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 24

- Review filtering
- Introduce steering control
- Finish content lectures!

Announcements

- VSCode has some issues with loop timing—use Putty
- HW 5 due 12/7 at 12:30 (lab start)
- Last lab content lecture
- Tuesday / Wednesday is prep for competition
- Thursday 12/8 is competition
- Report due on day of final (12/19) – details posted shortly

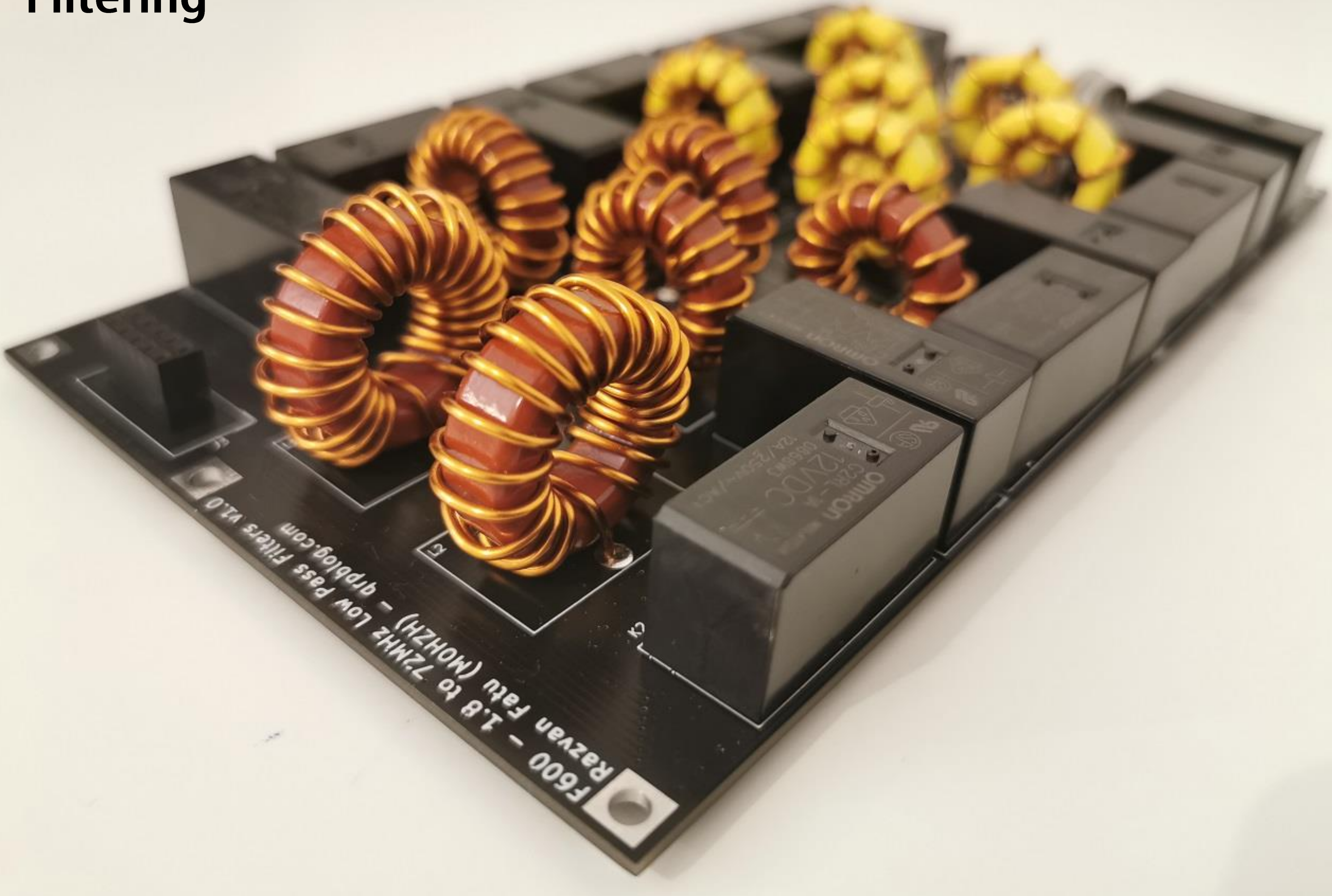
Final Competition

- Last lecture – Thursday December 8th
- The competition will be in the FRB atrium
- Main goal: assess and compare balance / steering controllers
- Your team / ball-bot will be scored on four tasks
- **Be ready when class starts with your ball-bots charged and connected**
- We will compete as well—if you beat us, you get an automatic A on the project
- Wear your ROB 311 shirts!

Events

- ~10 minutes of balancing—all ball-bots at once; last ball standing
- ~2 min max z-axis angular velocity
- ~2 min balancing with perturbations
- ~4 min steering around a 4′ x 4′ square loop

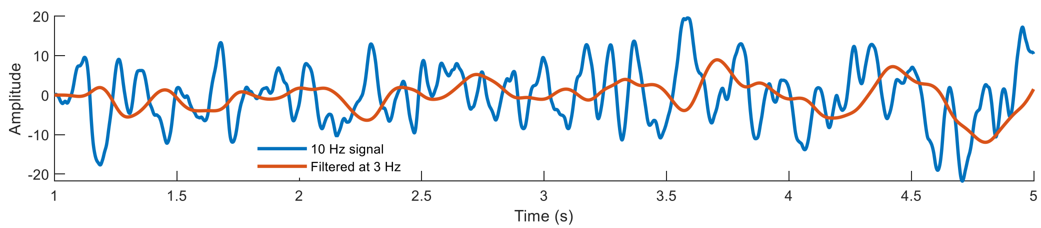
Filtering



Razvan Fatu (M0HZH) - qrpblog.com
F600 - 1.8 to 72MHz Low Pass Filters v1.0
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

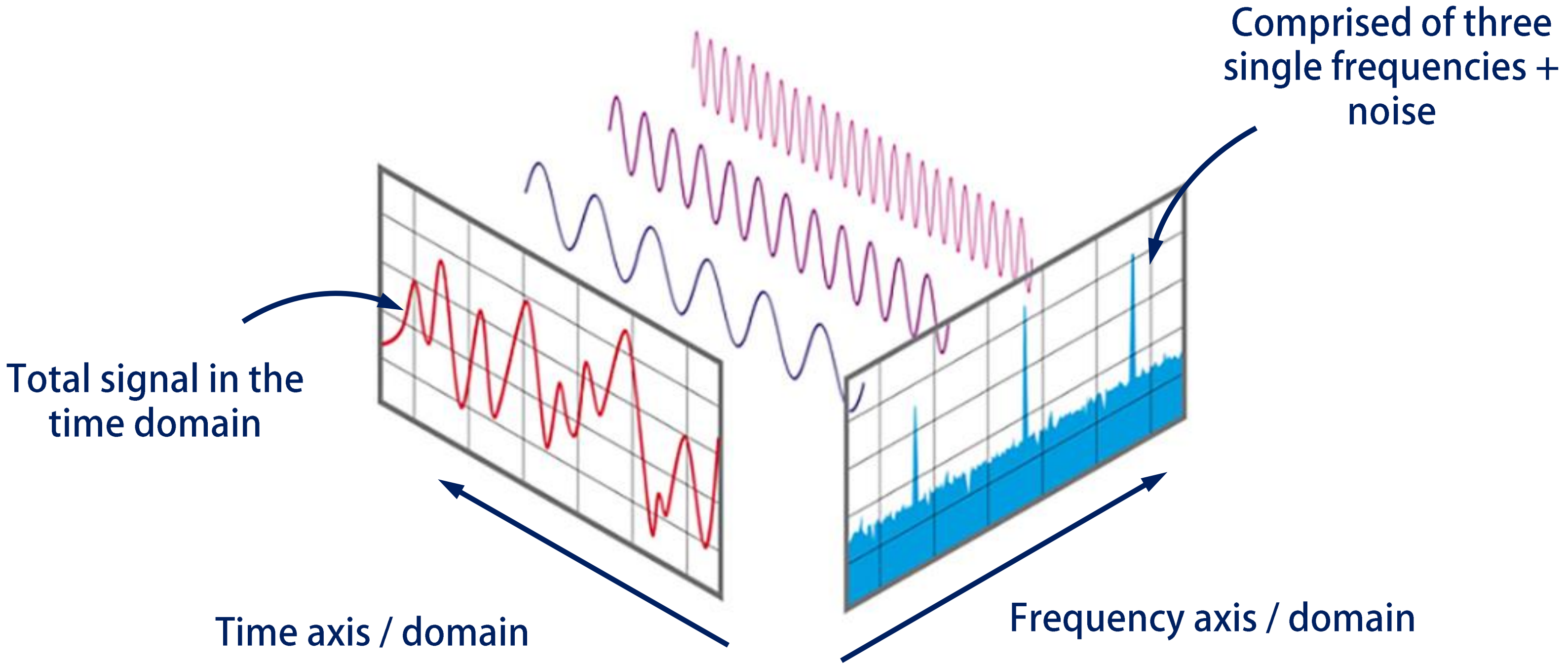
Original time-domain signal

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

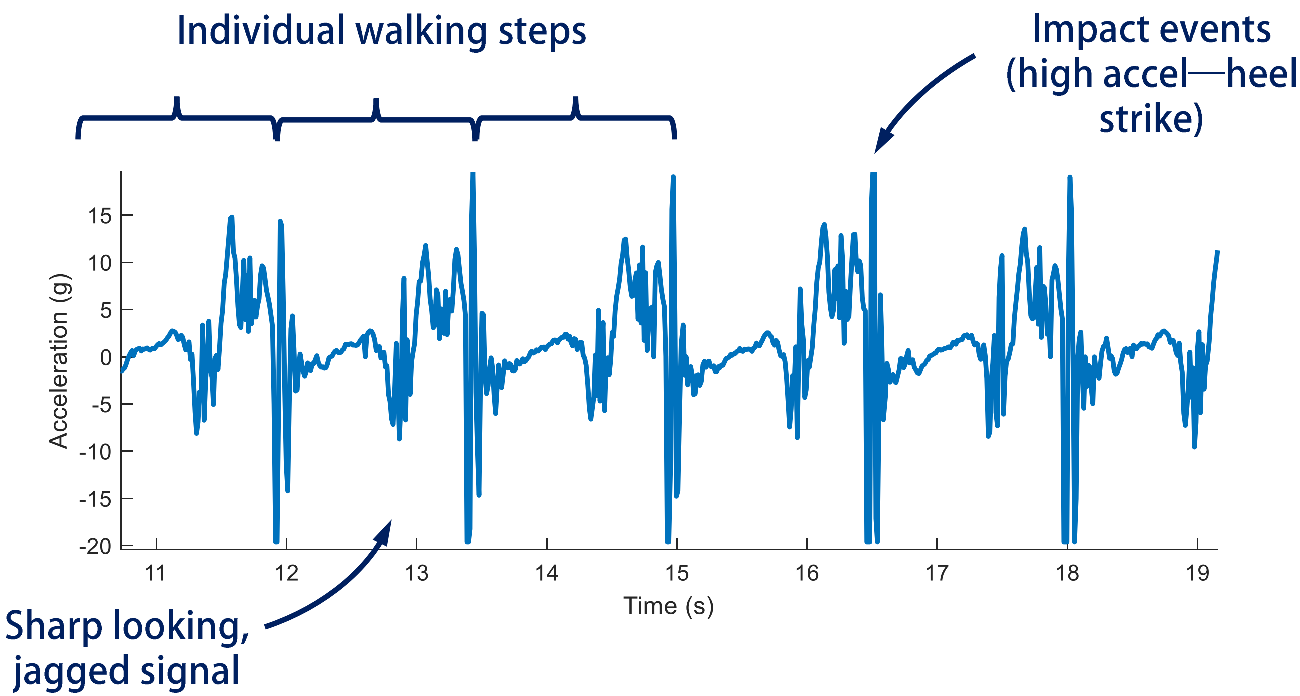
- Consider a recording of three people whistling into a microphone
- Each person is whistling at a different frequency
- What would that signal look like?



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

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?



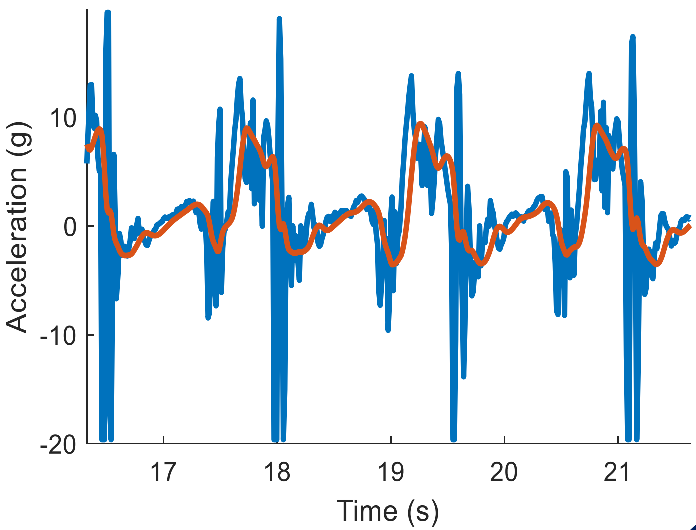
- This signal has some high frequency components—we could filter these out to remove them or isolate them, depending on what we needed

Example IMU Analysis

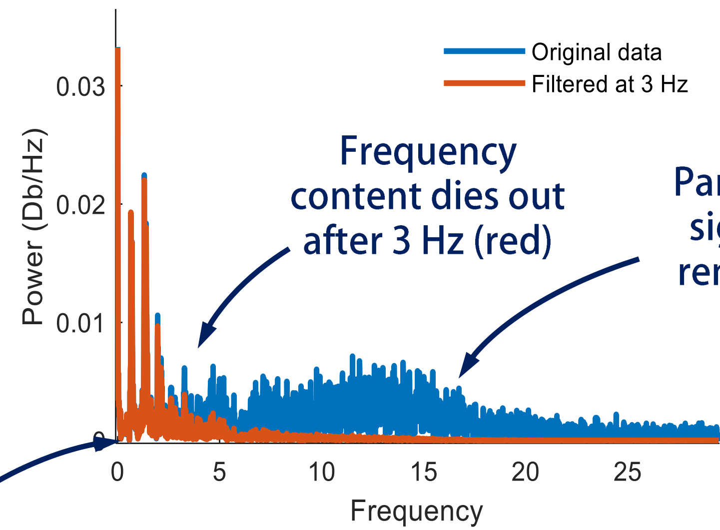
- If we filter the data with a 'low pass' filter, we can attenuate the higher frequency impacts
- This causes a slight delay, depending on the type of filter
- The effect of filtering can be viewed in both the time and frequency domains

Fast Fourier Transform converts back and forth

Time domain



Frequency domain



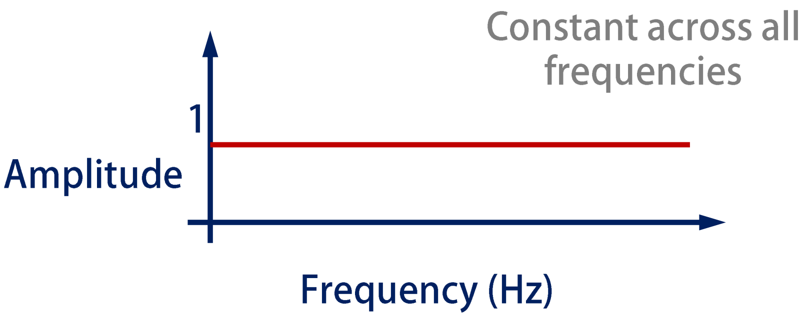
Frequency content dies out after 3 Hz (red)

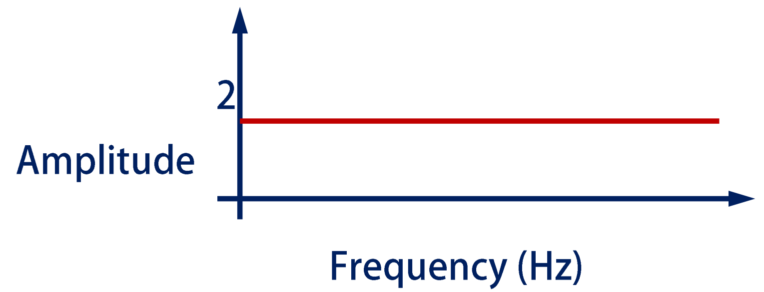
Parts of the blue signal that get removed by the filter

A value at 0 Hz means the signal has a non-zero mean

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?

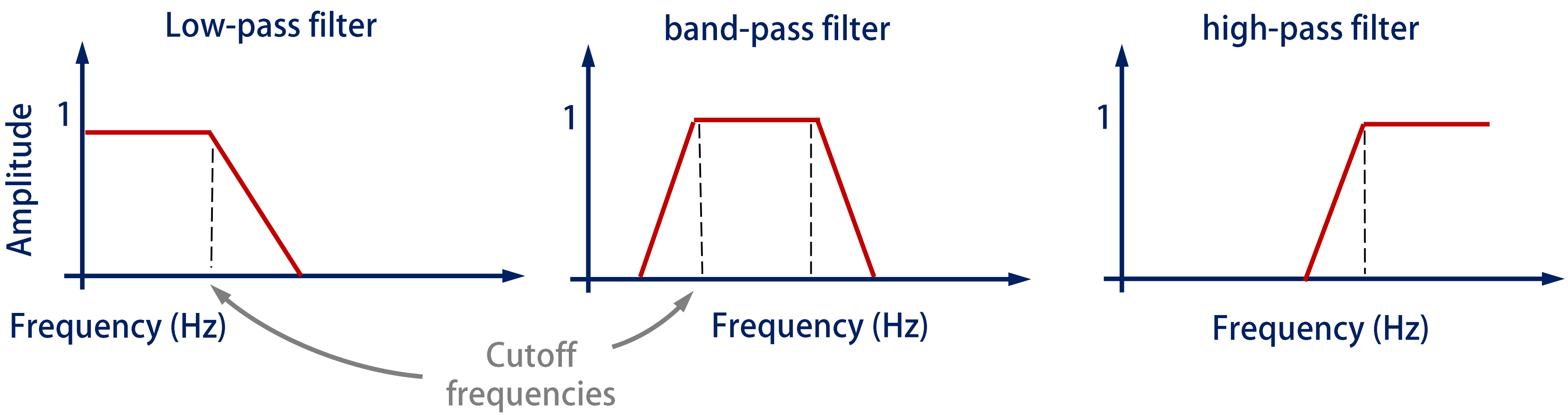




- The left would do nothing and the right would amplify by a factor of 2x
- Filtering is about specifying the exact shape of the multiplying function (red line)
- There are three types of filters, described based on their pass band
  - Low pass – allows low frequencies through
  - Band pass – allows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

Filtering As Multiplication

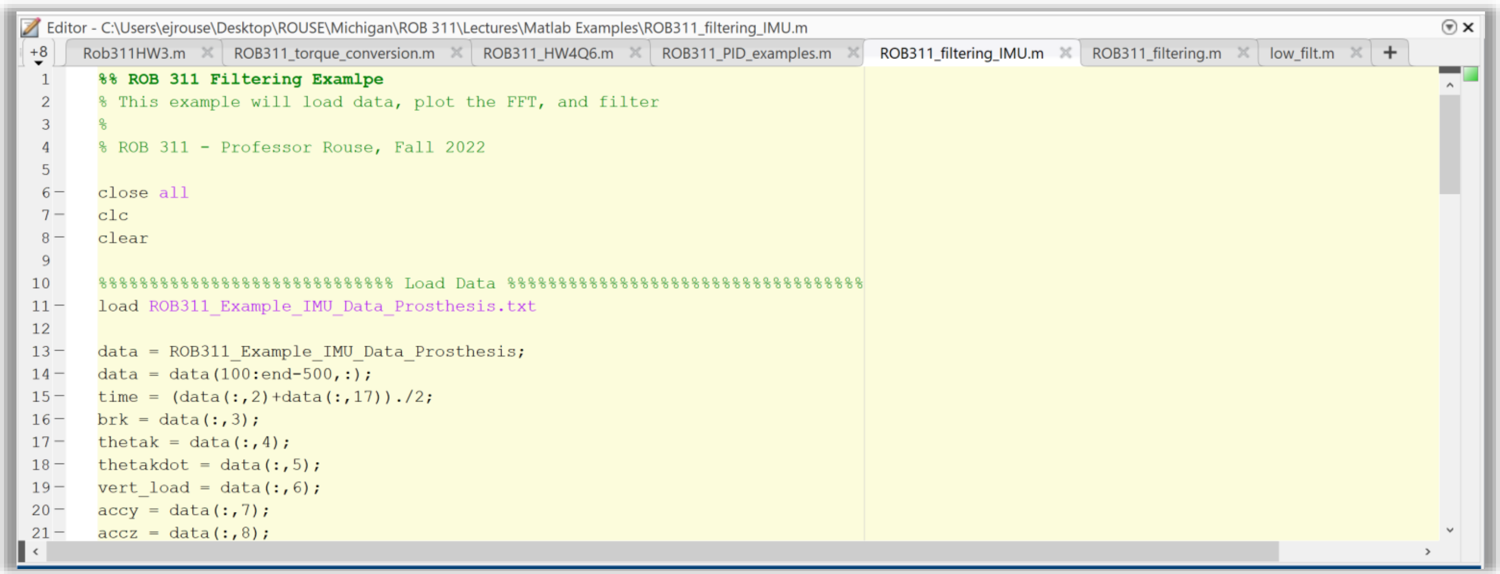
- Lets look at how different types of filter types affect the frequencies that pass through



- 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



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

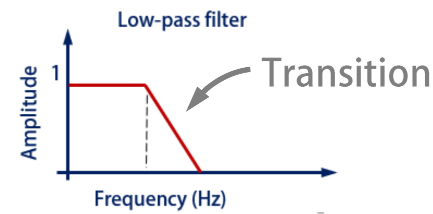
Rob311HW3.m | ROB311_torque_conversion.m | ROB311_HW4Q6.m | ROB311_PID_examples.m | ROB311_filtering_IMU.m | ROB311_filtering.m | low_filt.m

```
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)



M ROBOTICS

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

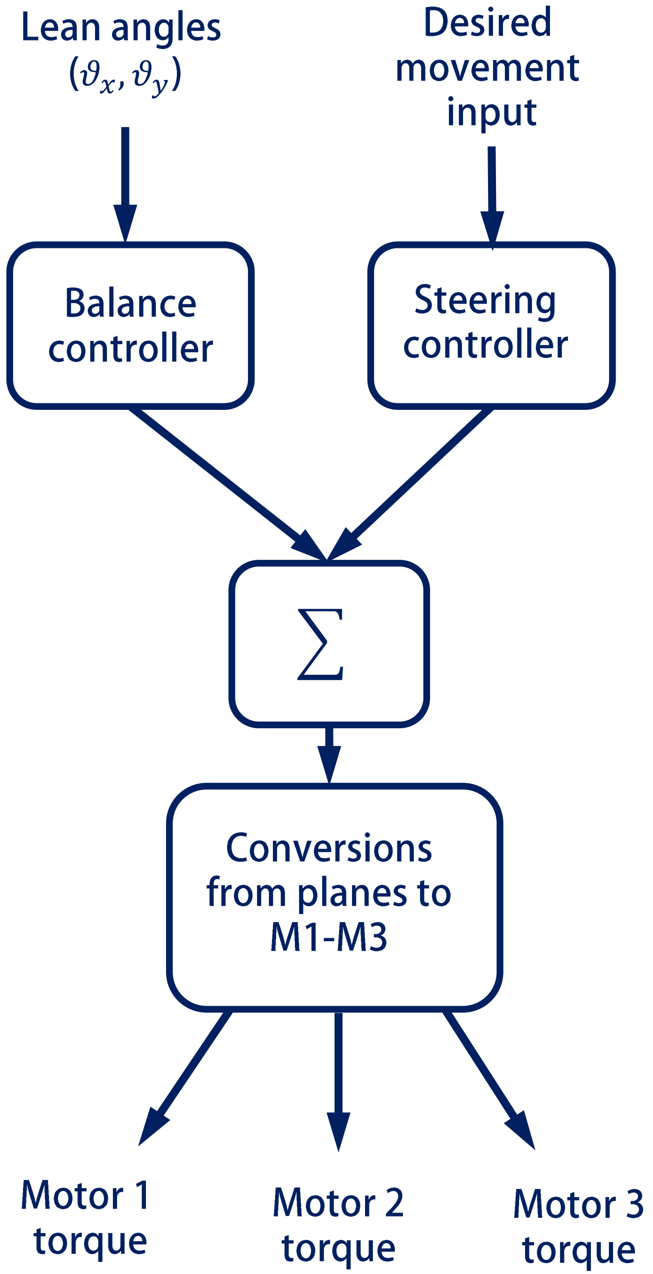
$$x[k] = \left(\frac{1}{n_f}\right) x[k] + \left(\frac{1}{n_f}\right) x[k-1] + \left(\frac{1}{n_f}\right) x[k-2] + \left(\frac{1}{n_f}\right) x[k-3] \ldots$$

  - Moving average is one (simple) type of low-pass filter
- Yves and Senthur showed you Python filtering this week

Steering Control

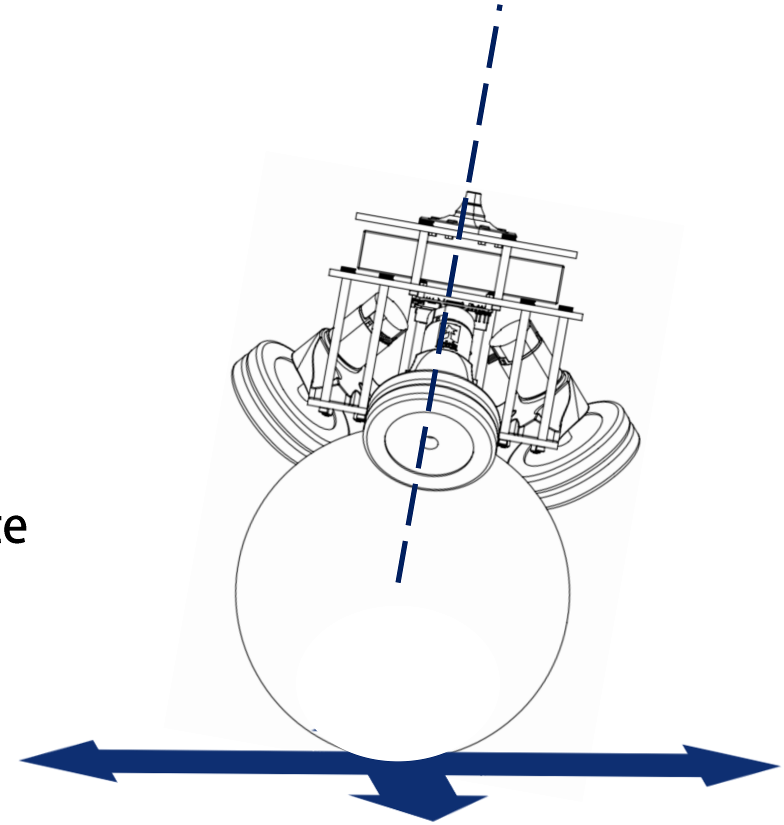
Controller Architecture

- 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
- First, we learned the balance controller
- Now we want to make it drive on command



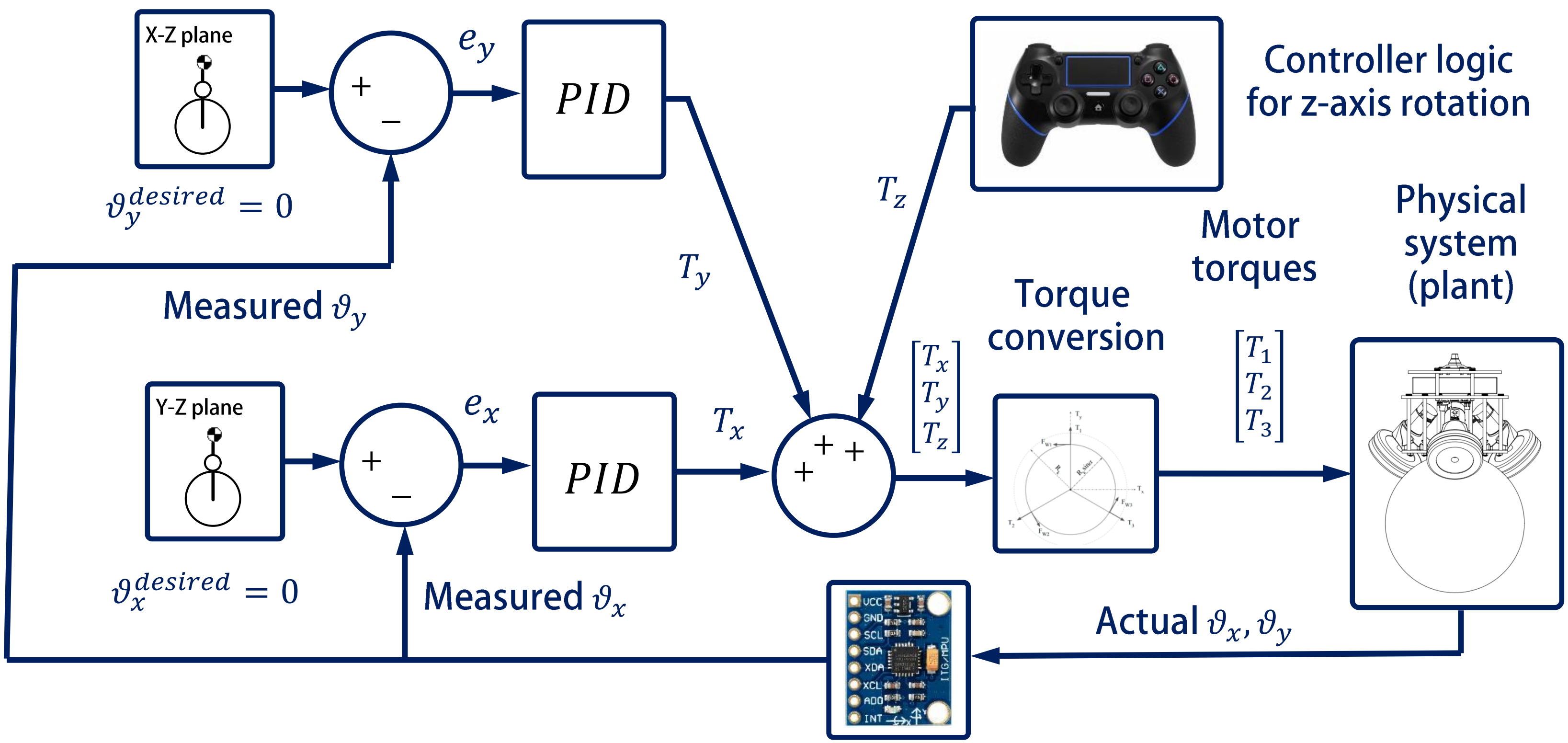
Steering Control

- There are many different ways to build a steering controller for our ball-bots
- We've tried many of the controllers and think we have a controller that will work well for steering
- But first, lets think about our options
- We already know the balance controller will be working
- To add to the balance controller, we could
  - Control ball position in the X-Y plane
  - Control ball velocity in X-Y plane
  - Add torques directly from the PS4 remote (no closed loop steering control)
- Lets review our block diagram and modify for a steering controller



Controller for Z-Axis Torque

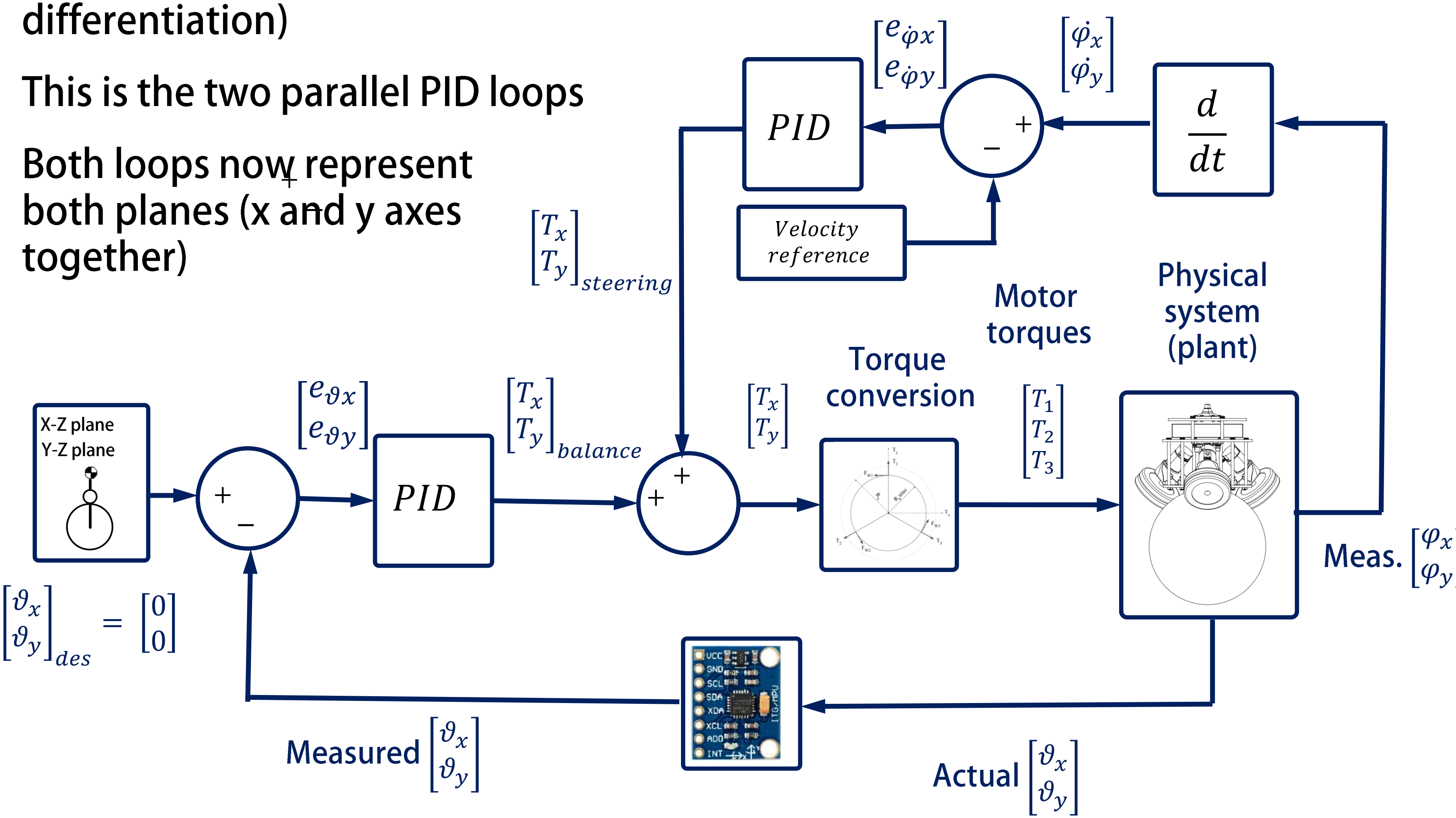
- We have added a z-axis torque that lets the ball rotate arbitrarily around the z-axis
- You developed your own version of this controller using the PS4 remote
- Now, we're going to switch gears and learn our steering controller



M | ROBOTICS

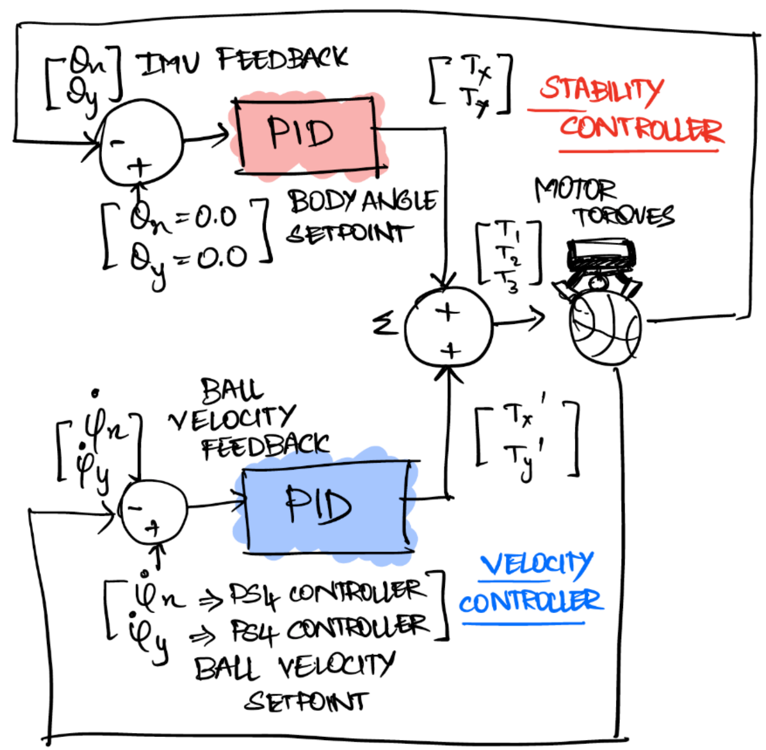
Steering Controller

- To steer, we're going to build a controller that monitors the ball's velocity
- We know the ball's velocity from the motor encoders (via numerical differentiation)
- This is the two parallel PID loops
- Both loops now represent both planes (x and y axes together)



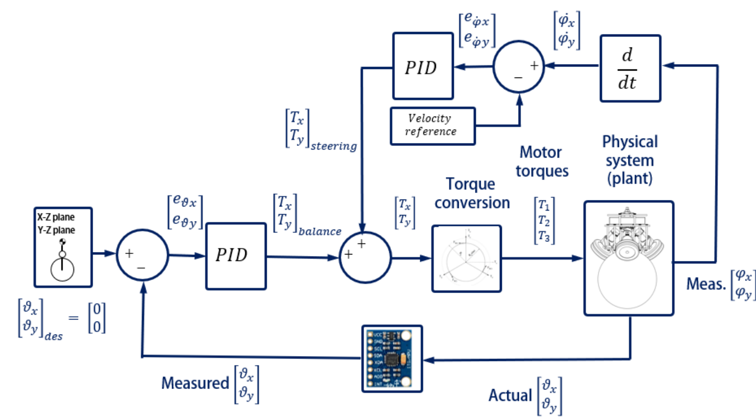
Steering Controller

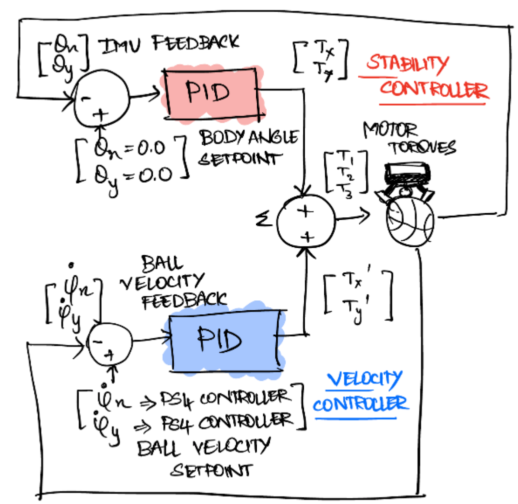
- This is the same diagram, sketched by Senthur
- This shows a little about how different concepts can be represented similarly
- This lets the balance controller and steering controller operate together
- Do we need to be concerned about the balance controller being overpowered?
- What could this cause? Loss of balance
- Lets think about how we combine the torques before they are converted to M1-3 torques
- What could we do to prevent the balance controller from being overpowered



Steering Controller

- What do these block diagrams say?
- Balance while maintaining a ball velocity
- What will this do?
  - Depends on the velocity reference
  - But it will keep the ball from rolling away while balancing
- What if we wanted to control the global position of the ball-bot in the X-Y plane?
- Take a few minutes to draw the block diagram for this type of position control



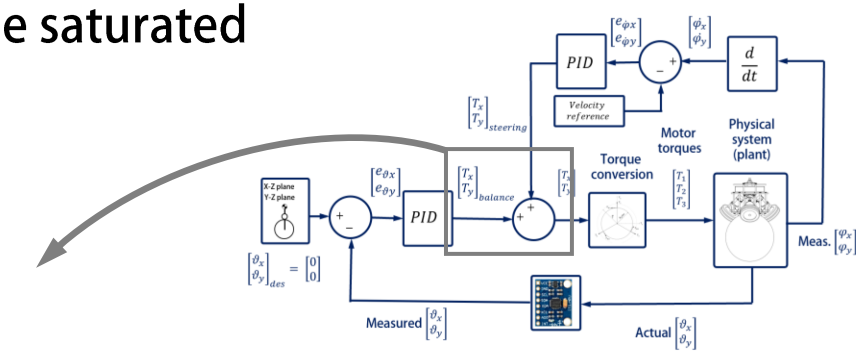


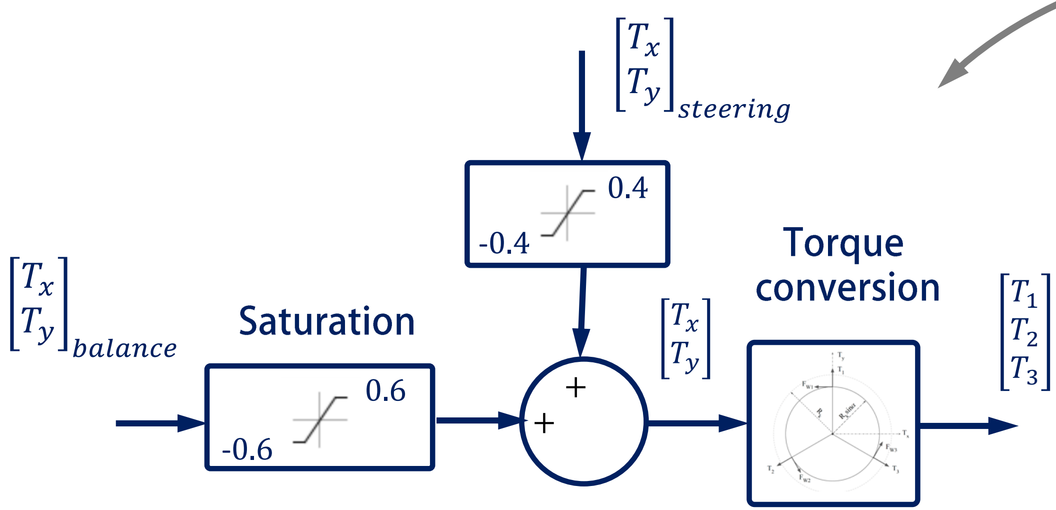
Steering Controller

- Position control block diagram:

Helpful Tips

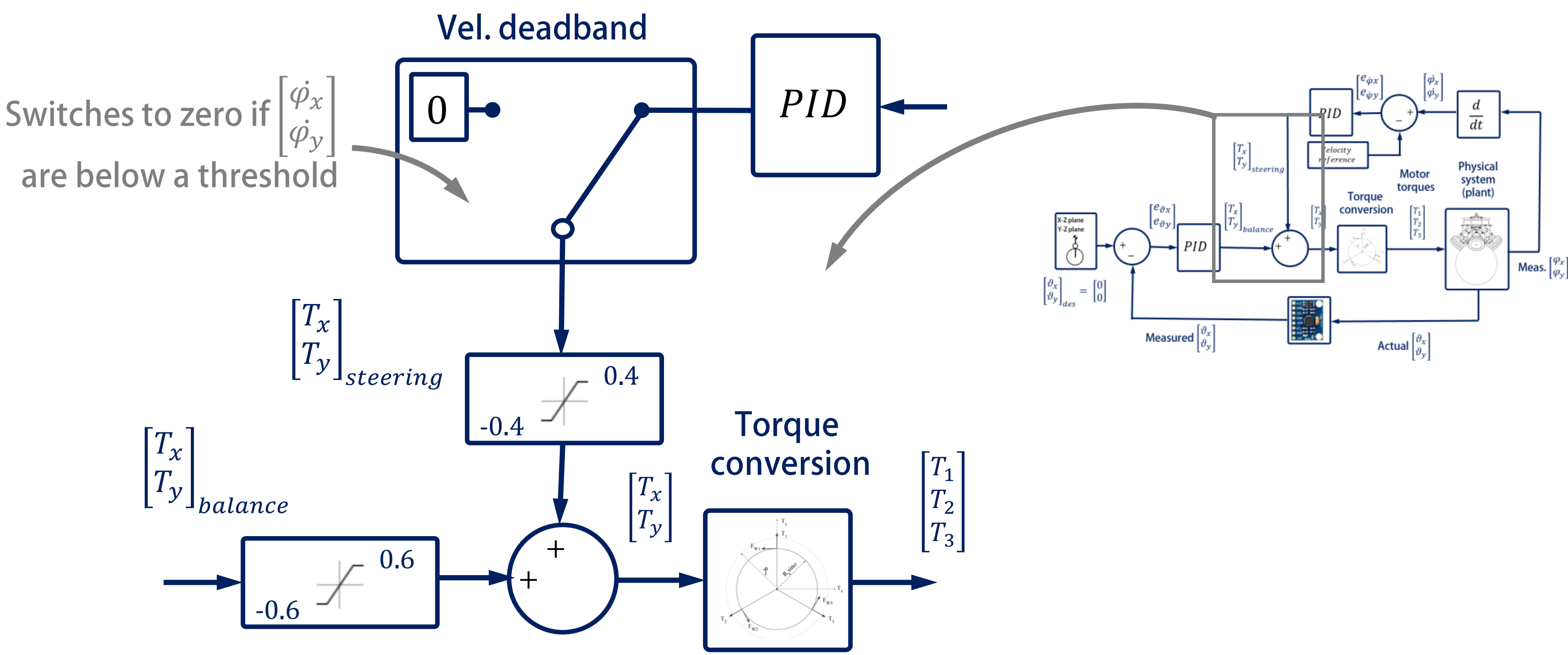
- Now, we will learn a few bells and whistles that help the system maintain controllability
- We want to limit the ability for the velocity controller to overpower the balance controller
- To this end, we will not allow the steering controller to add more than 40% of the max duty cycle / torque
- Both torques (balance and steering) can be saturated
- This helps make the system more stable





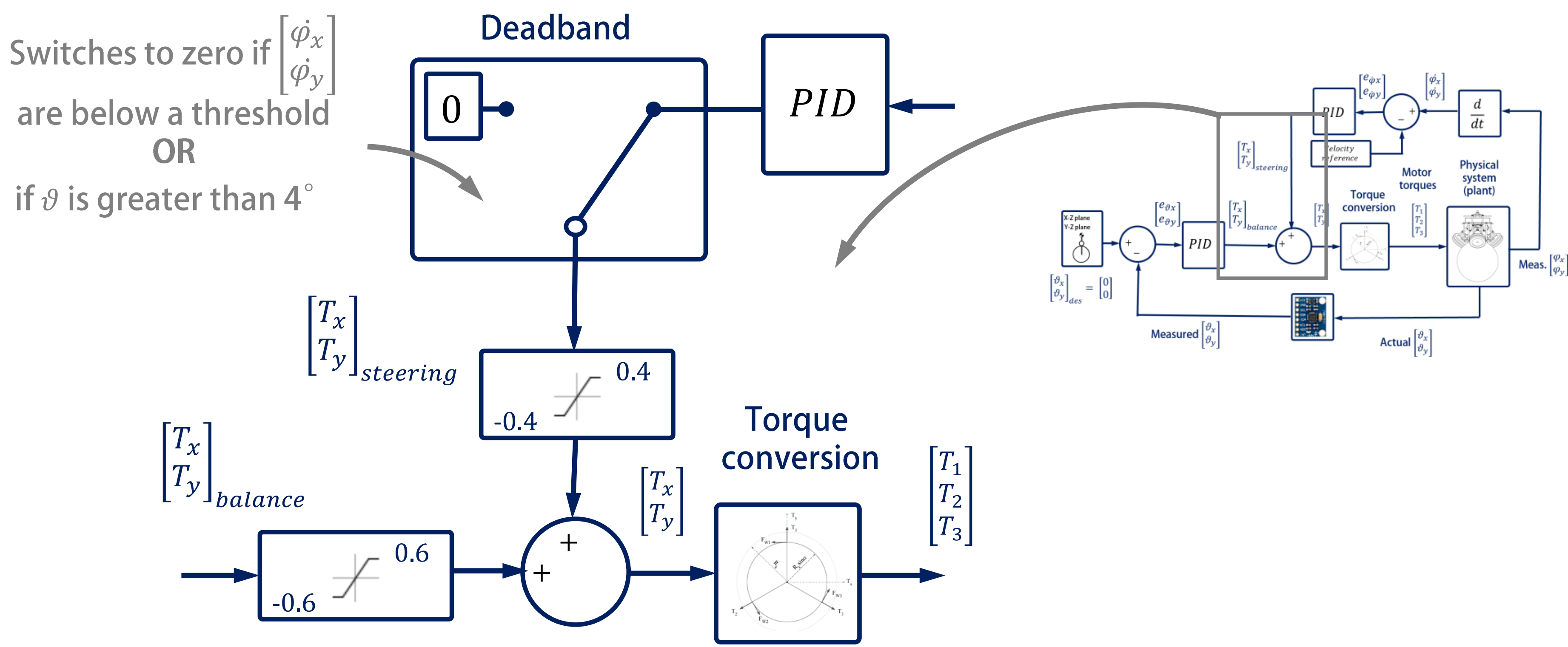
Helpful Tips

- We also find it helpful to add a deadband to the velocity controller
- A deadband tells the controller to do nothing when the error is sufficiently close to zero
- This helps avoid oscillations / transmission backlash



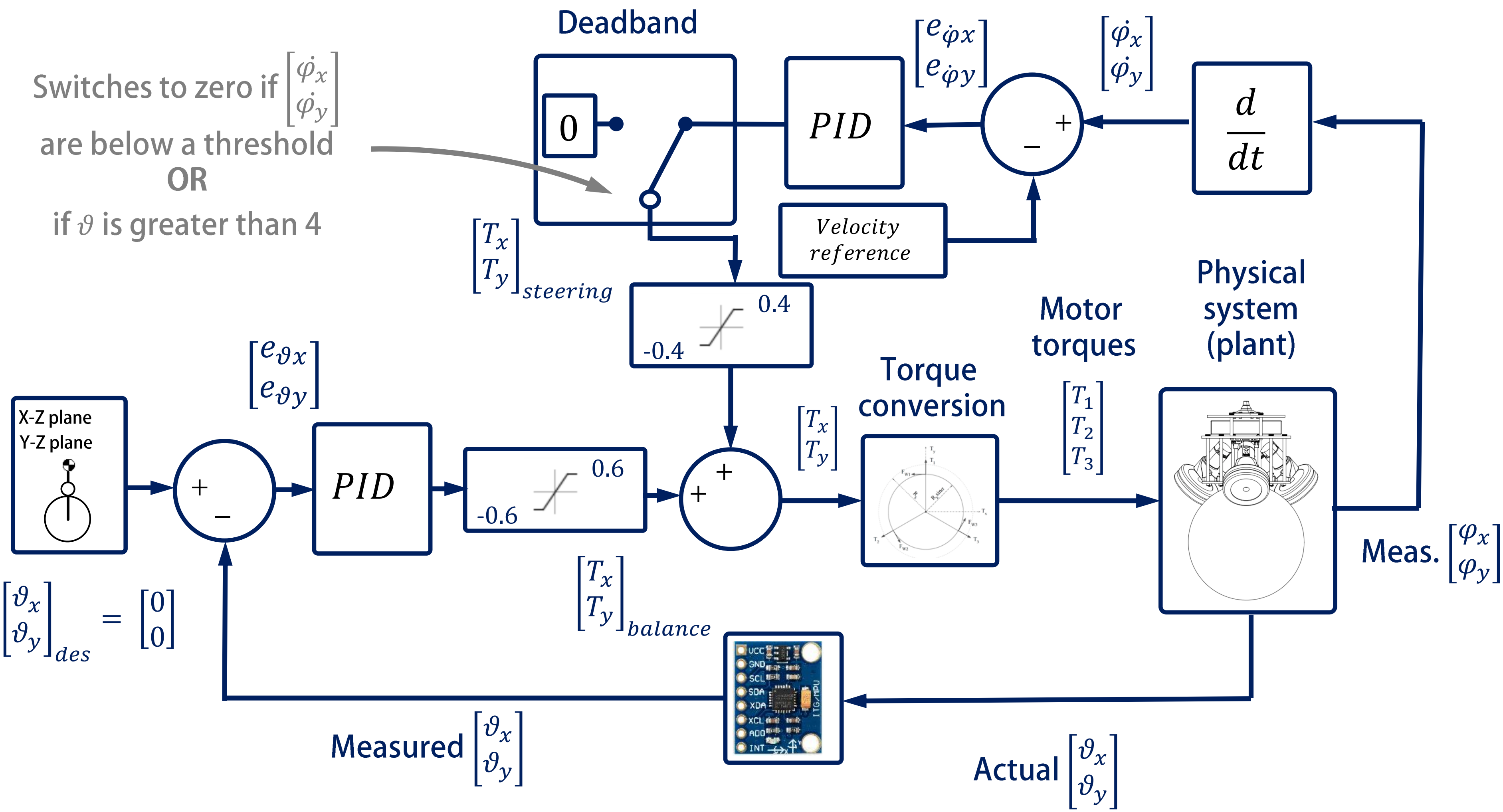
Helpful Tips

- We also want the system to pay attention to its lean angle
- If the lean angle is large, we want to turn the steering control off
- Lets add another logical operation to our deadband
- Now, if the lean angle is greater than 4° the steering controller turns off



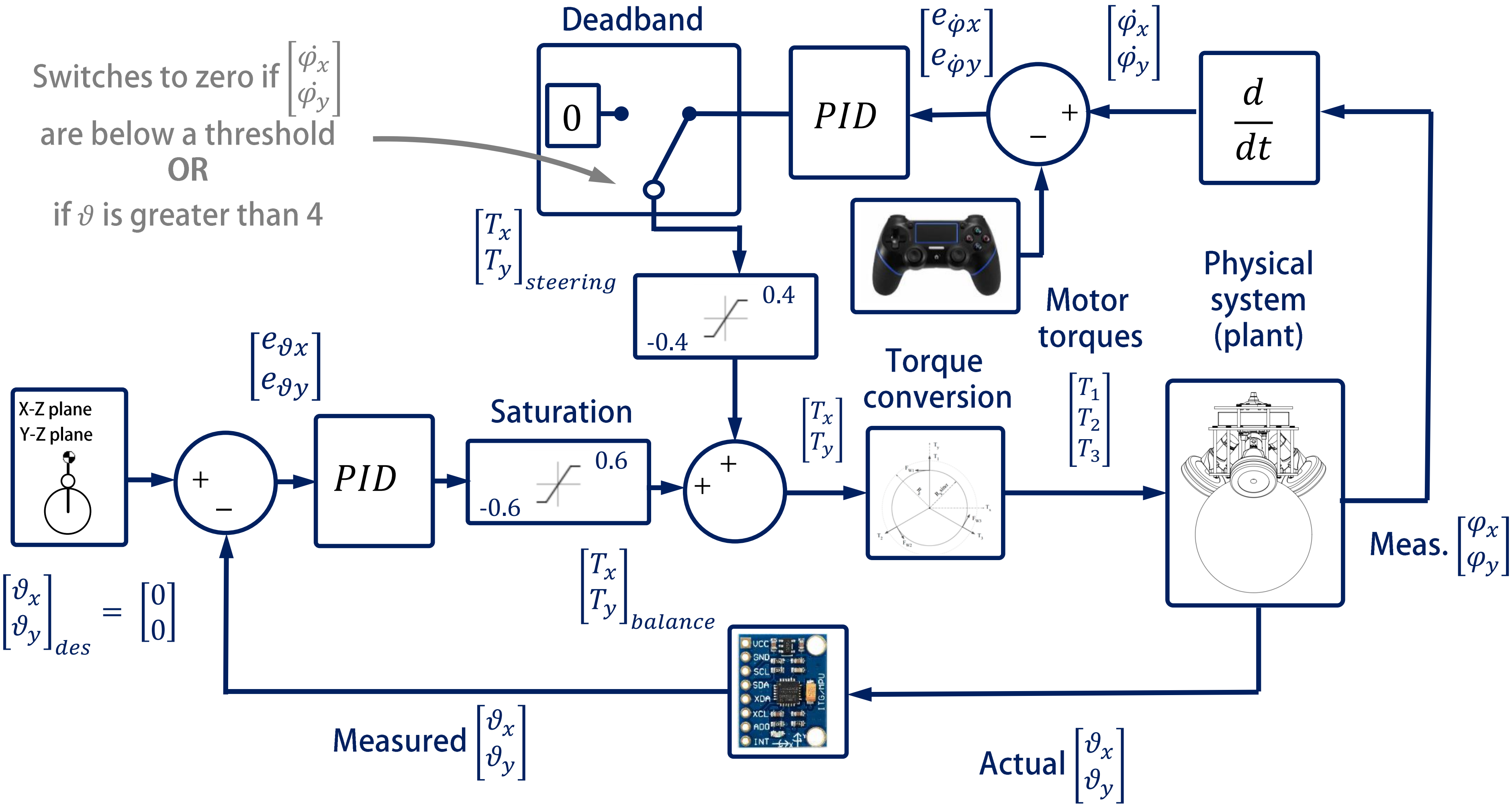
Now We Put it All Together

- Lets add all the components from today so far
- Where does the velocity reference come from?



Now We Put it All Together

- Lets add all the components from today so far
- Where does the velocity reference come from?



Now We Put it All Together

- We have provided the balance + steering controller script on Canvas
- You can use this script, and will likely need to tune your PID gains
  - How many PID controllers are there?
  - Two, each controlling two planes (so, four total)

C:\Users\ejrouse\Desktop\ROUSE\Michigan\ROB 311\Lectures\Content for Lectures\ROB311_steering_demo.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

ROB311_ball_kinematics_demo_vEJR.py | FIR.py | ROB311_Python_filtering.py | bbot_app.py | ROB311_steering_demo.py

```python
import sys
import threading
import time
import numpy as np
from threading import Thread
from MBot.Messages.message_defs import mo_states_dtype, mo_cmds_dtype, mo_pid_params_dtype
from MBot.SerialProtocol.protocol import SerialProtocol
from rtplot import client
from scipy.signal import butter, lfilter, filtfilt
from simple_pid import PID
from pyPS4Controller.controller import Controller
import board
import adafruit_dotstar as dotstar
from enum import Enum
from collections import deque
from DataLogger import dataLogger

import FIR as fir
"""
ROB 311 - Ball-bot steering demo

This program uses a soft realtime loop to enforce loop timing. Soft real time loop is a  class
designed to allow clean exits from infinite loops with the potential for post-loop cleanup operations executing.

The Loop Killer object watches for the key shutdown signals on the UNIX operating system (which runs on the PI)
when it detects a shutdown signal, it sets a flag, which is used by the Soft Realtime Loop to stop iterating.
Typically, it detects the CTRL-C from your keyboard, which sends a SIGTERM signal.

the function_in_loop argument to the Soft Realtime Loop's blocking_loop method is the function to be run every loop.
A typical usage would set function_in_loop to be a method of an object, so that the object could store program state.
See the 'ifmain' for two examples.

Authors: Senthur Raj, Gray Thomas, Yves Nazon and Elliott Rouse
Neurobionics Lab / Locomotor Control Lab
"""
```

What Did You Learn?

"I have an idea for a robot!"

Spec'ing
Motor
Ratio
Power

Layout
Geometry
Component selection

3D Modeling
3D design
File generation
Prep for mfg.

Manufacturing
Additive mfg.
Laser cutting
Water jetting

Sensing
IMUs
Encoders
DAQ

Communication
I^2C
SPI

Wheels
Omni-wheels
Mecanum wheels

Control
PID
Balance control
Steering control

- You have learned how to build robots and make them move!
- The Tuesday + Wednesday next week are implementation / finalizing
- Competition on Thursday 12/8!