

Inertial Measurement Units I



Gordon Wetzstein
Stanford University

EE 267 Virtual Reality

Lecture 9

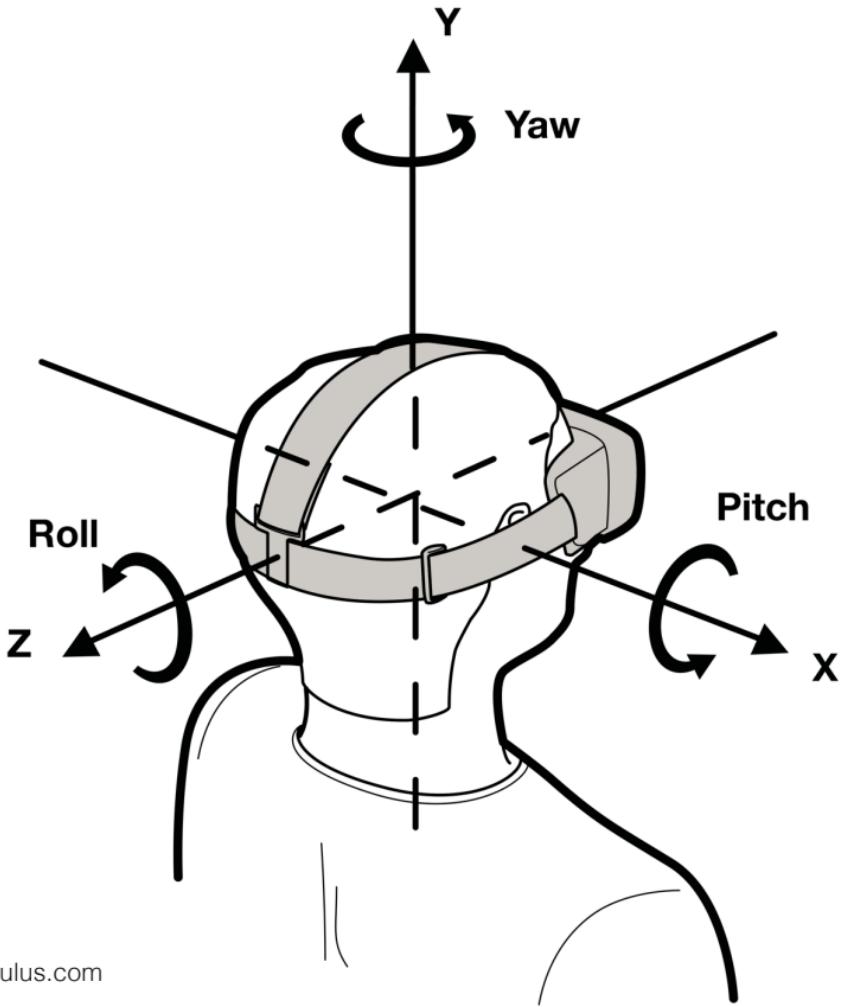
stanford.edu/class/ee267/

Logistics

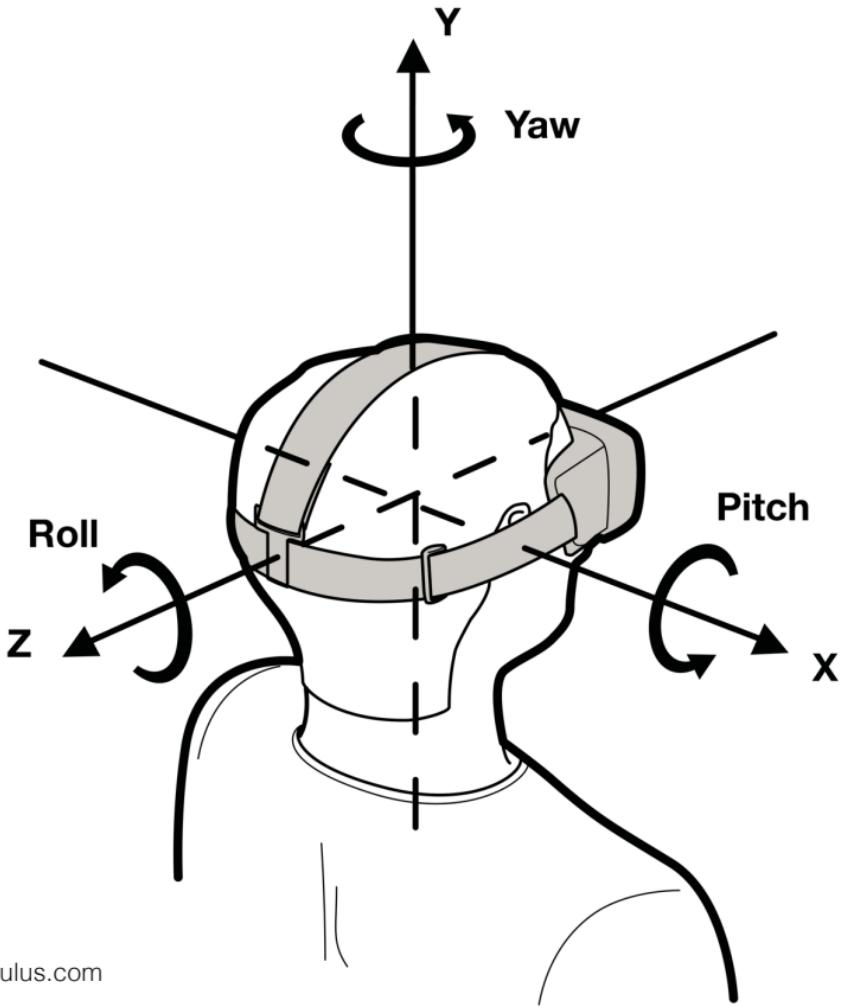
- Project proposal due this Fri

Lecture Overview

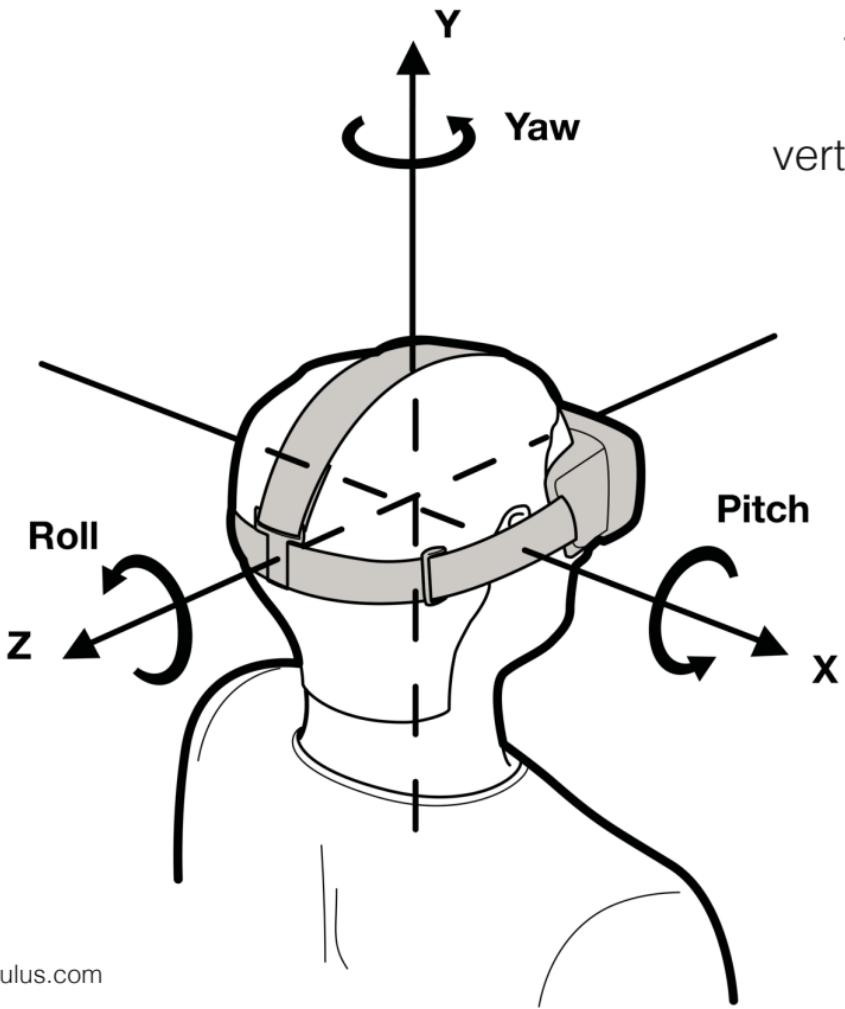
- coordinate systems (world, body/sensor, inertial, transforms)
- overview of inertial sensors: gyroscopes, accelerometers, and magnetometers
- gyro integration aka *dead reckoning*
- orientation tracking in *flatland*
- pitch & roll from accelerometer
- overview of VRduino



- primary goal: track orientation of head or other device
- orientation is the rotation of device w.r.t. world/earth or *inertial* frame
- rotations are represented by Euler angles (yaw, pitch, roll) or quaternions



- orientation tracked with IMU models relative rotation of sensor/body frame in world/inertial coordinates
- example: person on the left looks up → pitch=90° or rotation around x-axis by 90°
- similarly, the world rotates around the sensor frame by -90° (inverse rotation)



from lecture 2:

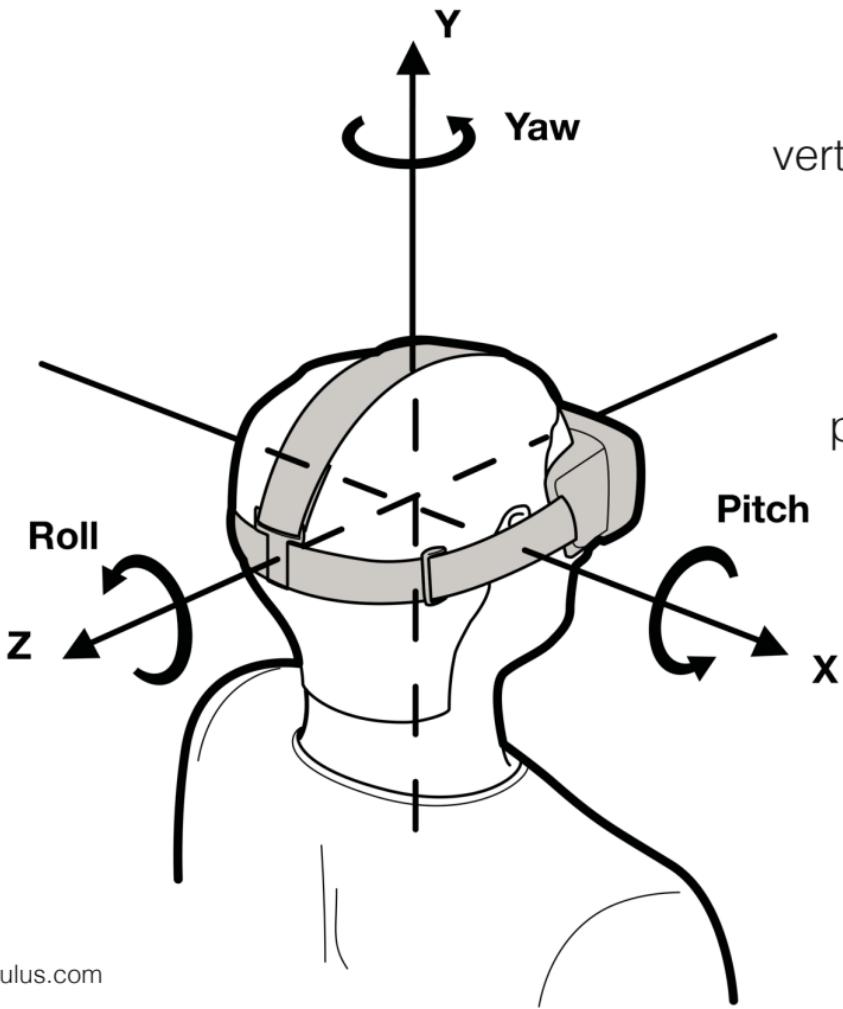
vertex in clip space



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

vertex





from lecture 2:

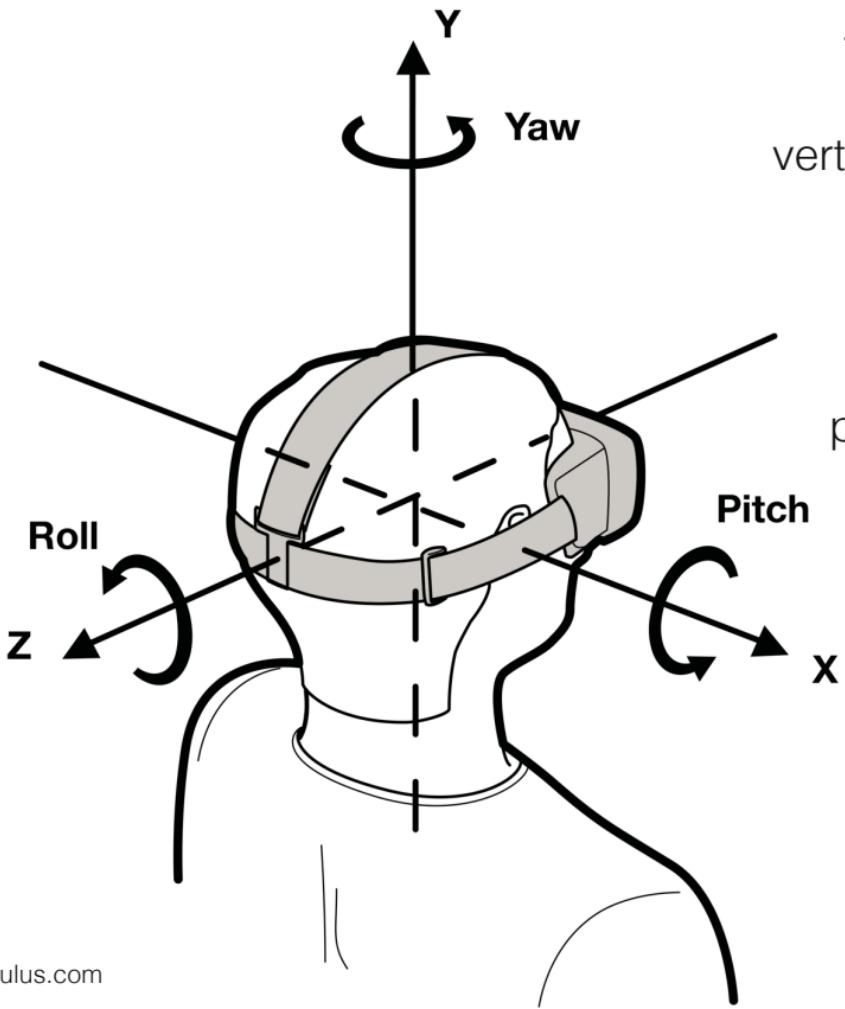
vertex in clip space

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

↓
projection matrix view matrix model matrix
↑ ↑ ↑

vertex





from lecture 2:

vertex in clip space

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

projection matrix

view matrix

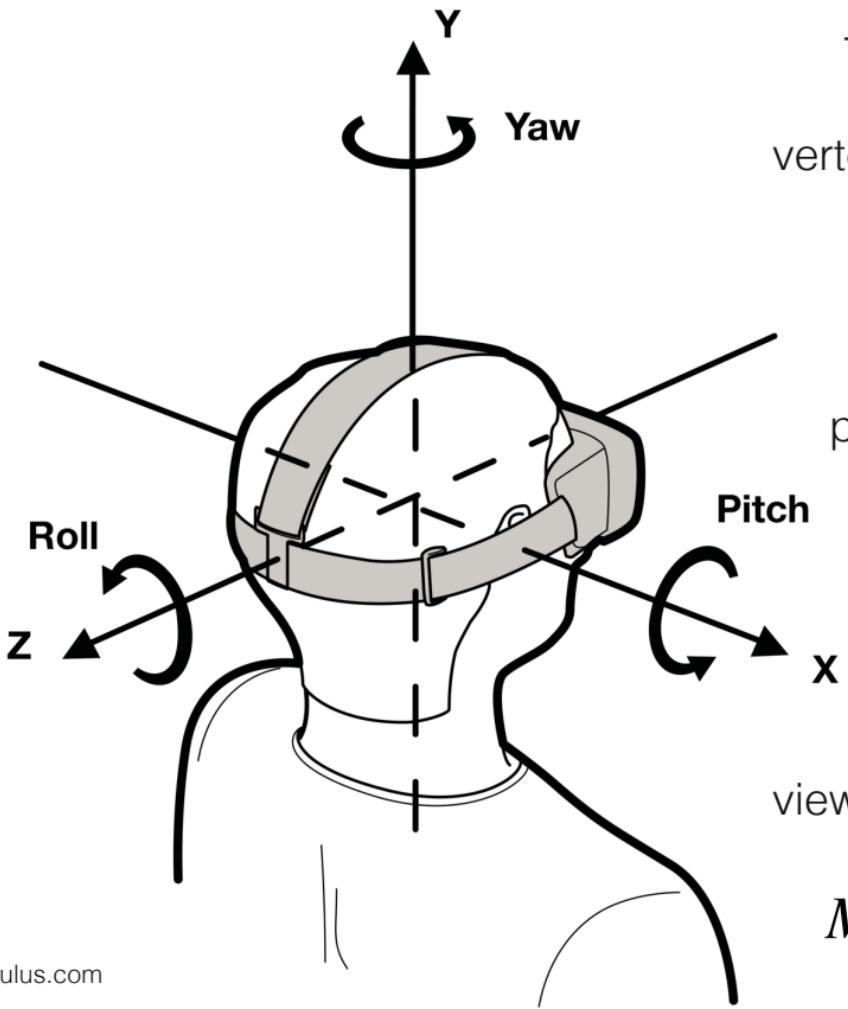
model matrix

rotation translation

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

vertex





from lecture 2:

vertex in clip space

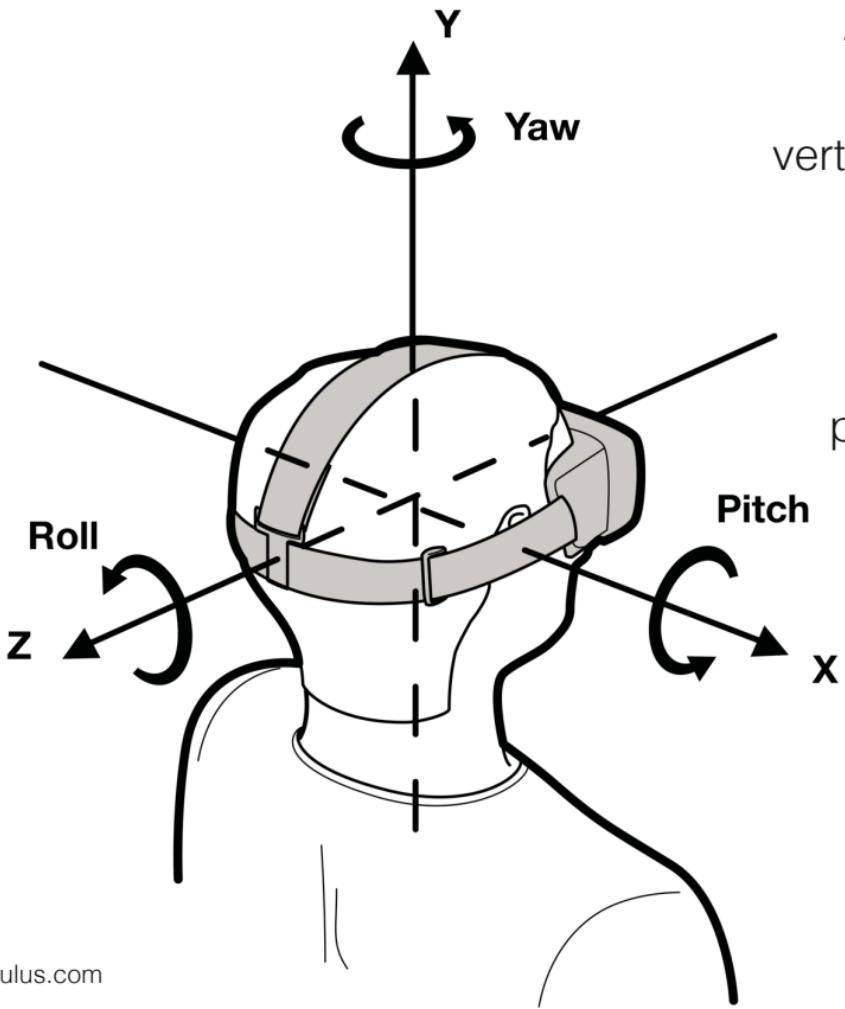
$$v_{clip} = M_{proj} \cdot M_{view} \cdot M_{model} \cdot v$$

↓ ↑ ↑ ↓
projection matrix view matrix model matrix rotation translation

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

view matrix for stereo camera:

$$M_{view}^{stereo} = T\left(\pm\frac{ipd}{2}, 0, 0\right) \cdot R \cdot T(-eye)$$



from lecture 2:

vertex in clip space

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

projection matrix

view matrix

model matrix

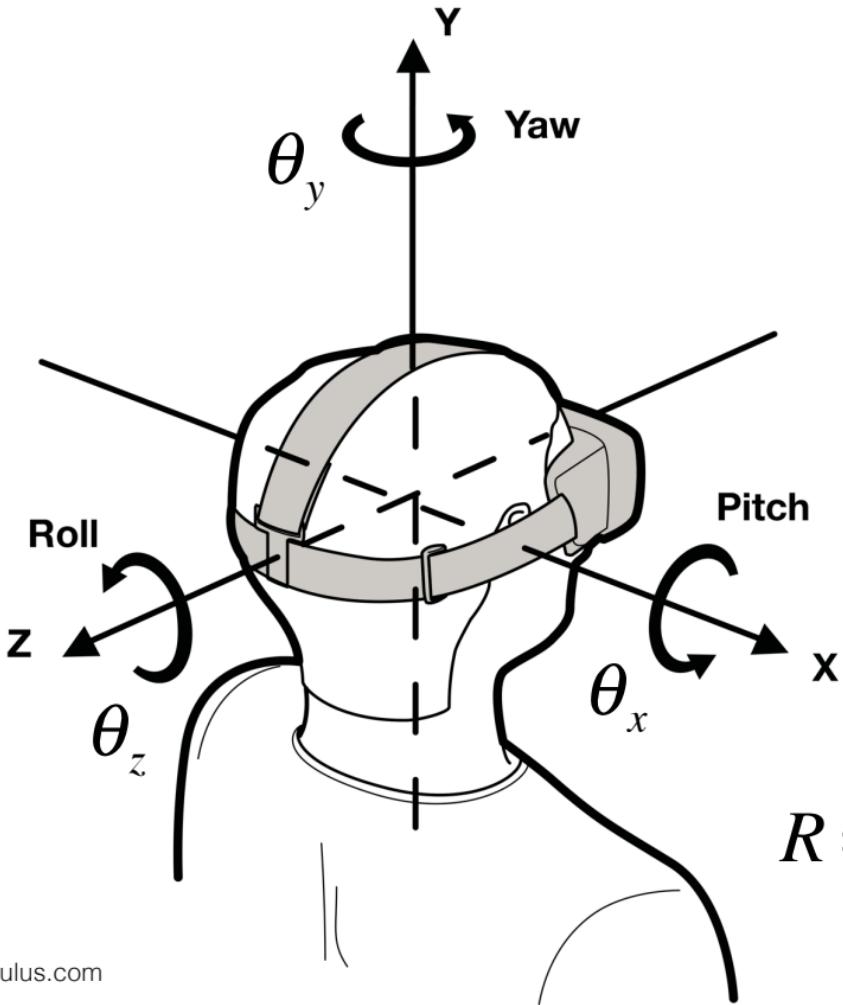
rotation translation

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

↑
↑

sensor/body
frame

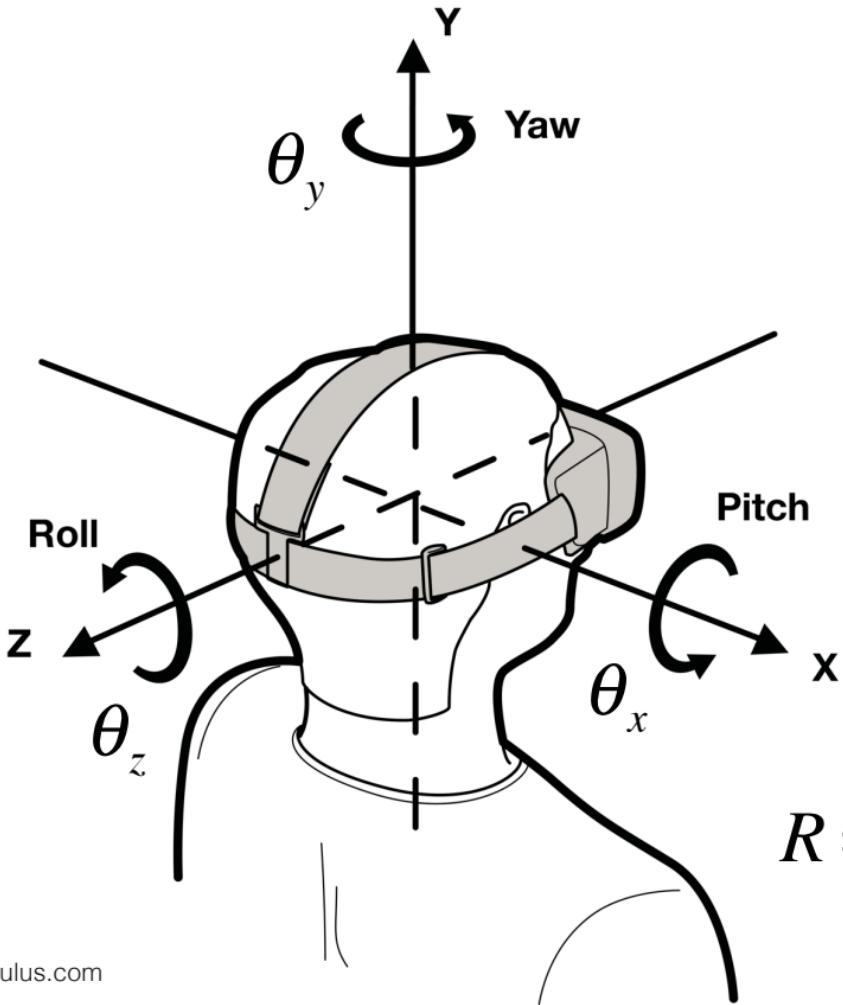
world/inertial
frame



rotation translation

$$M_{view} = R \cdot T(-eye)$$
$$R = R_z(-\theta_z) \cdot R_x(-\theta_x) \cdot R_y(-\theta_y)$$

order of rotations (world to body)



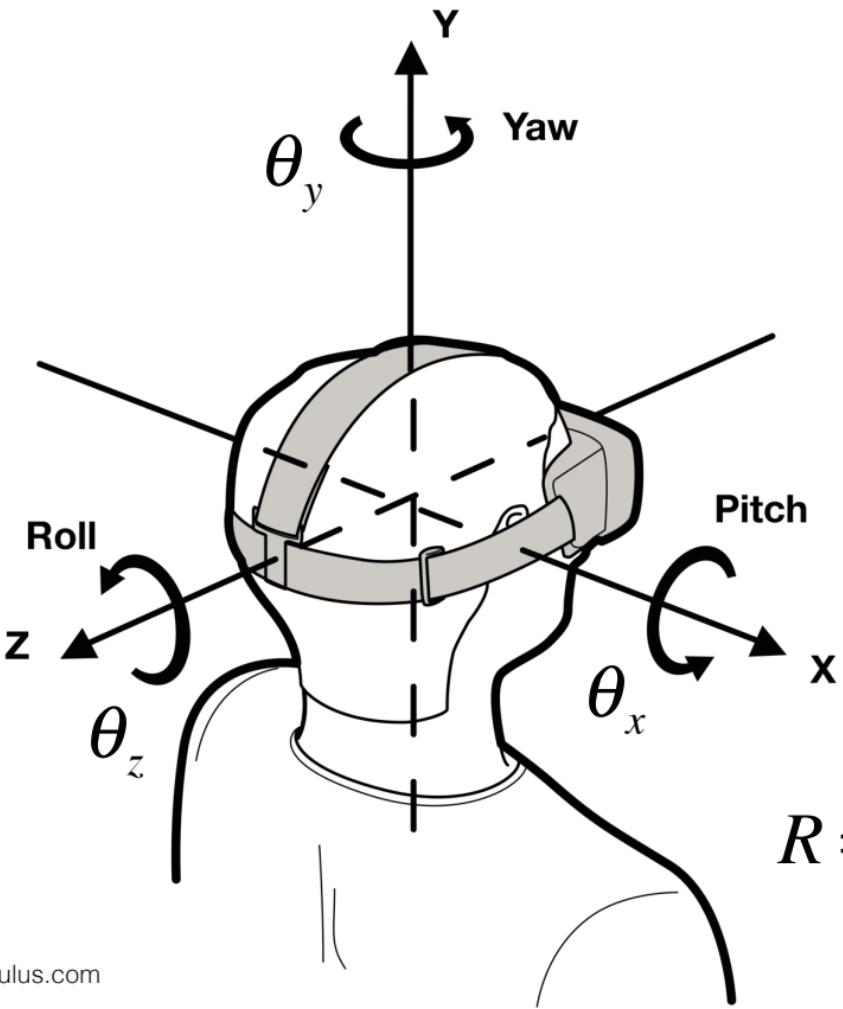
- this representation for a rotation is known as *Euler angles*

- need to specify order of rotation, e.g. yaw-pitch-roll

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

order of rotations (world to body)

ATTENTION!



- Euler angles are usually a terrible idea for orientation tracking with more than 1 axis
- one of several reasons:
rotations are not commutative

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

order of rotations (world to body)

What do Inertial Sensors Measure?

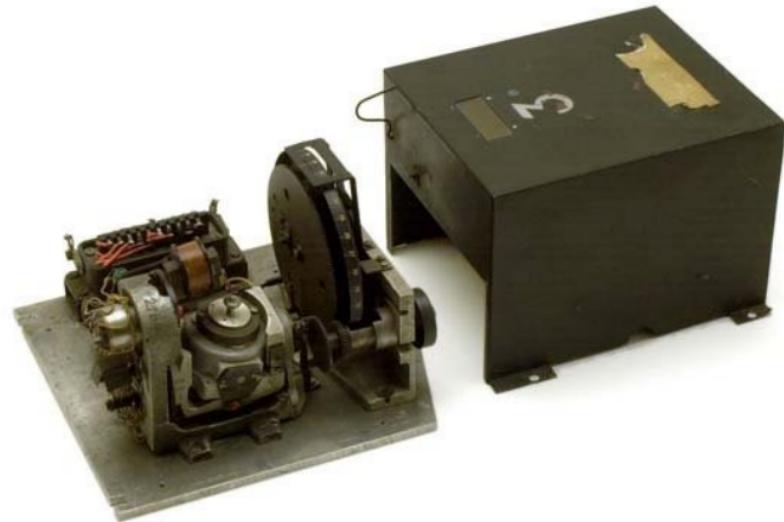
- gyroscope measures angular velocity $\tilde{\omega}$ in degrees/sec
- accelerometer measures linear acceleration \tilde{a} in m/s²
- magnetometer measures magnetic field strength \tilde{m} in uT
(micro Tesla) or Gauss → 1 Gauss = 100 uT

What do Inertial Sensors Measure?

- gyroscope measures angular velocity $\tilde{\omega}$ in degrees/sec
 - accelerometer measures linear acceleration \tilde{a} in m/s²
 - magnetometer measures magnetic field strength \tilde{m} in uT
(micro Tesla) or Gauss $\rightarrow 1$ Gauss = 100 uT
- ALL MEASUREMENTS TAKEN IN
SENSOR/BODY COORDINATES!**

History of Gyroscopes

- critical for inertial measurements in ballistic missiles, aircrafts, drones, the mars rover, pretty much anything that moves!

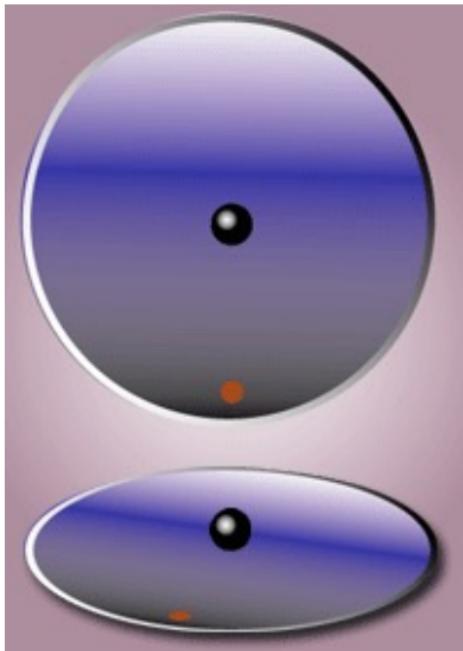


WWII era gyroscope used in the V2 rocket

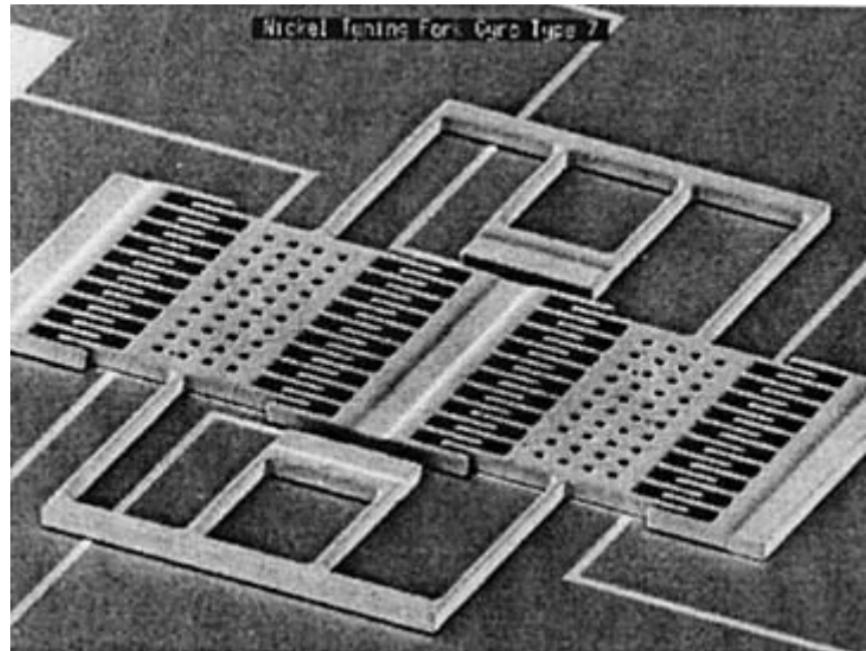
MEMS Gyroscopes

- today, we use microelectromechanical systems (MEMS)

Coriolis Force

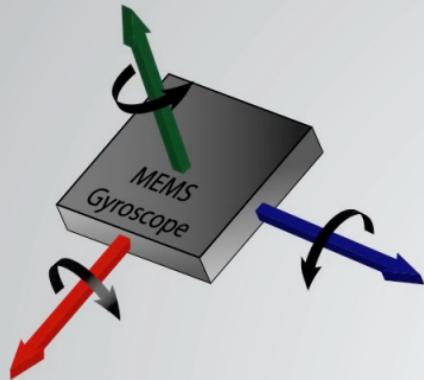


wikipedia



quora.com

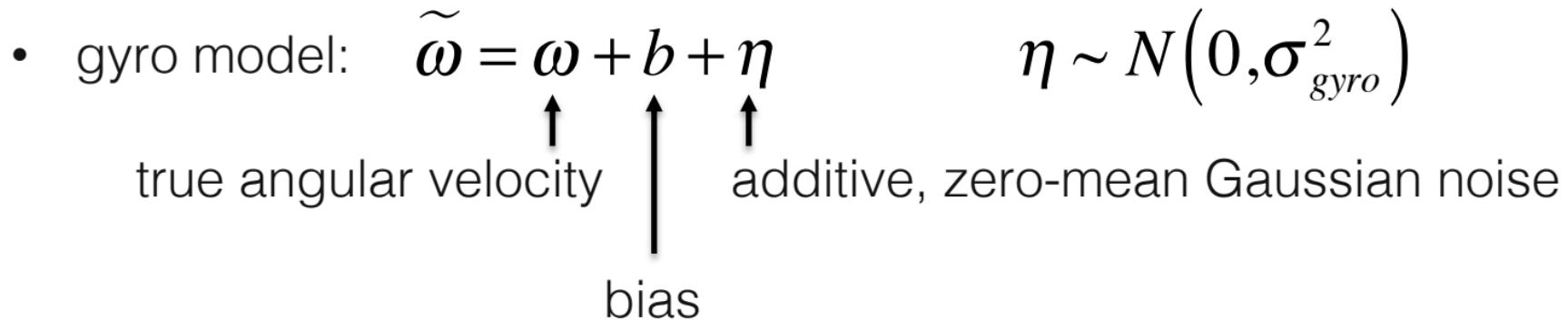
MEMS Gyroscope



Gyroscopes

- gyro model: $\tilde{\omega} = \omega + b + \eta$

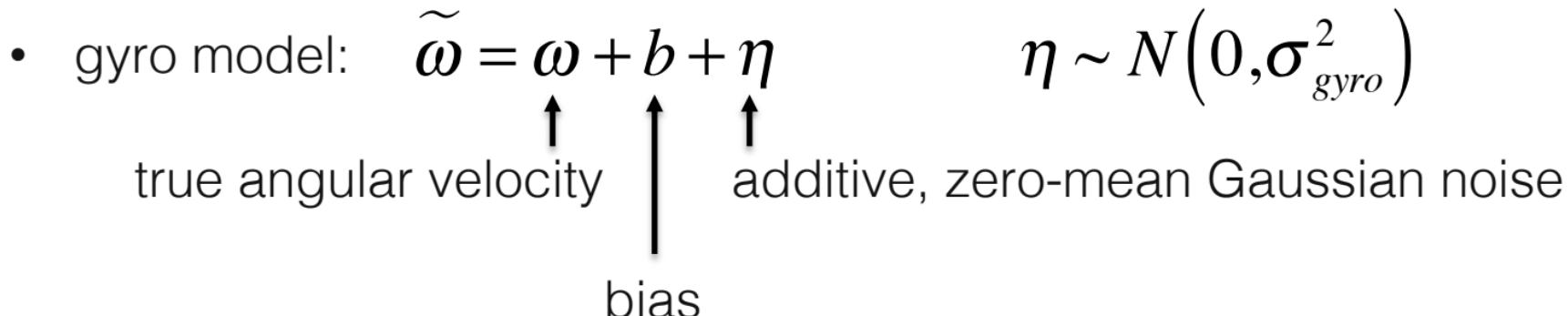
Gyroscopes

- gyro model: $\tilde{\omega} = \omega + b + \eta$ $\eta \sim N(0, \sigma_{gyro}^2)$


↑ ↑ ↑

true angular velocity bias additive, zero-mean Gaussian noise

Gyroscopes

- gyro model: $\tilde{\omega} = \omega + b + \eta$ $\eta \sim N(0, \sigma_{gyro}^2)$


The diagram illustrates the gyro model equation $\tilde{\omega} = \omega + b + \eta$. Three vertical arrows point upwards from the terms to their labels: the first arrow points to ω with the label "true angular velocity"; the second arrow points to b with the label "bias"; the third arrow points to η with the label "additive, zero-mean Gaussian noise".
- 3 DOF = 3-axis gyros that measures 3 orthogonal axes, assume no crosstalk
- bias is temperature-dependent and may change over time; can approximate as a constant
- additive measurement noise

Gyroscopes

- from gyro measurements to orientation – use Taylor expansion

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

Gyroscopes

- from gyro measurements to orientation – use Taylor expansion

have: angle at
last time step

have:
time step

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

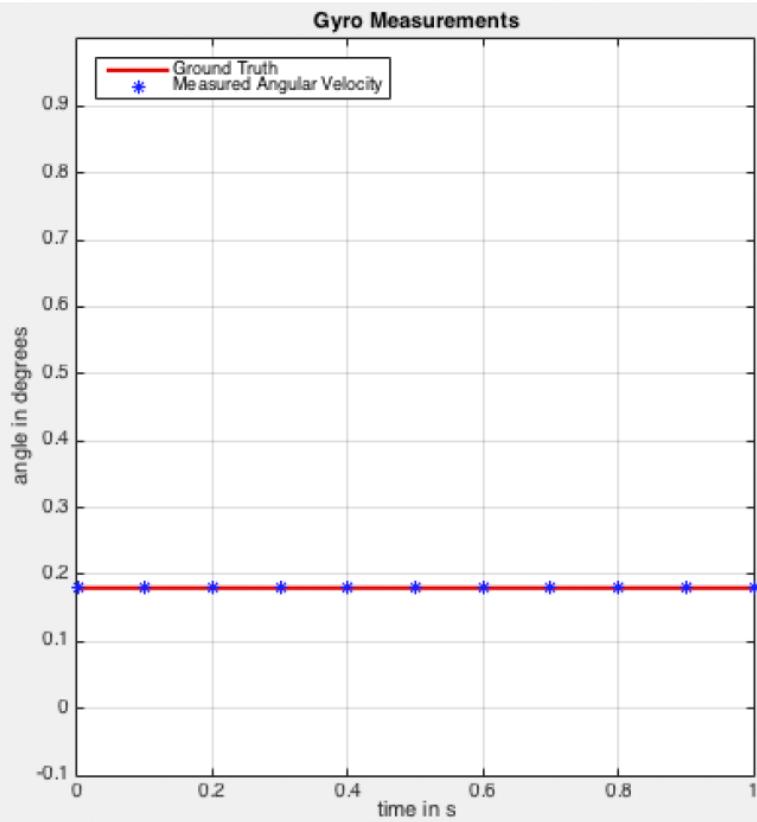
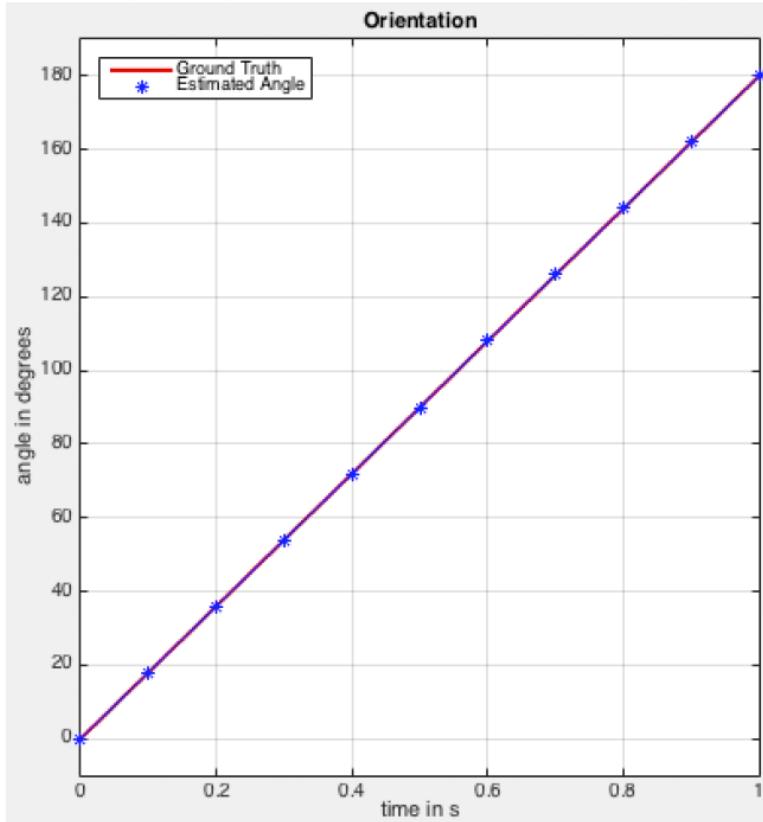
want: angle at
current time step

$= \omega$

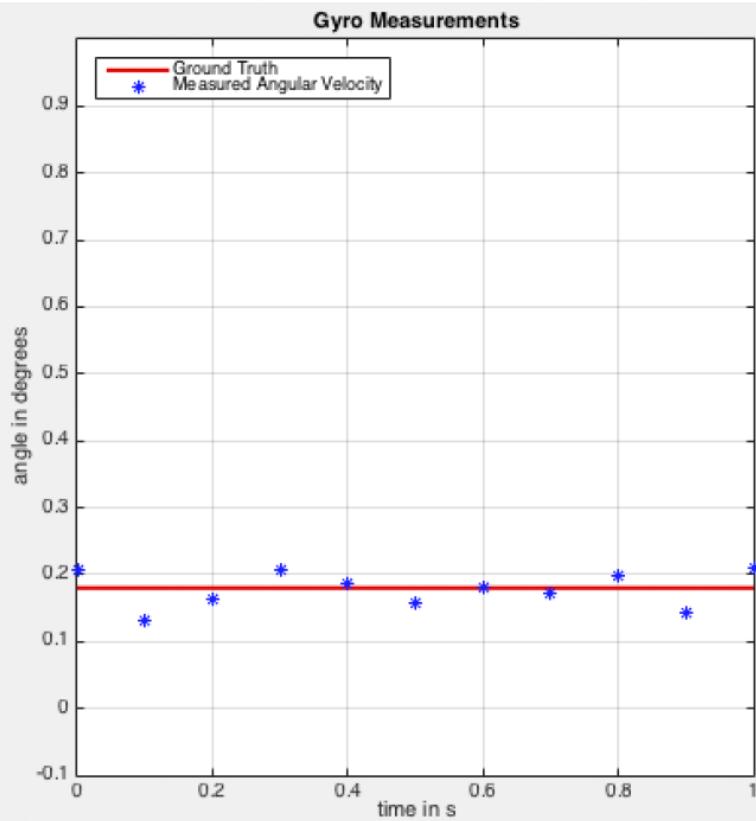
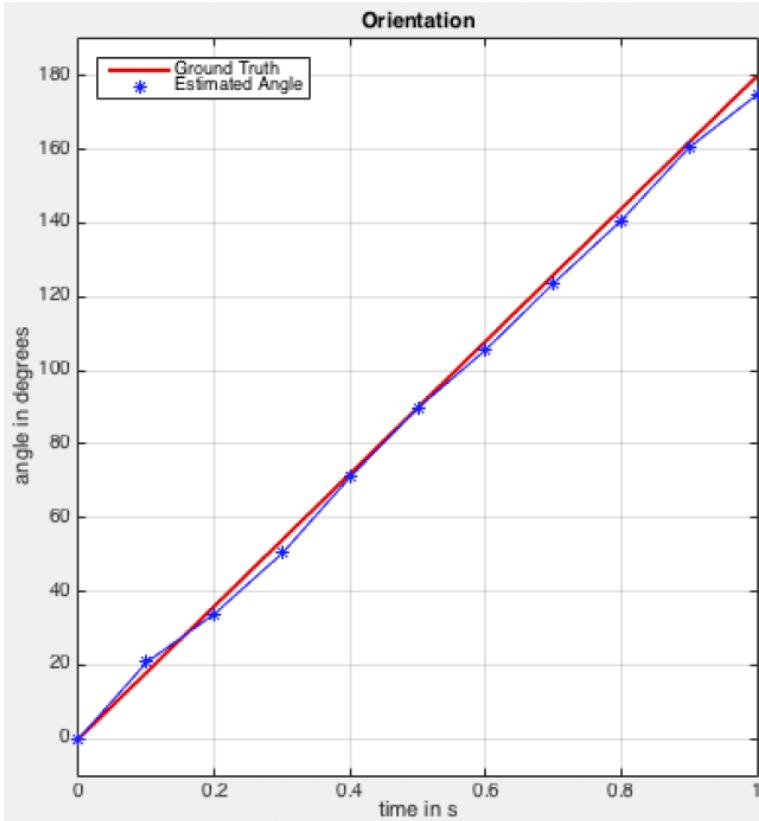
↑
approximation error!

have: gyro measurement
(angular velocity)

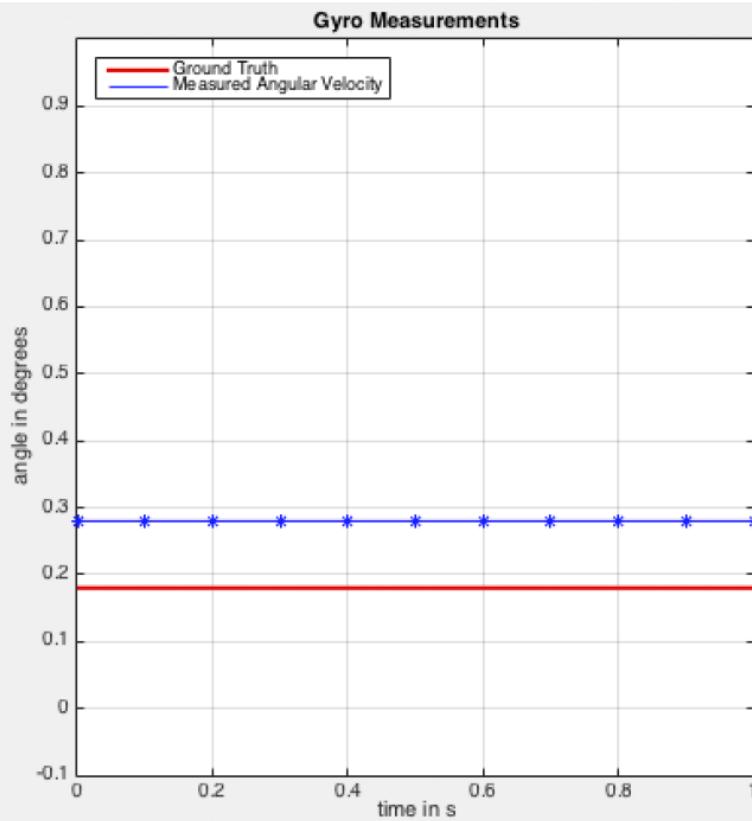
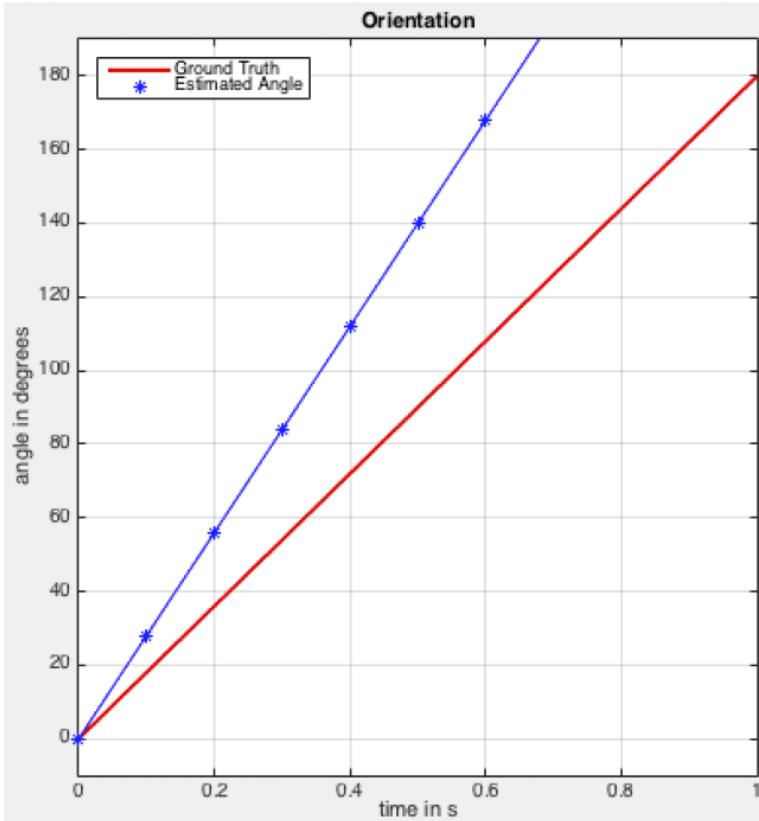
Gyro Integration: linear motion, no noise, no bias



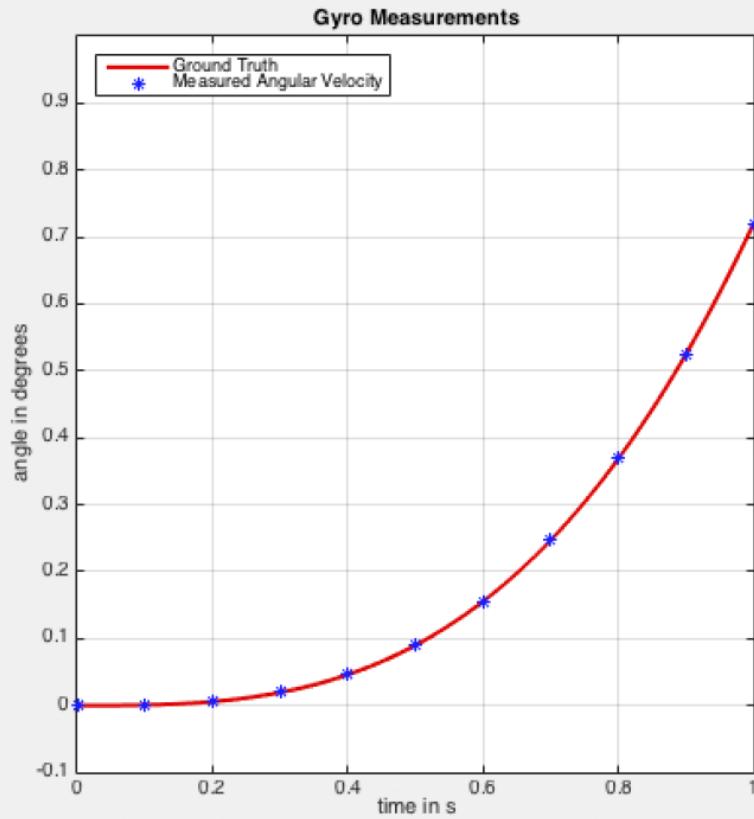
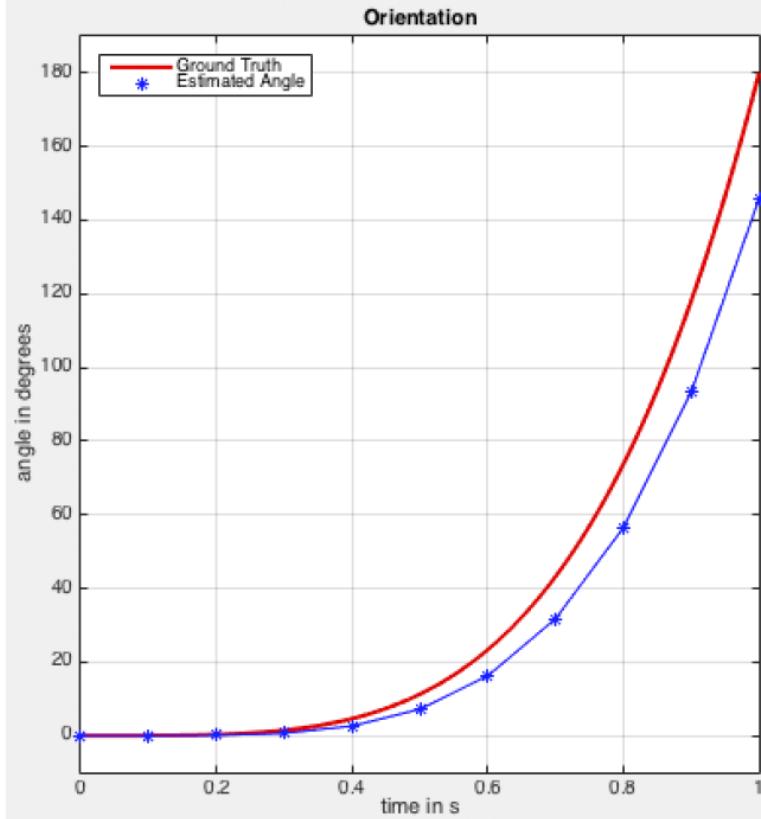
Gyro Integration: linear motion, noise, no bias



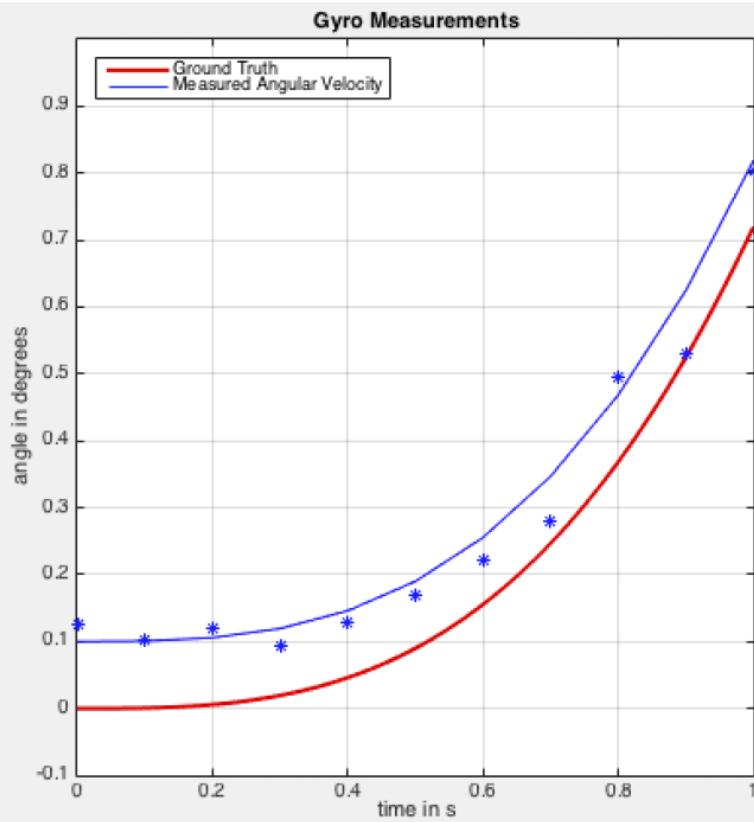
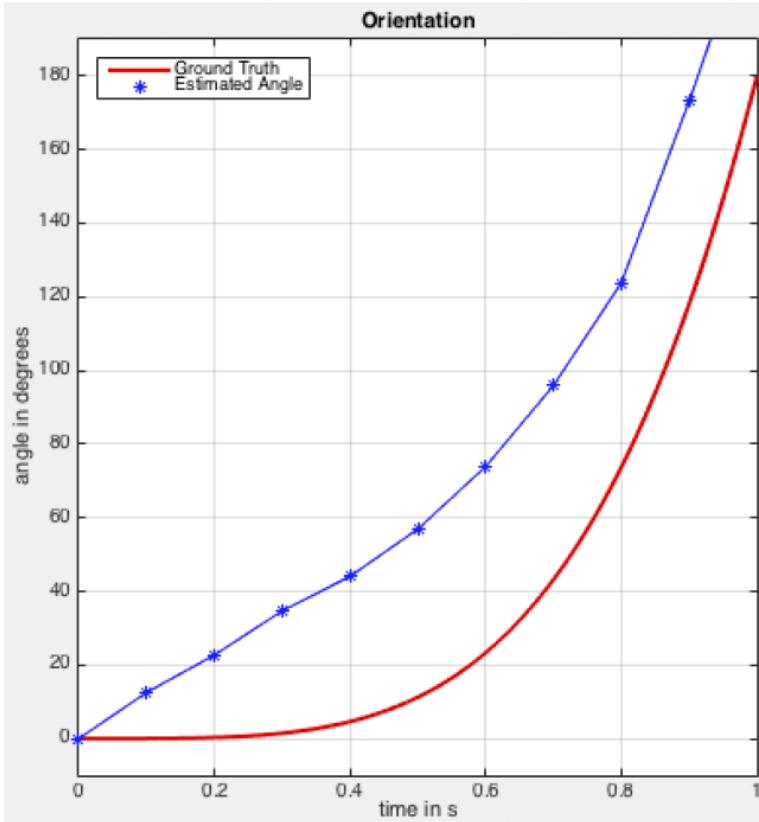
Gyro Integration: linear motion, no noise, bias



Gyro Integration: nonlinear motion, no noise, no bias



Gyro Integration: nonlinear motion, noise, bias



Gyro Integration aka *Dead Reckoning*

- works well for linear motion, no noise, no bias = unrealistic
- even if bias is known and noise is zero → drift (from integration)
- bias & noise variance can be estimated, other sensor measurements used to correct for drift (sensor fusion)
- accurate in short term, but not reliable in long term due to drift



map from game "pirates!"

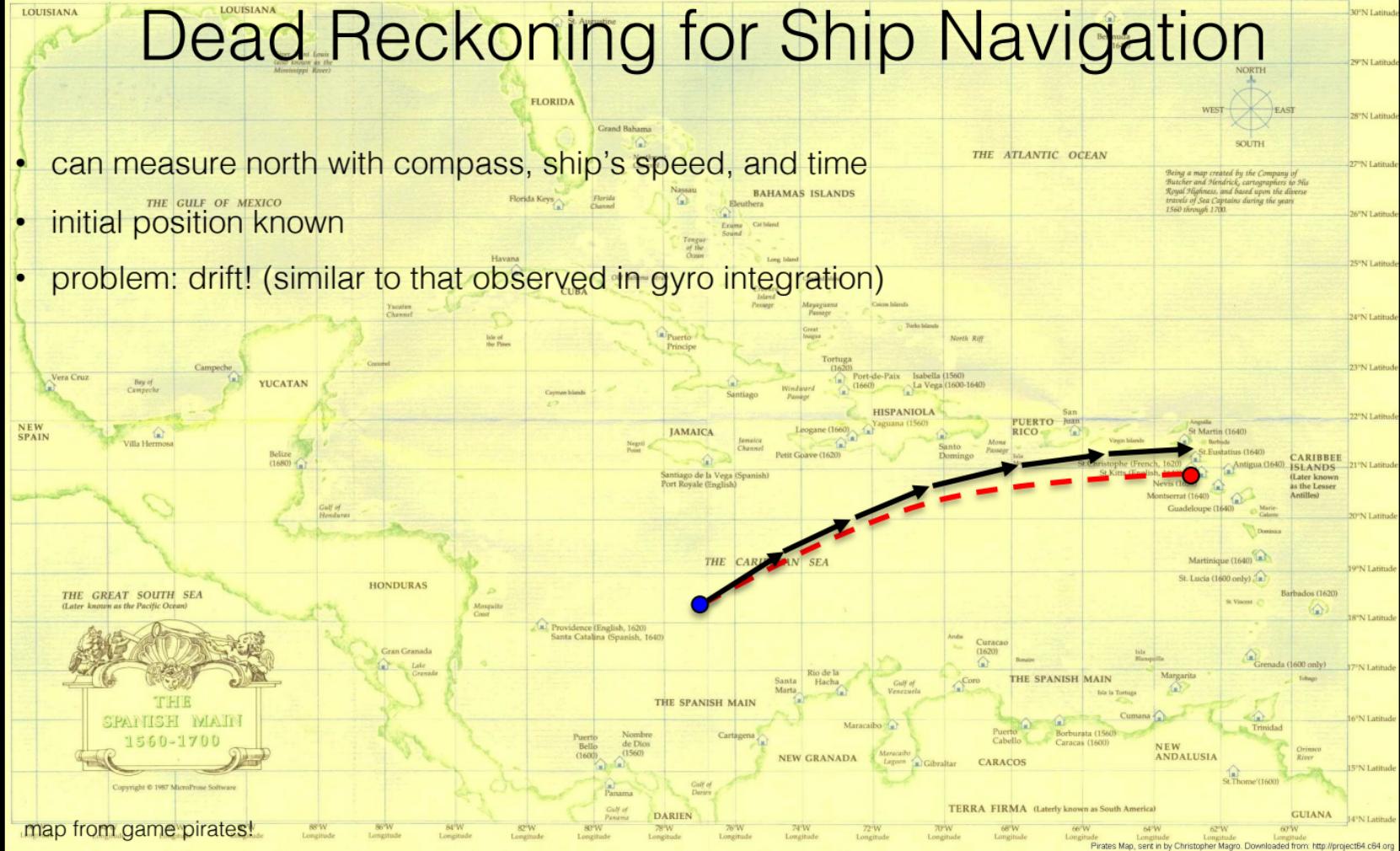
Copyright © 1987 MicroProse Software

Longitude

Latitude

Dead Reckoning for Ship Navigation

- can measure north with compass, ship's speed, and time
 - initial position known
 - problem: drift! (similar to that observed in gyro integration)

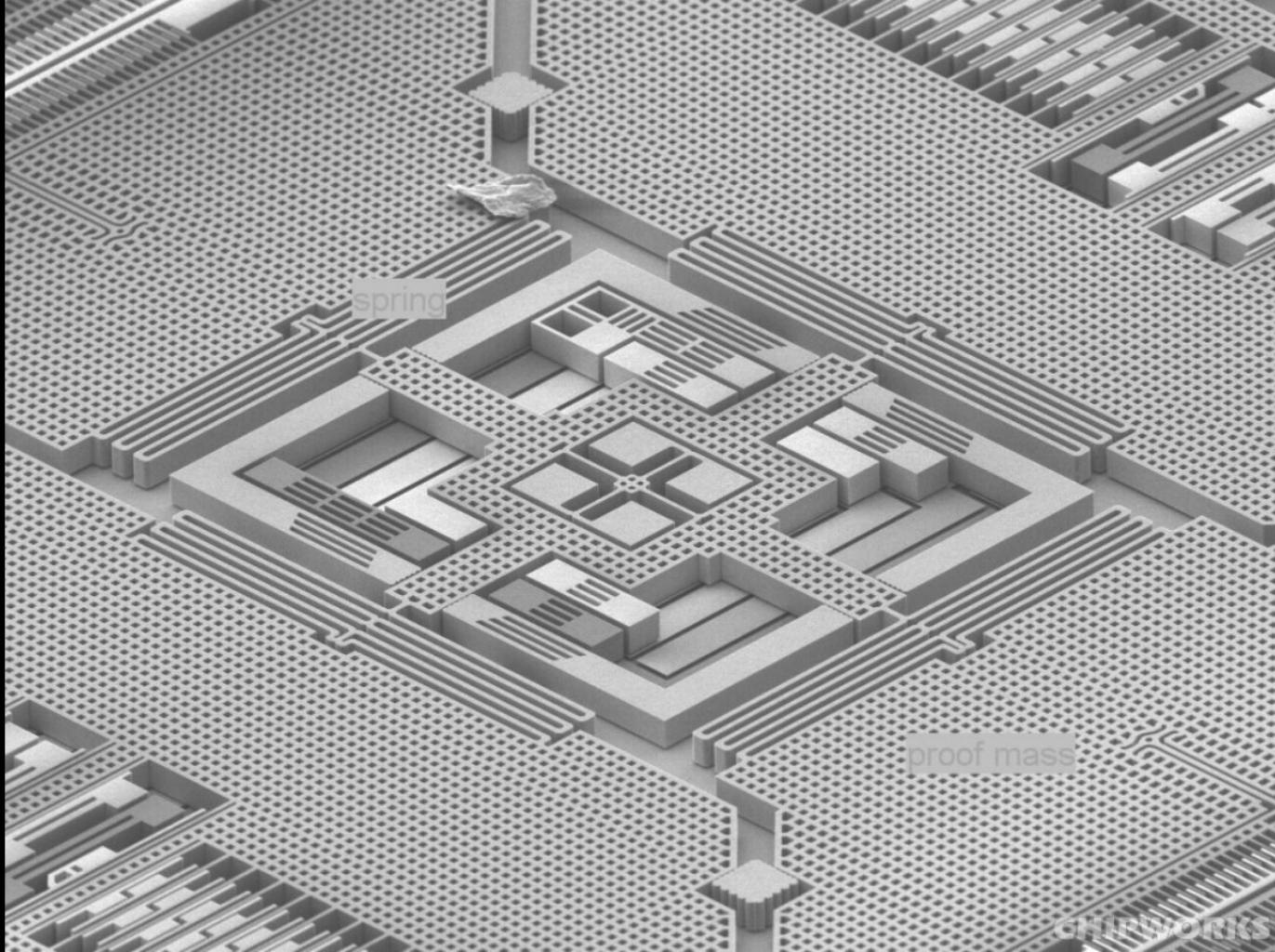


Gyro Advice

Always be aware of what units you are working with, degrees per second vs. radians per second!

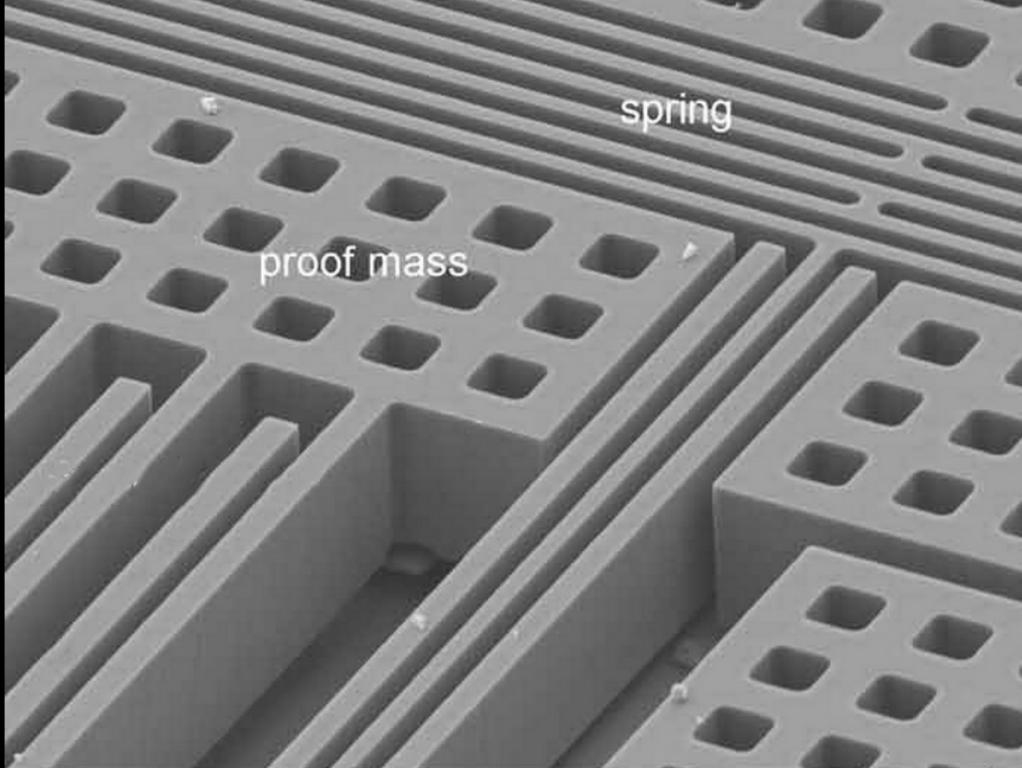
Accelerometers

- measure linear acceleration $\tilde{a} = a^{(g)} + a^{(l)} + \eta$, $\eta \sim N(0, \sigma_{acc}^2)$
- without motion: read noisy gravity vector $a^{(g)} + \eta$ pointing UP! with magnitude $9.81 \text{ m/s}^2 = 1\text{g}$
- with motion: combined gravity vector and external forces $a^{(l)}$

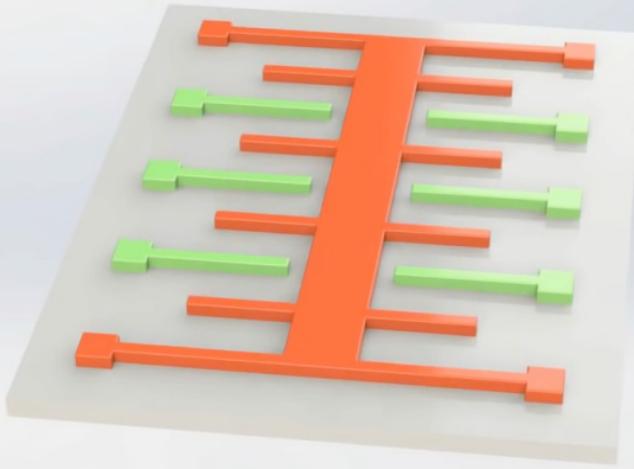


<https://stevecurd.wordpress.com/2015/08/25/mems-and-me-how-does-my-fitbit-know-im-walking/>

capacitive
plates



MEMS Accelerometer



Accelerometers

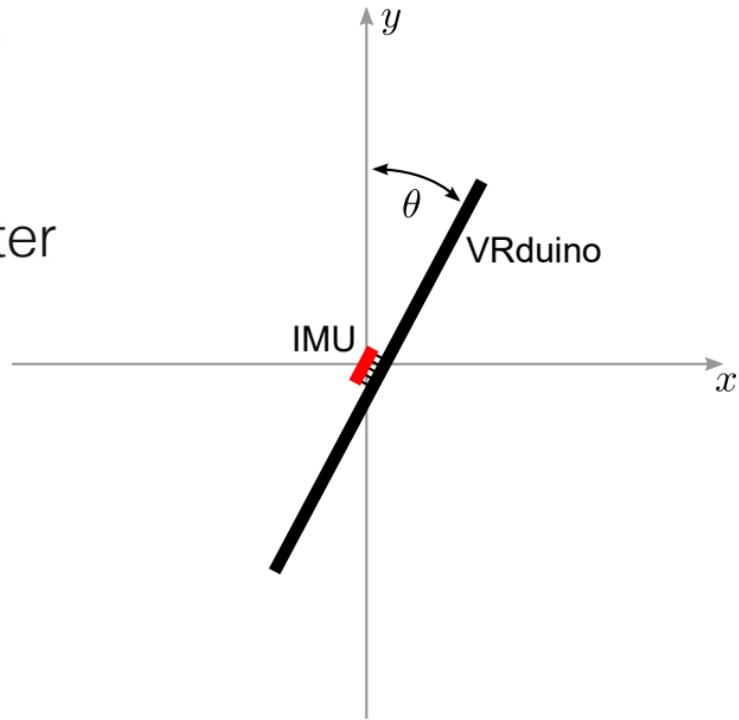
- advantages:
 - points up on average with magnitude of 1g
 - accurate in long term because no drift and the earth's center of gravity (usually) doesn't move
- problem:
 - noisy measurements
 - unreliable in short run due to motion (and noise)
- complementary to gyro measurements!

Accelerometers

- fusing gyro and accelerometer data = 6 DOF sensor fusion
- can correct tilt (i.e., pitch & roll) only – no information about yaw

Orientation Tracking in *Flatland*

- problem: track angle θ in 2D space
- sensors: 1 gyro, 2-axis accelerometer
- goal: understand sensor fusion

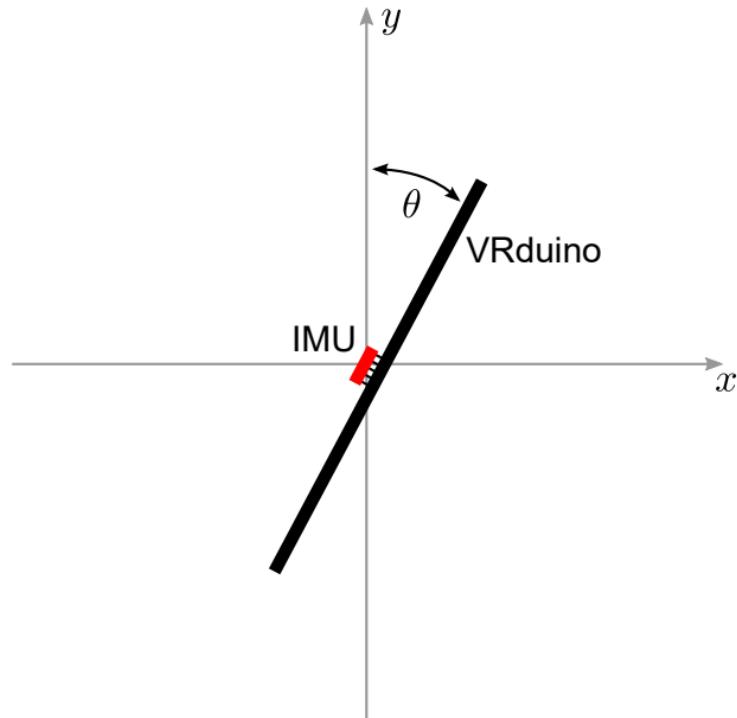


Orientation Tracking in *Flatland*

- gyro integration via Taylor series as

$$\theta_{gyro}^{(t)} = \theta_{gyro}^{(t-1)} + \tilde{\omega} \Delta t$$

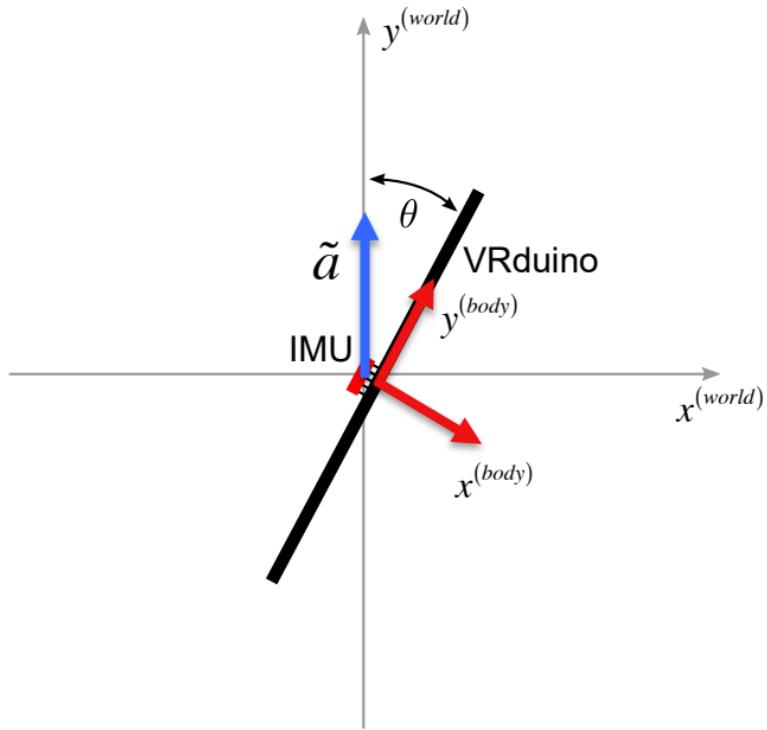
- get Δt from microcontroller
- set $\theta_{gyro}^{(0)} = 0$
- biggest problem: drift!



Orientation Tracking in *Flatland*

- angle from accelerometer

$$\theta_{acc} = \tan^{-1} \left(\frac{\tilde{a}_x}{\tilde{a}_y} \right)$$



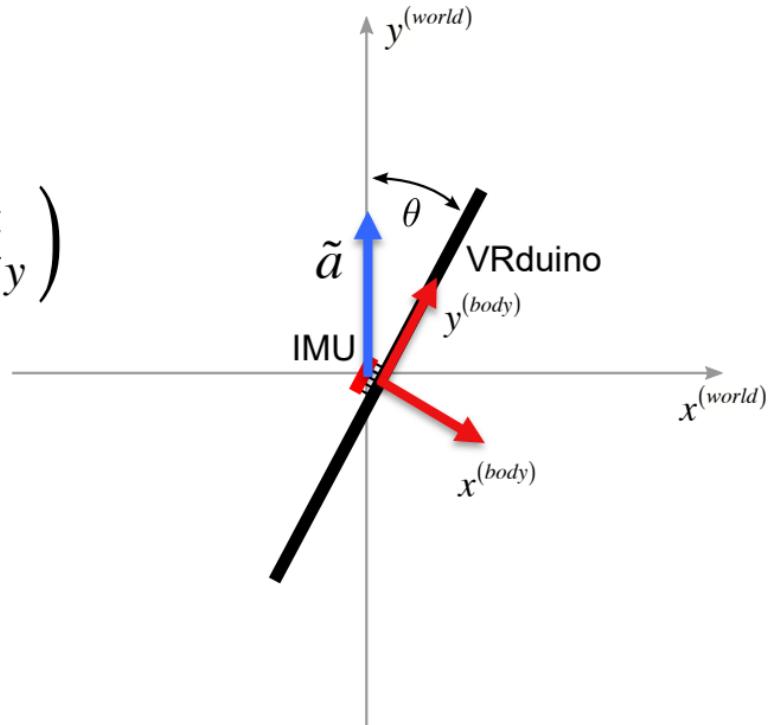
Orientation Tracking in *Flatland*

- angle from accelerometer

$$\theta_{acc} = \tan^{-1}\left(\frac{\tilde{a}_x}{\tilde{a}_y}\right) = \text{atan2}\left(\tilde{a}_x, \tilde{a}_y\right)$$

↑

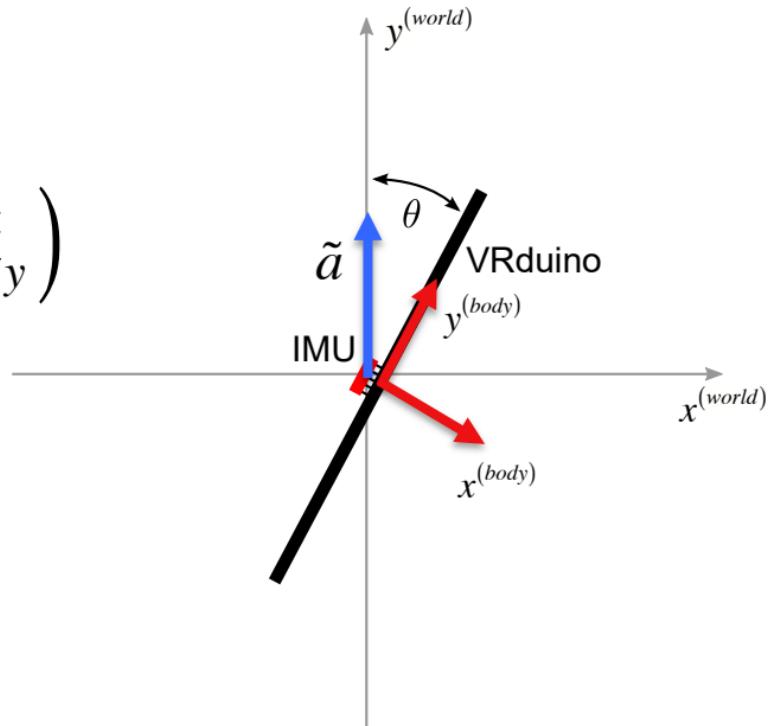
handles division by 0 and proper signs, provided by most programming languages



Orientation Tracking in *Flatland*

- angle from accelerometer

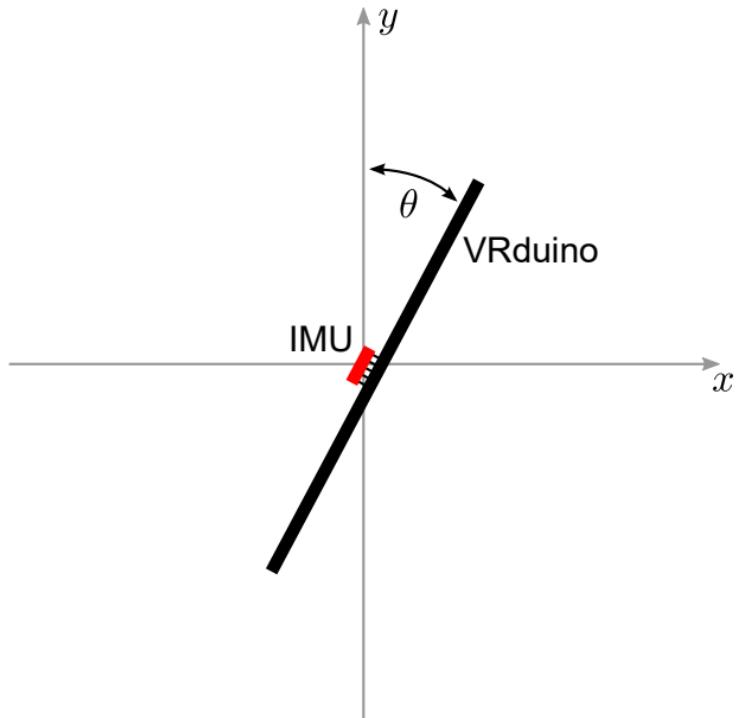
$$\theta_{acc} = \tan^{-1}\left(\frac{\tilde{a}_x}{\tilde{a}_y}\right) = \text{atan2}\left(\tilde{a}_x, \tilde{a}_y\right)$$



- biggest problem: noise

Orientation Tracking in *Flatland*

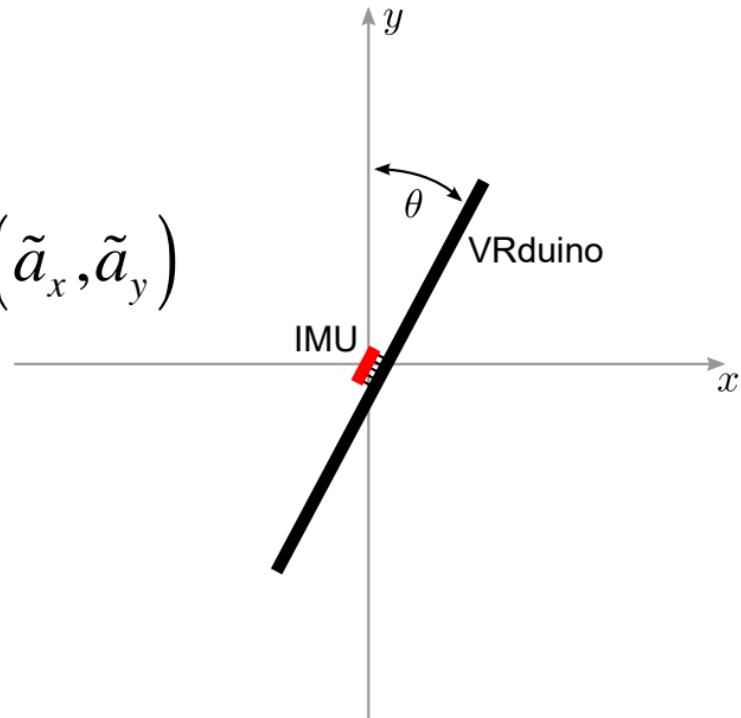
- sensor fusion: combine gyro and accelerometer measurements
- intuition:
 - remove drift from gyro via high-pass filter
 - remove noise from accelerometer via low-pass filter



Orientation Tracking in *Flatland*

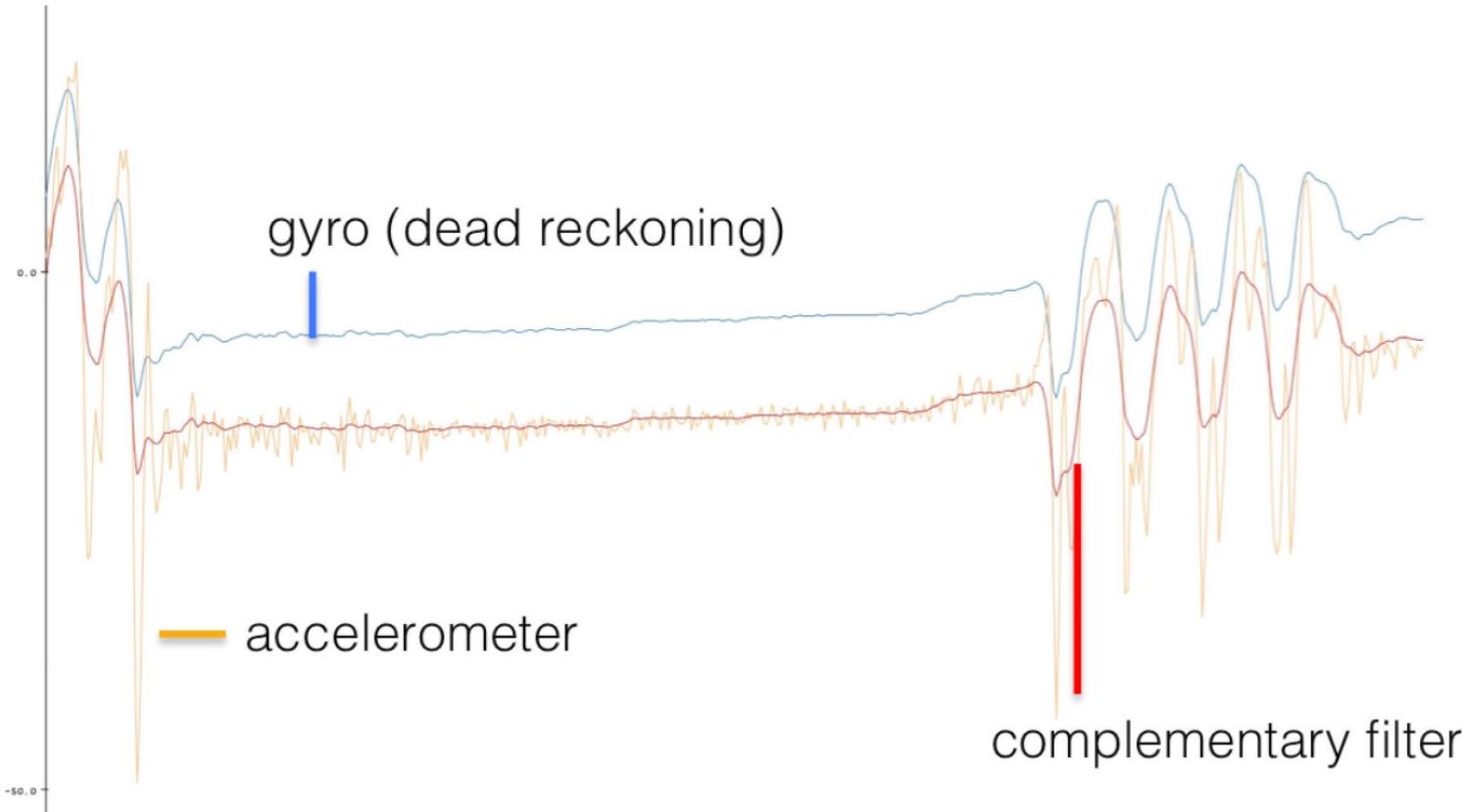
- sensor fusion with complementary filter, i.e. linear interpolation

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



- no drift, no noise!

Orientation Tracking in *Flatland*



Pitch and Roll from 3-axis Accelerometer

- problem: estimate pitch and roll angles in 3D, from 3-axis accelerometer
- together, pitch & roll angles are known as *tilt*
- goal: understand tilt estimation in 3D

Pitch and Roll from 3-axis Accelerometer

- use only accelerometer data – can estimate pitch & roll, not yaw
- assume no external forces (only gravity) – acc is pointing UP!

normalize gravity vector in
inertial coordinates

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



normalize gravity vector rotated into
sensor coordinates

Pitch and Roll from 3-axis Accelerometer

- use only accelerometer data – can estimate pitch & roll, not yaw
- assume no external forces (only gravity) – acc is pointing UP!

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

Pitch and Roll from 3-axis Accelerometer

- use only accelerometer data – can estimate pitch & roll, not yaw
- assume no external forces (only gravity) – acc is pointing UP!

$$\hat{a} = \frac{\tilde{a}}{\|\tilde{a}\|} = \begin{pmatrix} -\cos(-\theta_x)\sin(-\theta_z) \\ \cos(-\theta_x)\cos(-\theta_z) \\ \sin(-\theta_x) \end{pmatrix}$$

Pitch and Roll from 3-axis Accelerometer

- use only accelerometer data – can estimate pitch & roll, not yaw
- assume no external forces (only gravity) – acc is pointing UP!

$$\hat{a} = \frac{\tilde{a}}{\|\tilde{a}\|} = \begin{pmatrix} -\cos(-\theta_x)\sin(-\theta_z) \\ \cos(-\theta_x)\cos(-\theta_z) \\ \sin(-\theta_x) \end{pmatrix} \quad \text{roll} \quad \hat{a}_x = \frac{-\sin(-\theta_z)}{\cos(-\theta_z)} = -\tan(-\theta_z)$$

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

Pitch and Roll from 3-axis Accelerometer

- use only accelerometer data – can estimate pitch & roll, not yaw
- assume no external forces (only gravity) – acc is pointing UP!

pitch

$$\hat{a} = \frac{\tilde{a}}{\|\tilde{a}\|} = \begin{pmatrix} -\cos(-\theta_x)\sin(-\theta_z) \\ \cos(-\theta_x)\cos(-\theta_z) \\ \sin(-\theta_x) \end{pmatrix} \quad \Rightarrow \quad \frac{\hat{a}_z}{\sqrt{\hat{a}_x^2 + \hat{a}_y^2}} = \frac{\sin(-\theta_x)}{\sqrt{\cos^2(-\theta_x)(\sin^2(-\theta_z) + \cos^2(-\theta_z))}} = 1$$
$$= \frac{\sin(-\theta_x)}{\cos(-\theta_x)} = \tan(-\theta_x)$$

Pitch and Roll from 3-axis Accelerometer

- use only accelerometer data – can estimate pitch & roll, not yaw
- assume no external forces (only gravity) – acc is pointing UP!

$$\hat{a} = \frac{\tilde{a}}{\|\tilde{a}\|} = \begin{pmatrix} -\cos(-\theta_x)\sin(-\theta_z) \\ \cos(-\theta_x)\cos(-\theta_z) \\ \sin(-\theta_x) \end{pmatrix} \quad \text{pitch}$$

 $\frac{\hat{a}_z}{\sqrt{\hat{a}_x^2 + \hat{a}_y^2}} = \frac{\sin(-\theta_x)}{\sqrt{\cos^2(-\theta_x)(\sin^2(-\theta_z) + \cos^2(-\theta_z))}} = 1$

$$\theta_x = -\text{atan2}\left(\hat{a}_z, \sqrt{\hat{a}_x^2 + \hat{a}_y^2}\right) \text{ in rad } \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$$

Pitch and Roll from 3-axis Accelerometer

- use only accelerometer data – can estimate pitch & roll, not yaw
- assume no external forces (only gravity) – acc is pointing UP!

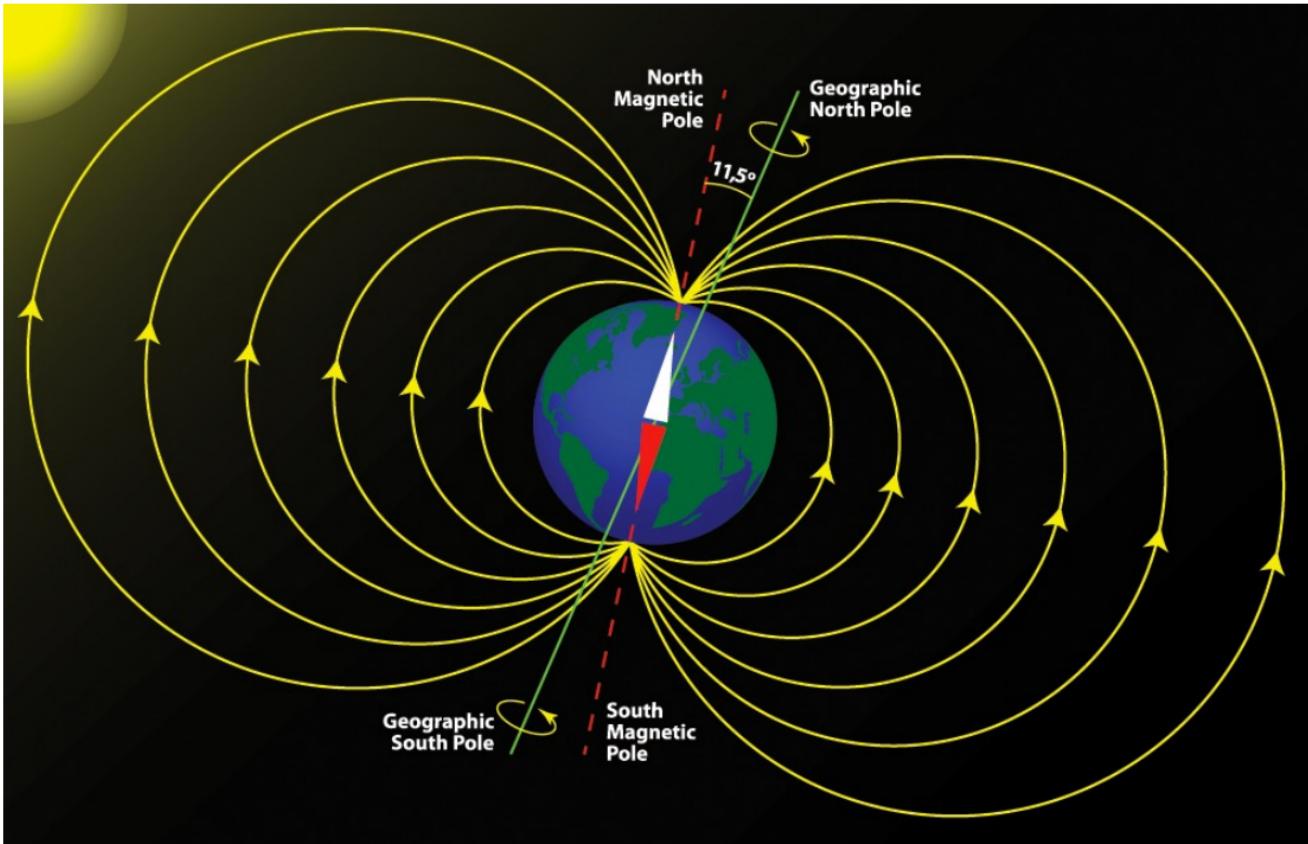
pitch

$$\frac{\hat{a}_z}{\sqrt{\hat{a}_x^2 + \hat{a}_y^2}} = \frac{\sin(-\theta_x)}{\sqrt{\cos^2(-\theta_x)(\sin^2(-\theta_z) + \cos^2(-\theta_z))}}$$
$$= 1$$

$$\theta_x = -\text{atan2}(\hat{a}_z, \text{sign}(\hat{a}_y) \cdot \sqrt{\hat{a}_x^2 + \hat{a}_y^2}) \text{ in rad } \in [-\pi, \pi]$$



Magnetometers



MEMS Magnetometer

Hall Effect

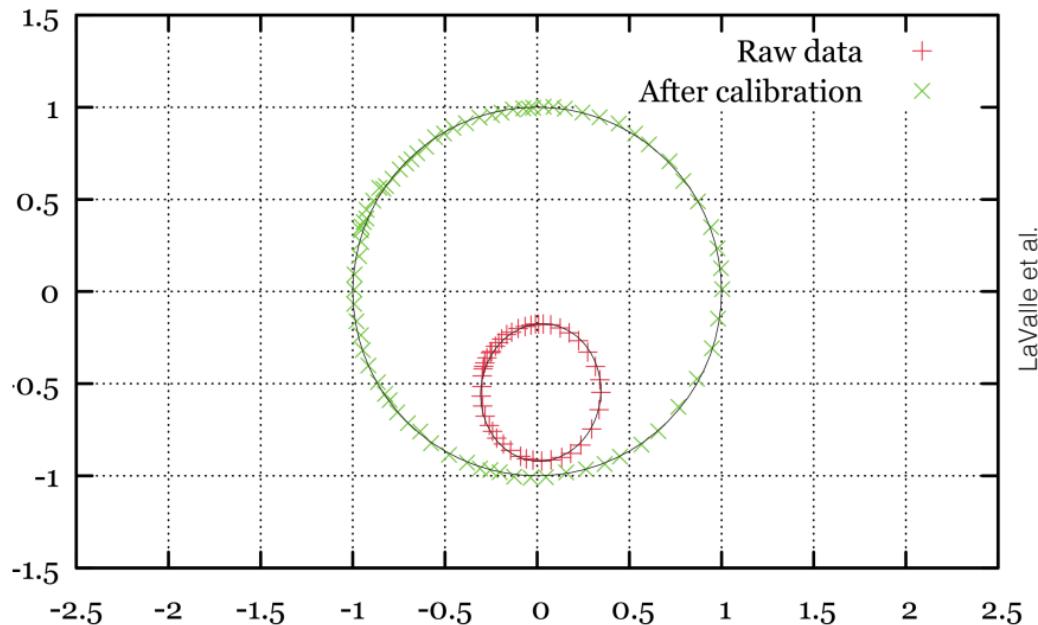
Magneto-resistive effect

Magnetometers

- measure earth's magnetic field in Gauss or uT
- 3 orthogonal axes = vector pointing along the magnetic field
- actual direction depends on latitude and longitude!
- distortions due to metal / electronics objects in the room or in HMD

Magnetometers

difficult to work with magnetometers without proper calibration →
we will not use the magnetometer in the HW!



LaValle et al.

Magnetometers

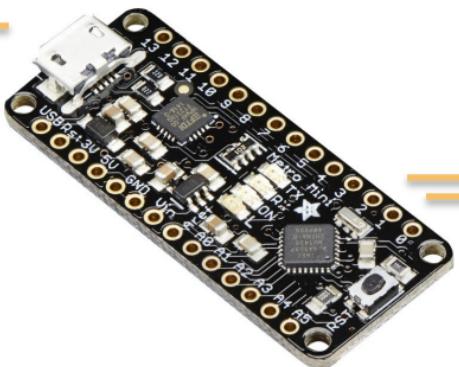
- advantages:
 - complementary to accelerometer – gives yaw (heading)
- problems:
 - affected by metal, distortions of magnetic field
 - need to know location, even when calibrated (e.g. GPS)
- together with gyro + accelerometer = 9 DOF sensor fusion

Prototype IMU

- 9 DOF IMU: InvenSense MPU-9250 = updated model of what was in the Oculus DK2
- 3-axis gyro, 3-axis accelerometer, 3-axis magnetometer all on 1 chip (we'll only use gyro and acc, but we'll give you code to read mag if you want to use it in your project)
- interface with I2C (serial bus) from Arduino

Prototype IMU

to host:
serial via USB

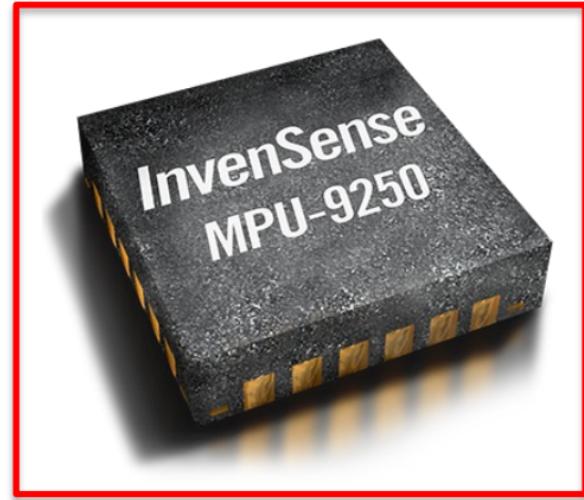


I2C



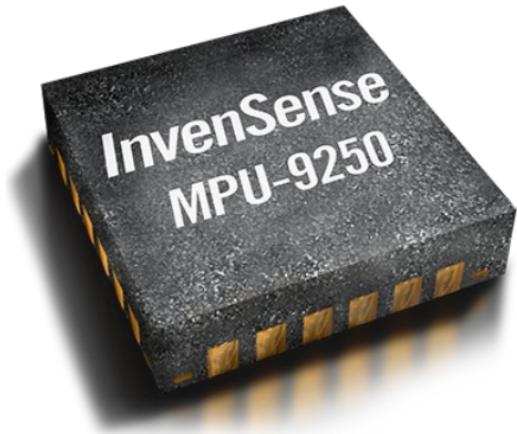
e.g. Arduino

InvenSense MPU-9250



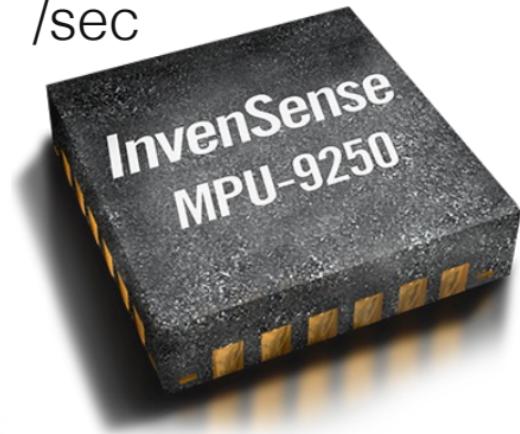
MPU-9250 Specs

- multi-chip module: 1 die houses gyro & accelerometer, the other the magnetometer
- magnetometer: Asahi Kasei Microdevices AK8963 (“3rd party device”)
- 9x 16 bit ADCs for digitizing 9DOF data



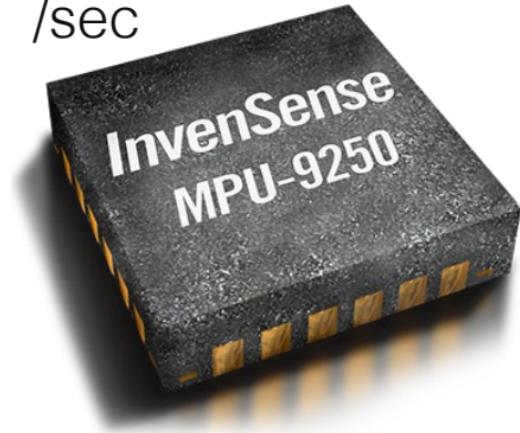
MPU-9250 Specs

- gyro modes: ± 250 , ± 500 , ± 1000 , ± 2000 ° /sec
- accelerometer: ± 2 , ± 4 , ± 8 , ± 16 g
- magnetometer: ± 4800 uT
- configure using registers (see specs) via I2C
- also supports on-board Digital Motion Processing™ (DMP™) sorry, we don't have access
- we'll provide starter code for Arduino in lab (easy to use for beginners, not consumer product grade!)



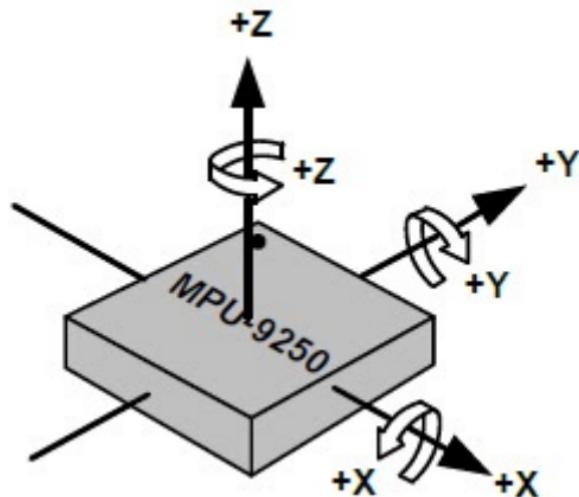
MPU-9250 Specs

- gyro modes: $\pm 250, \pm 500, \pm 1000, \pm 2000$ ° /sec
- accelerometer: $\pm 2, \pm 4, \pm 8, \pm 16$ g
- magnetometer: ± 4800 uT

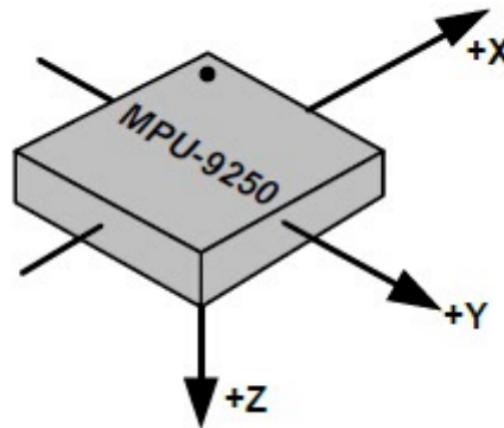


$$metric_value = \frac{raw_sensor_value}{2^{15} - 1} \cdot max_range$$

MPU-9250 Coordinate Systems



gyro & accelerometer

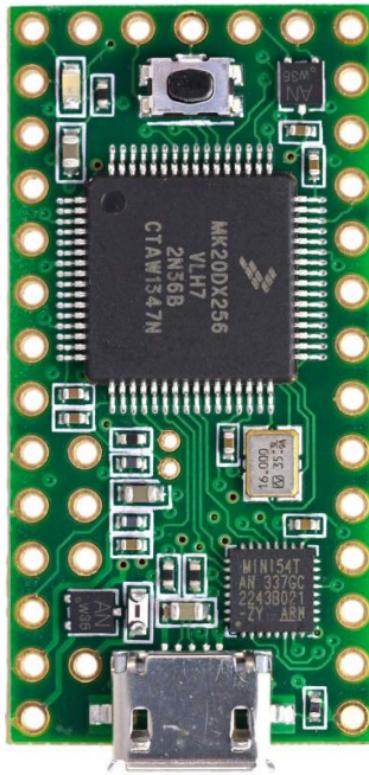


magnetometer

How to read data from IMU

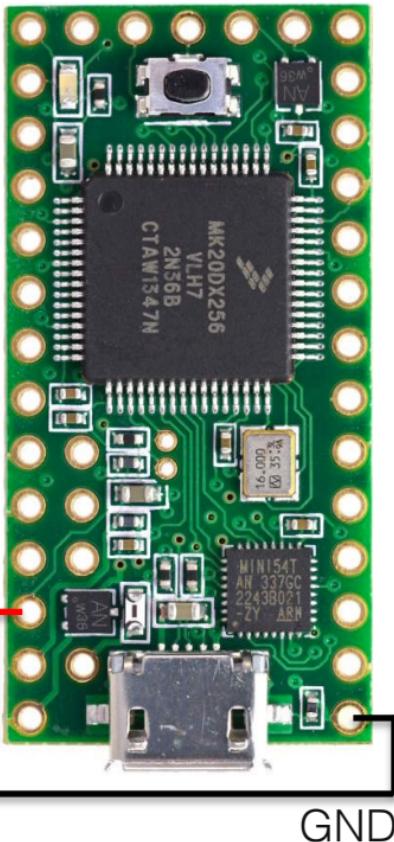
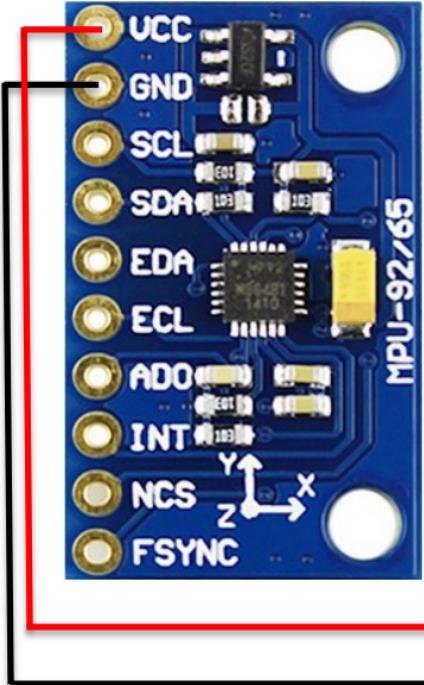
- I2C = serial interface with 2 wires (also see next lab)
- microcontroller to read, we'll use Teensy 3.2, but any Arduino can be used, e.g. past offerings used Metro Mini
- schematics - which pins to connect where
- quick intro to Arduino
- Wire library to stream out data via serial
- serial client using node server

How to read data from IMU



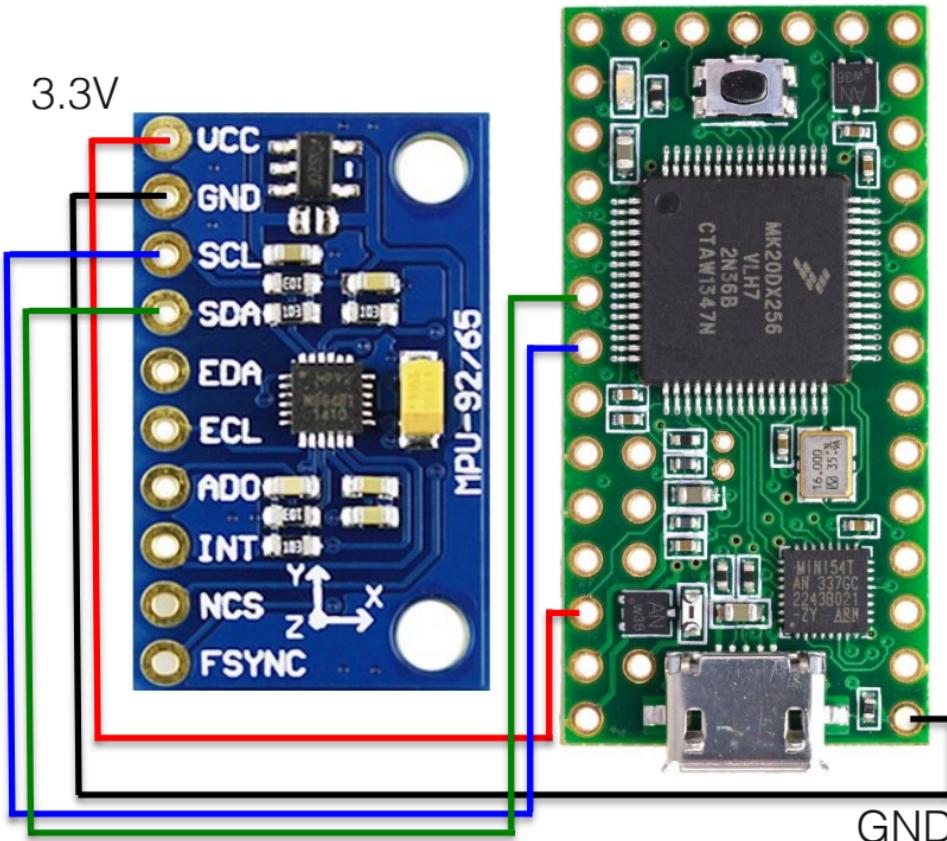
How to read data from IMU

3.3V



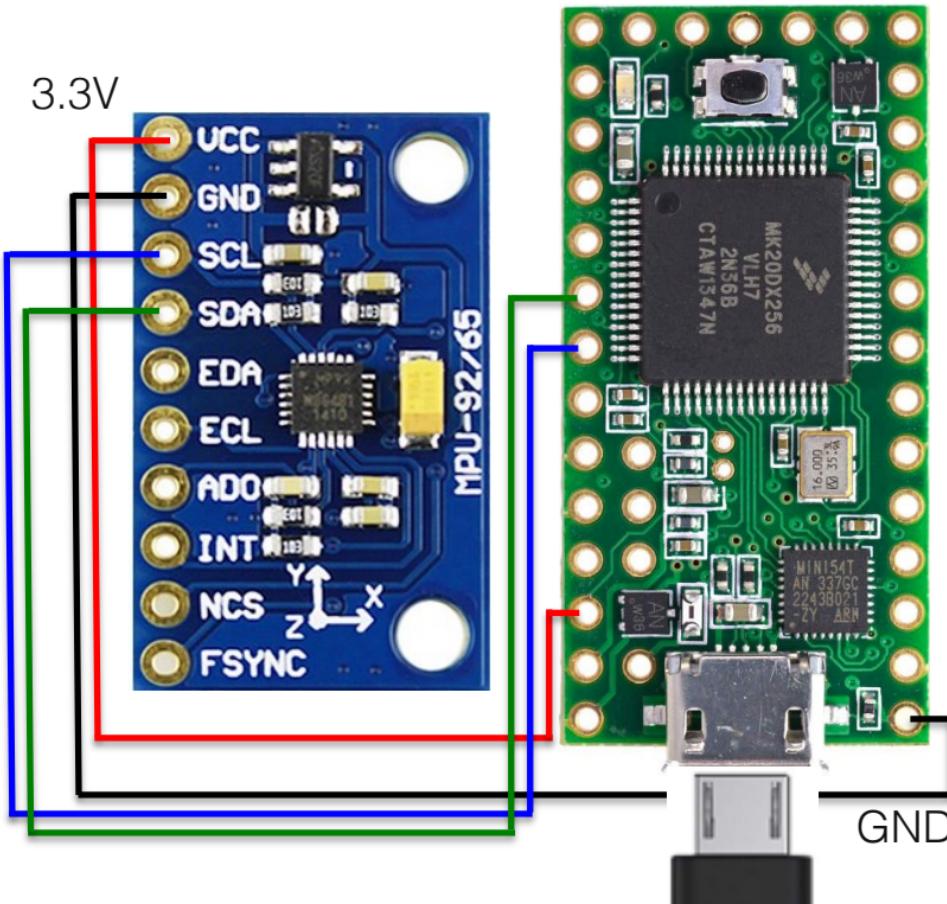
- connect power & ground

How to read data from IMU



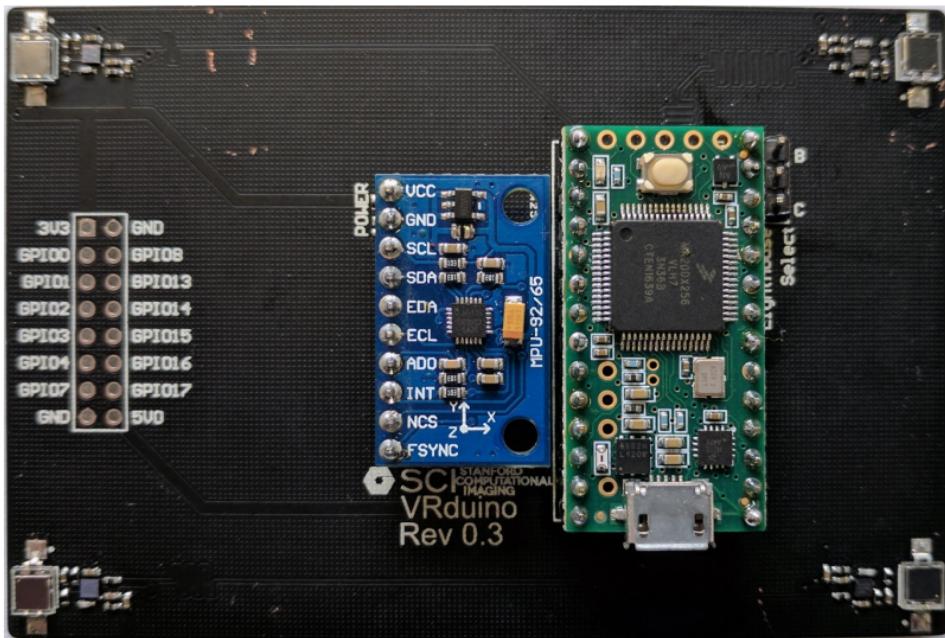
- connect power & ground
- connect I2C clock (SCL, pin19/A5) and data (SDA, pin18/A4) lines

How to read data from IMU



- connect power & ground
- connect I2C clock (SCL, pin19/A5) and data (SDA, pin18/A4) lines
- connect micro USB for power and data transfer

VRduino

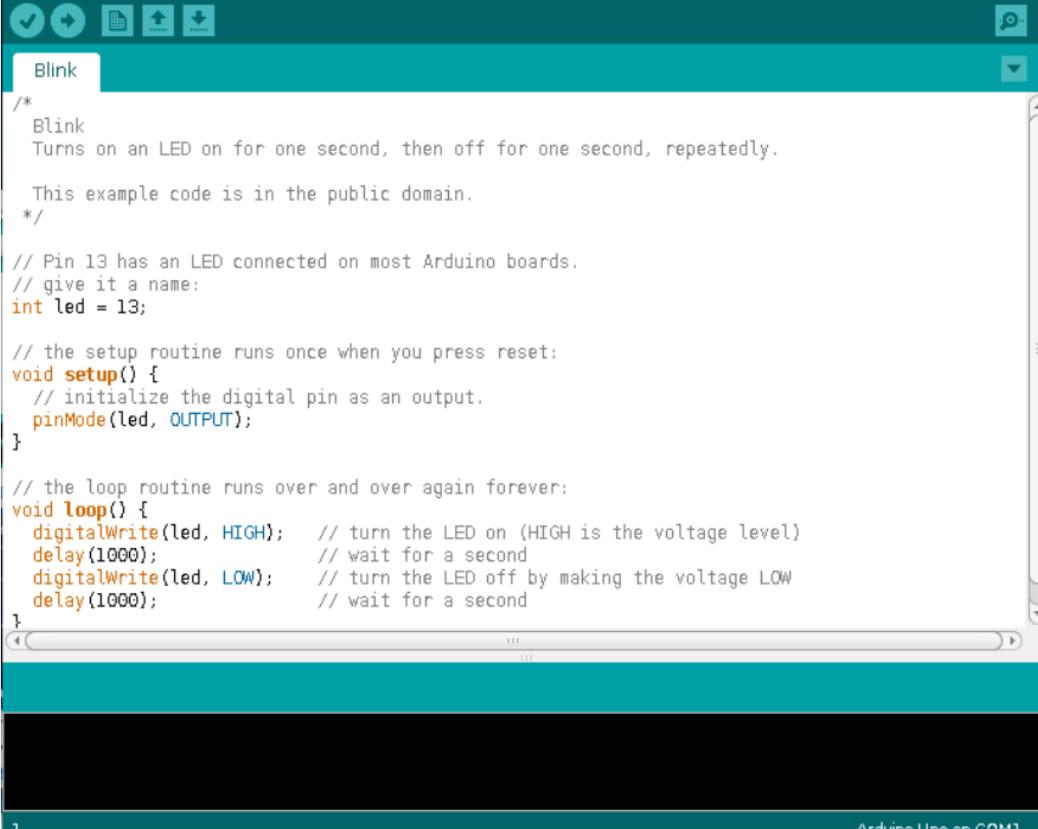


- Teensy 3.2 & IMU already connected through PCB
- also has 4 photodiodes (more details next week)
- GPIO pins for additional sensors or other add-ons

Introduction to Arduino

- open source microcontroller hardware & software
- directly interface with sensors (i.e. IMU) and process raw data
- we will be working with Teensy 3.2 (Arduino compatible)
- use Arduino IDE for all software development, installed on all lab machines
- if you want to install it on your laptop, make sure to get:
 - IDE: <https://www.arduino.cc/en/Main/Software>
 - Teensyduino: <https://www.pjrc.com/teensy/teensyduino.html>
 - Wire library (for serial & I2C): <http://www.arduino.cc/en/Reference/Wire>
 - FTDI drivers: <http://www.ftdichip.com/Drivers/VCP.htm>

Introduction to Arduino (Random Test Program)



The screenshot shows the Arduino IDE interface with the 'Blink' example sketch open. The code is as follows:

```
/*
Blink
Turns on an LED on for one second, then off for one second, repeatedly.

This example code is in the public domain.
*/

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);    // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);     // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

The status bar at the bottom right indicates "Arduino Uno on COM1".

← variable definition

← setup function = initialization

← loop function = runtime callback

← connected to COM1 serial port

Introduction to Arduino

- need to stream data from Arduino to host PC
- use Wire library for all serial & I2C communication
- use node server to read from host PC and connect to JavaScript (see lab)

Introduction to Arduino

The screenshot shows the Arduino IDE interface with the sketch titled "imu_test_9dof_corrected". The code is as follows:

```
imu_test_9dof_corrected | Arduino 1.6.5

imu_test_9dof_corrected
// Put read bytes starting at register Register in the Data array.

void I2Cread(uint8_t Address, uint8_t Register, uint8_t Nbytes, uint8_t* Data)
{
    // Set register address
    Wire.beginTransmission(Address);
    Wire.write(Register);
    Wire.endTransmission();

    // Read Nbytes
    Wire.requestFrom(Address, Nbytes);
    uint8_t index=0;
    while (Wire.available())
        Data[index++]=Wire.read();
}

///////////////////////////////
// Write a byte (Data) in device (Address) at register (Register)

void I2CwriteByte(uint8_t Address, uint8_t Register, uint8_t Data)
{
    // Set register address
    Wire.beginTransmission(Address);
    Wire.write(Register);
    Wire.write(Data);
    Wire.endTransmission();
}

///////////////////////////////
// Initializations
void setup()
{
    // Arduino initializations
    Wire.begin();
    Serial.begin(115200);

    // Configure gyroscope range
    I2CwriteByte(MPU9250_ADDRESS, 27, GYRO_FULL_SCALE_2000_DPS);
    I2CwriteByte(MPU9250_ADDRESS, 27, GYRO_FULL_SCALE_500_DPS);

    // Configure accelerometers range
    I2CwriteByte(MPU9250_ADDRESS, 28, ACC_FULL_SCALE_16_G);
    I2CwriteByte(MPU9250_ADDRESS, 28, ACC_FULL_SCALE_2_G);

    // Set by pass mode for the magnetometers
    I2CwriteByte(MPU9250_ADDRESS, 0x37, 0x02);

    // Request first magnetometer single measurement
    I2CwriteByte(MAG_ADDRESS, 0x0A, 0x01);
}

///////////////////////////////
// Main loop, read and display data
while (true)
{
    // Done uploading.
}

Sketch uses 6,894 bytes (21%) of program storage space. Maximum is 32,256 bytes.
Global variables use 410 bytes (20%) of dynamic memory, leaving 1,638 bytes for local variables. Maximum is 2,048 bytes.
```

← read from I2C (connected to IMU)

← write to I2C (connected to IMU)

← setup function = one time initialization

← open serial connection to
communicate with host PC

← set registers to configure IMU

Read Serial Data in Windows

- serial ports called COMx (USB serial usually COM3-COM7)
 1. establish connection to correct COM port (choose appropriate baud rate)
 2. read incoming data (in a thread)

Summary

- coordinate systems (world, body/sensor, inertial, transforms)
- overview of inertial sensors: gyroscopes, accelerometers, and magnetometers
- gyro integration aka *dead reckoning*
- orientation tracking in *flatland*
- pitch & roll from accelerometer
- overview of VRduino

Next Lecture

- quaternions and rotations with quaternions
- 6 DOF sensor fusion with quaternions & complementary filtering

Must read: course notes on IMUs!

Additional Information

- D. Sachs “Sensor Fusion on Android Devices: A Revolution in Motion Processing”, Google Tech Talks 2010, Video on youtube.com (<https://www.youtube.com/watch?v=C7JQ7Rpwn2k>)
- S. LaValle, A. Yershova, M. Katsev, M. Antonov “Head Tracking for the Oculus Rift”, Proc. ICRA 2014
- <http://www.chrobotics.com/library>