq k
\]
Quaternions

• think about quaternions as an extension of complex numbers to having 3 (different) imaginary numbers or fundamental quaternion units $i, j, k$

$$q = q_w + iq_x + jq_y + kq_z$$

• quaternion algebra is well-defined and will give us a powerful tool to work with rotations in axis-angle representation in practice
Quaternions

- axis-angle to quaternion (need normalized axis $\mathbf{v}$)

$$q(\theta, \mathbf{v}) = \cos\left(\frac{\theta}{2}\right) + i v_x \sin\left(\frac{\theta}{2}\right) + j v_y \sin\left(\frac{\theta}{2}\right) + k v_z \sin\left(\frac{\theta}{2}\right)$$
Quaternions

- axis-angle to quaternion (need normalized axis \( \nu \))

\[
q(\theta, \nu) = \cos\left(\frac{\theta}{2}\right) + i\nu_x \sin\left(\frac{\theta}{2}\right) + j\nu_y \sin\left(\frac{\theta}{2}\right) + k\nu_z \sin\left(\frac{\theta}{2}\right)
\]

- valid rotation quaternions have unit length

\[
\|q\| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} = 1
\]
Two Types of Quaternions

- **vector quaternions** represent 3D points or vectors \( \mathbf{u} = (u_x, u_y, u_z) \) can have arbitrary length

\[
q_u = 0 + iu_x + ju_y + ku_z
\]

- **valid rotation quaternions** have unit length

\[
\|q\| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} = 1
\]
Quaternion Algebra

- quaternion addition:
  \[ q + p = (q_w + p_w) + i(q_x + p_x) + j(q_y + p_y) + k(q_z + p_z) \]

- quaternion multiplication:
  \[ qp = (q_w + iq_x + jq_y + kq_z)(p_w + ip_x + jp_y + kp_z) \]
  \[ = (q_wp_w - qxpx - qypy - q_zpz) + i(q_wp_x + qxpw + qypz - q_zpy) + j(q_wp_y - qxpz + qypw + q_zpx) + k(q_wp_z + qxpy - q_ypx + q_zpw) \]
Quaternion Algebra

- quaternion conjugate: \( q^* = q_w - iq_x - jq_y - kq_z \)
- quaternion inverse: \( q^{-1} = \frac{q^*}{||q||^2} \)
- rotation of vector quaternion \( q_u \) by \( q \): \( q'_u = qq_uq^{-1} \)
- inverse rotation:
- successive rotations by \( q_1 \) then \( q_2 \): \( q'_u = q_2 q_1 q_u q_1^{-1} q_2^{-1} \)
Quaternion Algebra

- detailed derivations and reference of general quaternion algebra and rotations with quaternions in course notes

- please read *course notes* for more details!
Quaternion-based
6-DOF Orientation Tracking
Quaternelion-based Orientation Tracking

1. 3-axis gyro integration

2. computing the tilt correction quaternion

3. applying a complementary filter
Gyro Integration with Quaternions

• start with initial quaternion: \[ q^{(0)} = 1 + i0 + j0 + k0 \]

• convert 3-axis gyro measurements \( \tilde{\omega} = (\tilde{\omega}_x, \tilde{\omega}_y, \tilde{\omega}_z) \) to instantaneous rotation quaternion as

\[ q_\Delta = q \left( \Delta t \| \tilde{\omega} \|, \frac{\tilde{\omega}}{\| \tilde{\omega} \|} \right) \]

\( \text{angle} \) \hspace{1cm} \( \text{rotation axis} \)

• integrate as

\[ q_{\omega}^{(t+\Delta t)} = q^{(t)} q_\Delta \]
Gyro Integration with Quaternions

- integrated gyro rotation quaternion $q_{\omega}^{(t+\Delta t)}$ represents rotation from body to world frame, i.e.

$$q_{(world)} = q_{\omega}^{(t+\Delta t)} q_{u}^{(body)} q_{\omega}^{(t+\Delta t)\^{-1}}$$

- last estimate $q^{(t)}$ is either from gyro-only (for dead reckoning) or from last complementary filter

- integrate as

$$q_{\omega}^{(t+\Delta t)} = q^{(t)} q_{\Delta}$$
Tilt Correction with Quaternions

• assume accelerometer measures gravity vector in body (sensor) coordinates

\[ \vec{a} = \left( \vec{a}_x, \vec{a}_y, \vec{a}_z \right) \]

• transform vector quaternion of \( \vec{a} \) into current estimation of world space as

\[
q_a^{(\text{world})} = q_{\omega}^{(t+\Delta t)} q_a^{(\text{body})} q_{\omega}^{(t+\Delta t)^{-1}}
\]

\[
q_a^{(\text{body})} = 0 + i\vec{a}_x + j\vec{a}_y + k\vec{a}_z
\]
Tilt Correction with Quaternions

• assume accelerometer measures gravity vector in body (sensor) coordinates

\[ \tilde{a} = (\tilde{a}_x, \tilde{a}_y, \tilde{a}_z) \]

• transform vector quaternion of \( \tilde{a} \) into current estimation of world space as

\[ q_a^{(\text{world})} = q_{\omega}^{(t+\Delta t)} q_a^{(\text{body})} q_{\omega}^{(t+\Delta t)^{-1}} \]

• if gyro quaternion is correct, then accelerometer world vector points up, i.e.

\[ q_a^{(\text{world})} = 0 + i0 + j9.81 + k0 \]
Tilt Correction with Quaternions

- gyro quaternion likely includes drift
- accelerometer measurements are noisy and also include forces other than gravity, so it’s unlikely that accelerometer world vector actually points up

- if gyro quaternion is correct, then accelerometer world vector points up, i.e.
  \[ q_{a}^{(\text{world})} = 0 + i0 + j9.81 + k0 \]
Tilt Correction with Quaternions

solution: compute tilt correction quaternion that would rotate $q^{(\text{world})}_a$ into up direction

how? get normalized vector part of vector quaternion $q^{(\text{world})}_a$

$$v = \begin{pmatrix} \frac{q_a^{(\text{world})}_{a_x}}{||q_a^{(\text{world})}||}, \frac{q_a^{(\text{world})}_{a_y}}{||q_a^{(\text{world})}||}, \frac{q_a^{(\text{world})}_{a_z}}{||q_a^{(\text{world})}||} \end{pmatrix}$$
Tilt Correction with Quaternions

solution: compute tilt correction quaternion that would rotate $q_a^{(\text{world})}$ into up direction

$$q_t = q\left(\phi, \frac{n}{||n||}\right)$$

$$\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \cos(\phi) \quad \Rightarrow \quad \phi = \cos^{-1}(v_y)$$

$$n = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -v_z \\ 0 \\ v_x \end{pmatrix}$$
Complementary Filter with Quaternions

- complementary filter: rotate into gyro world space first, then rotate “a bit” into the direction of the tilt correction quaternion

\[ q^{(t+\Delta t)}_c = q \left( (1 - \alpha) \phi, \frac{n}{\|n\|} \right) q^{(t+\Delta t)}_\omega \quad 0 \leq \alpha \leq 1 \]

- rotation of any vector quaternion is then

\[ q^{(\text{world})}_u = q^{(t+\Delta t)}_c q^{(\text{body})}_u q^{(t+\Delta t)^{-1}}_c \]
Integration into Graphics Pipeline

- compute $q_{c}^{(t+\Delta t)}$ via quaternion complementary filter first
- stream from microcontroller to PC
- convert to 4x4 rotation matrix (see course notes) $q_{c}^{(t+\Delta t)} \Rightarrow R_{c}$
- set view matrix to $M_{\text{view}} = R_{c}^{-1}$ to rotate the world in front of the virtual camera
Head and Neck Model

pitch around base of neck!

roll around base of neck!
Head and Neck Model

• why? there is not always positional tracking! this gives some motion parallax

• can extend to torso, and using other kinematic constraints

• integrate into pipeline as

\[
M_{\text{view}} = T(0, -l_n, -l_h) \cdot R \cdot T(0, l_n, l_h) \cdot T(-\text{eye})
\]
Must read: course notes on IMUs!