Robotics 311 : How to build robots and make them move

Prof. Elliott Rouse

GSI Yves Nazon MS

Fall 2022

Photo credit: Brenda Ahearn / U-M CoE

ROB 311 – Lecture 23

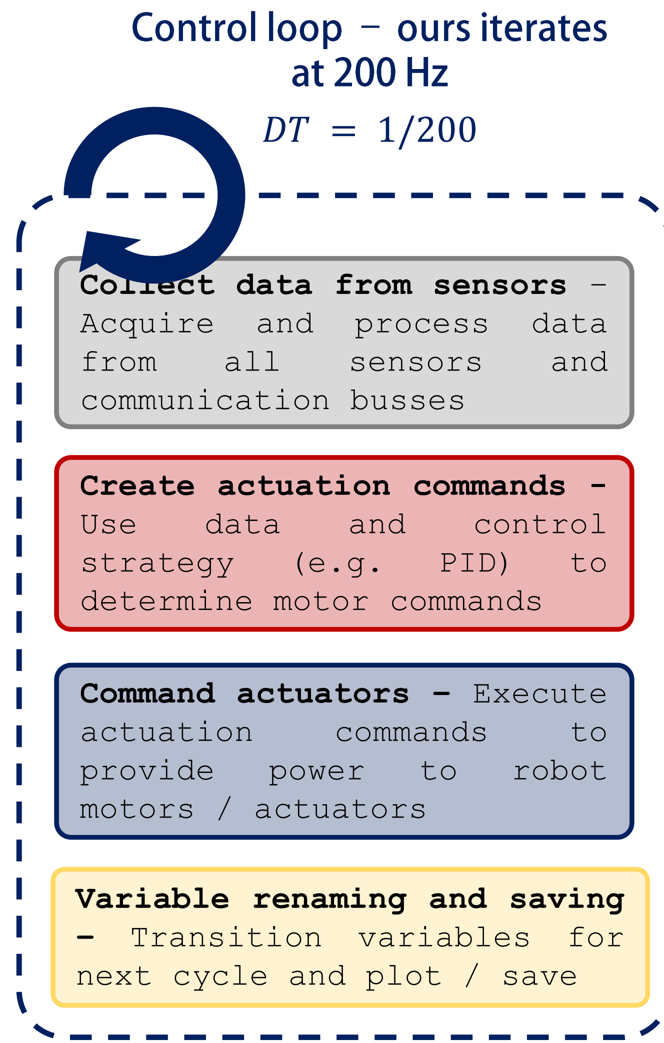
- Review PID / implementation
- Discuss system frequency response
- Learn basic filtering in Python and MATLAB

Announcements

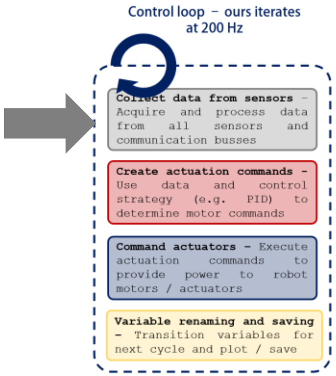
- HW5 will be posted today
- Only one more HW assignment
- After Thanksgiving, we have 3 lectures, 2 labs, and a competition

How To Implement Control

- How to practically implement PID in your ball-bot
- Points to consider
  - How do we calculate the derivative?
  - How can we deal with nonlinearities (e.g. saturation)?
- Let's discuss the overall control loop structure
- Your control loop needs to do four things
  - Refresh data / communication
  - Use feedback to determine commands
  - Send commands to the motors
  - Save data and transition variables



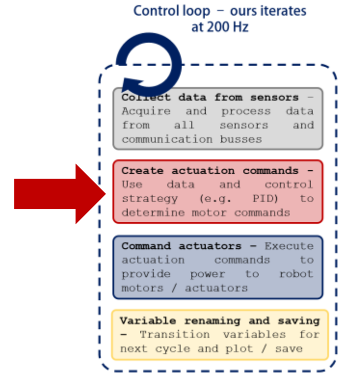
Data Collection



- In our control loop, the first thing that must be done is to refresh data from all sensors
- This would be a series of communication library calls, typically one for each component
- In our system, your data are refreshed automatically
  - How? The Pico collects and sends data to the RPi
  - We have streamlined this process, but you could do it on your own

```
# Define variables for saving / analysis here - below you can create variables from the available states in message_defs.py

# Motor rotations
psi_1 = states['psi_1']
psi_2 = states['psi_2']
psi_3 = states['psi_3']

# Body lean angles
theta_x = (states['theta_roll'])
theta_y = (states['theta_pitch'])

# Controller error terms
error_x = desired_theta_x - theta_x
error_y = desired_theta_y - theta_y
```

Actuation Commands



- Setting actuation commands comes from our control law
- We will use PID
- Let's go through the calculation of each term
    - Proportional term

Easy to implement

$$u_p[k] = K_p \cdot (y[k] - r[k]) = K_p e[k]$$

- Most of your controller effort will likely come from this term
- Integral term

This is your integral—it can be a running sum

$$e_{sum} = e[k] + e_{sum}$$

The DT can be left out, which just scales K_i

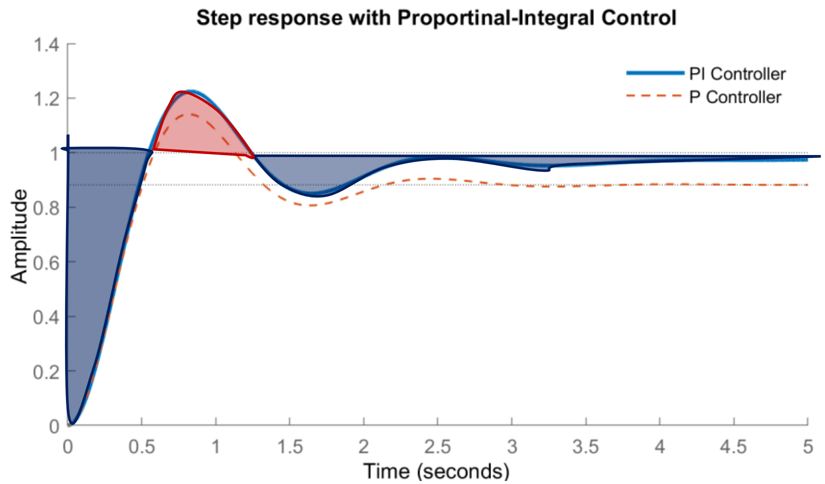
$$u_i(t) = K_i \cdot e_{sum} \cdot DT = K_i \int_{t_0}^{t} e(\tau)\, d\tau$$

- The integral term is susceptible to saturation
- Saturation is a type of nonlinearity

$[k]$ used to mean the value of loop iteration k

Integral Commands

- Integration term – saturation
- Reminder – the integral effort will keep track of the difference between the reference and the output



- This idea allows the controller to add as much effort as possible
- Steps could be arbitrarily large in your application—are can be very large
- Can your controller always provide this effort? Think about flooring your gas pedal...

PWM

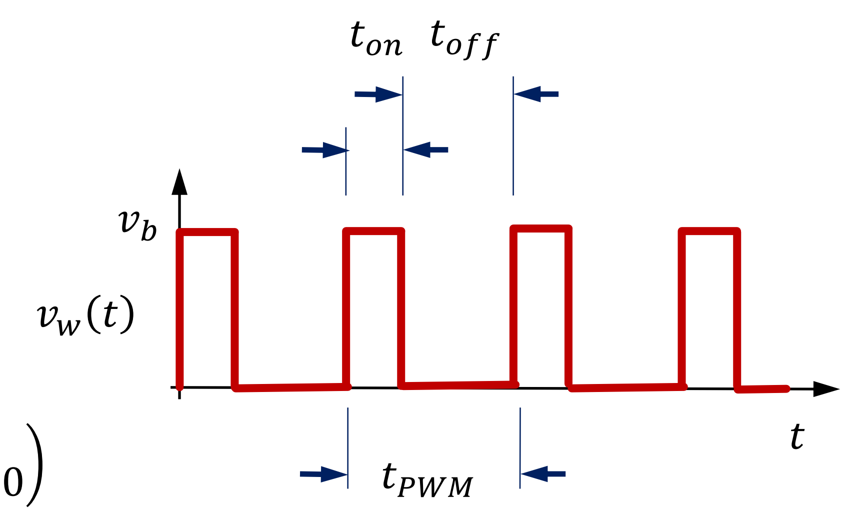
Battery voltage: v_b

Applied motor voltage: v_w

- Pulse Width Modulation (PWM)
- This is how our actuation command (voltage) is provided to the motors
- A quickly-oscillating voltage that varies between 0 V and the battery voltage
- By varying the ‘duty cycle’ the applied voltage can be varied
- The dynamics of the motor and physical system smooth the rapidly oscillating voltage
- What’s the max PWM value? 100%
- This creates a nonlinearity when the controller maxes out

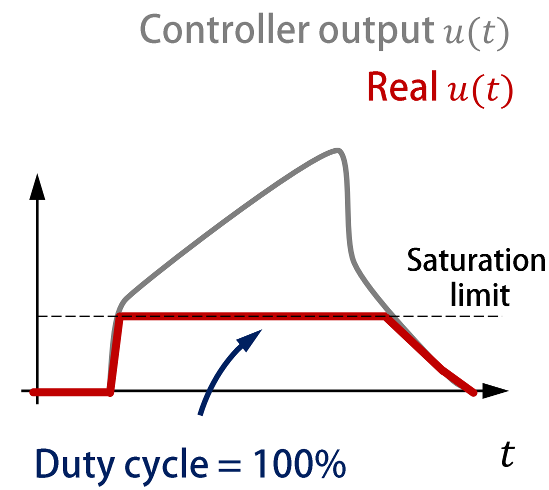
$$F_{PWM} = \frac{1}{t_{PWM}} \qquad Duty\ Cycle = \frac{t_{on}}{t_{PWM}} \cdot 100$$

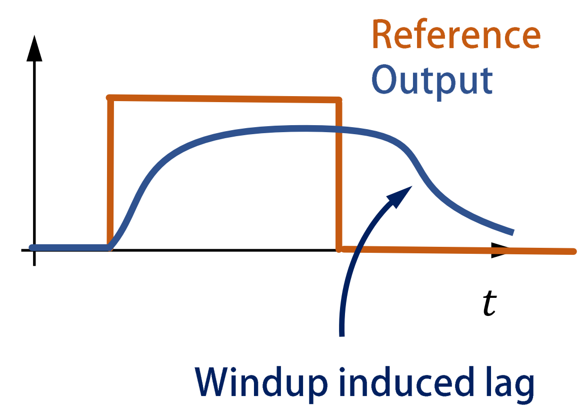
$$v_{applied} = v_b \cdot \left({}^{Duty\ Cycle}/_{100} \right)$$



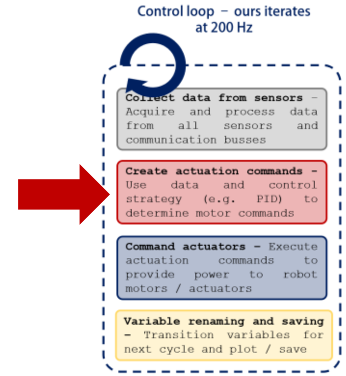
Saturation and Windup

- The maximum-effort nature of actuators is known as 'saturation'
- Saturation can cause excessive overshooting / inefficiency
- To address saturation, sometimes a saturation value is used to limit $u_i(t)$
- To implement in Python, an `if-then` statement can be used to check the magnitude of $u_i[k]$
- You can limit $u_i[k]$ to a maximum of $X\%$ of the maximum effort
  - 50% could be a good starting point, but it will need to be adjusted
- Integral terms add a delay or lag
- Known as 'windup'





Derivative Commands



- We've so far described the nuances of calculating P and I commands
- The derivative term requires a numerical derivative
- Most common is the two-point / finite difference method

This is the value from one loop iteration go

$$\frac{de}{dk}[k] = (e[k] - e[k-1])/DT$$

Finite difference derivative

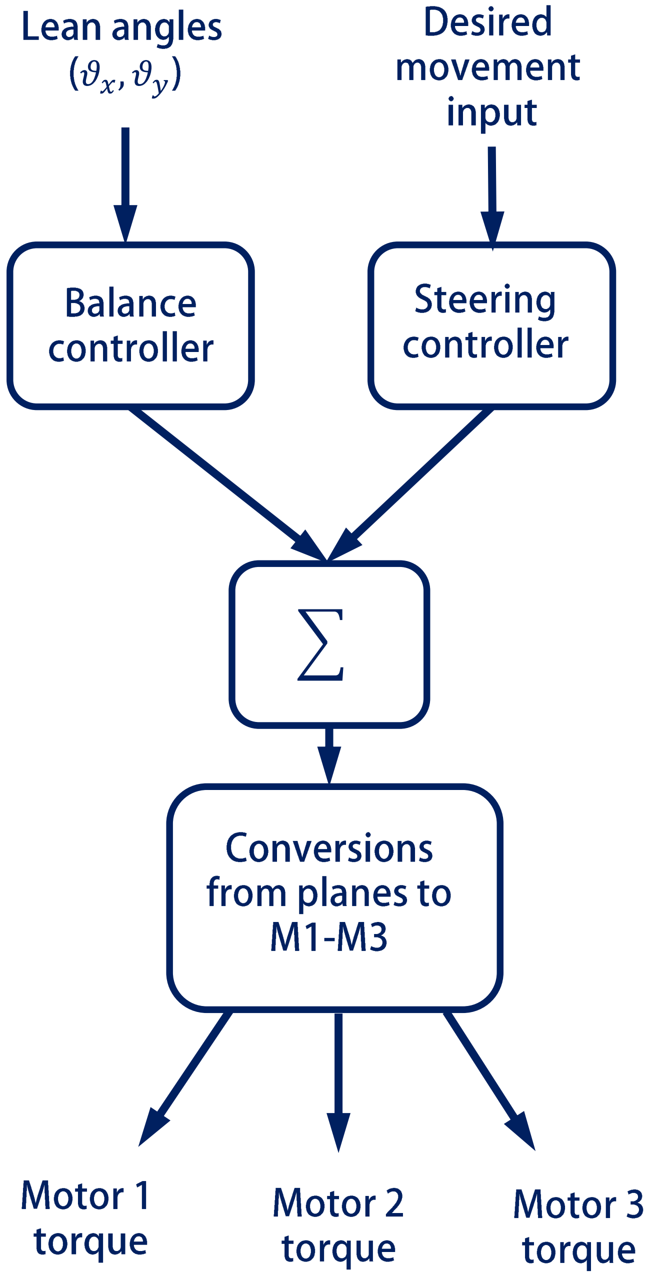
Variable saved / transitioned at the end of each loop

$$u_d[k] = K_d \cdot \frac{de}{dk}[k]$$

- Watch out for noise! Clean signals are needed to gain useful information
- Are our signals noisy?
  - IMU – no it's actually pretty clean - view the data to confirm
  - Encoder data – clean when viewed at larger time scales, but quantized by nature
- These data can be filtered, but this adds delay

Controller Architecture

- We break the controller into the two planes
- Each plane will be handled independently
- Each plane has two controllers that run in parallel
    - Balance controller / steering controller
    - They will be separate but will run simultaneously
- There will be four total controllers in parallel
- We will superimpose the torques from the balance and steering controllers
- Simultaneous balance and steering
- We will begin with the **balance controller**
- Let's think about how this controller should be designed

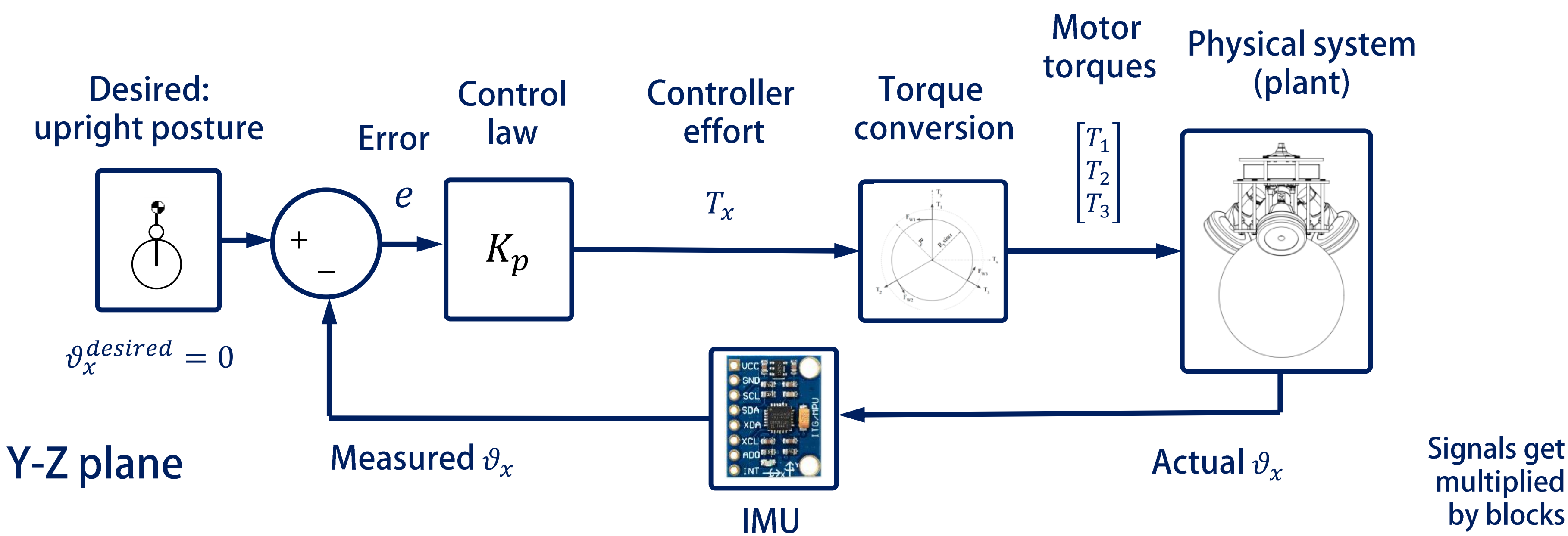


Balance Controller

- In lab, we began with a simple P-controller
- We knew we wanted the system to maintain upright balance

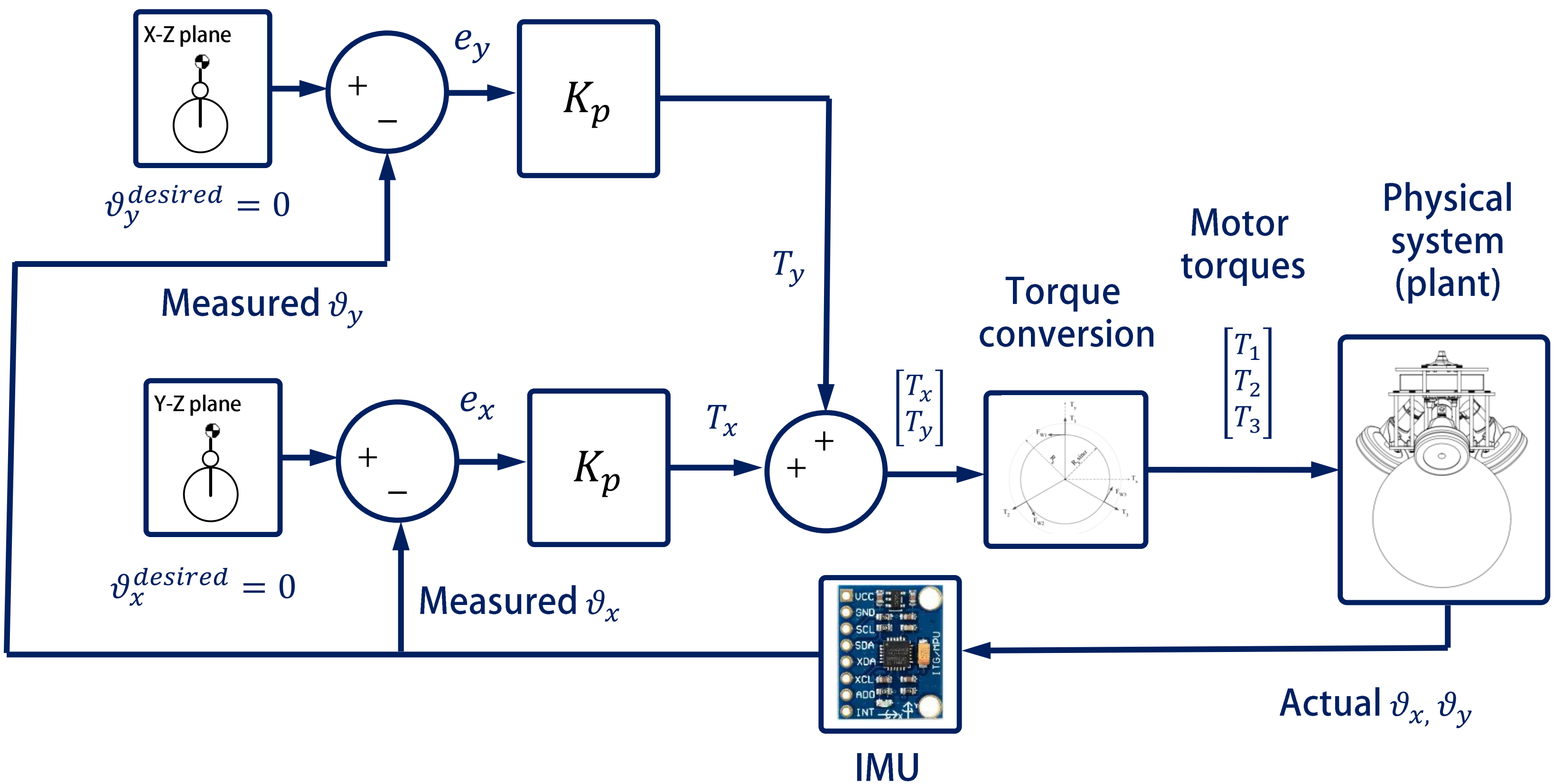
$$T[k] = -K_p \cdot \vartheta_{axis}[k]$$

- Our reference trajectory is upright posture ($\vartheta_x^{desired} = \vartheta_y^{desired} = 0$)
- Putting into the technical framework of feedback control
- Control law: $T[k] = e[k] \cdot K_p = \left(\vartheta^{desired}[k] - \vartheta[k]\right) \cdot K_p = -\vartheta[k] \cdot K_p$



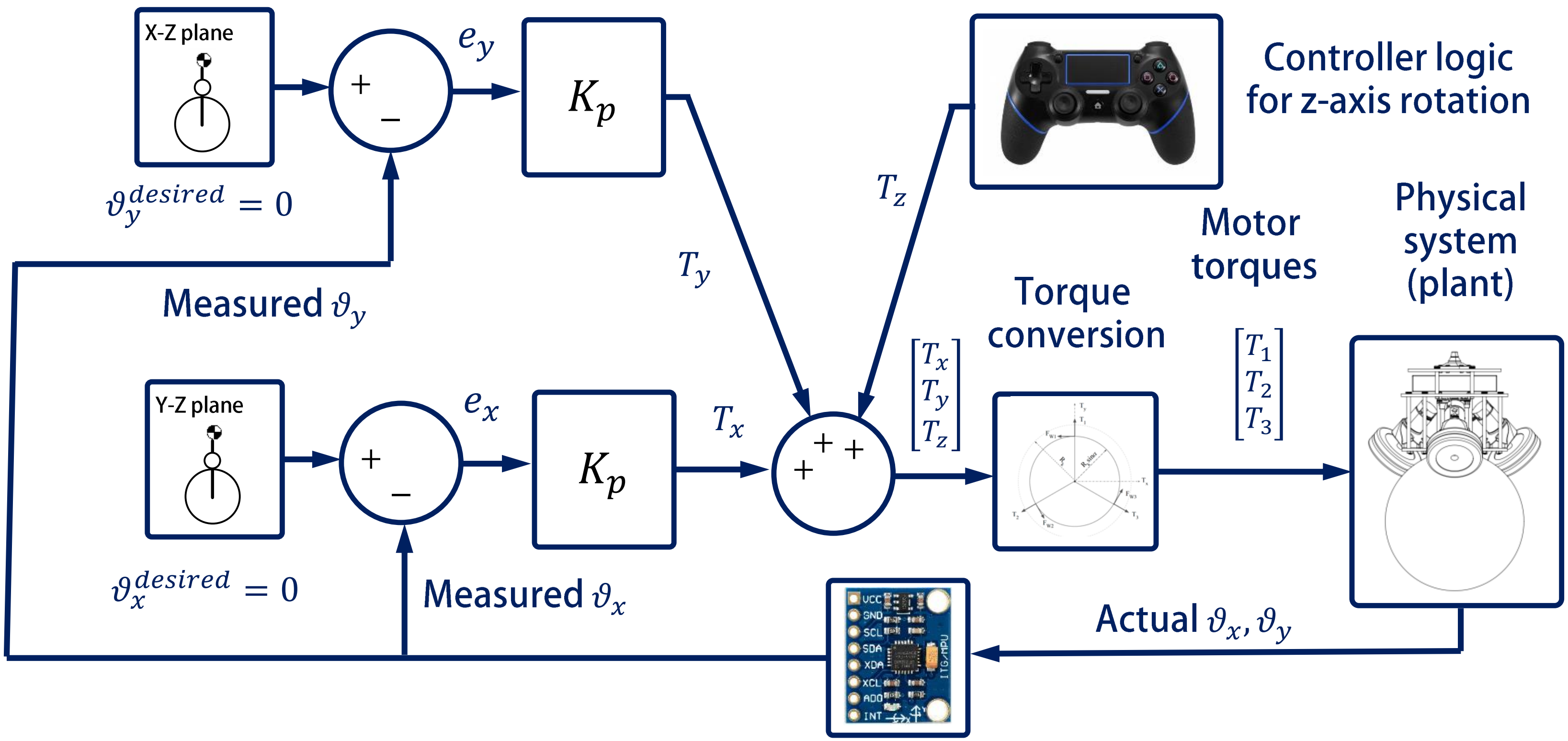
Balance Controller

- We extended to both planes at once (X-Z and Y-Z planes)
- Control law: $T[k] = e[k] \cdot K_p = \left(\vartheta^{desired}[k] - \vartheta[k]\right) \cdot K_p = -\vartheta[k] \cdot K_p$
- We learned to choose K_p by tuning the controller (lab)



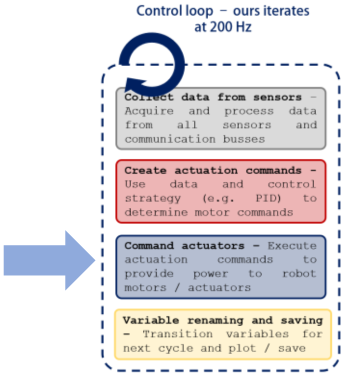
Controller for Z-Axis Torque

- Now we will add the z-axis torque from button commands you choose
- Triggers provide continuous values and buttons provide binary values
- Create a torque function using the button presses and add to the torque commands (or use the demos)
- You will need to set the z-axis torque to the torque value from the PS4 controller



M ROBOTICS

Sending Actuator Commands

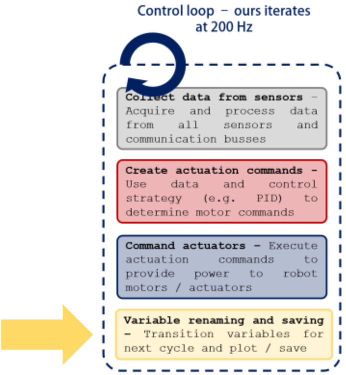


- After constructing the torque commands, they need to be sent to the motors
- We take care of this for you with an API, but you could also do it

```
# -----------------------------------------------------------

print("Iteration no. {}, T1: {:.2f}, T2: {:.2f}, T3: {:.2f}".format(i, T1, T2, T3))
commands['motor_1_duty'] = T1
commands['motor_2_duty'] = T2
commands['motor_3_duty'] = T3
ser_dev.send_topic_data(101, commands) # Send motor torques
```

- This applies voltage to the motor, the command of which comes from the required torque → required current
- The torque commands are sent to the Pico, which converts them into a motor duty cycle command
- Sending commands is usually only a few lines of code

Saving Data and Transitioning Variables

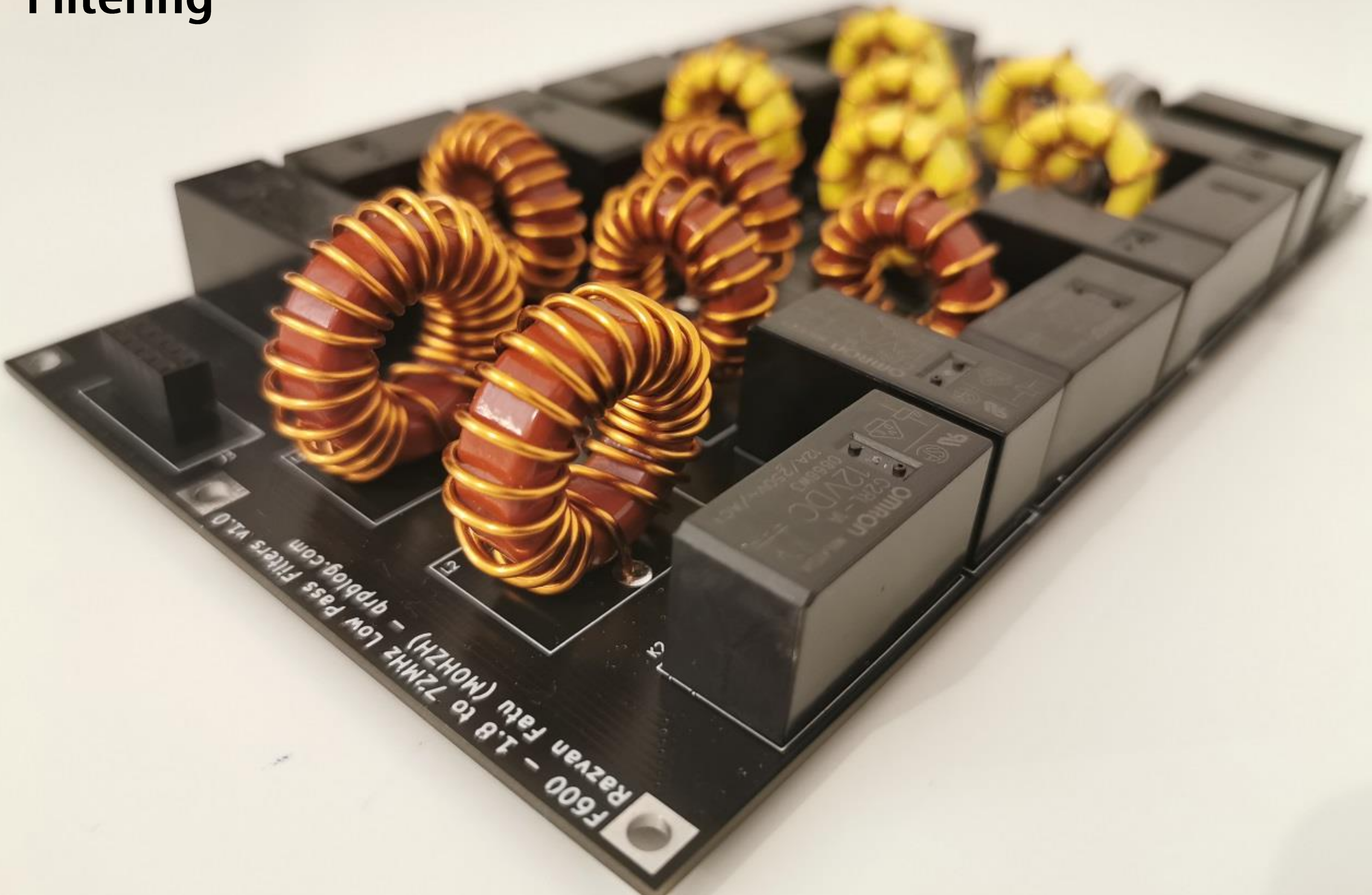


- At the end of your loop, you will need to
  - Create your data matrix
  - Rename variables (any variables for prev. loop iteration)

```
# Construct the data matrix for saving - you can add more variables by replicating the format below
data = [i] + [t_now] + [theta_x] + [theta_y] + [T1] + [T2] + [T3] + [phi_x] + [phi_y] + [phi_z] + [psi_1] + [psi_2] + [psi_3]
dl.appendData(data)

# Transition variables
error_x_prev = error_x
error_y_prev = error_y
```

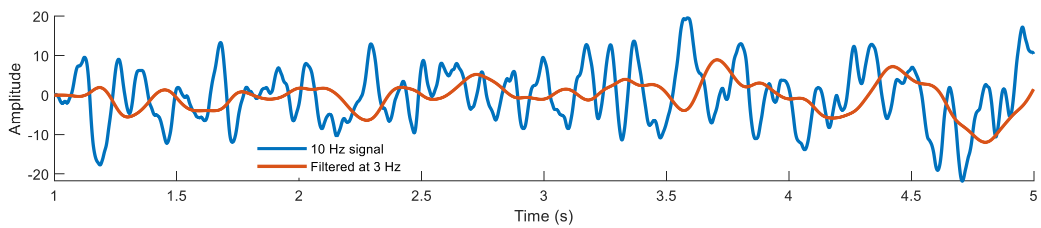
- If error values from the previous iteration are used, they need to be transitioned
- This is likely if you're taking a finite difference derivative in the loop
- This sets up the variables for your next loop iteration
- Data matrix gets created and appended to

Filtering


F600 - 1.8 to 72MHz Low Pass Filters v1.0
Razvan Fatu (M0HZH) - qrpblog.com
omron
G2RL-1E
12VDC

What is Filtering and Why Do We Need It?

- Filtering is a ubiquitous tool in robotics and engineering
- It's usually a critical step with any real-world data
- We use filtering to remove noise, which can be introduced in many ways
- Unwanted corruption of your signal
  - Sensor noise
  - 60 Hz electrical interference
  - Corrupted or uncertain data
  - Infinite examples
- Filtering passes signals or images through a dynamic system
- This changes the signal, often (but not always) smoothing it out over time



Frequency Content

- Let's begin by thinking remembering that time series data can be represented as a combination of frequencies
- This is provided using a mathematical tool known as a Fourier Transform

$$F(j\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt$$

Transformed signal

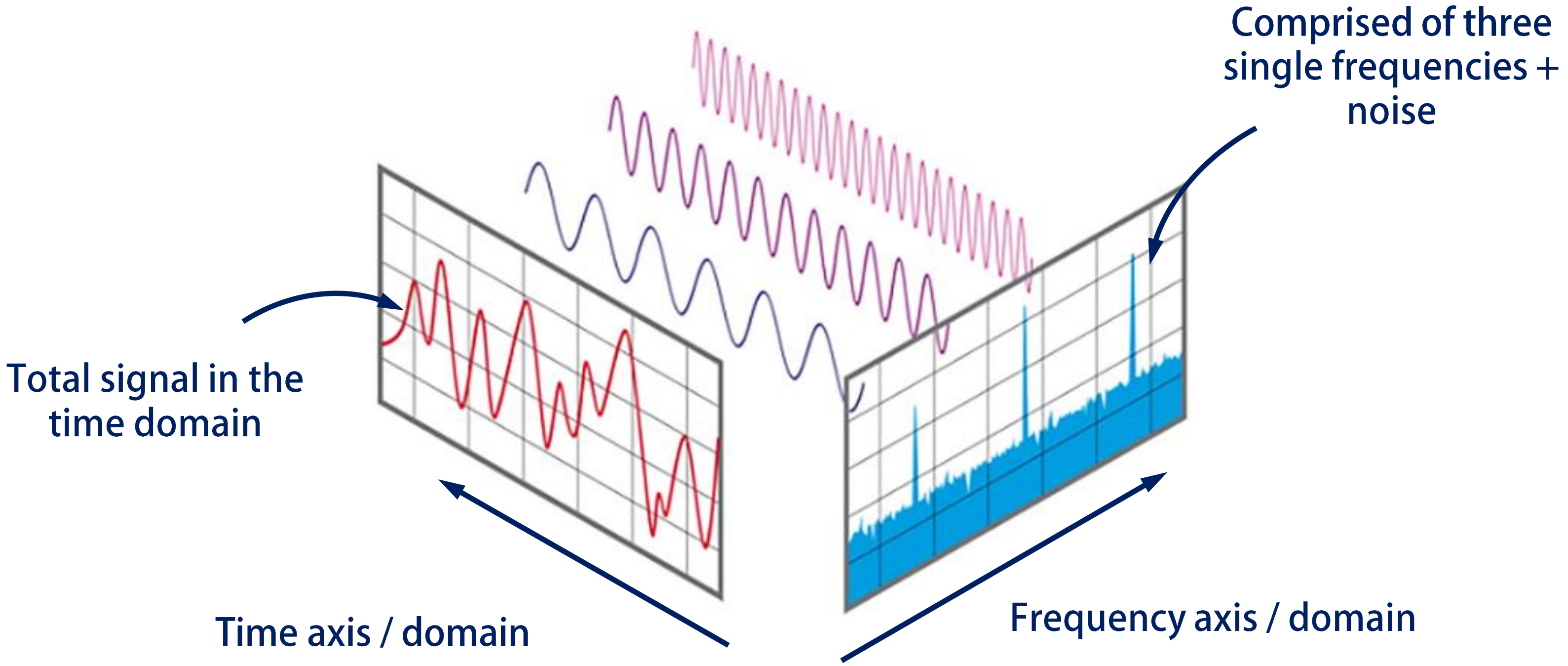
Original time-domain signal

Complex exponential

- Signals can be 'transformed' by this equation, describing the frequency and phase information of a signal
- Complex signal—describes magnitude and phase
- Audio signals provide a convenient context for explanation
- Sounds are composed of multiple frequencies
- We can view this information as a function of as time or frequencies

Frequency Content

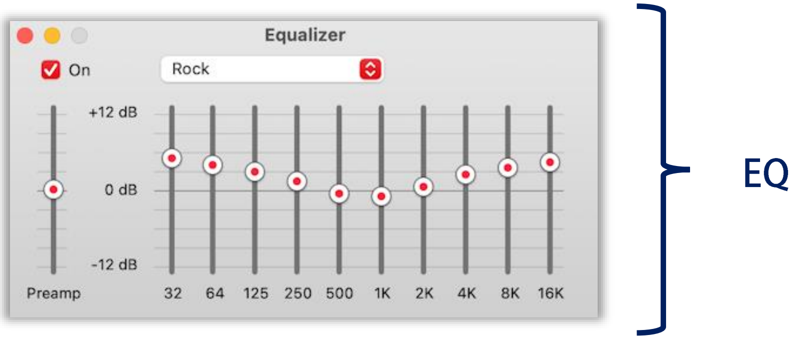
- Consider a recording of three people whistling into a microphone
- Each person is whistling at a different frequency
- What would that signal look like?

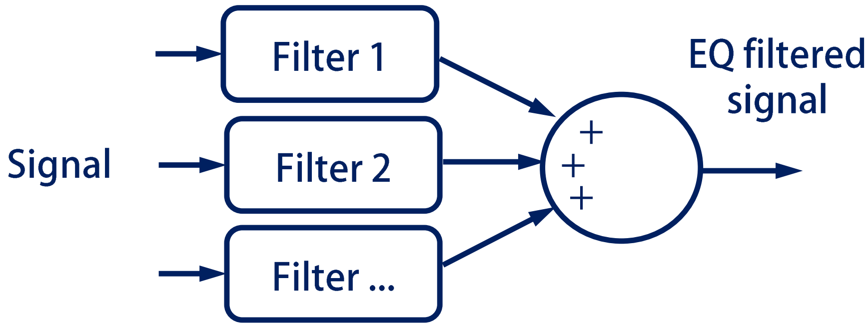


- This lets us begin to think about signals and systems in the *frequency domain*

Frequency Content

- We can use the Fourier Transform to compute the frequency content of signals
- Often implemented in software using the Fast Fourier Transform algorithm (FFT)
- The Fourier Transforms lets us selectively remove frequencies from data
- An music equalizer (EQ) is a set of parallel filters

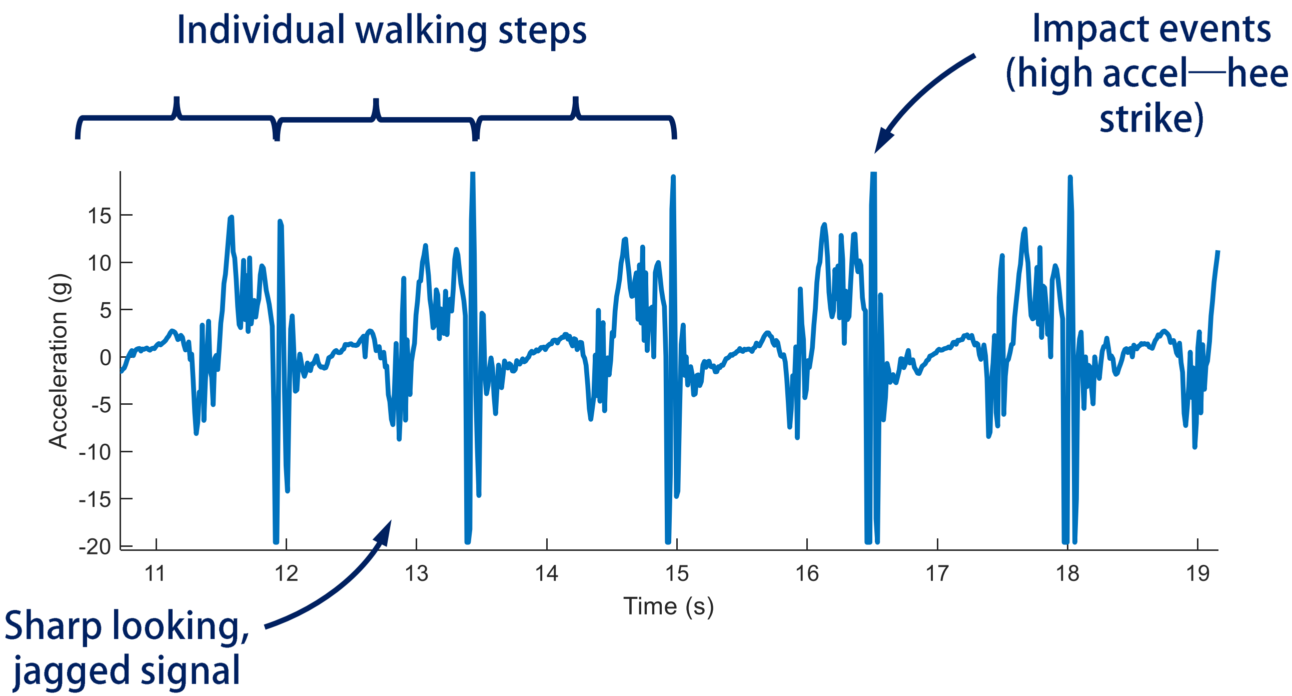




- Filters are characterized by what frequencies are ‘allowed through’ and which are attenuated
- There are many types that differ in the specific frequencies they attenuate / how exactly the attenuation is defined
- They can be run on previously-collected data (‘offline’) or run in real time in your control loop

Example IMU Analysis

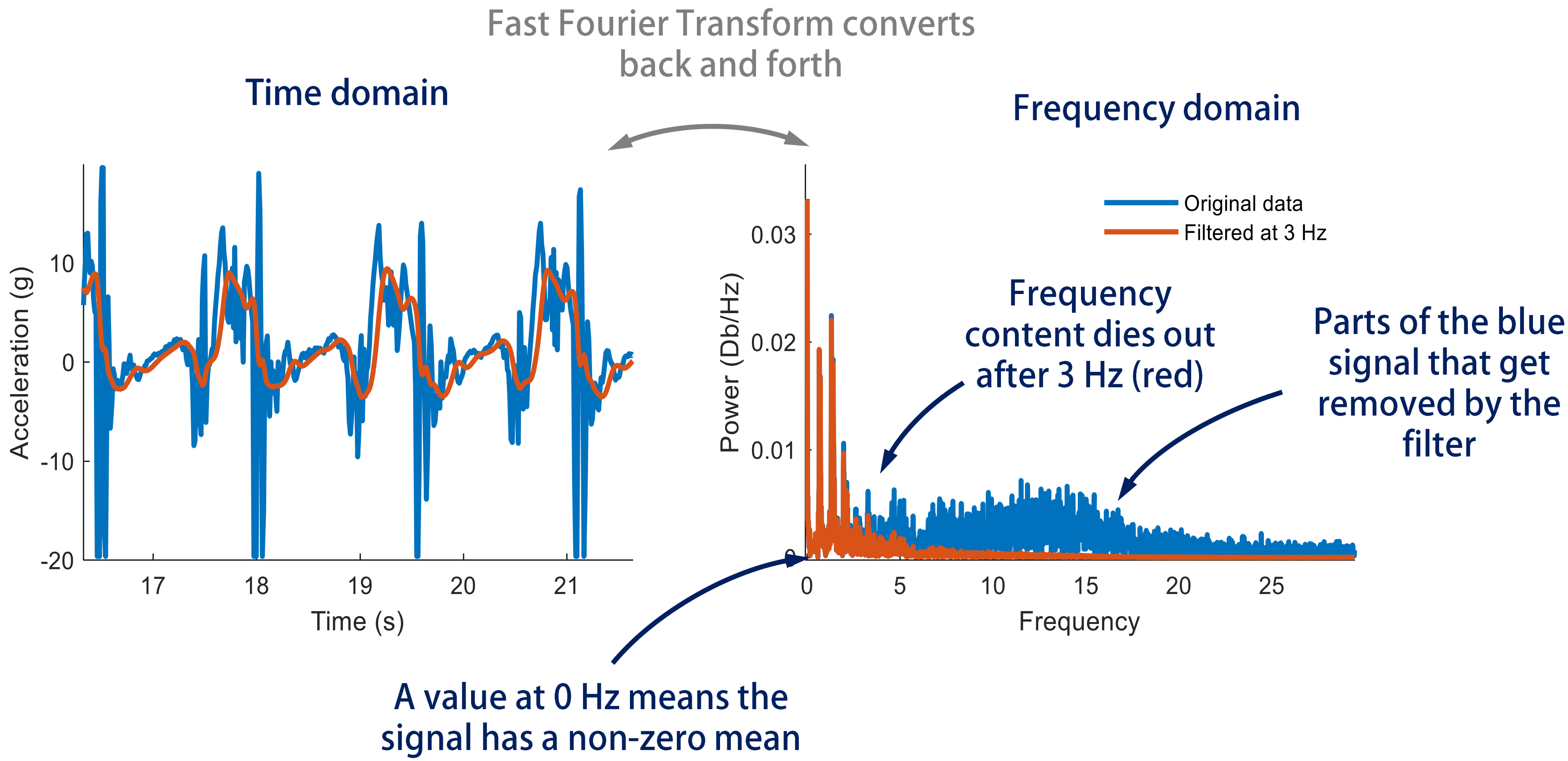
- Lets think about data collected from a robot
- These data come from me wearing [this](#) knee prosthesis (right)
- Lets look at vertical-axis acceleration—what do you see?



- This signal has some high frequency components—we could filter these out to remove them or isolate them, depending on what we needed

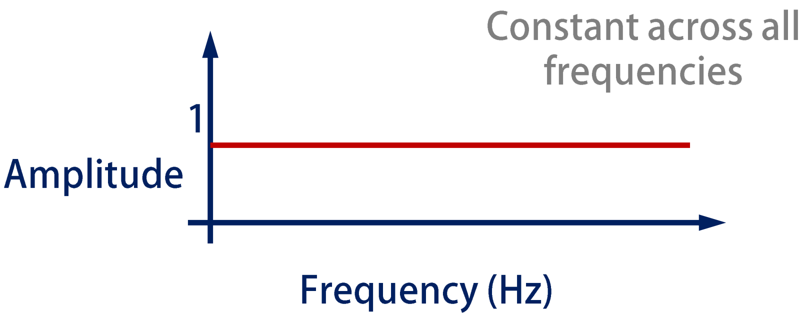
Example IMU Analysis

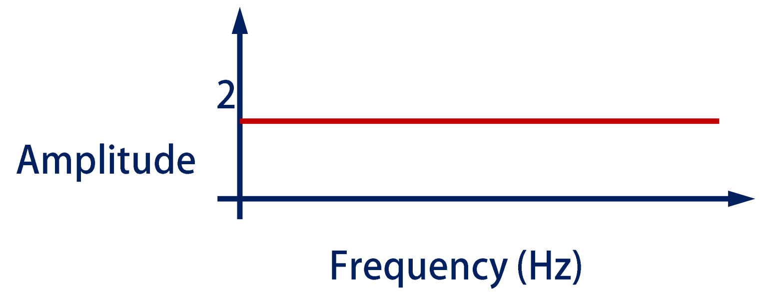
- If we filter the data with a ‘low pass’ filter, we can attenuate the higher frequency impacts
- This causes a slight delay, depending on the type of filter
- The effect of filtering can be viewed in both the time and frequency domains



Filtering As Multiplication

- We can think about filters as multiplying our signal by a function in the frequency domain – two signals in the frequency domain, multiplied point by point
- What if we multiplied a signal by these functions in the frequency domain?

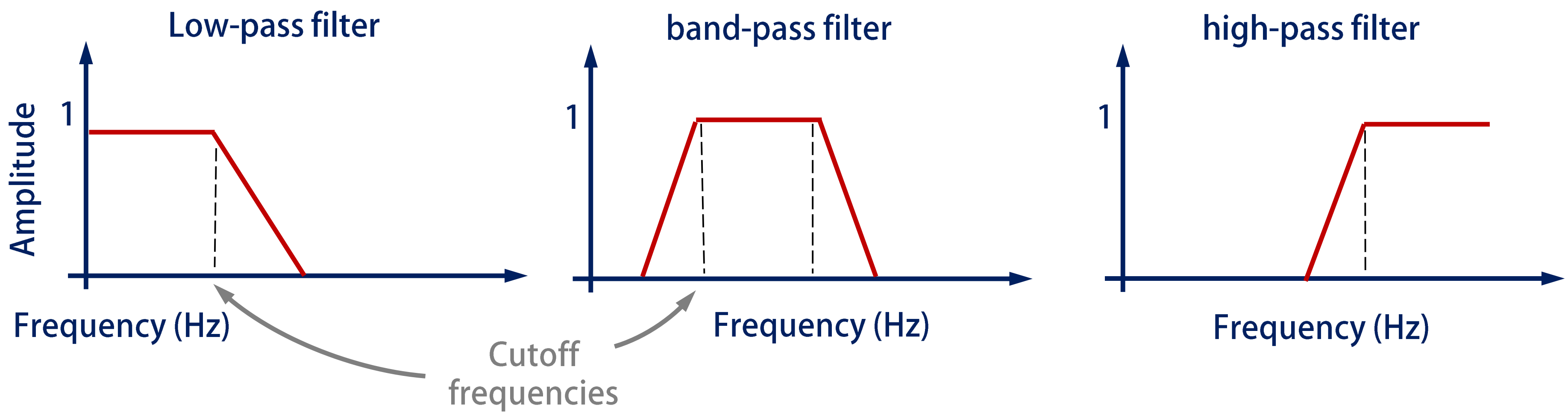




- The left would do nothing and the right would amplify by a factor of 2x
- Filtering is about specifying the exact shape of the multiplying function (red line)
- There are three types of filters, described based on their pass band
    - Low pass – allows low frequencies through
    - Band pass – allows a closed range of frequencies through
    - High pass – allows high frequencies through

Filtering As Multiplication

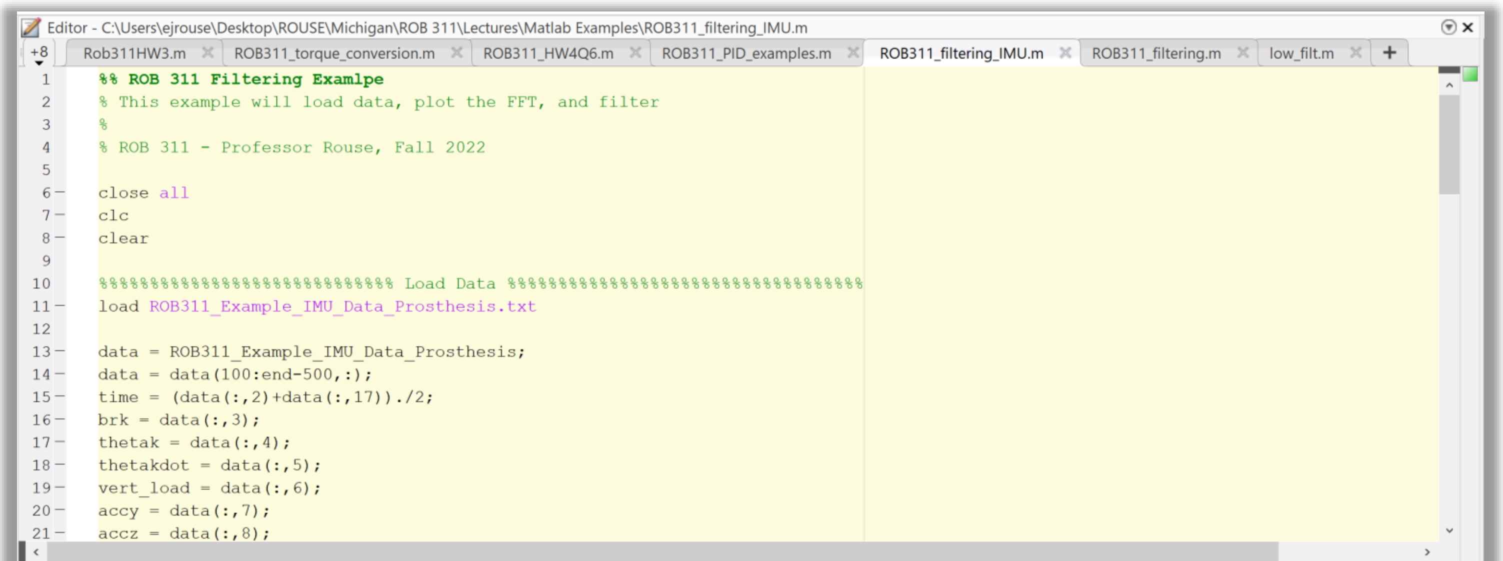
- Lets look at how different types of filter types affect the frequencies that pass through



- The ‘cutoff frequency’ defines what frequencies can pass through
- Filters can be also used to apply a gain at the same time
- Remember, the frequency domain is complex, having both magnitude and phase
- Phase describes how the frequencies begin to lag, and is described in degrees
- A phase shift of 360° is one cycle / period, and so on

How Do I Choose the Cutoff Frequency?

- This is depends on your application / task
- And where noise or artefacts may come from
- Use MATLAB to look at signal content
- Download posted MATLAB file and play with changing the frequency and data


Editor - C:\Users\ejrouse\Desktop\ROUSE\Michigan\ROB 311\Lectures\Matlab Examples\ROB311_filtering_IMU.m

+8 | Rob311HW3.m | ROB311_torque_conversion.m | ROB311_HW4Q6.m | ROB311_PID_examples.m | ROB311_filtering_IMU.m | ROB311_filtering.m | low_filt.m

```
%% ROB 311 Filtering Examlpe
% This example will load data, plot the FFT, and filter
%
% ROB 311 - Professor Rouse, Fall 2022

close all
clc
clear

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Load Data %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load ROB311_Example_IMU_Data_Prosthesis.txt

data = ROB311_Example_IMU_Data_Prosthesis;
data = data(100:end-500,:);
time = (data(:,2)+data(:,17))./2;
brk = data(:,3);
thetak = data(:,4);
thetakdot = data(:,5);
vert_load = data(:,6);
accy = data(:,7);
accz = data(:,8);
```

How is Filtering Implemented in Software?

- Lets look at our MATLAB filtering function `low_filt.m`
- Needs sample rate, filter order, cutoff freq., and data

Sample rate Fs

Filter order N

Editor - C:\Users\ejrouse\Desktop\ROUSE\Matlab\low_filt.m

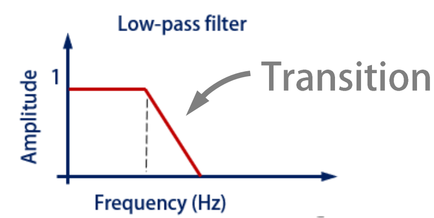
Rob311HW3.m | ROB311_torque_conversion.m | ROB311_HW4Q6.m | ROB311_PID_examples.m | ROB311_filtering_IMU.m | ROB311_filtering.m | low_filt.m

```
function filt_data = low_filt(Fs,N,Fc,data)

%%%%%%%%%%%%%%%%%%%%%%
%This function low-passfilters the EMG data to reduce the motion artifact
%Usage: filt_data = low_filt(Fs,N,Fc,data)
%Fs - sampling frequency
%N - Filter order
%Fc - cutoff frequency
%data - data to be filtered
%
%
%%%%%%%%%%%%%%%%%%%%%%%

[B,A] = butter(N, Fc/(Fs/2),'low');                % Butterworth filter design

for i=1:size(data,2)
%    filt_data(:,i) = filtfilt(B,A, data(:,i))     % For non-causal / bidirectional 0-phase filtering
     filt_data(:,i) = filter(B,A, data(:,i));      % For causal filtering
end
```

Cutoff freq. Fc

Creates filter coefficients (`B, A`)

- Order N: How fast the transition is in the frequency domain
- Butter: Specific shape of multiplying function (red shape)



How is Filtering Implemented in Software?

- Filtering can be applied to the entire signal at once (‘offline’ or post-processing) or it can be applied to a signal in real time
- In real time, filtering can be implemented by a short set of products and sums
- We will use filter libraries in MATLAB and Python, so you will not need to implement yourself
- Filters cleverly use the previous values to construct the filtered output
- Lets think of a moving average filter (low pass)
  - Moving average filters are defined by their length—how many samples are included (2 – 5 samples is common)
  - n_f is the number of samples included in the moving average

Filtered signal

$$x[k] = \left(\frac{1}{n_f}\right)x[k] + \left(\frac{1}{n_f}\right)x[k-1] + \left(\frac{1}{n_f}\right)x[k-2] + \left(\frac{1}{n_f}\right)x[k-3] \ldots$$

  - Moving average is one (simple) type of low-pass filter
  - Next lecture we will discuss how to implement in Python