# Sensors in the Control Loop

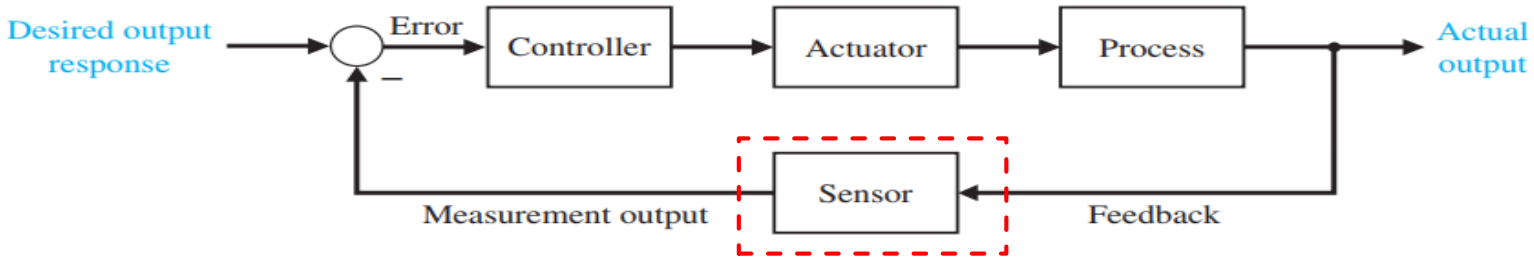
Dr. ASHRAF SUYYAGH
UNIVERSITY OF JORDAN
DEPARTMENT OF COMPUTER ENGINEERING

# Types of Sensors

- Analogue Sensors with Analogue O/P
- Digital Sensors with SPI or $I^2C$ Interface
  - Digital Sensors have internal configuration registers for different operation modes
  - Have built-in A/Ds
  - Many provide interrupt capabilities
- Most modern sensors are MEMS (**M**icro**E**lectro**M**echanical **S**ystems )

# Sensors in Feedback Control Systems

- When designing modern process plants / control systems, one needs to specify sensors to measure process variables, such as flow, level, pressure, temperature, acceleration, rotational speed, etc.
- The readings are essential part of the feedback control loop.
- We need proper values of these quantities to ensure correct and safe operation.
- Must guarantee that the actual value is sensed and passed to the controller. Values must also be error free.
- We need to ***calibrate*** the sensor, ***filter noise out*** of sensor readings, and in some applications, ***fuse the readings*** of multiple (same or different) sensors.



# Sensor Calibration - Introduction

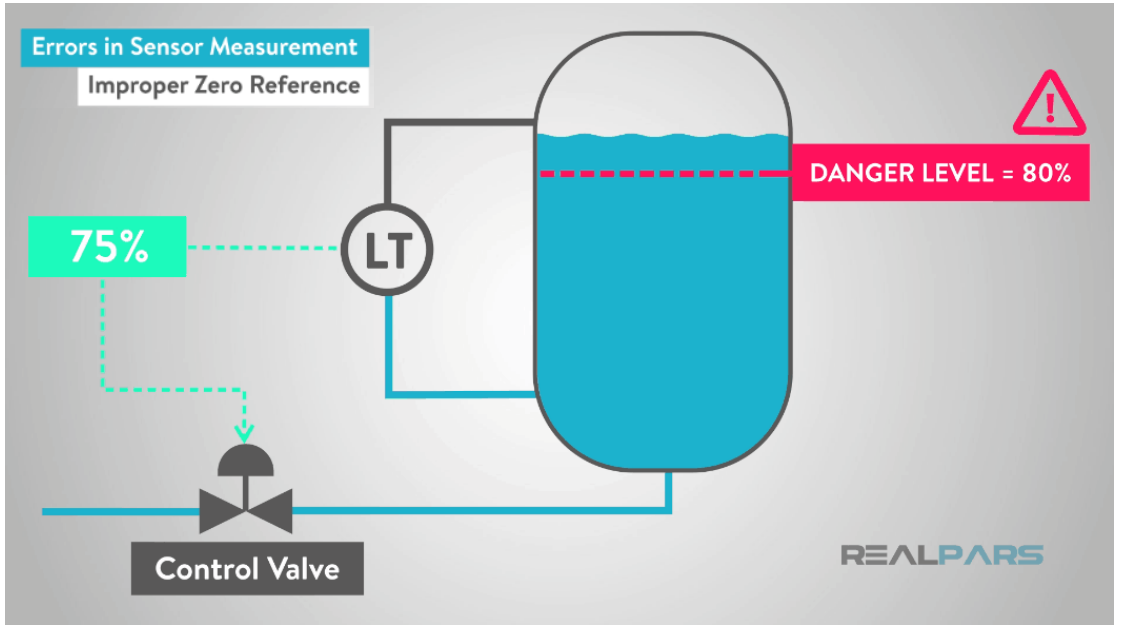
- Definition: Sensor calibration is an adjustment or set of adjustments performed on a sensor or instrument to make that instrument function as accurately, or error free, as possible.
- Calibration might be a one-time step, or carried out periodically (if possible).
- Many digital sensors are factory calibrated, and the calibration coefficients are stored internally in registers.
- Even with factory calibration, one might need to do a second step of calibration.
  - Mounting, soldering, and subjecting sensors to heat during assembly might affect it.
- Over time, sensor performance might degrade or change over time, recalibration if possible might be needed!
- Advanced industrial analogue sensors may have knobs for zero and span calibration.
- Digital Calibration can be done in software.

# Errors in Sensor Measurement

- Sensor error is the difference between the sensed and actual value of measured variable.
- Types of sensor errors:
  1. Errors due to Improper Zero Reference
  2. Error due to Shift in Sensor's Range
  3. Error due to Mechanical Wear or Damage
- The errors could be Systematic (deterministic) or Random (stochastic)
- Systematic errors could be observed as a bias or scaled values, dead sensor zones (regions), or non-linearity.
- Random errors could be observed as a bias drift or scale factor instability.

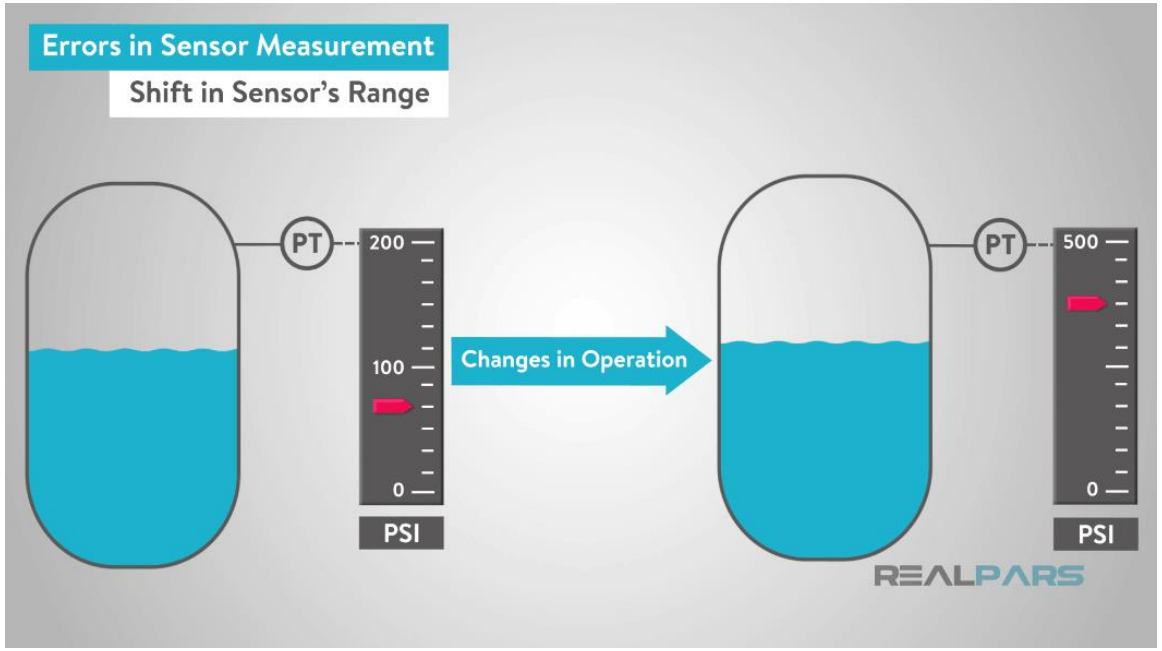
# Error due to Improper Zero Reference

- The reference voltage, or signal, may drift over time due to temperature, pressure, or change in ambient conditions.
- Suppose that the sensor zero reference is at 5% of liquid level instead of 0%.
- Suppose that the liquid level must not exceed 75% for safe operation.
- The liquid level can exceed safe limits (e.g. 80%) while the sensor reports liquid is still at safe threshold of 75%.



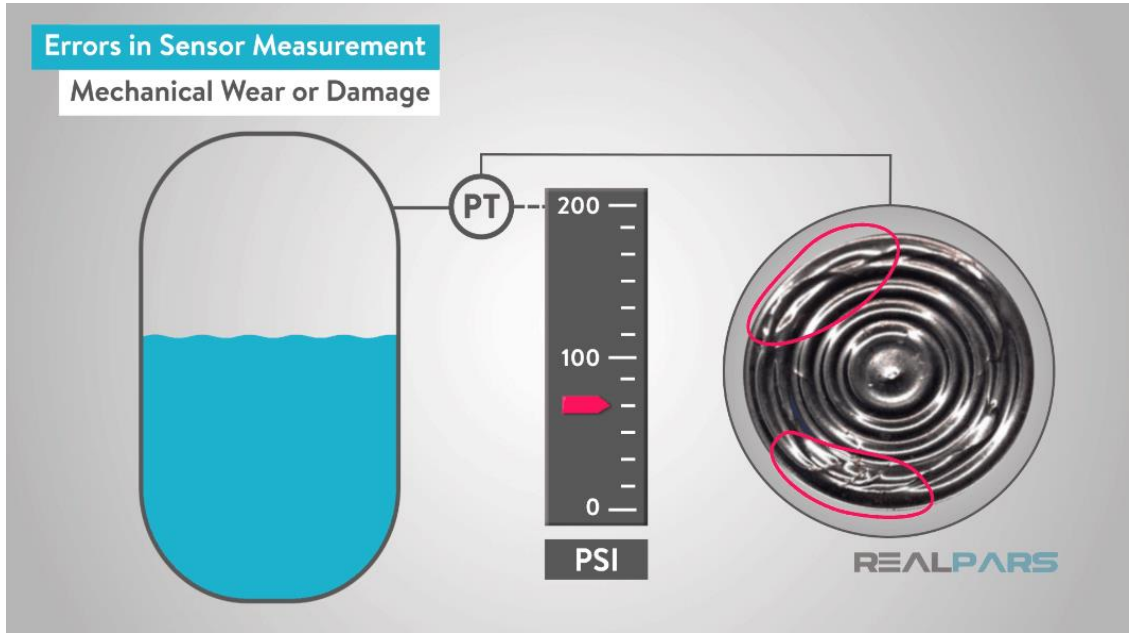
# Error due to Shift in Sensor's Range

- Sensors have different response and transfer functions given varying ambient conditions (e.g. temperature).
- Or due to system updates/design changes, the range of sensor requires re-adjustment.



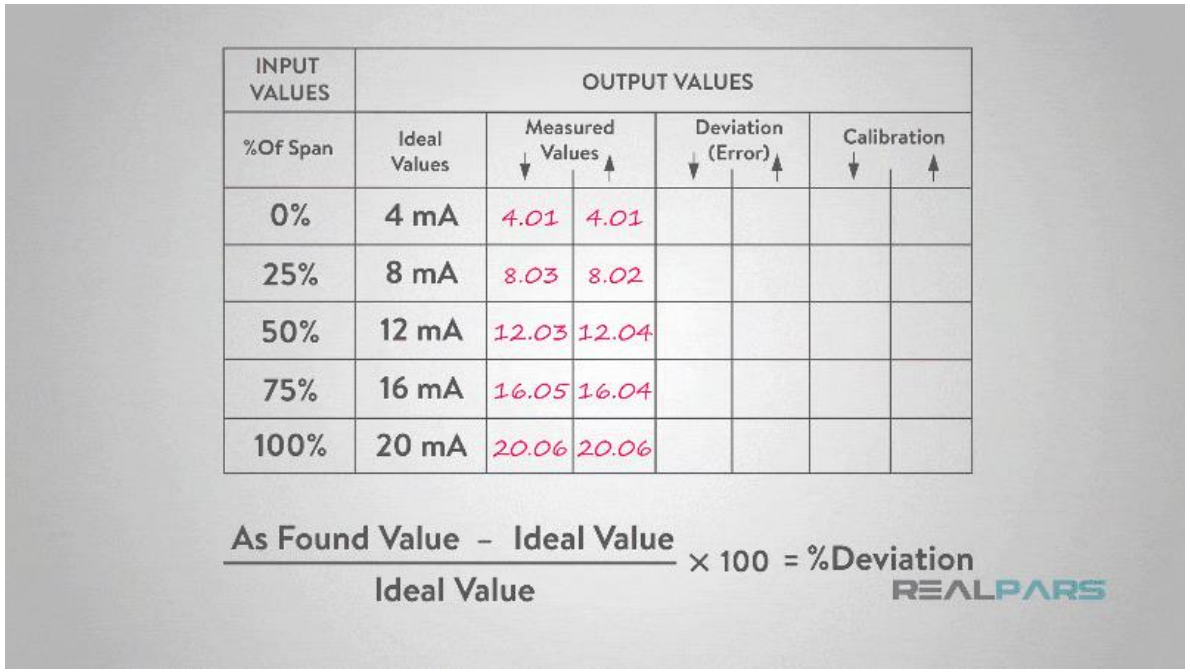
# Error due to Mechanical Wear or Damage

- Third, error in sensor measurement may occur because of mechanical wear, or damage. Usually, this type of error will require repair or replacement of the device.



# As-Found Check or "Five-Point" Calibration

- Calibration can take a considerable period of time, especially if the device is hard to reach or requires special tools.
- "As-Found" Check or "Five-point" Check minimizes calibration time.
- Checks for deviation errors at five points of sensor span (range) corresponding to 100%, 75%, 50%, 25%, and 0%.
- Hysteresis, a phenomenon whereby the sensor output for a process value is different going 'downscale' as it is going 'upscale'.
- Deviations are checked if they are within the maximum tolerated, if so, no calibration needed.



| INPUT VALUES | OUTPUT VALUES | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| %Of Span | Ideal Values | Measured Values ↓ | Measured Values ↑ | Deviation (Error) ↓ | Deviation (Error) ↑ | Calibration ↓ | Calibration ↑ |
| 0% | 4 mA | 4.01 | 4.01 | | | | |
| 25% | 8 mA | 8.03 | 8.02 | | | | |
| 50% | 12 mA | 12.03 | 12.04 | | | | |
| 75% | 16 mA | 16.05 | 16.04 | | | | |
| 100% | 20 mA | 20.06 | 20.06 | | | | |

$$\frac{\text{As Found Value} - \text{Ideal Value}}{\text{Ideal Value}} \times 100 = \%\text{Deviation}$$

REALPARS

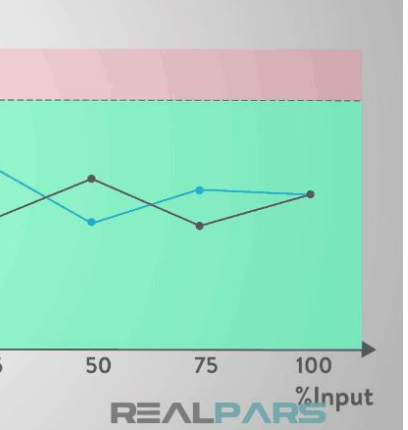
# Five-Point Check Example

0.5% Tolerance

**Calibration Sheet**

| INPUT VALUES | OUTPUT VALUES | | | | |
|---|---|---|---|---|---|
| %Of Span | Ideal Values | Measured Values ↓ | Measured Values ↑ | Deviation (Error) ↓ | Deviation (Error) ↑ |
| 0% | 4 mA | 4.01 | 4.01 | 0.25 | 0.25 |
| 25% | 8 mA | 8.03 | 8.02 | 0.38 | 0.25 |
| 50% | 12 mA | 12.03 | 12.04 | 0.25 | 0.33 |
| 75% | 16 mA | 16.05 | 16.04 | 0.31 | 0.23 |
| 100% | 20 mA | 20.06 | 20.06 | 0.30 | 0.30 |

**Deviation Graph**



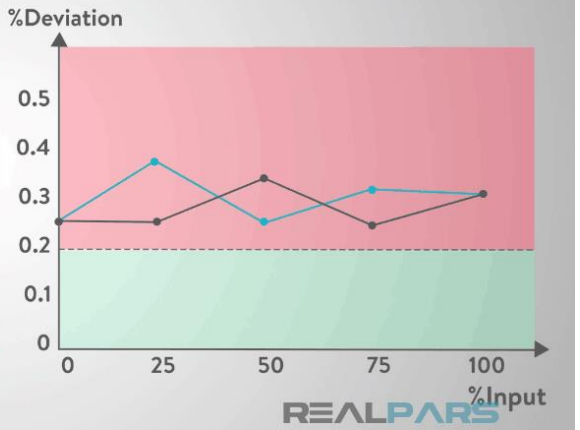

REALPARS

0.2% Tolerance

**Calibration Sheet**

| INPUT VALUES | OUTPUT VALUES | | | | | |
|---|---|---|---|---|---|---|
| %Of Span | Ideal Values | Measured Values ↓ | Measured Values ↑ | Deviation (Error) ↓ | Deviation (Error) ↑ | Calibration ↓ | Calibration ↑ |
| 0% | 4 mA | 4.01 | 4.01 | 0.25 | 0.25 | 4.00 | 4.00 |
| 25% | 8 mA | 8.03 | 8.02 | 0.38 | 0.25 | 8.01 | 8.00 |
| 50% | 12 mA | 12.03 | 12.04 | 0.25 | 0.33 | 12.02 | 12.02 |
| 75% | 16 mA | 16.05 | 16.04 | 0.31 | 0.23 | 16.03 | 16.02 |
| 100% | 20 mA | 20.06 | 20.06 | 0.30 | 0.30 | 20.00 | 20.00 |

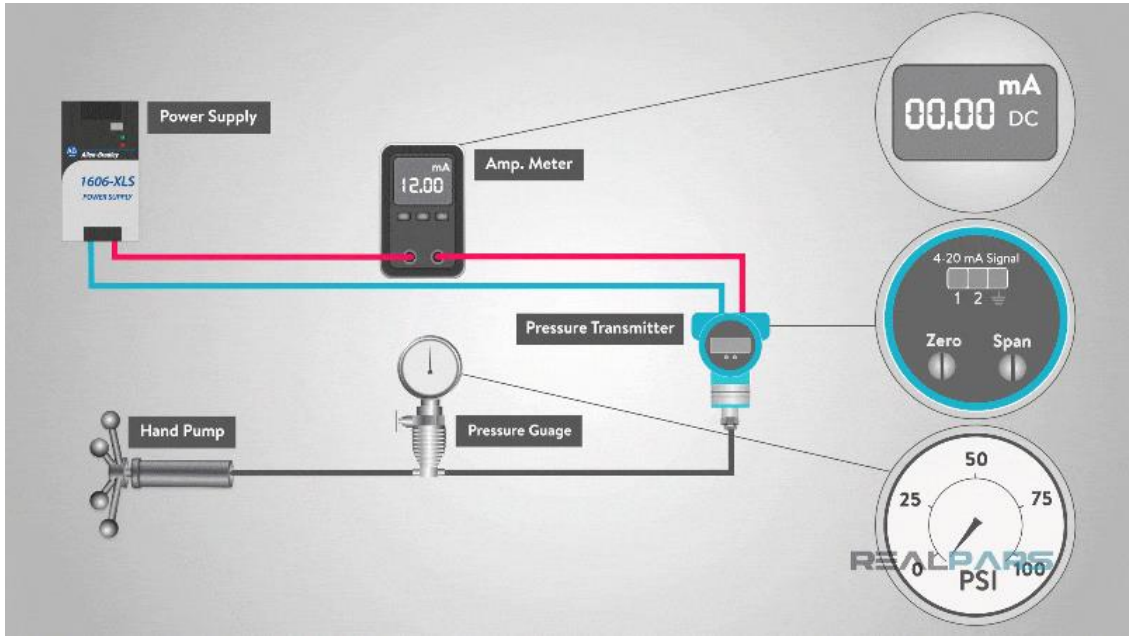
**Deviation Graph**




REALPARS

# How to perform calibration?

- **Analog Sensor Calibration (Zero and Span Adjustment)**
- One needs a very accurate process simulator → Pressure supply
- Also needs very accurate current measurement devices.
- Unfortunately, with analog transmitters, the zero and span adjustments are interactive; that is, adjusting one moves the other → Iterative process
- Usually 2-3 iterations.

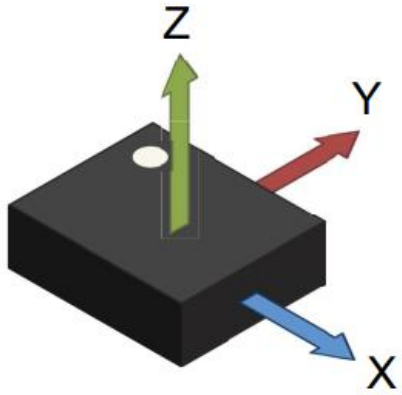


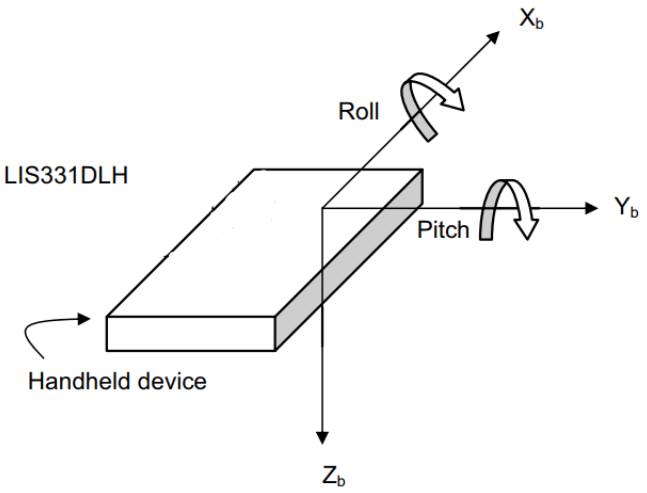
# Digital Sensor Calibration

- Many digital sensors are factory calibrated, calibration coefficients are stored internally.
- Depending on the device, you can enable a built-in algorithm in the MEMS to automatically apply the calibration and provide calibrated output.
- Other MEMS devices only allow you to read the calibration coefficients and apply the algorithm in software (in your own code)
- Even with factory calibration, there might be a case where you're the factory calibrated values are above your tolerance level, or the ambient conditions affected the device between calibration and usage.
- Additional Calibration Procedures are found in ***Application Notes***, or ***Design notes.***

# Digital Sensor Calibration – Accelerometer Example (1)

- This is one technique that can be used to calibrate some tri-axial sensor (i.e. sensors that produce outputs over the three cartesian axis)
- Examples include triaxial accelerometers or triaxial magnetometers.
- Ideally, when the sensor axis is along the same axis and direction of the G vector (earth gravitational force), the sensor must provide a reading of 1G (1000mg).
- If the axis is along the same axis but opposite direction of the G vector (earth gravitational force), the sensor must provide a reading of -1G (-1000mg).
- All other axis that are perpendicular to the gravitational axis must produce 0G since sin(90) = 0.
- Accelerometers can be used to calculate Roll, Pitch and Yaw angles (more later).



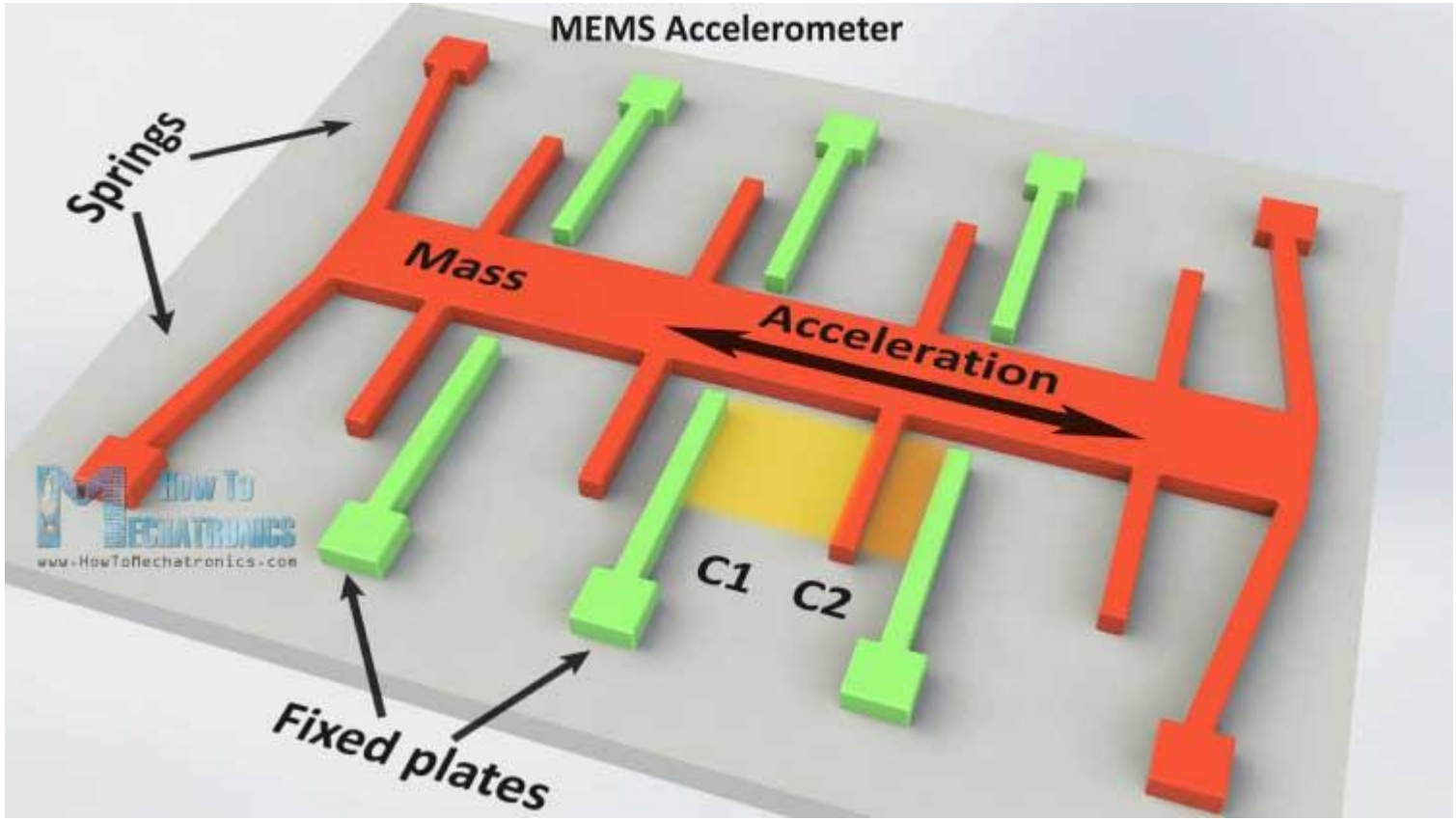
# Digital Sensor Calibration – Accelerometer Example (2)

**Sign definition of LIS331DLH sensor raw measurements**

| Stationary position | Accelerometer (signed integer) | | |
|---|---|---|---|
| | $A_x$ | $A_y$ | $A_z$ |
| $Z_b$ down | 0 | 0 | +1 *g* |
| $Z_b$ up | 0 | 0 | -1 *g* |
| $Y_b$ down | 0 | +1 *g* | 0 |
| $Y_b$ up | 0 | -1 *g* | 0 |
| $X_b$ down | +1 *g* | 0 | 0 |
| $X_b$ up | -1 *g* | 0 | 0 |

# Digital Sensor Calibration – Accelerometer Example (3)

- However, when perfectly aligned, the values read ,might not be equal to one (calibration errors (e.g. misaligned axes), and noise).
- Misalignment during fabrication, due to excessive heat during mounting and soldering on PCB (Printed Circuit Board).
- Acceleration is measured by change of capacitance.



# Digital Sensor Calibration – Accelerometer Example (4)

- This is the Least Squares Method for Calibration
- Ideal (Target Output) = Raw Input * Calibration Matrix

Y = W * X

$$\begin{bmatrix} A_{x1} & A_{y1} & A_{z1} \end{bmatrix} = \begin{bmatrix} A_{rx1} & A_{ry1} & A_{rz1} & 1 \end{bmatrix} \times \begin{bmatrix} CC_{x1} & CC_{x2} & CC_{x3} \\ CC_{y1} & CC_{y2} & CC_{y3} \\ CC_{z1} & CC_{z2} & CC_{z3} \\ O_1 & O_2 & O_3 \end{bmatrix}$$

1x3 | 1x4 | 4x3 | Offset

$$A_{x1} = A_{rx1}CC_{x1} + A_{ry1}CC_{y1} + A_{rz1}CC_{z1} + O1$$
$$A_{y1} = A_{rx2}CC_{x2} + A_{ry2}CC_{y2} + A_{rz2}CC_{z2} + O2$$
$$A_{z1} = A_{rx3}CC_{x3} + A_{ry3}CC_{y3} + A_{rz3}CC_{z3} + O3$$

Where $[A_{x1}, A_{y1}, A_{z1}]$ is any of the possible vectors corresponding to the sensor orientation (see slide 14)

# Digital Sensor Calibration – Accelerometer Example (5)

- Each of the $A_r$ values is the average of 500 or 1000 samples collected at each calibration position.
- However, this procedure must be repeated for all 6-positions. As such the matrix will be:

$$\begin{bmatrix} A_{x1} & A_{y1} & A_{z1} \\ A_{x2} & A_{y2} & A_{z2} \\ A_{x3} & A_{y3} & A_{z3} \\ A_{x4} & A_{y4} & A_{z4} \\ A_{x5} & A_{y5} & A_{z5} \\ A_{x6} & A_{y6} & A_{z6} \end{bmatrix} = \begin{bmatrix} A_{rx1} & A_{ry1} & A_{rz1} & 1 \\ A_{rx2} & A_{ry2} & A_{rz2} & 1 \\ A_{rx3} & A_{ry3} & A_{rz3} & 1 \\ A_{rx4} & A_{ry4} & A_{rz4} & 1 \\ A_{rx5} & A_{ry5} & A_{rz5} & 1 \\ A_{rx6} & A_{ry6} & A_{rz6} & 1 \end{bmatrix} \times \begin{bmatrix} CC_{x1} & CC_{x2} & CC_{x3} \\ CC_{y1} & CC_{y2} & CC_{y3} \\ CC_{z1} & CC_{z2} & CC_{z3} \\ O_1 & O_2 & O_3 \end{bmatrix}$$

- And then you solve for X (Calibration Matrix) by using this equation (Use MATLAB):

$$X = \left[W^T W\right]^{-1} W^T Y$$

X = inv(W'*W)*W'*Y

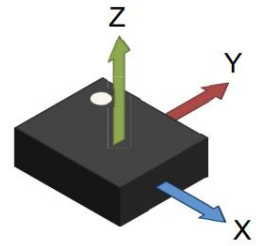
# Digital Sensor Calibration – Accelerometer Example (6)

- Suppose that while having the accelerometer positioned as shown in the adjacent figure, ideally we expect the readings to be [0 0 -1], however, the output of the accelerometer was recorded as:

Z = -0.9689 -1.0929 -0.9302 -0.9132 -0.9643 -0.9485 -0.9514 -1.0216 -0.9689 -1.0658

Y = -0.0123 -0.0237 0.0531 0.0590 -0.0626 -0.0020 -0.0109 0.0293 0.0419 0.0509

X = -0.0448 0.0359 0.0310 -0.0675 -0.0762 -0.0003 0.0919 -0.0319 0.0171 -0.0552



Direction of detectable acceleration (top view)

Then by taking the average of each of the raw data collected at this position:

$A_{rz}$ =-0.9826,

$A_{ry}$ = 0.0123,

$A_{rx}$ = -0.0100

This procedure should be repeated another five times for the remaining six positions, and then the we fill them in their respected locations in the matrices and in the correct order

# Digital Sensor Calibration – Accelerometer Example (7)

- Resulting equation could be something like this:

$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1.0249 & 0.0196 & -0.0222 \\ -1.0321 & -0.0016 & -0.0058 \\ 0.0016 & 0.9805 & -0.0157 \\ -0.0279 & -0.9774 & 0.0155 \\ -0.0177 & 0.0161 & 1.0135 \\ -0.0100 & 0.0123 & -0.9826 \end{bmatrix} \times \begin{bmatrix} CC_{x1} & CC_{x2} & CC_{x3} \\ CC_{y1} & CC_{y2} & CC_{y3} \\ CC_{z1} & CC_{z2} & CC_{z3} \\ O_1 & O_2 & O_3 \end{bmatrix}$$

- *Note: In this example, the values were generated by the **rand** function in MATLAB, and not from actual accelerometer trace. Thus, do not expect proper operation when applying the equation matrix.*

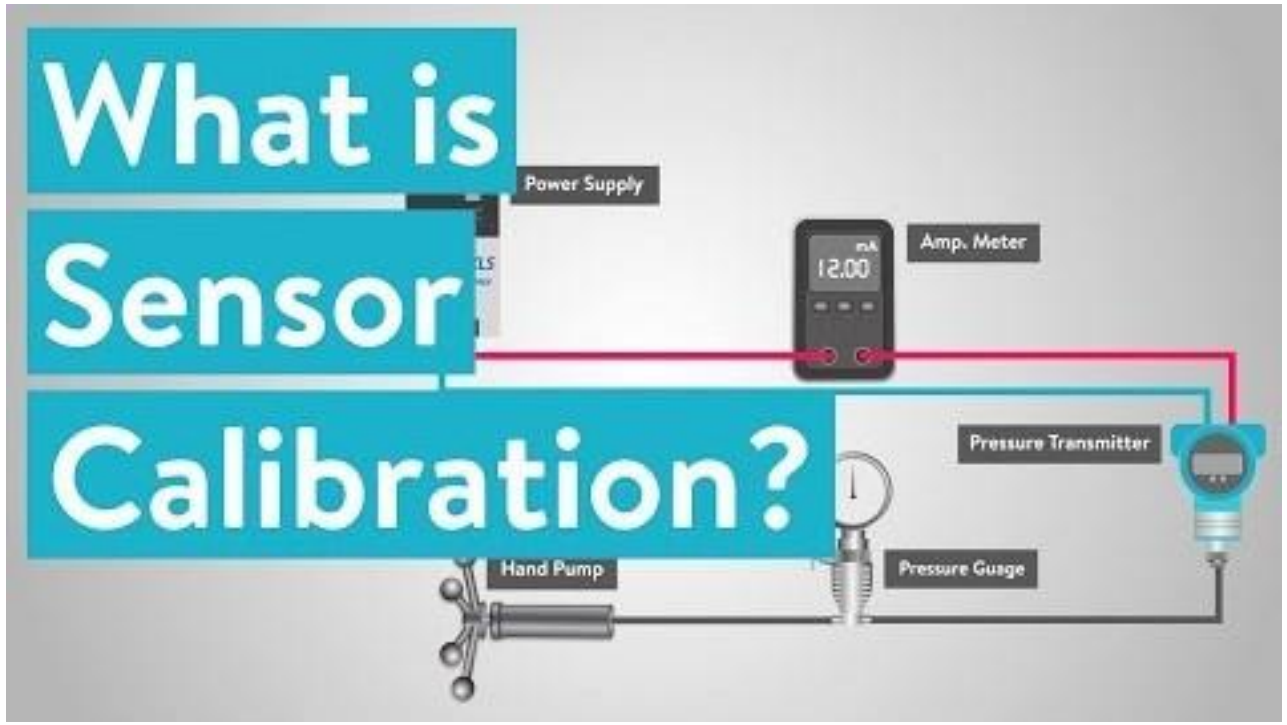
# Digital Sensor Calibration – Accelerometer Example (8)

- Solving for the coefficients matrix:

$$\begin{bmatrix} CC_{x1} & CC_{x2} & CC_{x3} \\ CC_{y1} & CC_{y2} & CC_{y3} \\ CC_{z1} & CC_{z2} & CC_{z3} \\ O_1 & O_2 & O_3 \end{bmatrix} = \begin{bmatrix} 0.9724 & -0.0105 & 0.0080 \\ -0.0146 & 1.0215 & 0.0158 \\ 0.0039 & -0.0021 & 1.0015 \\ 0.0100 & -0.0085 & -0.0005 \end{bmatrix}$$

- So now suppose that we have a new reading [ 1.05 0.05 - 0.09], if we multiply it by the first three rows of the coefficients matrix and then add the offsets to each, the reading will become [1.0300 0.0317 -0.0815].
- Evaluating the magnitude of the vector $\sqrt{A_x^2 + A_y^2 + A_z^2}$ we find that the magnitudes of the input data and the calibrated data are 1.055 and 1.0337, respectively.
- Calibration has reduced the error by approximately 2%.

# Summary Video on Sensor Calibration



# Filters (Anti-Aliasing and Digital)

- Never forget Nyquist–Shannon sampling theorem:
  *If a signal x(t) contains no frequencies higher than B Hz, then perfect reconstruction is guaranteed for a given sample rate $f_s$ when $f_s > 2B$*
- The signal x(t) might be subject to high frequency noises such that x(t)' = x(t) + r(t)
  where r(t) is a random noise signal with frequencies higher than B → This causes artificats!
- High frequency noise must be filtered out **prior** to ADC sampling.
- We use a low pass filter which is called Anti-Aliasing Filter (AAF).
- The Anti-Aliasing Filter is an analogue filter placed before the ADC channel!

# Anti-aliasing while using Digital MEMS Sensors

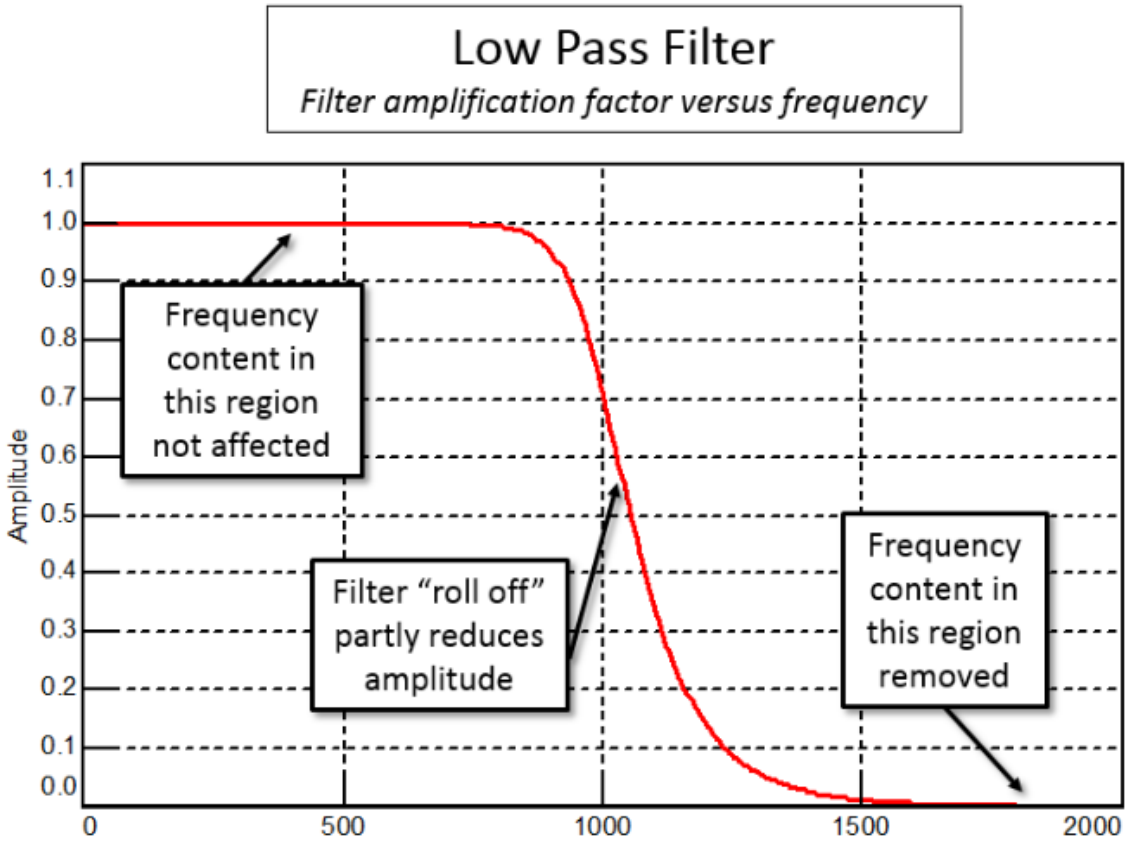
- Digital Sensors have internal ADCs. Interface to the controller through SPI or I2C.
- Usually configurable for multiple sampling rates.
- Usually *no anti-aliasing filter circuitry* between the sensing element and the internal ADC (check datasheets).
- If one is using a digital sensor, one must be careful in configuring the ODR (Output Data Rate (i.e. sampling rate)) of the sensor.
- Designers tend to oversample by selecting higher ODR than needed, the deal with the noise by digital filters.
- Suppose that B = 49 Hz, and maximum noise frequency is 95Hz
  1. If you configure the ODR to be 100Hz, then the noise will cause folding and aliasing in the original signal.
  2. If you configure the ODR to be 200Hz, then you are getting a faithful reproduction of the signal and the noise component without aliasing/folding onto the original signal → Smooth and filter noise using digital sensor!

# The Low Pass Filter

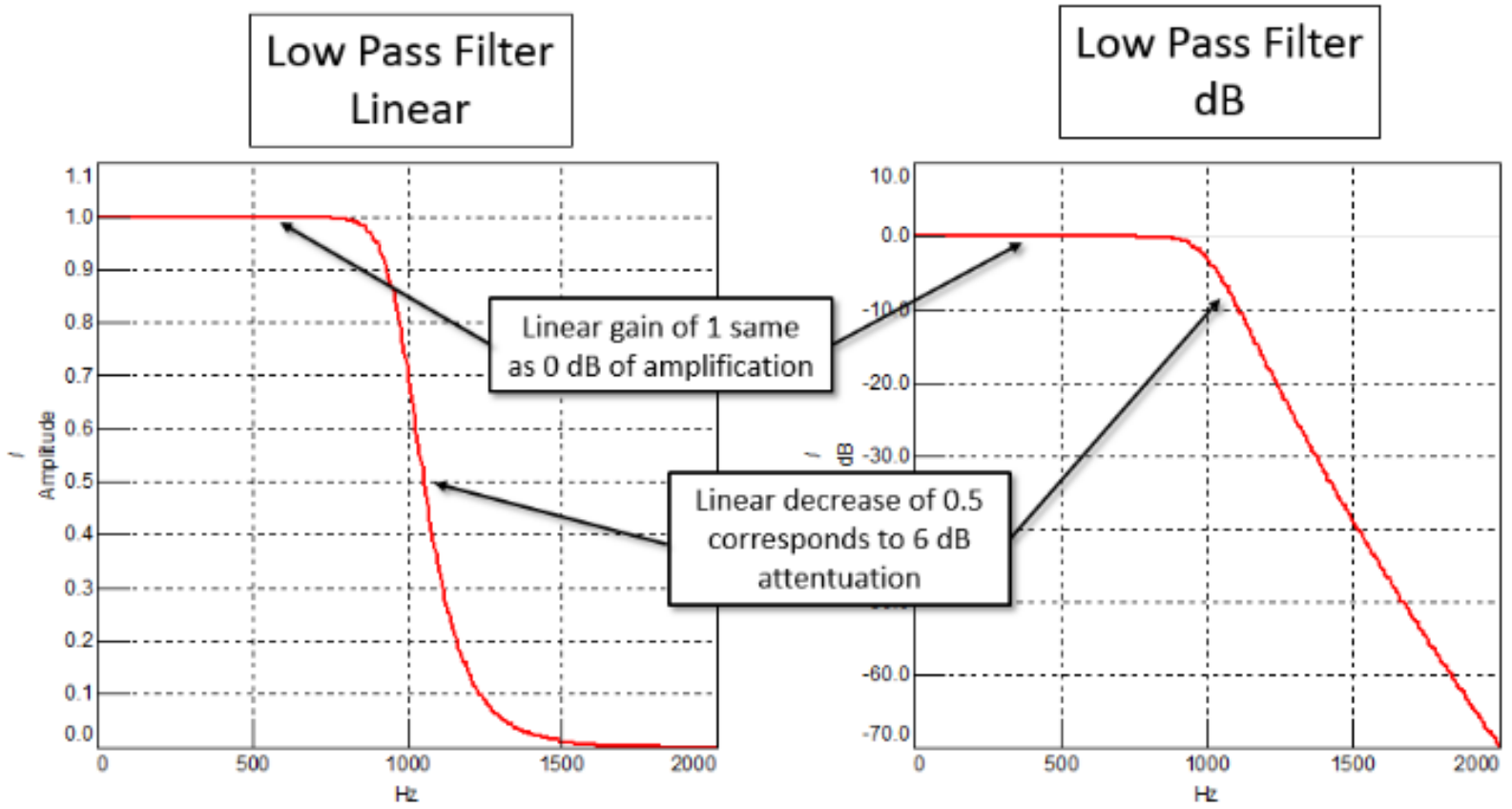
- A low pass filter passes low frequencies unaltered (left) and removes high frequencies (right)
- Notice the three regions:
  1. Unaltered frequencies 0 – 800Hz
  2. Roll off region 800 - 1500
  3. Filtered out region > 1500 (cut-off frequency)

Can convert to Decibel (dB) graph by using

$$\mathbf{20\ log_{10}(x/x_{ref})}$$

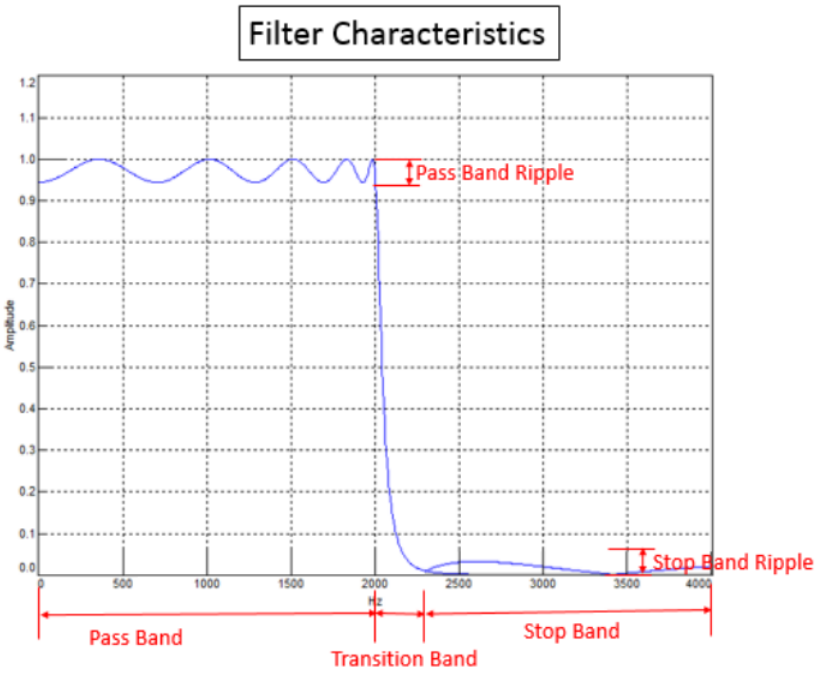


# The Low Pass Filter – Linear vs dB Graph



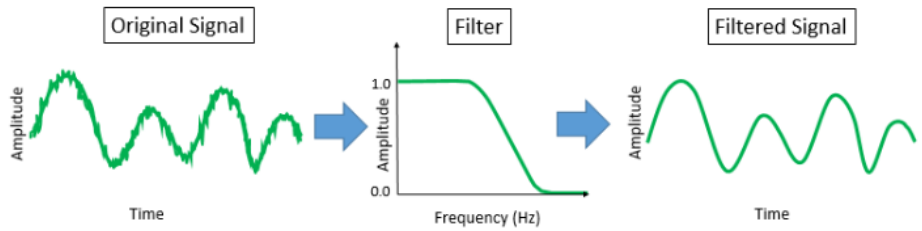
# Filter Attributes

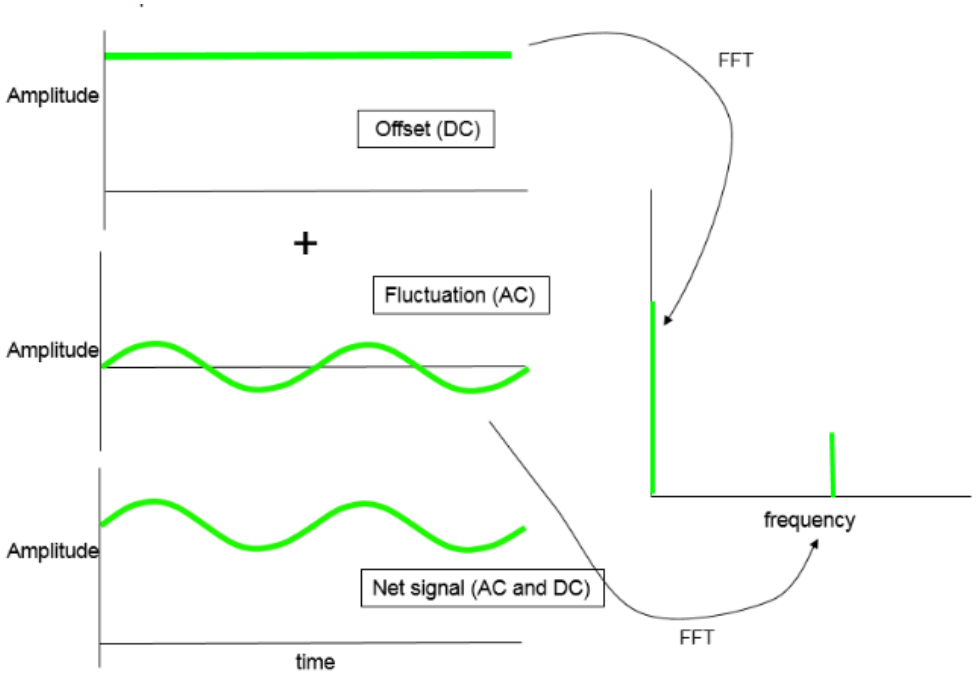
- The filter attributes depend on the type of the filter:
- **Pass Band** – Ideally, in this band, the filter should have an amplitude of exactly one. To ensure the data in the pass band is identical to the original time history data, there should be no ripple in the filter. Ripple is a slight variation in amplitude as function of frequency. However; in many case, this is not avoidable.
- **Transition Width (roll-off)** – Depending on the application, it might be desirable to have the transition between the pass and stop bands be as narrow as possible in terms of frequency. Both the method and the filter order determine how rapidly the transition between pass and stop bands occurs.
- **Stop Band** – The stop band could also contain data if the filter has ripple (e.g. IIR filters). In some applications, the amplitude may be so small it does not matter. In others, the ripple might not be acceptable.
- **Group Delay/Phase** – Filters create a delay in the output time history, and this can even vary as a function of frequency.



# Applications of Filters

- Anti-Aliasing Filter – An anti-aliasing filter is used to remove signal content that cannot be properly digitized before Analog-to-Digital conversion.
- Drift Removal – Drift or large offsets can be removed from a signal via a high pass filter or AC coupling.
- Noise Removal – Filters could be used to remove unwanted high frequency noise from a signal, for example, a hiss in a musical recording.



# Types of Digital Filters - Definitions

- Finite Impulse Response filters (FIR)
- Infinite Impulse Response filters (IIR)
- Remember the impulse function **$\delta(t)$** → Dirac Delta Function which is a very short burst with high amplitude.
- Think of it as being in a wide room, or over a top of adjacent hills. Suddenly shouting your name very loudly is like a impulse function. The echos of your name is how the system (in this case, room design, or physical formation of hills) responds to your sudden loud shout.
- A finite impulse response means that the response does not last forever.
- Infinite response means that at least mathematically, the response can go forever (e.g. infinite ripple)

# Types of Digital Filters - Equations

FIR Filter Equation: $y(n) = \sum_{k=0}^{N} a(k)x(n-k)$

IIR Filter Equation: $y(n) = \sum_{k=0}^{N} a(k)x(n-k) + \sum_{j=0}^{P} b(j)y(n-j)$

Output used recursively

- ***x(n)*** is the incoming data samples series, where ***x(1)*** is the first sample, and ***n*** the total number of samples (the time difference between ***x(n)*** and ***x(n+1)*** is the sampling time $1/f_s$
- ***a(k)*** and ***b(k)*** are the filter coefficients. The value of these coefficients dictate different types of filters within the same category.
- ***y(n)*** is the output filtered data samples series.
- ***N*** and ***P*** are the number of filter "taps" (i.e. filter width). These correspond to the number of coefficients in the filter. They would affect the filter output delay where

$$\text{Delay} \propto N, P$$

- The higher the number N, the higher the order of the filter (N= 1→ First order filter, N = 10 → $10^{th}$ order filter, etc)
- ***Filtering is a convolution process***

# Unrolling the FIR Equation

FIR Filter Equation: $y(n) = \sum_{k=0}^{N} a(k)x(n-k)$

For n = 0, k = 0 to N:

$y(0) = a(0)x(0) + a(1)x(-1) + a(2)x(-2) + a(3)x(-3) \ldots .$

For n = 1, k = 0 to N:

$y(1) = a(0)x(1) + a(1)x(0) + a(2)x(-1) + a(3)x(-2) \ldots .$

For n = 2, k = 0 to N:

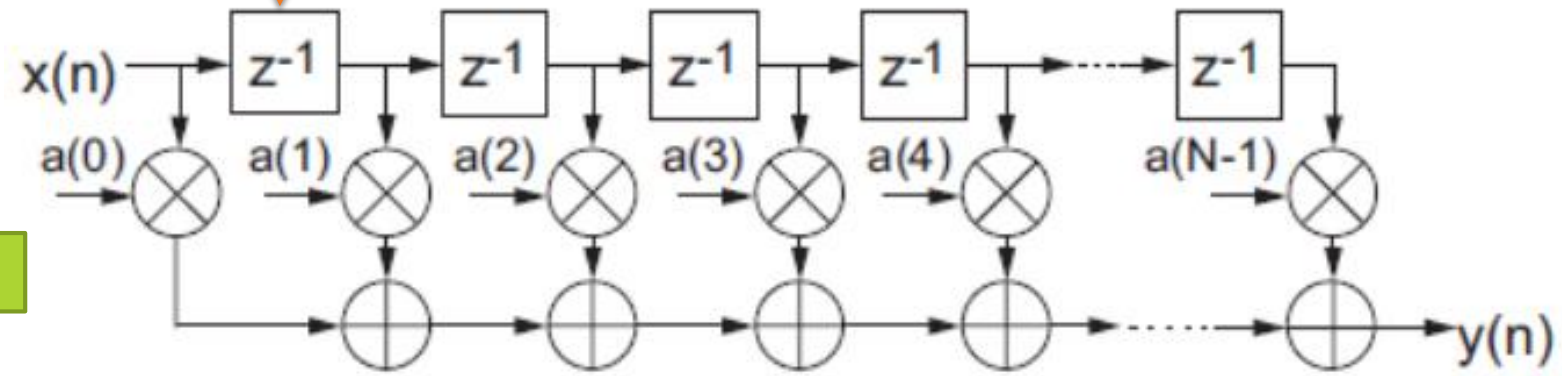
$y(2) = a(0)x(2) + a(1)x(1) + a(2)x(0) + a(3)x(-1) \ldots .$

Terms highlighted in orange do not exist, filter does not become fully realized until n > N

# FIR Tapped-Delay Design

Unit Delay Time

Feed-Forward Equation




Multiplication

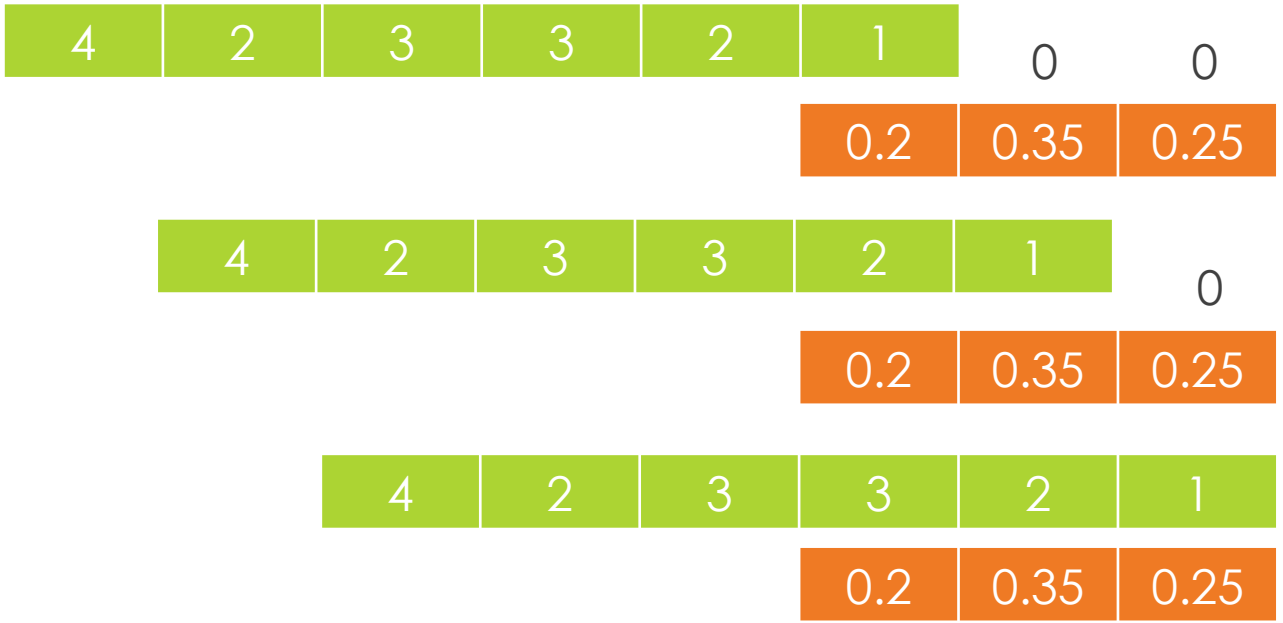
Summation

Remember, MATLAB indices start at 1, C/C++ and other languages start at 0

# FIR Example (1)

- Suppose x(n) = [1, 2, 3, 3, 2, 4] and suppose N = 2, where a(0) = 0.2, a(1) = 0.35, and a(2) = 0.25

1st Sample, don't confuse the order of inputs




$y(0) = 0.2$

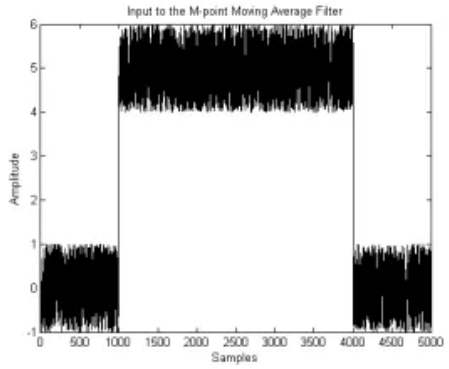
$y(1) = 2\times0.2 + 1\times0.35 = 0.75$

$y(2) = 3\times0.2 + 2\times0.35 + 0.25 = 1.55$

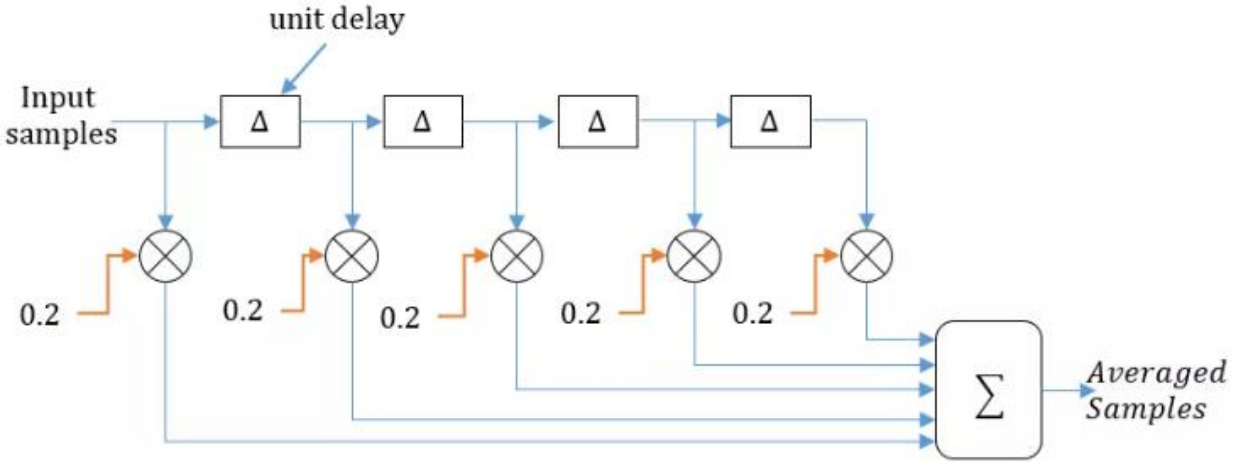
# FIR Example (2)

# A Special Case of the FIR – Moving Average

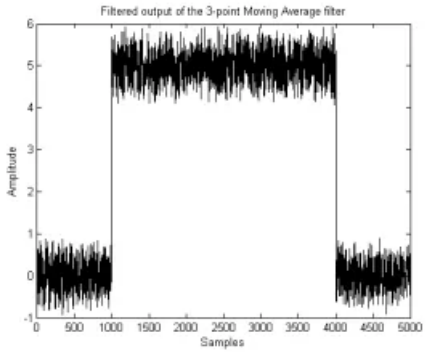
- The Moving Average Filter is a special case of the low pass FIR where the filter coefficients are all the same and are equal to 1/N.
- This is basically taking the average of the most recent N sensor readings (smoothing operation).
- The rule is: Good performance in the time domain, poor performance in the frequency domain.
- Susceptible to rare events (e.g. rapid and large momentary change) → Try median filter instead!



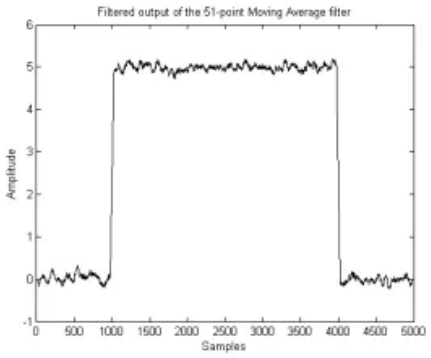

Input to Moving average filter



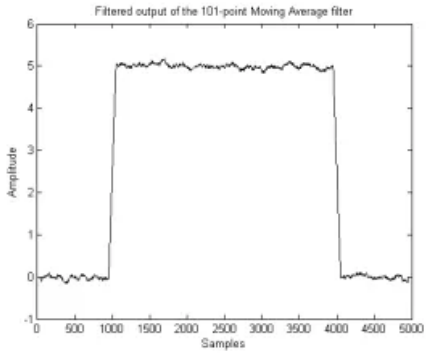
# Moving Average Filter - Case Study



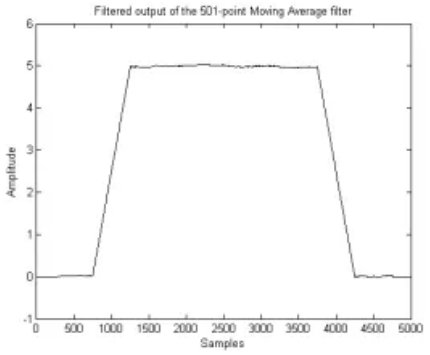

Response of 3 point Moving average filter



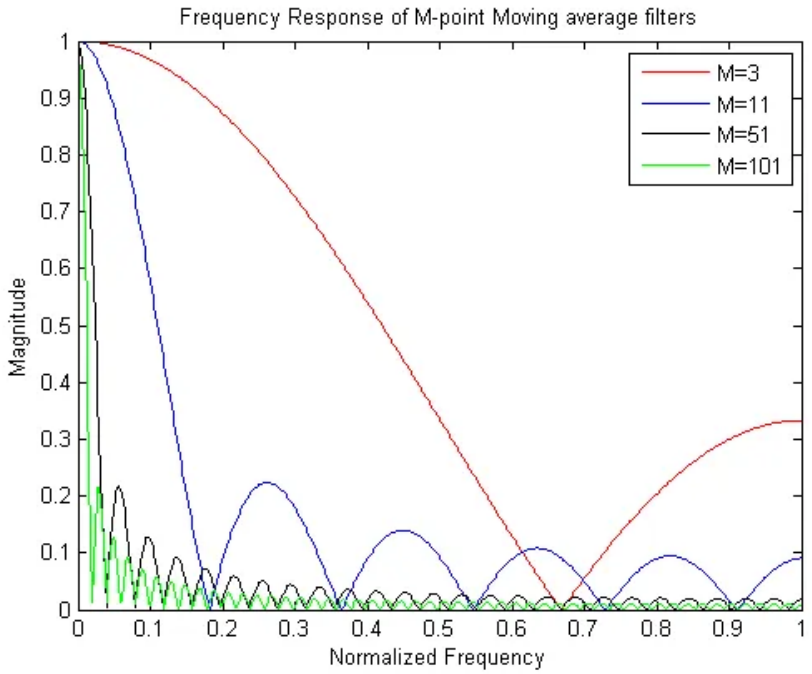

Response of 51-point Moving average filter




Response of 101-point Moving average filter



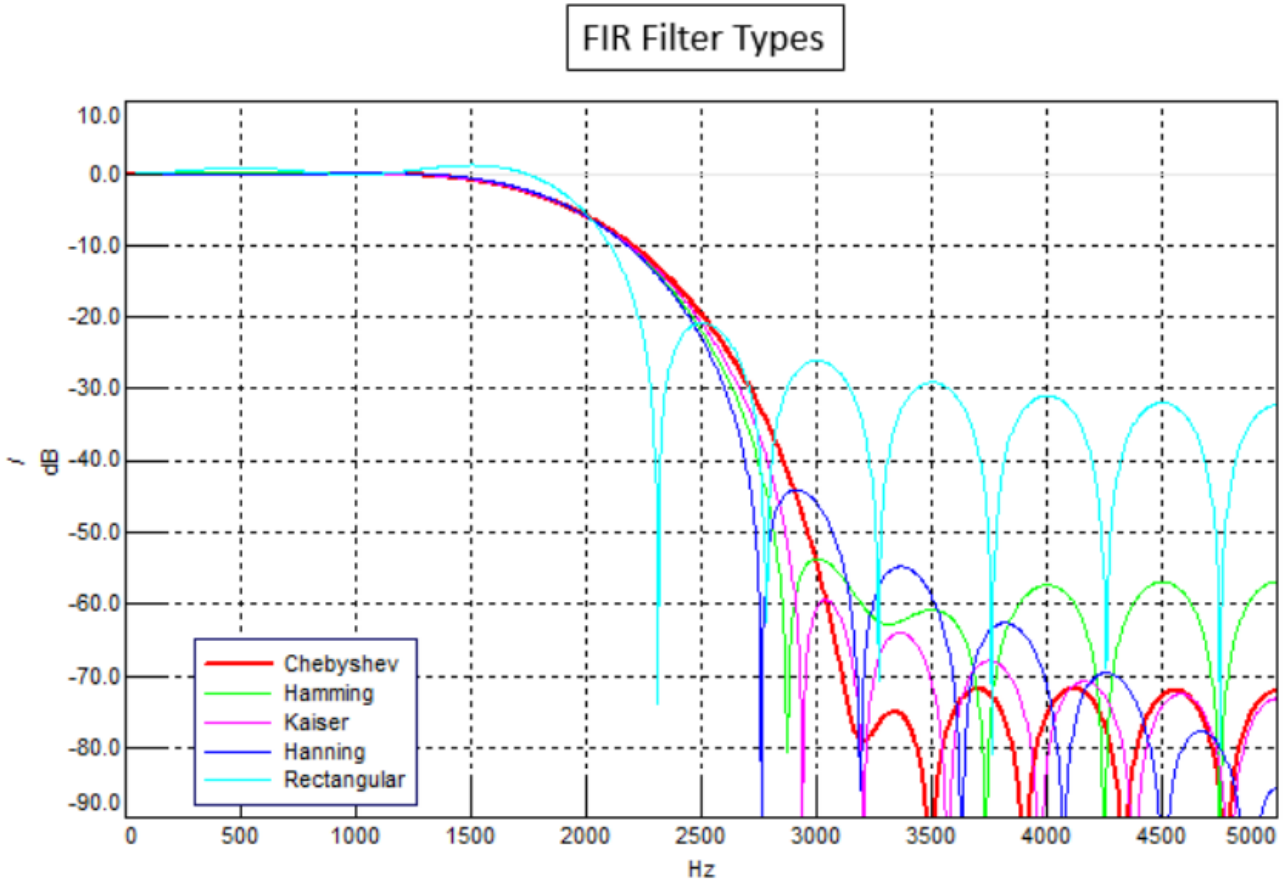

Response of 501 point Moving average filter




The roll-off is very slow and the stopband attenuation is ghastly

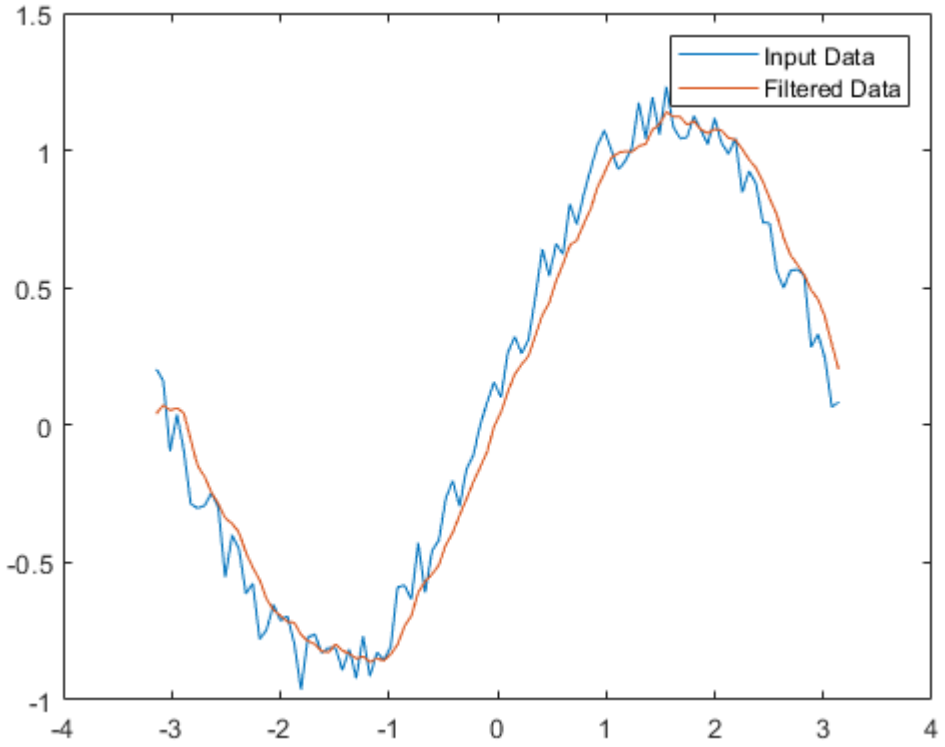
# FIR Filter Methods

- **Chebyshev** – Has the lowest amount of ripple in stop band, but widest transition band.
- **Hamming** – Narrow transition zone, smaller ripple than Hanning.
- **Kaiser** – The Kaiser window has small amplitude ripple in stop zone, only the wide transition width Chebyshev has lower amplitude ripple.
- **Hanning** – Narrowest transition band, but large ripple in stop band.
- **Rectangular** – Largest amount of ripple/lobes, even affects pass band.



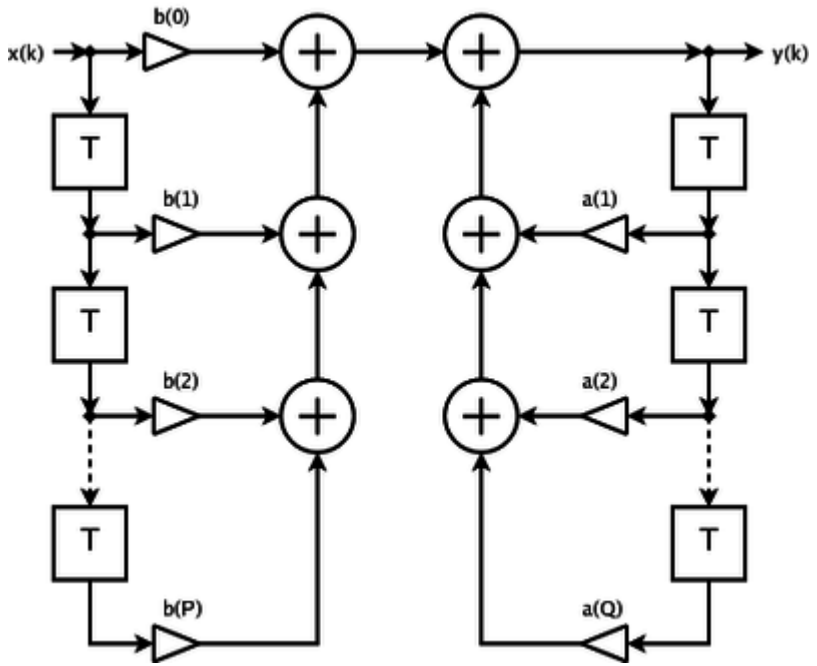
# FIR Implementation in MATLAB

```
t = linspace(-pi,pi,100);
rng default  %initialize random number generator
x = sin(t) + 0.25*rand(size(t));
windowSize = 5;
b = (1/windowSize)*ones(1,windowSize);
a = 1;
y = filter(b,a,x);
plot(t,x)
hold on
plot(t,y)
legend('Input Data','Filtered Data')
```



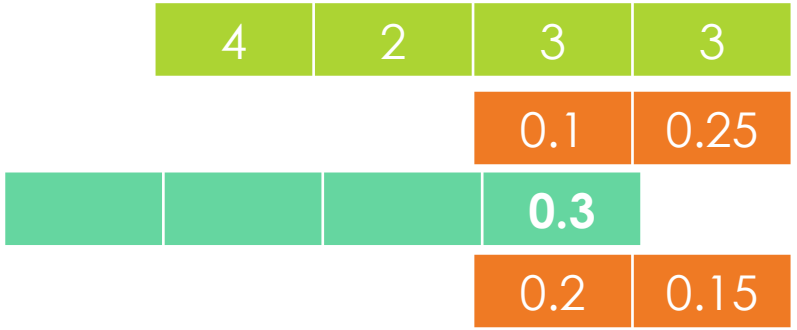
# Infinite Impulse Response Filters (IIR)

- IIR Filters use previous outputs in a feedback loop to process the new current output (i.e. IIRs have “memory”).
- The filter impulse response never becomes exactly 0 but rather approaches it.
- Main issue is the stability of such filters.
- BIBO stability (Bounded Input, Bounded Output)
- Due to the feedback loop, the output could grow toward ∞
- IIR filters must have stability conditions (imposed by the choice of filter coefficients)



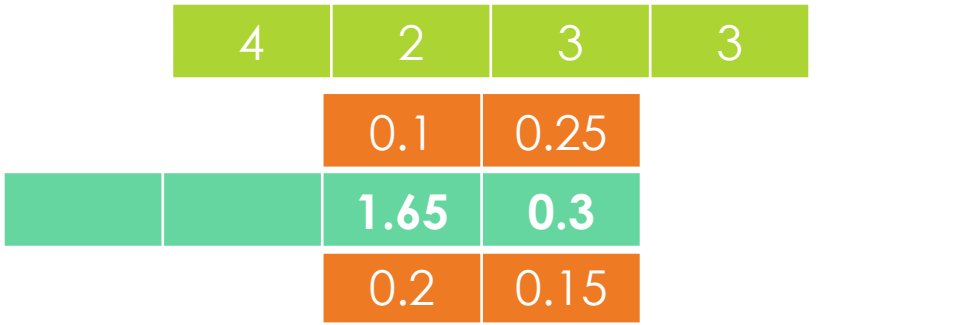
# IIR Filter Example with N, P = 2 (1)


$y(0) = 3 \times 0.1 = 0.3$

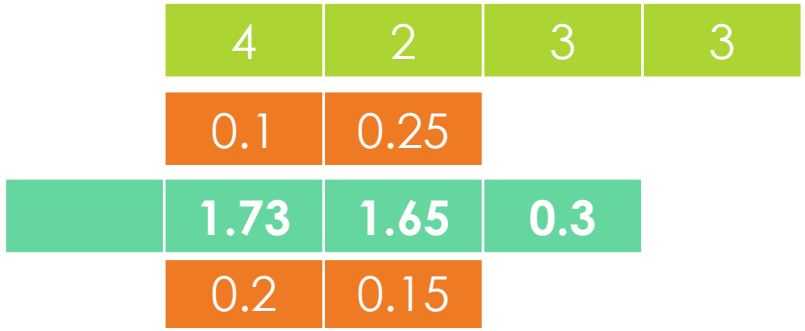



$y(1) = 3 \times 0.25 + 3 \times 0.1 + 0.3 \times 0.2 = 1.65$

# IIR Filter Example with N, P = 2 (2)




$y(2) = 3\times0.25 + 2\times0.1 + 0.3\times0.15 + 1.65\times0.2 = 1.73$




$y(3) = 2\times0.25 + 4\times0.1 + 1.65\times0.15 + 1.73\times0.2 = 1.4935$

# IIR Filter Example with N, P = 2 (3)

| | 4 | 2 | 3 | 3 |
|---|---|---|---|---|
| 0.1 | 0.25 | | | |
| **1.49** | **1.73** | **1.65** | **0.3** | |
| 0.2 | 0.15 | | | |

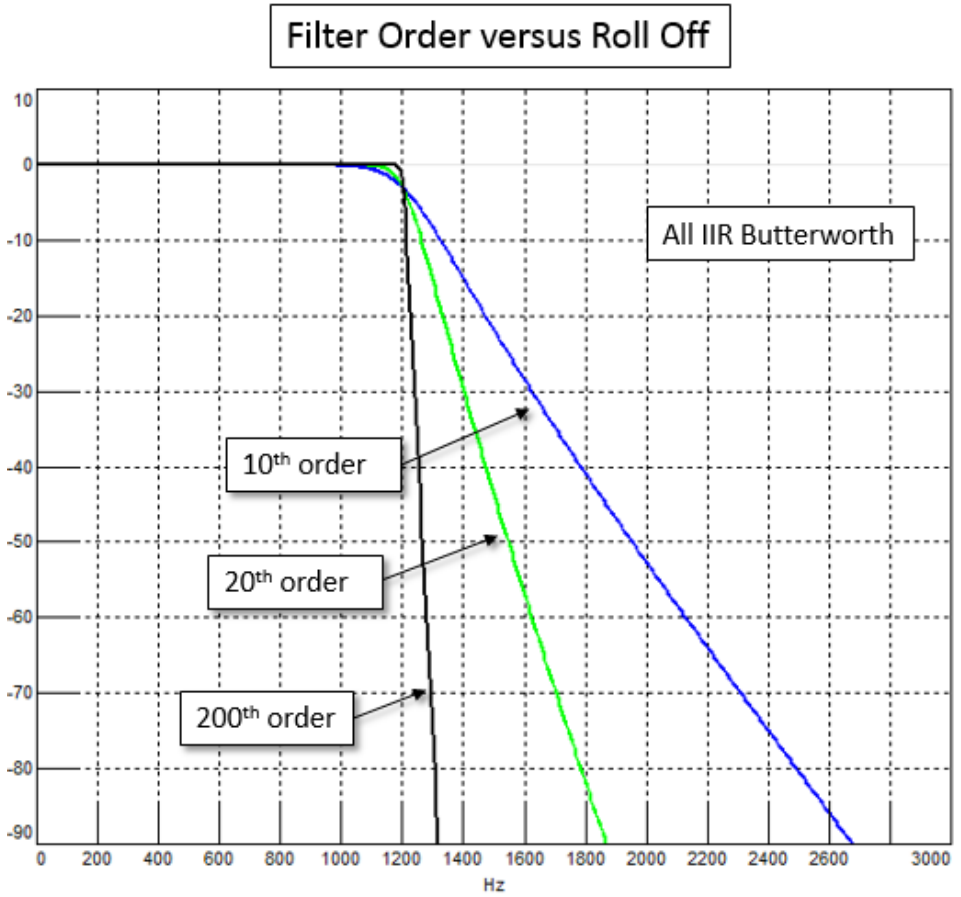
$y(4) = 4\times0.25 + 1.73\times0.15 + 1.49\times0.2 = 1.5575$

| | | 4 | 2 | 3 | 3 |
|---|---|---|---|---|---|
| 0.1 | 0.25 | | | | |
| | **1.49** | **1.73** | **1.65** | **0.3** | |
| 0.2 | 0.15 | | | | |

$y(5) = 1.49\times0.15 = 0.2235$

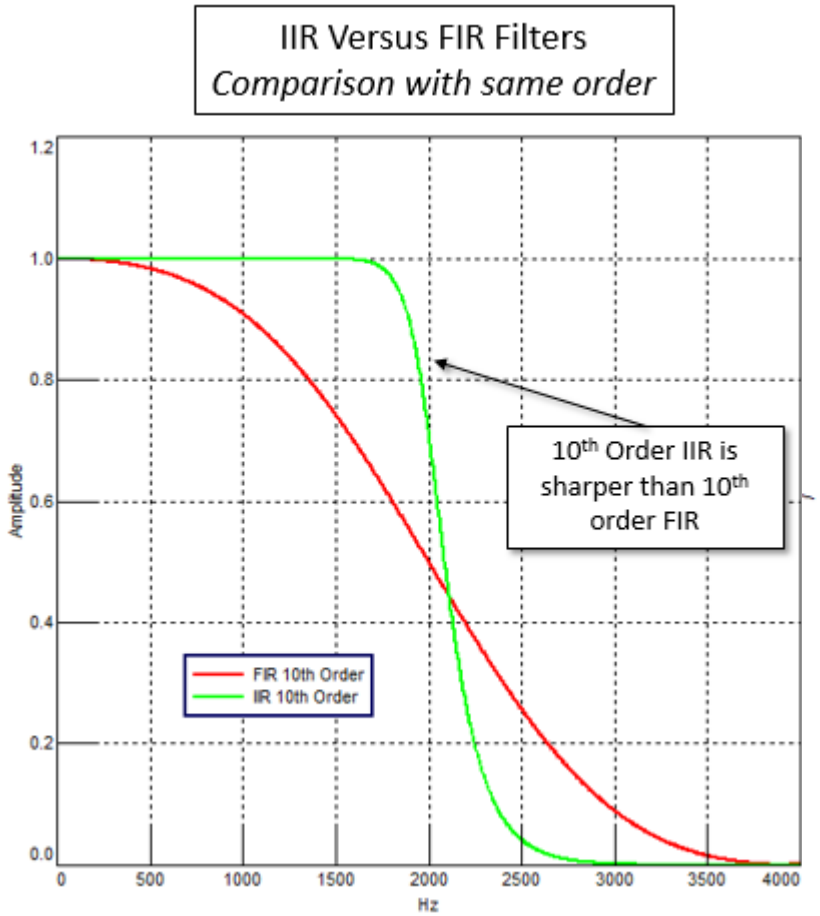
# FIR and IIR Filters Order vs Roll-off

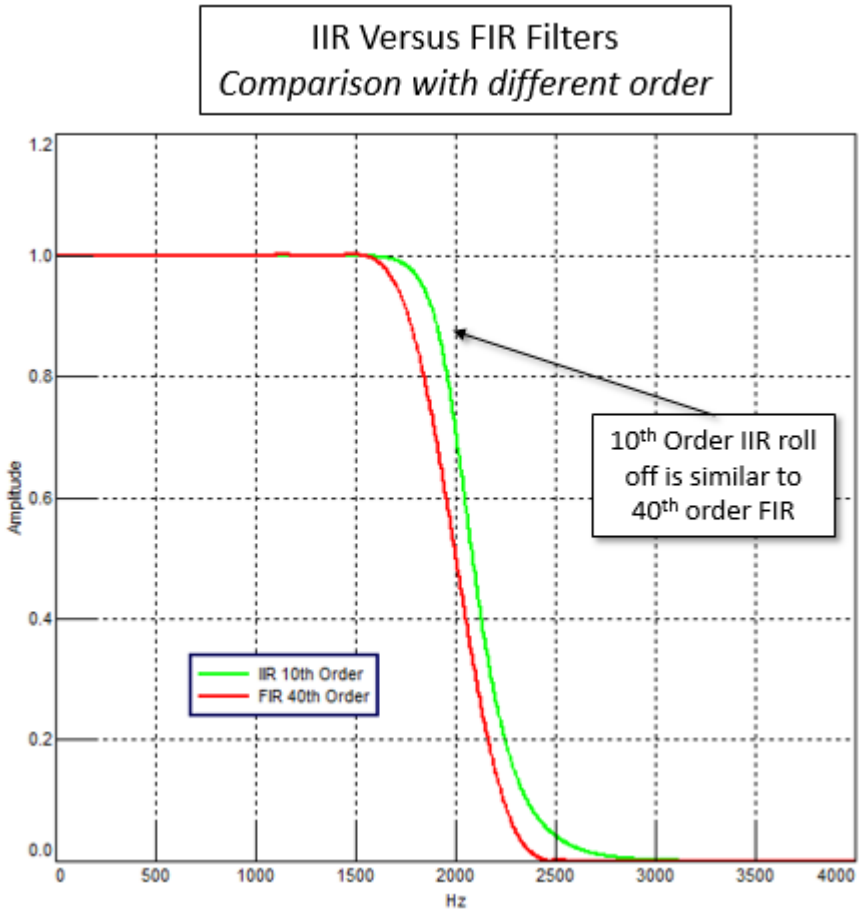
- Increasing the order of filter, but keeping all else the same, increases the sharpness of the filter roll off → **sharper transition** between the frequencies being passed and the frequencies being stopped.
- Sharper roll-off requires more calculations (more taps to do the math for).
- Impacts the filter time delay



# FIR vs IIR Order impact on Roll-off

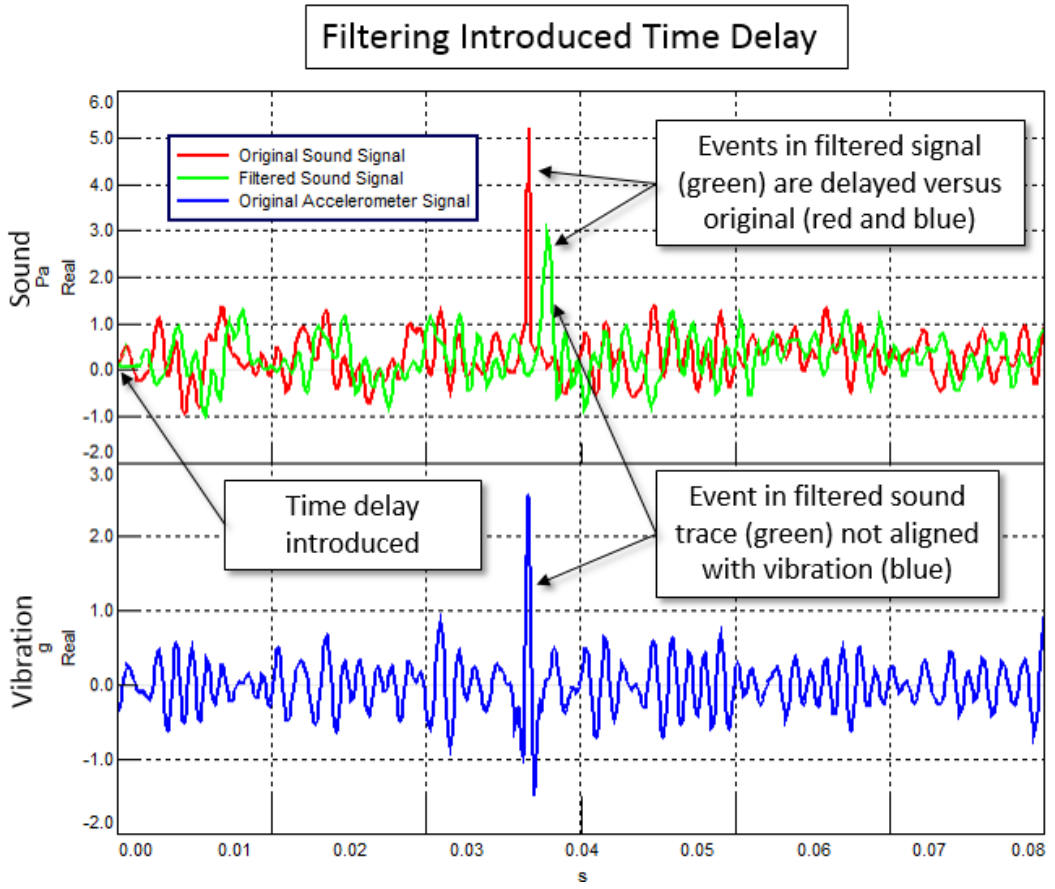
- Due to the recursive and feedback nature of IIR filters, they achieve sharper roll-off at the same order as that of an FIR filter.
- IIR filters faster from a computational point of view than FIR filters.
- IIR ***might*** be better for real-time applications



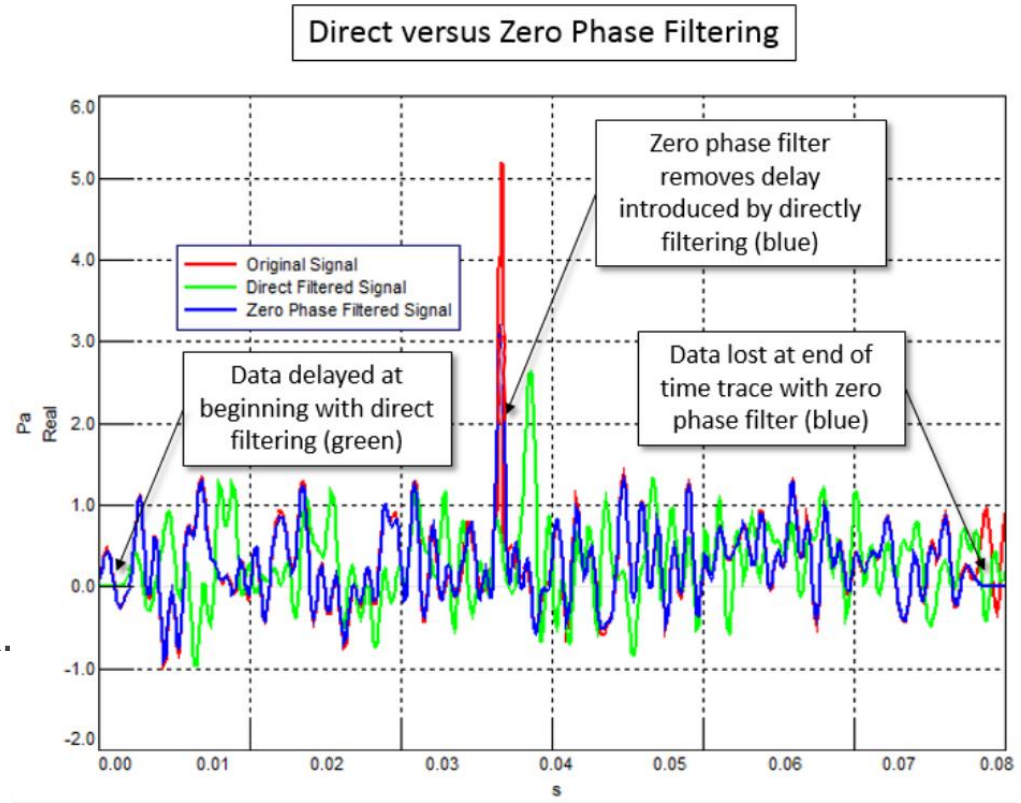
# Filters and Time Delays

- After applying a filter to time data, and comparing to the original time history, a slight shift, or time delay, can be seen in the data.
- If the offset is known → no problem in some applications
- The higher the filter order, the longer the delay → Could be an issue in real-time control applications.
- Suppose you have different sensor streams, each with different filter designs → Misalignment



# Eliminating Delay - Zero Phase Filtering

- The delay in the output time history can be eliminated by filtering the data ‘forward and backward’.
- After the signal(n) is filtered, and the new signal y(n) is created, it can be fed backward into the filter.
- The data points in y(n) are reversed in order in time to do this, and fed into the filter again. This is referred to as “zero phase filtering”
- Works well if you are working offline on signal traces, but does not work at all in online real-time (control) application.

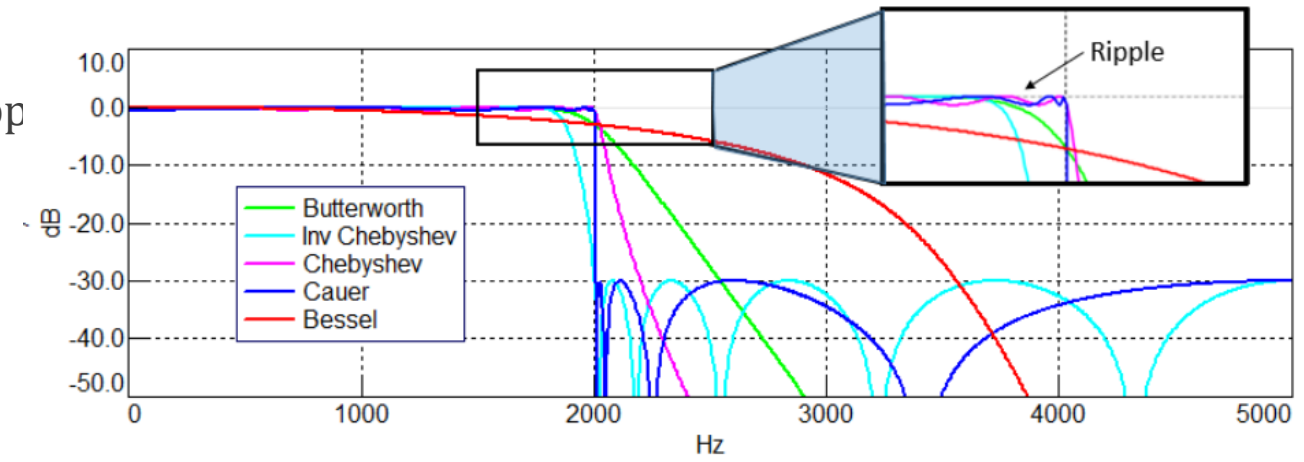


# Types of IIR Filters

- **Butterworth** – Flat response in both the pass and stop band, but has a wide transition zone.
- **Inverse Chebyshev** – Flat in the pass band, with a narrower transition width than the Butterworth filter, but has ripple in the stop band. If ripple in the stop band is not a problem, might be preferred for a given application over the Butterworth filter.
- **Chebyshev** – Can have ripple in pass band, but has steeper rolloff than Inverse Chebyshev.
- **Cauer** – Narrowest transition zone. Ripple in both stop and pass bands. Sometimes called an **Elliptic** filter.
- **Bessel** – Sloping amplitude in both the pass and stop band, with a very wide transition zone. The delay versus frequency in the filter is the flattest in this list.

Comparison of IIR Filters

| Method | Pass Band | Transition Width | Stop Band |
|---|---|---|---|
| Butterworth | Flat | Wide | Monotonic |
| Inv. Chebyshev | Flat | Narrow | Ripple |
| Chebyshev | Ripple | Narrow | Monotonic |
| Cauer | Ripple | Narrowest | Ripple |
| Bessel | Sloping | Very Wide | Sloping |



# Final Words on FIR and IIR Filters

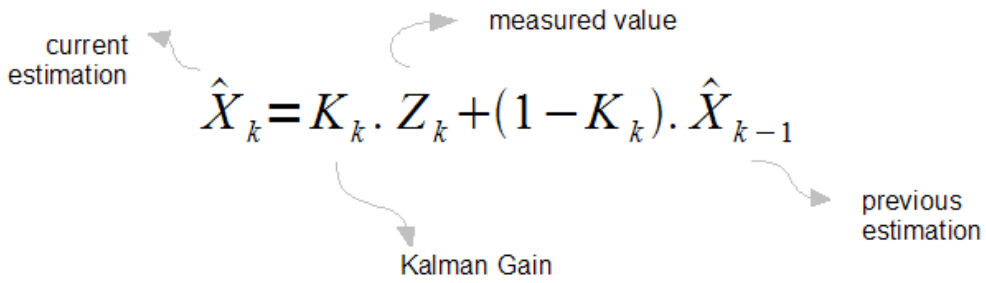
| | FIR | IIR |
|---|---|---|
| Computational Speed | Slow – High Order | Fast – Low Order |
| Phase/Delay | Constant | Non-Constant |
| Stability | Always | Sometimes |

# Stochastic Filters – The Kalman Filter

- The Kalman filter is one of the most powerful tools in engineering. Its also known as Linear Quadratic Estimation (LQE)
- The Kalman filter is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies.
- The Kalman filter produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe.
- The Kalman filter  is a state-based adaptive estimator of a physical process that is known to be optimal with respect to the estimation error when the noise is Gaussian and the system is linear.
- Widely used in control applications
- Some view its higher order versions quite mathematically challenging.
- Can be used for filtering (1D version), or sensor fusion (e.g. 2D version).
- The algorithm works in a two-step process: Estimate and Update

# The 1D Kalman Filter – Main Idea

- The equations of the Kalman filter involve matrices and matrix operations; however, when used on filtering noise of one variable, these equations could be extremely simplified (i.e. scalars rather than matrices).




- $\hat{X}_k, Z_k, K_k$ are systems states where $k$ is the sample number.
- We need to define and know each of these variables

# Kalman Filter – The Signal and its Measurement

- Any physical signal values includes noise $x'_k = x_k + r_k$ where $x_k$ is the original uncorrupted signal, and $r_k$ is white Gaussian noise.
- The measured values $Z_k$ have measurement uncertainty (dependant on the accuracy of the measurement tool), this measurement uncertainty can also be thought of or modelled as noise (measurement noise):

$$Z_k = x'_k + v_k$$

- For the Kalman filter to work, the signal noise and the measurement noise must be Gaussian *(most times they are).* Also, the signal noise and measurement noise must be statistically independent.
- Therefore, the value we read from the outside world can be expressed as:

$$Z_k = x_k + r_k + v_k$$

# The Iterative Time and Measurement Update Equations

- It is not easy to estimate the noise parameters. In the Kalman filter, we have three variables that are related to noise estimation: Q, P, and R. Each one of these variables relates to the uncertainty of our knowledge of the parameter it represent.
  - Q represents the uncertainty of our knowledge of the signal noise.
  - R represents the uncertainty of our knowledge of the measurement noise.
  - P represents the uncertainty of our knowledge of the estimation noise.
  - Mathematically, these variables represent the covariance of the signal, measurement, and estimation noise.
- Many times, even if your initial estimates are bad, the Kalman filter will still do quite well over time as it updates those estimates during operation.

| Time Update (prediction) | Measurement Update (correction) |
|---|---|
| $\bar{\hat{x}}_k = \bar{\hat{x}}_{k-1}$ | $K_k = \frac{\bar{P}_k}{\bar{P}_k + R}$ |
| $\bar{P}_k = P_{k-1} + \mathrm{Q}$ | $\hat{x}_k = K_k Z_k + (1 - K_k)\bar{\hat{x}}_k$ |
| | $P_k = (1 - K_k)\,\bar{P}_k$ |

# Kalman Filter Example

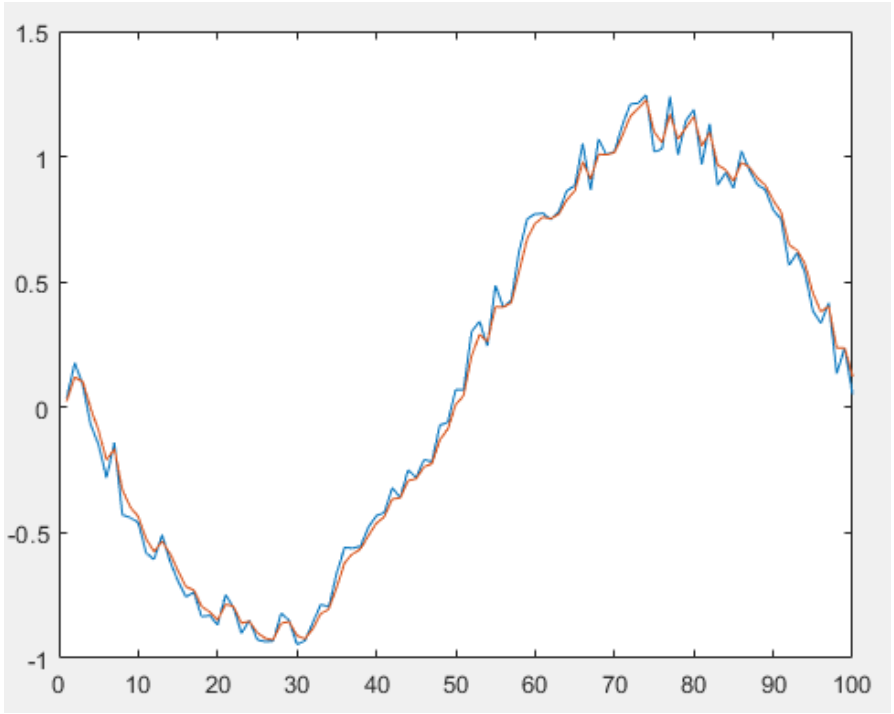
$P_0 = 1, Q = 1, R = 5$

$X_k = 0.0385 \quad 0.1770 \quad 0.0925 \quad -0.06709 \quad -0.1494 \quad -0.2803 \quad -0.1403 \quad -0.4284 \quad -0.4395$

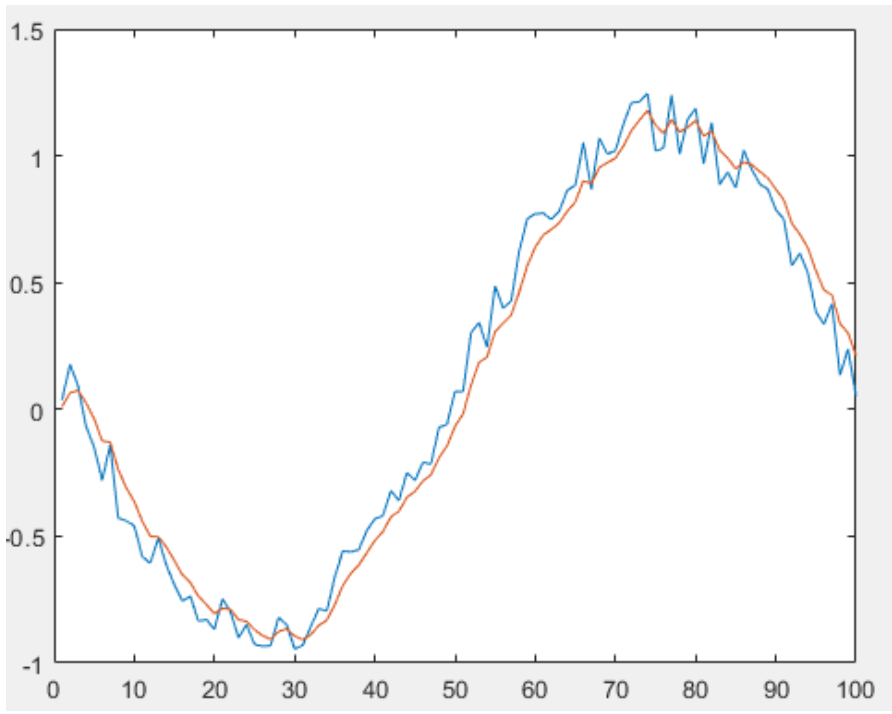
| $k$ | $Z_k$ | $\hat{\bar{x}}_{k-1}$ | $\bar{P}_k$ | Time Update | Measurement Update | $\hat{x}_k$ | $P_k$ |
|---|---|---|---|---|---|---|---|
| 1 | 0.0385 | 0 | 1 | $\hat{\bar{x}}_k = \hat{\bar{x}}_{k-1} = 0$<br>$\bar{P}_k = P_{k-1} + Q = 2$ | $K_k = \frac{\bar{P}_k}{\bar{P}_k + R} = 2/(2 + 5) = 0.2857$<br>$\hat{x}_k = 0.2857*0.0385 + (1 - 0.2857)*0 = 0.011$<br>$P_k = (1-0.2857)*2 = 1.4286$ | 0.011 | 1.4286 |
| 2 | 0.1770 | 0.011 | 1.4286 | $\hat{\bar{x}}_k = \hat{\bar{x}}_{k-1} = 0.011$<br>$\bar{P}_k = P_{k-1} + Q = 1.4286 + 1 = 2.4286$ | $K_k = 2.4286/(2.4286 + 5) = 0.3269$<br>$\hat{x}_k = 0.3269*0.177 + (1 - 0.3269)*0.011 = 0.0653$<br>$P_k = (1-0.3269)*2.4286 = 1.6347$ | 0.0653 | 1.6347 |
| 3 | 0.0925 | 0.0653 | 1.6347 | $\hat{\bar{x}}_k = \hat{\bar{x}}_{k-1} = 0.0653$<br>$\bar{P}_k = 2.6346$ | $K_k = 0.3451$<br>$\hat{x}_k = -0.1888$<br>$P_k = 1.7254$ | -0.1888 | 1.7254 |
| 4 | 0.06709 | -0.1888 | 1.7254 | $\hat{\bar{x}}_k = \hat{\bar{x}}_{k-1} = 0.1888$<br>$\bar{P}_k = 2.7254$ | $K_k = .352$<br>$\hat{x}_k = -0.1749$<br>$P_k = 1.7639$ | -0.1749 | 1.7639 |

# Kalman Filter MATLAB Example

```
function y =  kalman1d(X, P, Q, R)

K = 0;
x = 0;
y = zeros(1, length(X));
   for i = 1: length(X)
      P = P + Q;
      K = P / (P + R);
      x =  x + K*(X(i) - x);
      P = (1 - K)* P;
      y(i) = x;
   end
end
```

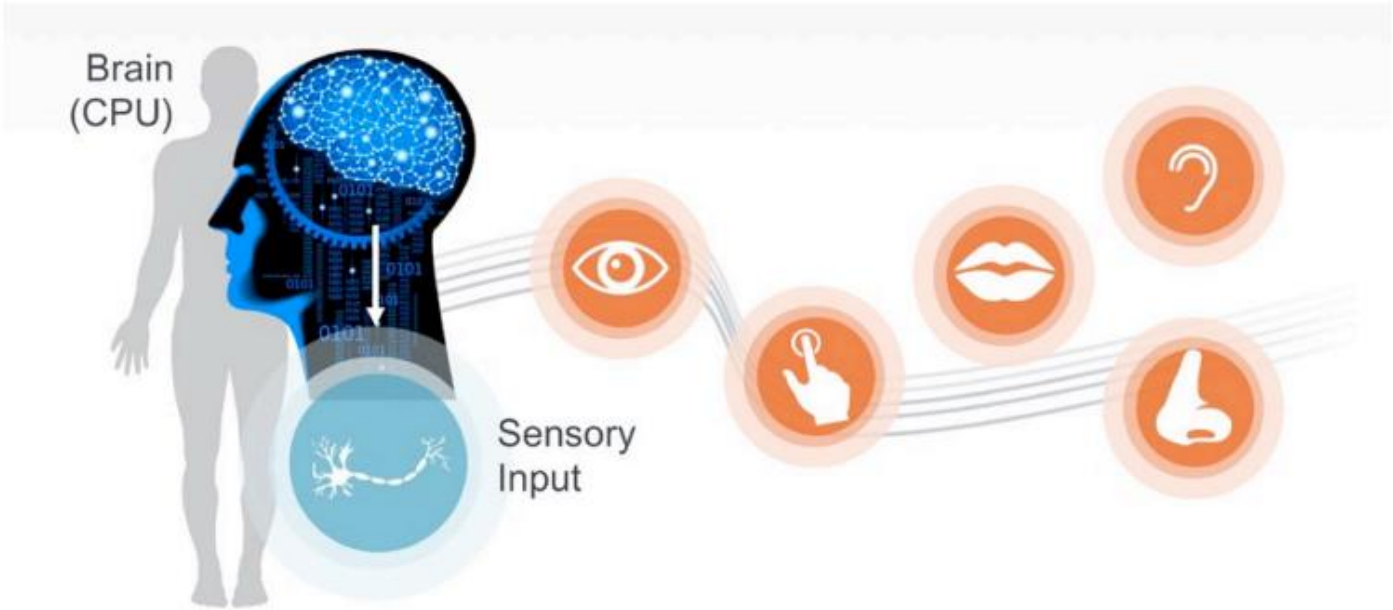



P = Q = R = 1




P = Q = 1,  R = 5

# The Human Model: The Ultimate Sensing Example




Sensory information (vision, hearing, smell, taste and touch) is gathered from one's surroundings and travels through the peripheral nervous system to the brain for processing and response.

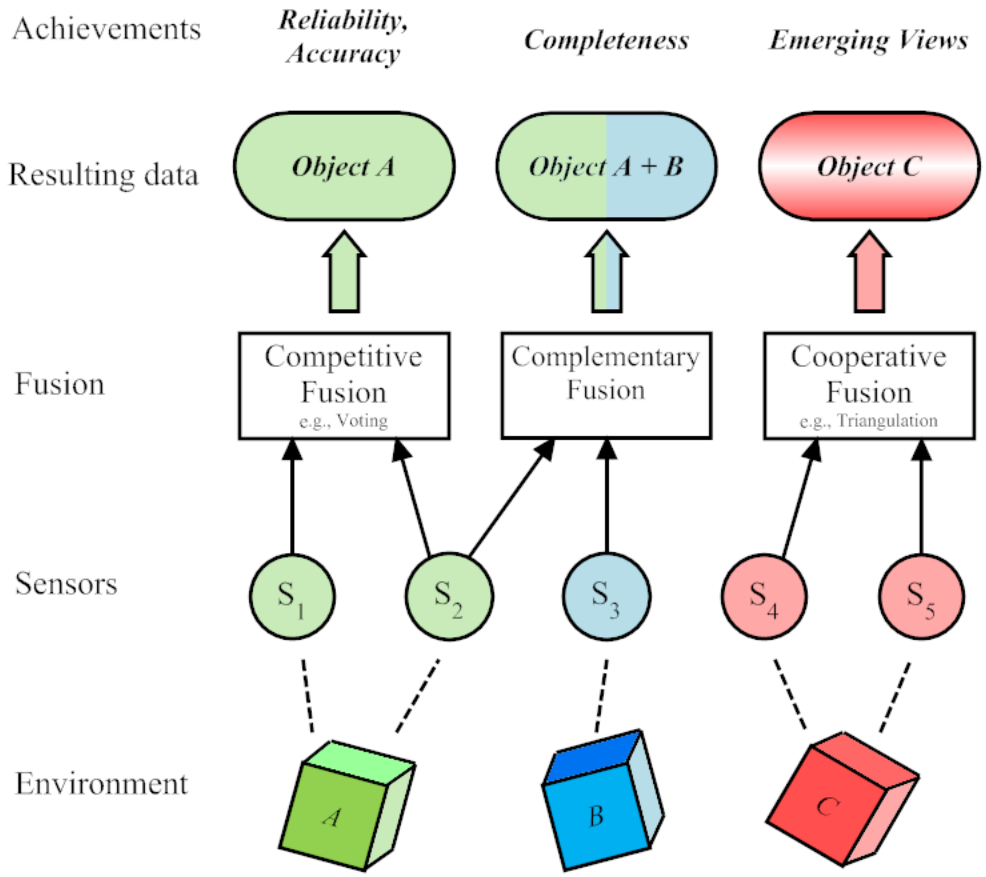
**"The Whole Is Greater Than the Sum of Its Parts"**

# Sensor Fusion – Gaming Platform

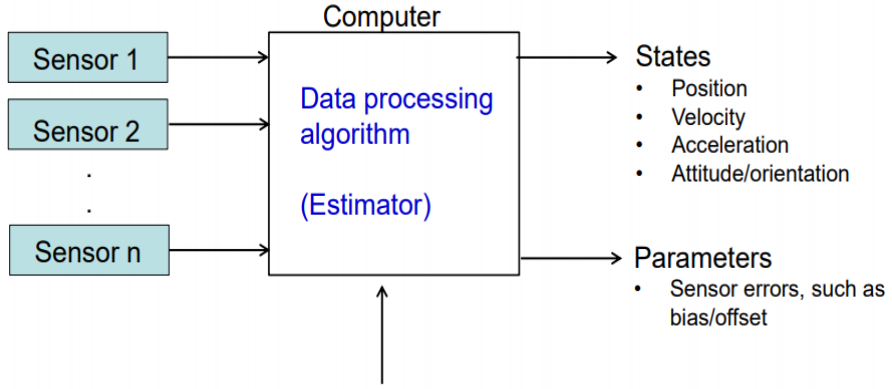
- Muscle relaxation (MR) and Muscle contraction (MC)—via a pressure sensor
- Heart rate variability (HRV)—via a two-electrode ECG on a chip
- Sweat (S)—via a capacitive sensor
- Attitude (A)—via an accelerometer monitoring a person's state of relaxation (jerky movements vs. steady hands)

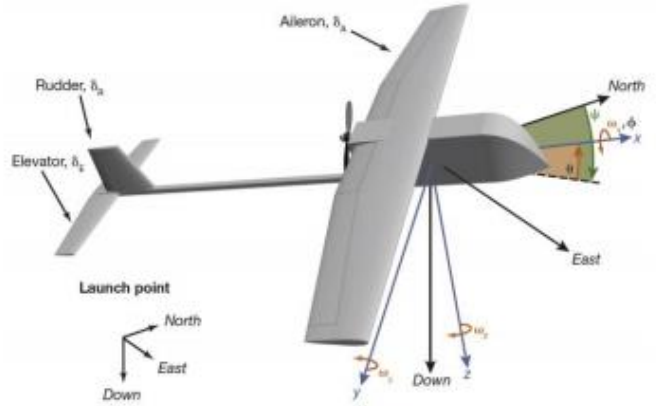
# Sensor Fusion Overview

- Three major types of sensor fusion:
  1. Competitive Fusion / Redundant: two or more input sources provide information about the same target and could thus be fused to increment the confidence.
  2. Complementary Fusion: the information provided by the input sources represents different parts of the scene and could thus be used to obtain more complete global information (two cameras (scenes), Acc + Gyro, etc)
  3. Cooperative Fusion: when the provided information is combined into new information that is typically more complex than the original information. For example, multi-modal (audio and video) data fusion is considered cooperative.



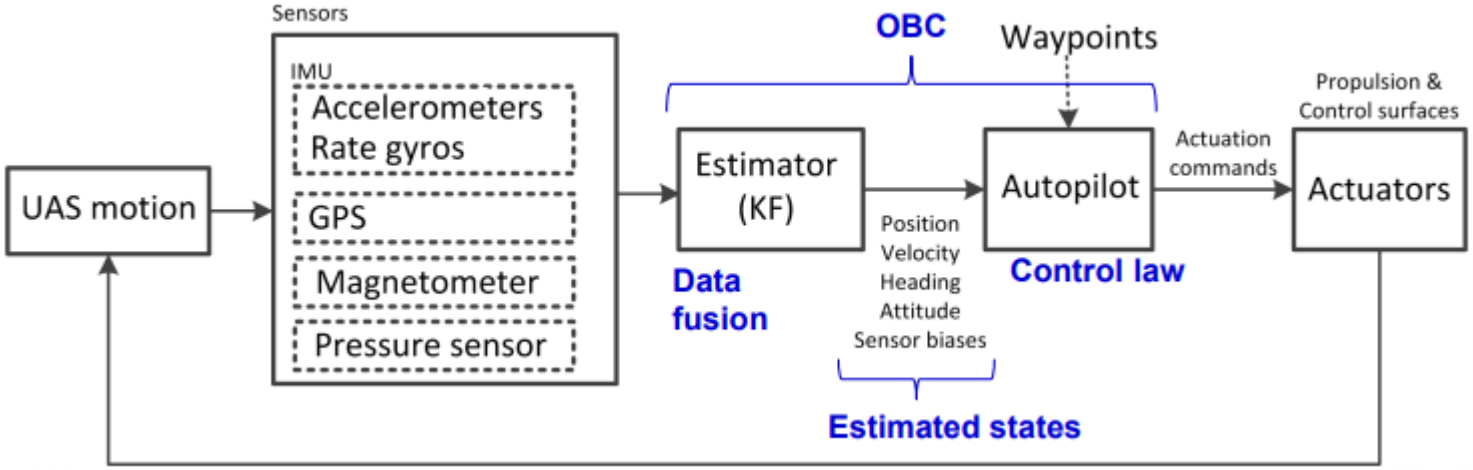
# Multi-sensor Systems



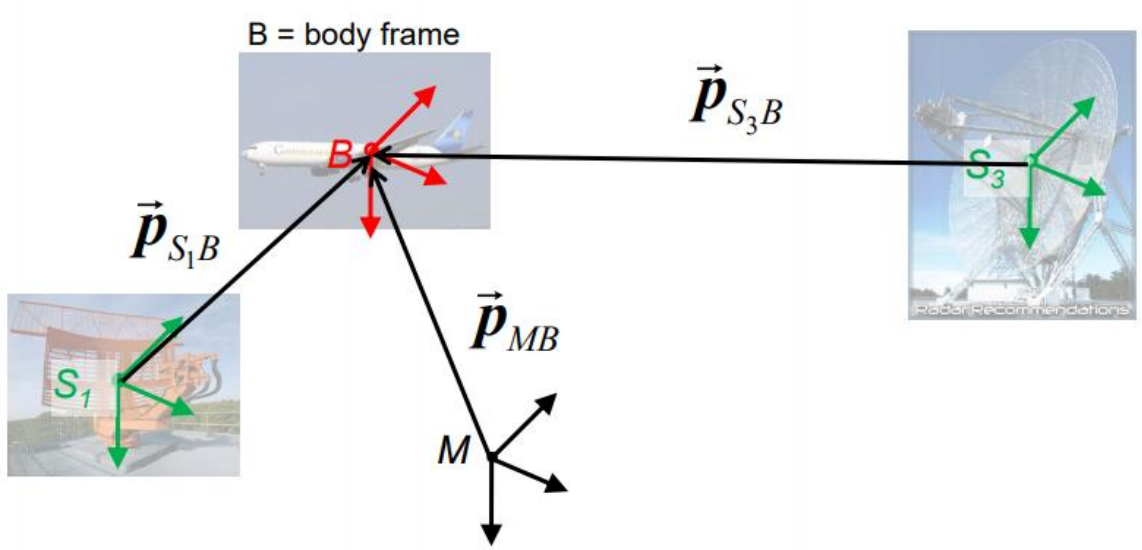

**GNC-UAS**

Guidance, Navigation and Control: Unmanned Aircraft System



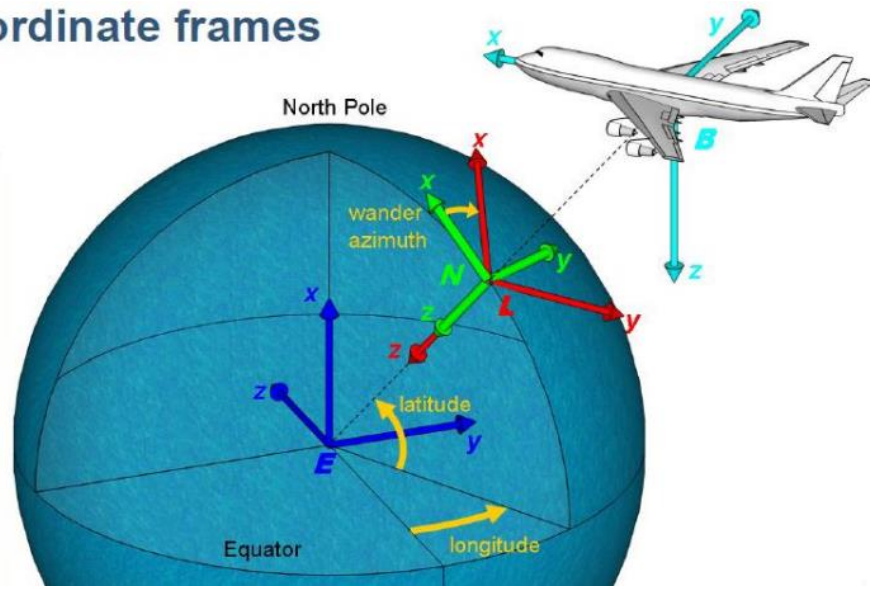
# Reference Frame for Multi-sensor Fusion

- Data measured in different coordinate frames (S1 and S3 in example)
- Before data fusion all vector measurements must be transformed into a common coordinate system M




**Important coordinate frames**

| Frame symbol | Description |
|---|---|
| *I* | Inertial |
| *E* | Earth-fixed |
| *B* | Body-fixed |
| *N* | North-East-Down (local level) |



# Some Techniques of Sensor Fusion Algorithms

- Bayesian Sensor Fusion: Based on probabilities
- Complimentary Sensor Fusion *(We will cover this technique in this course)*
- Dempster–Shafer Theory (DST): Generalization of Bayesian theory
- Kalman Filter

# Two-sensor Data Fusion - Complimentary Sensor Fusion

- Both sensors take a measurement Z of a constant but unknown parameter x, in the presence of noise v with standard deviation σ

$$Z_1 = x + v_1 \qquad \text{and} \qquad Z_2 = x + v_2$$

- The fusion is a linear combination of the measurements:

$$\hat{x} = k_1 Z_1 + (1 - k_1) Z_2$$

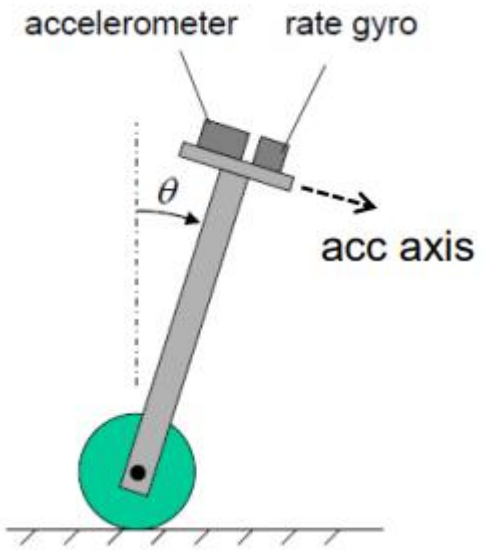
$$\hat{x} = \frac{\sigma_2^2}{\sigma_1^2+\sigma_2^2} Z_1 + \left(1 - \frac{\sigma_2^2}{\sigma_1^2+\sigma_2^2}\right) Z_2 = \frac{\sigma_2^2}{\sigma_1^2+\sigma_2^2} Z_1 + \frac{\sigma_1^2}{\sigma_1^2+\sigma_2^2} Z_2$$

- *Special Case 1:* $\sigma_1^2 = \sigma_2^2$ *What does this mean?*
- *Special Case 2:* $\sigma_1^2 = 0$, or $\sigma_2^2 = 0$ *What does this mean?*
- Not an optimal solution for a properly modelled random process

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} (Z_i - \mu)^2$$

# Complementary Filter – The Balance Robot

- Two different Measurement Sources for estimating one variable, both with different noise properties.
- For example, one gives good information in the low frequency region, the other in the high frequency region
- This one directional balancing robot uses one-axis accelerometer and one-axis gyroscope
- Gyros provides faster sensing on the angular rate of change, accelerometer are slow in this regard!
- Gyros suffer from drift due to integration, accelerometers do not!
- Sensor fusion helps mitigate this problem!




$$\theta \approx \int (\text{angular rate})\, dt$$

- not good in long term due to integration

$$\theta \approx \sin^{-1}\left(\frac{\text{accel. output}}{g}\right)$$

- only good in long term
- not proper during fast motion

# Calculating Roll and Pitch Angles

- Suppose our balance robot needs to balance itself not only in one direction but the two directions, then we need to upgrade our sensors to tri-axial ones.
- For the accelerometer, we need to calculate the pitch and roll angles.
- These can then be fused with the angular rates from the gyro and fed to the controller.

$$\text{Pitch} = \alpha = \arctan\left(\frac{A_{x1}}{\sqrt{(A_{y1})^2 + (A_{z1})^2}}\right)$$

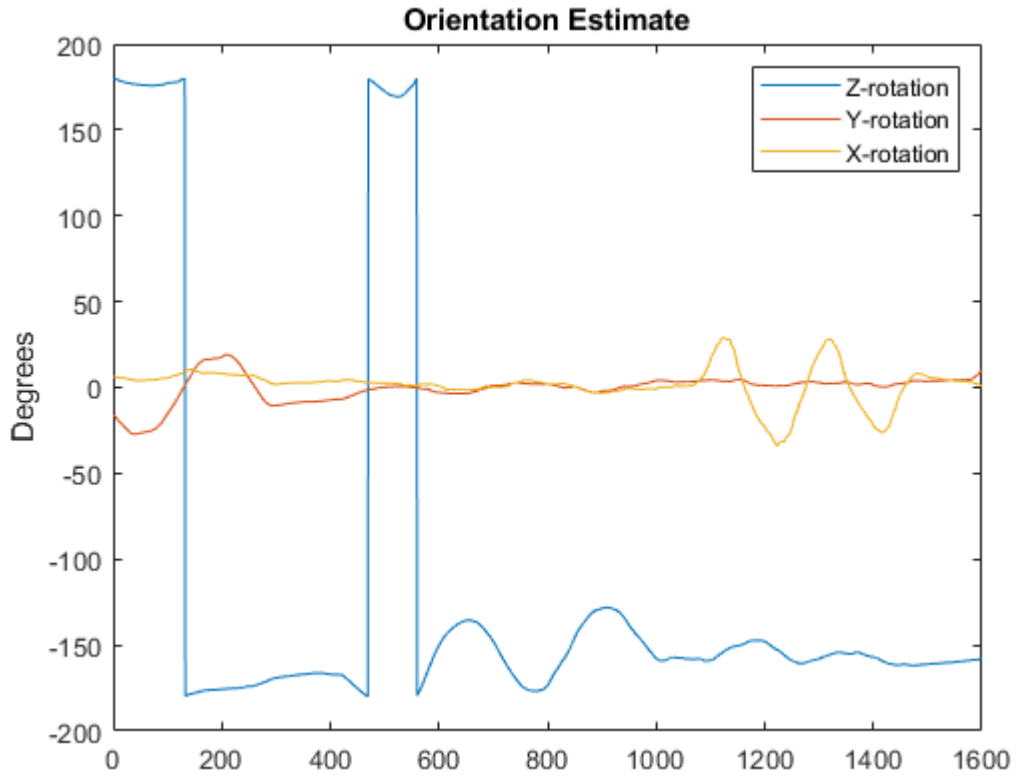
$$\text{Roll} = \beta = \arctan\left(\frac{A_{y1}}{\sqrt{(A_{x1})^2 + (A_{z1})^2}}\right)$$

# IMU Fusion using MATLAB Built-in Functions

```
ld = load('rpy_9axis.mat');

accel = ld.sensorData.Acceleration;
gyro = ld.sensorData.AngularVelocity;
mag = ld.sensorData.MagneticField;
Fs  = ld.Fs;  % Hz

fuse = complementaryFilter('SampleRate', Fs);
q = fuse(accel, gyro, mag);

plot(eulerd( q, 'ZYX', 'frame'));
title('Orientation Estimate');
legend('Yaw:Z-rotation', 'Pitch:Y-rotation',
'Roll:X-rotation');
ylabel('Degrees');
```

- You can implement the previous equations yourself, or you can make use of MATLAB features.
- For the complimentary filter, MATLAB has an example that uses recorded data from an accelerometer, gyroscope, and magnetometer stored in the file `'rpy_9axis.mat'` provided by MATLAB if the proper boxes are installed.
- Other Examples exist that use the Kalman Filter Fusion Algorithm

# Example Output



# References


- https://realpars.com/sensor-calibration/
- **AN3182 Application note -** Tilt measurement using a low-g 3-axis accelerometer (Appendix A)
- https://community.sw.siemens.com/s/article/introduction-to-filters-fir-versus-iir
- https://community.sw.siemens.com/s/article/ac-and-dc-coupling-what-s-the-difference
- https://cache.freescale.com/files/32bit/doc/white_paper/SENFEIOTLFWP.pdf
- https://www.uio.no/studier/emner/matnat/fys/FYS3240/v17/lecturenotes/l13---data-fusion.pdf