Inertial Measurement Units II

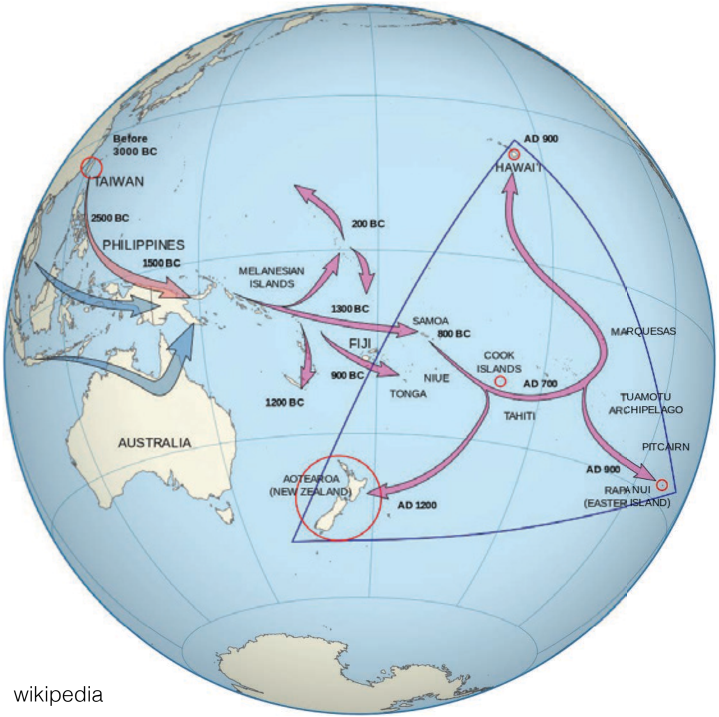
Gordon Wetzstein
Stanford University

EE 267 Virtual Reality

Lecture 10

stanford.edu/class/ee267/

Polynesian Migration

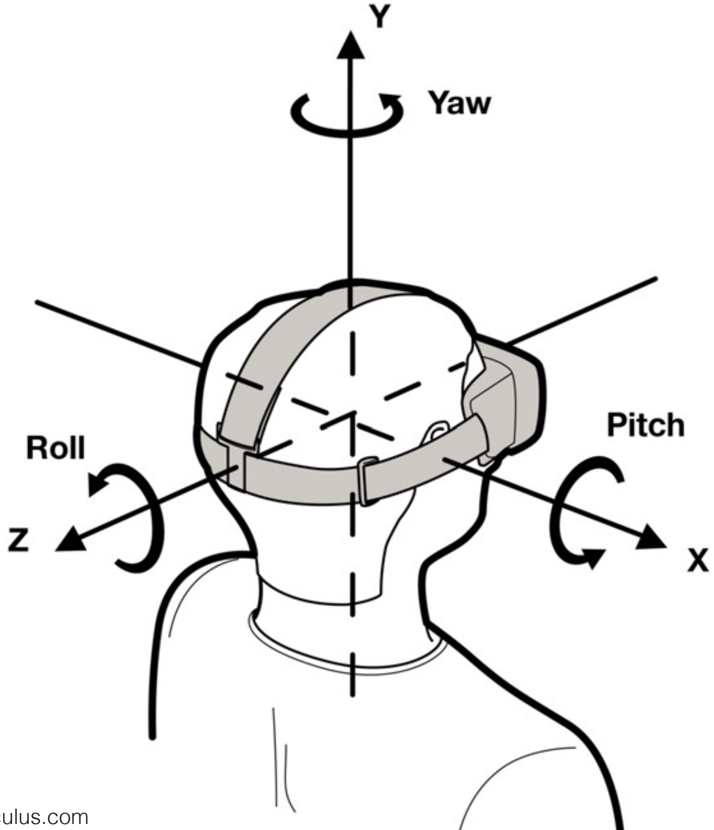


wikipedia

Lecture Overview

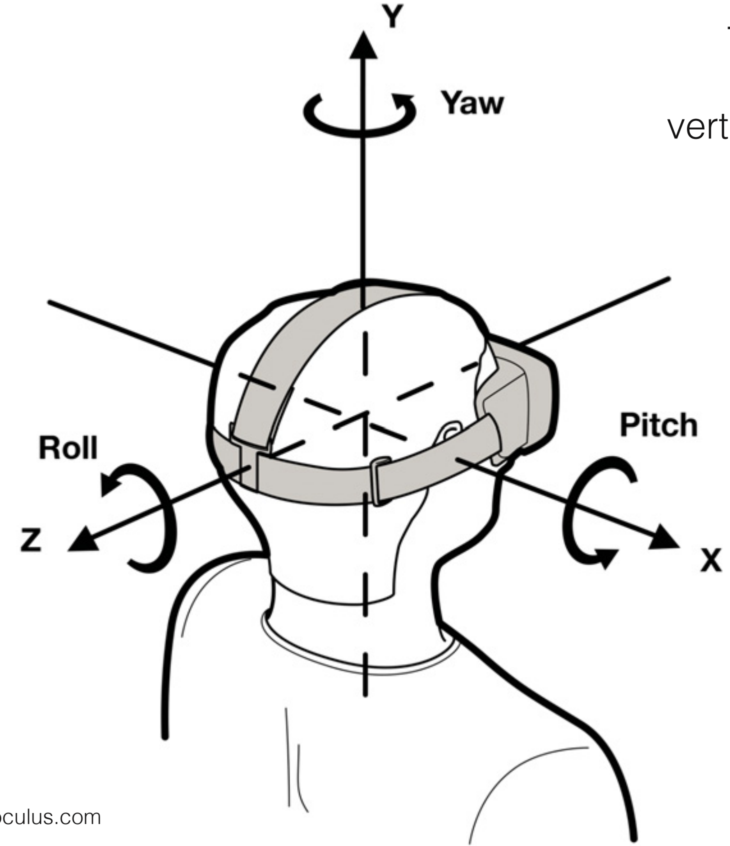
- short review of coordinate systems, tracking in flatland, and accelerometer-only tracking

- rotations: Euler angles, axis & angle, gimbal lock
- rotations with quaternions
- 6-DOF IMU sensor fusion with quaternions



oculus.com

- primary goal: track orientation of head or device
- inertial sensors required pitch, yaw, and roll to be determined



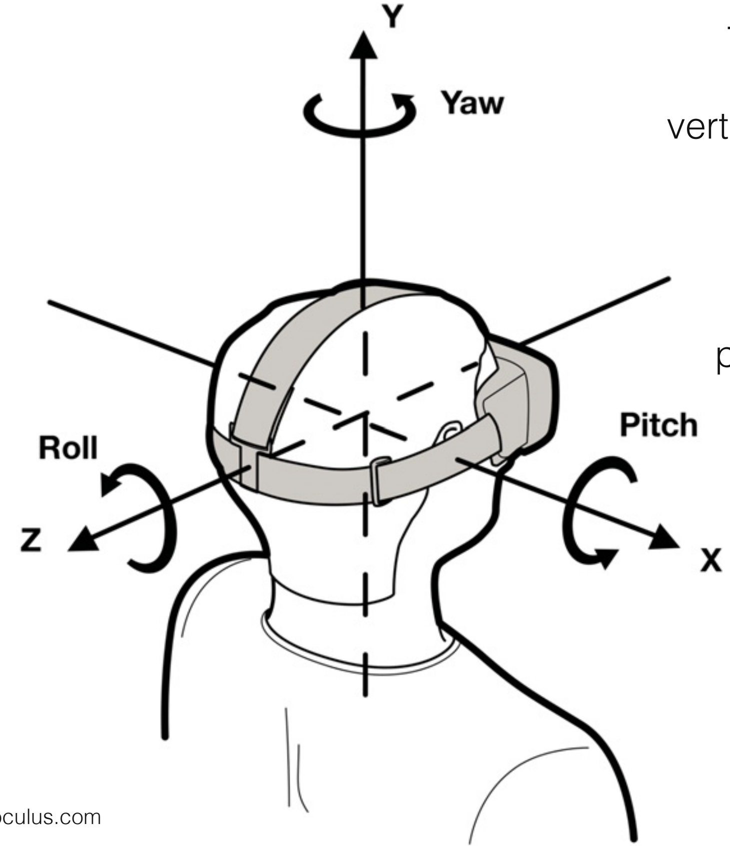
oculus.com

from lecture 2:

vertex in clip space

vertex

$$v_{clip} = M_{proj} \cdot M_{view} \cdot M_{model} \cdot v$$



oculus.com

from lecture 2:

vertex in clip space

vertex

$$v_{clip} = M_{proj} \cdot M_{view} \cdot M_{model} \cdot v$$

projection matrix

view matrix

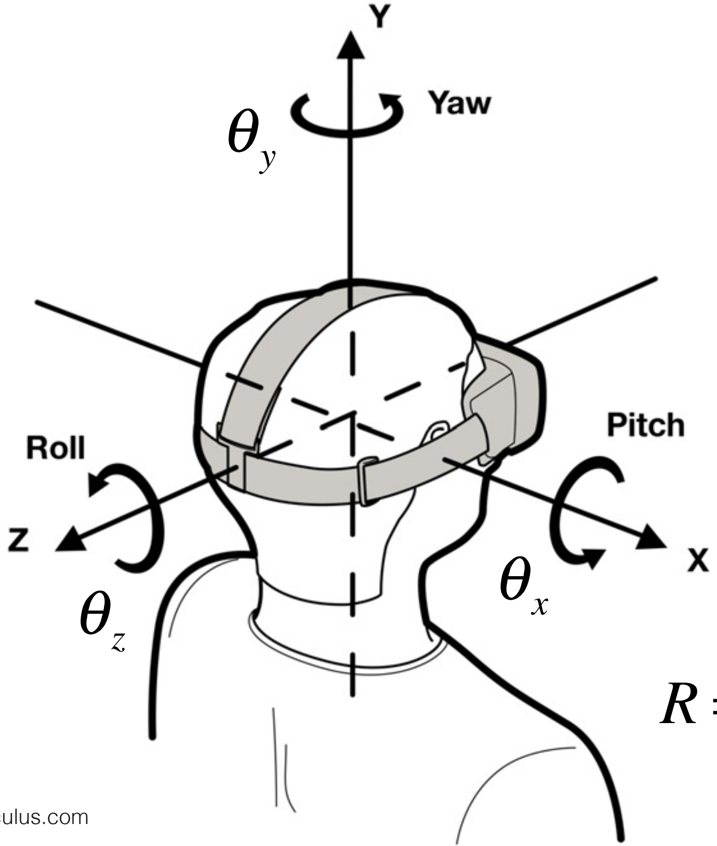
model matrix

rotation

translation

$$M_{view} = R \cdot T(-eye)$$

Euler angles



oculus.com

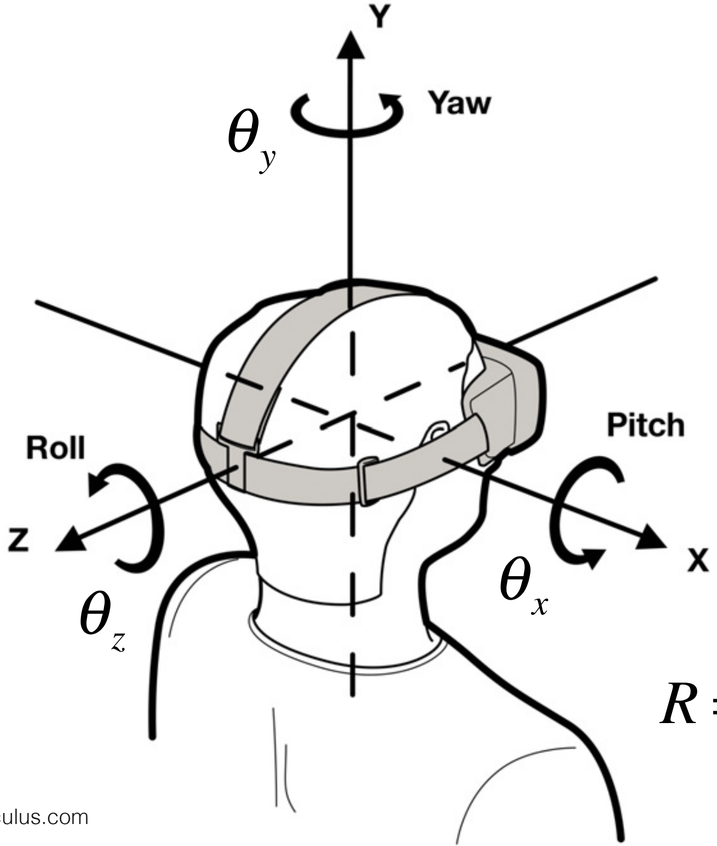
rotation translation

$$M_{view} = R \cdot T(-eye)$$

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

roll pitch yaw

Euler angles



oculus.com

2 important coordinate systems:

<u>body/sensor frame</u> <u>world/inertial frame</u>

$$M_{view} = R \cdot T(-eye)$$

$$R = \underbrace{R_z(-\theta_z)}_{\text{roll}} \cdot \underbrace{R_x(-\theta_x)}_{\text{pitch}} \cdot \underbrace{R_y(-\theta_y)}_{\text{yaw}}$$

Gyro Integration aka *Dead Reckoning*

- from gyro measurements to orientation – use Taylor expansion

<u>have</u>: angle at last time step

<u>have</u>: time step

$$\theta(t+\Delta t) \approx \theta(t) + \frac{\partial}{\partial t}\theta(t)\Delta t + \varepsilon, \quad \varepsilon \sim O\left(\Delta t^2\right)$$

<u>want</u>: angle at current time step

approximation error!

$$= \omega$$

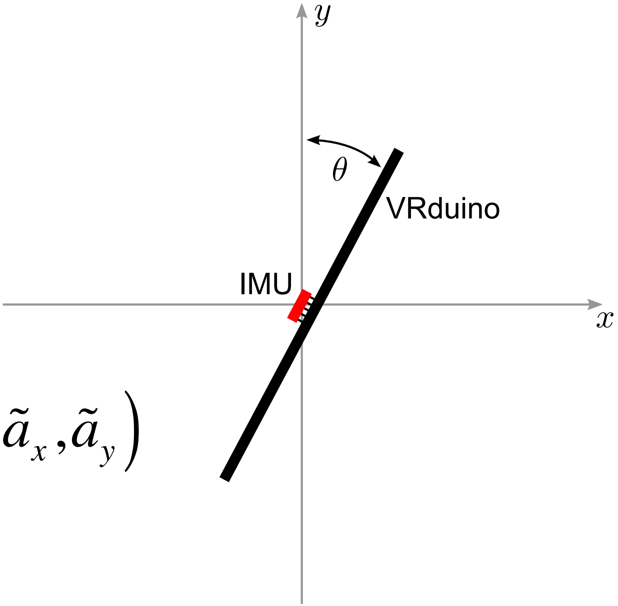
<u>have</u>: gyro measurement (angular velocity)

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:

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

- 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

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

$$= \begin{pmatrix} -\cos(-\theta_x)\sin(-\theta_z) \\ \cos(-\theta_x)\cos(-\theta_z) \\ \sin(-\theta_x) \end{pmatrix} \Rightarrow$$

$$\theta_x = -\text{atan2}\left(\hat{a}_z, \text{sign}(\hat{a}_y)\cdot\sqrt{\hat{a}_x^2 + \hat{a}_y^2}\right)$$

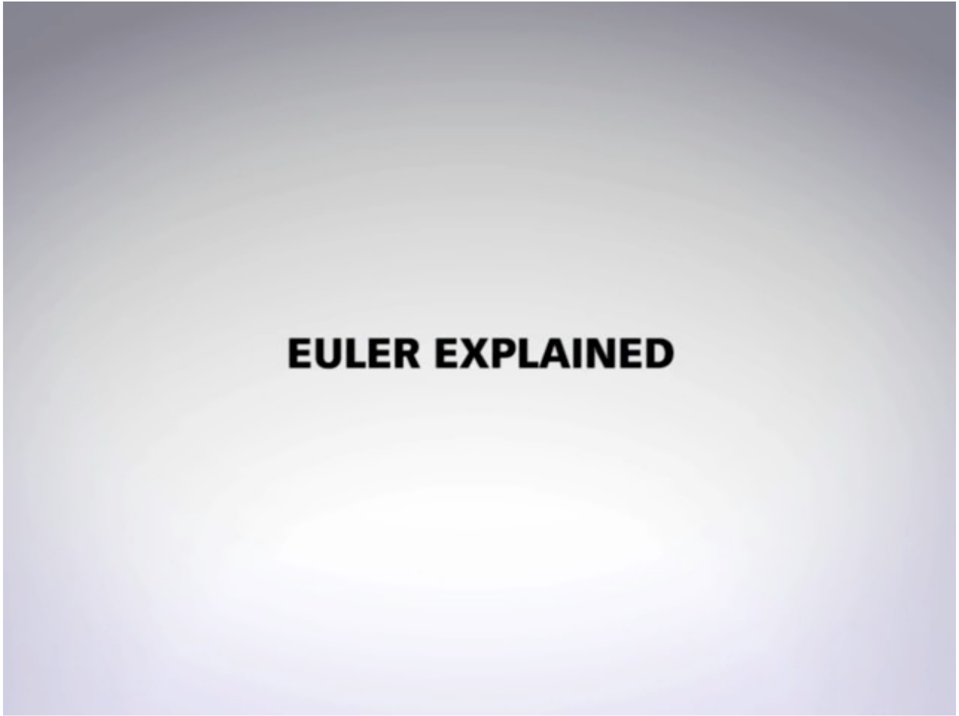
$$\theta_z = -\text{atan2}(-\hat{a}_x, \hat{a}_y) \text{ both in rad}$$

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 → singularities

Gimbal Lock

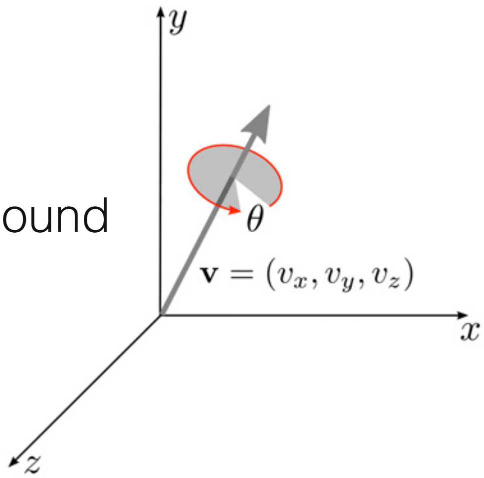


The Guerrilla CG Project, The Euler (gimbal lock) Explained – see: https://www.youtube.com/watch?v=zc8b2Jo7mno

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 v by angle θ
- 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*

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

$$ij = -ji = k$$

$$i \neq j \neq k$$

$$ki = -ik = j$$

$$i^2 = j^2 = k^2 = ijk = -1$$

$$jk = -kj = i$$

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 *v*)

$$q(\theta,v) = \underbrace{\cos\left(\frac{\theta}{2}\right)}_{q_w} + i\,\underbrace{v_x \sin\left(\frac{\theta}{2}\right)}_{q_x} + j\,\underbrace{v_y \sin\left(\frac{\theta}{2}\right)}_{q_y} + k\,\underbrace{v_z \sin\left(\frac{\theta}{2}\right)}_{q_z}$$

Quaternions

- axis-angle to quaternion (need normalized axis *v*)

$$q(\theta,v) = \underbrace{\cos\left(\frac{\theta}{2}\right)}_{q_w} + i\,\underbrace{v_x \sin\left(\frac{\theta}{2}\right)}_{q_x} + j\,\underbrace{v_y \sin\left(\frac{\theta}{2}\right)}_{q_y} + k\,\underbrace{v_z \sin\left(\frac{\theta}{2}\right)}_{q_z}$$

- 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

- <u>vector quaternions</u> represent 3D points or vectors $\boldsymbol{u}=(u_x,u_y,u_z)$ can have arbitrary length

$$q_u = 0 + i\,u_x + j\,u_y + k\,u_z$$

- valid <u>rotation quaternions</u> 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:

$$\begin{aligned} qp &= (q_w + iq_x + jq_y + kq_z)(p_w + ip_x + jp_y + kp_z) \\ &= (q_w p_w - q_x p_x - q_y p_y - q_z p_z) + \\ &\quad i(q_w p_x + q_x p_w + q_y p_z - q_z p_y) + \\ &\quad j(q_w p_y - q_x p_z + q_y p_w + q_z p_x) + \\ &\quad k(q_w p_z + q_x p_y - q_y p_x + q_z p_w) + \end{aligned}$$

Quaternion Algebra

- quaternion conjugate: $q^* = q_w - iq_x - jq_y - kq_z$

- quaternion inverse: $q^{-1} = \dfrac{q^*}{\|q\|^2}$

- rotation of vector quaternion q_u by q : $q'_u = q q_u q^{-1}$
- inverse rotation: $q_u = q^{-1} q'_u q$

- 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

Quaternion-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 $\tilde{\omega} = \left(\tilde{\omega}_x, \tilde{\omega}_y, \tilde{\omega}_z\right)$ to instantaneous rotation quaternion as

avoid division by 0!

$$q_\Delta = q\left(\underbrace{\Delta t \|\tilde{\omega}\|}_{\text{angle}}, \underbrace{\frac{\tilde{\omega}}{\|\tilde{\omega}\|}}_{\text{rotation axis}} \right)$$

- 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.

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

- 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 $\tilde{a} = \left(\tilde{a}_x, \tilde{a}_y, \tilde{a}_z\right)$

- transform vector quaternion of $\tilde{a}$ into current estimation of world space as

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

$$q_a^{(body)} = 0 + i\tilde{a}_x + j\tilde{a}_y + k\tilde{a}_z$$

Tilt Correction with Quaternions

- assume accelerometer measures gravity vector in body (sensor) coordinates $\tilde{a} = \left(\tilde{a}_x, \tilde{a}_y, \tilde{a}_z\right)$

- 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 $q_a^{(\text{world})}$ into up direction

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

$$v = \left(\frac{q_{a_x}^{(\text{world})}}{\left\| q_a^{(\text{world})} \right\|}, \frac{q_{a_y}^{(\text{world})}}{\left\| q_a^{(\text{world})} \right\|}, \frac{q_{a_z}^{(\text{world})}}{\left\| q_a^{(\text{world})} \right\|} \right)$$

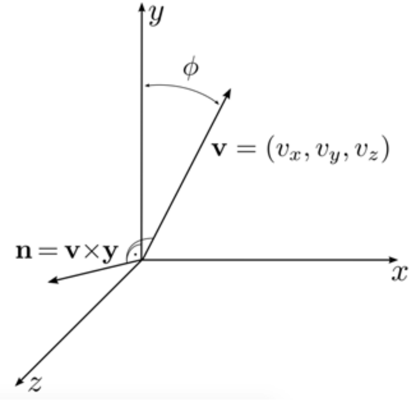
Tilt Correction with Quaternions

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

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

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

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



Complementary Filter with Quaternions

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

$$q_c^{(t+\Delta t)} = q\left((1-\alpha)\phi, \frac{n}{\|n\|} \right) q_\omega^{(t+\Delta t)} \qquad 0 \le \alpha \le 1$$

- rotation of any vector quaternion is then $q_u^{(\mathrm{world})} = q_c^{(t+\Delta t)} q_u^{(\mathrm{body})} {q_c^{(t+\Delta t)}}^{-1}$

Integration into Graphics Pipeline

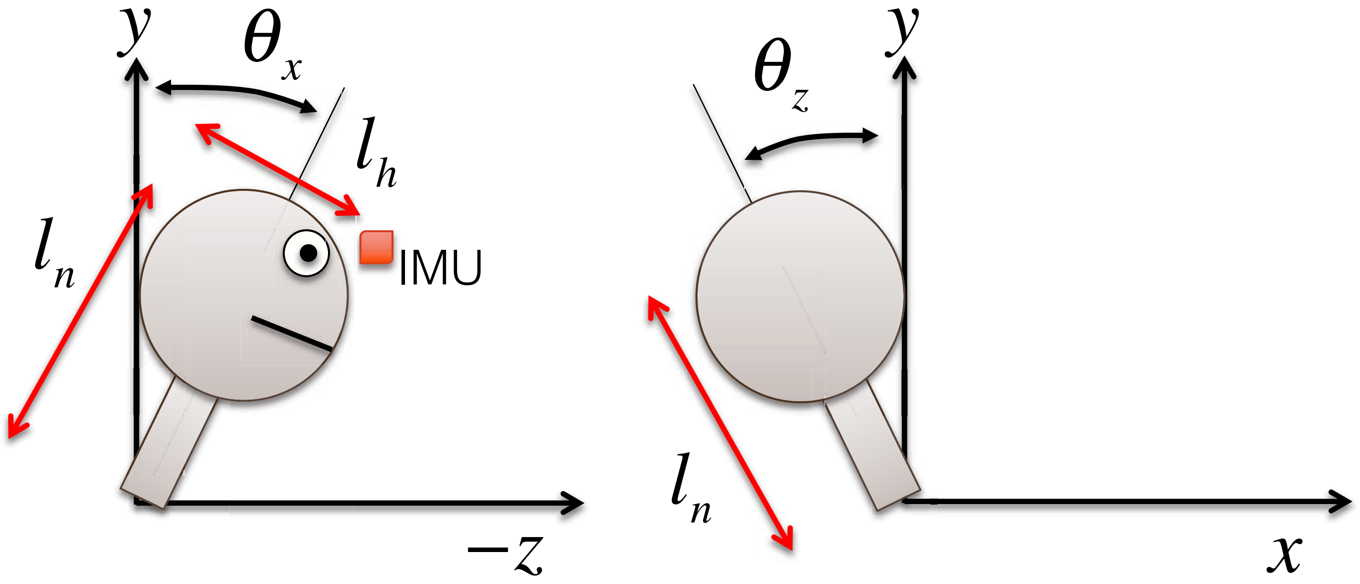
- compute $q_c^{(t+\Delta t)}$ via quaternion complementary filter first

- stream from microcontroller to PC

- convert to 4x4 rotation matrix (see course notes) $\quad q_c^{(t+\Delta t)} \Rightarrow R_c$

- set view matrix to $M_{view} = R_c^{-1}$ 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

$$M_{view} = T\left(0, -l_n, -l_h\right) \cdot R \cdot T\left(0, l_n, l_h\right) \cdot T\left(-eye\right)$$

Must read: course notes on IMUs!