2.004 Lab 8 Intro:
Self-Balancing Robot Control
Part I: Stabilization Using PID Control

Fall, 2021

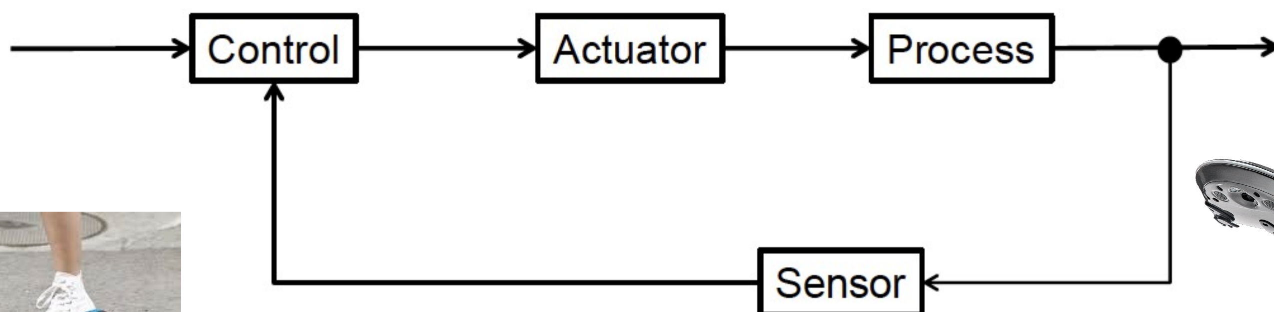
Objectives

- **Given the mechanical properties of the 2.004 self-balancing robot and its open-loop transfer function, design a stabilization feedback control (PD controller) using Matlab's sisotool.**
- **Implement the stabilization feedback control on the self-balancing robot and test the controller.**
- **Improve the controller design by adding an integral term to make it a PID controller.**
- **Fine-tune your PID controller to achieve long-term stability.**
- **Stability Competition (last 20 minutes of lab).**
- **Deliverable (one per group): Segway robot control (Labs 8 and 9) report**
  - Due dates:
    - Monday's sections: 11/30 midnight
    - Tuesday's sections: 12/1 midnight
    - No submission after 12/2

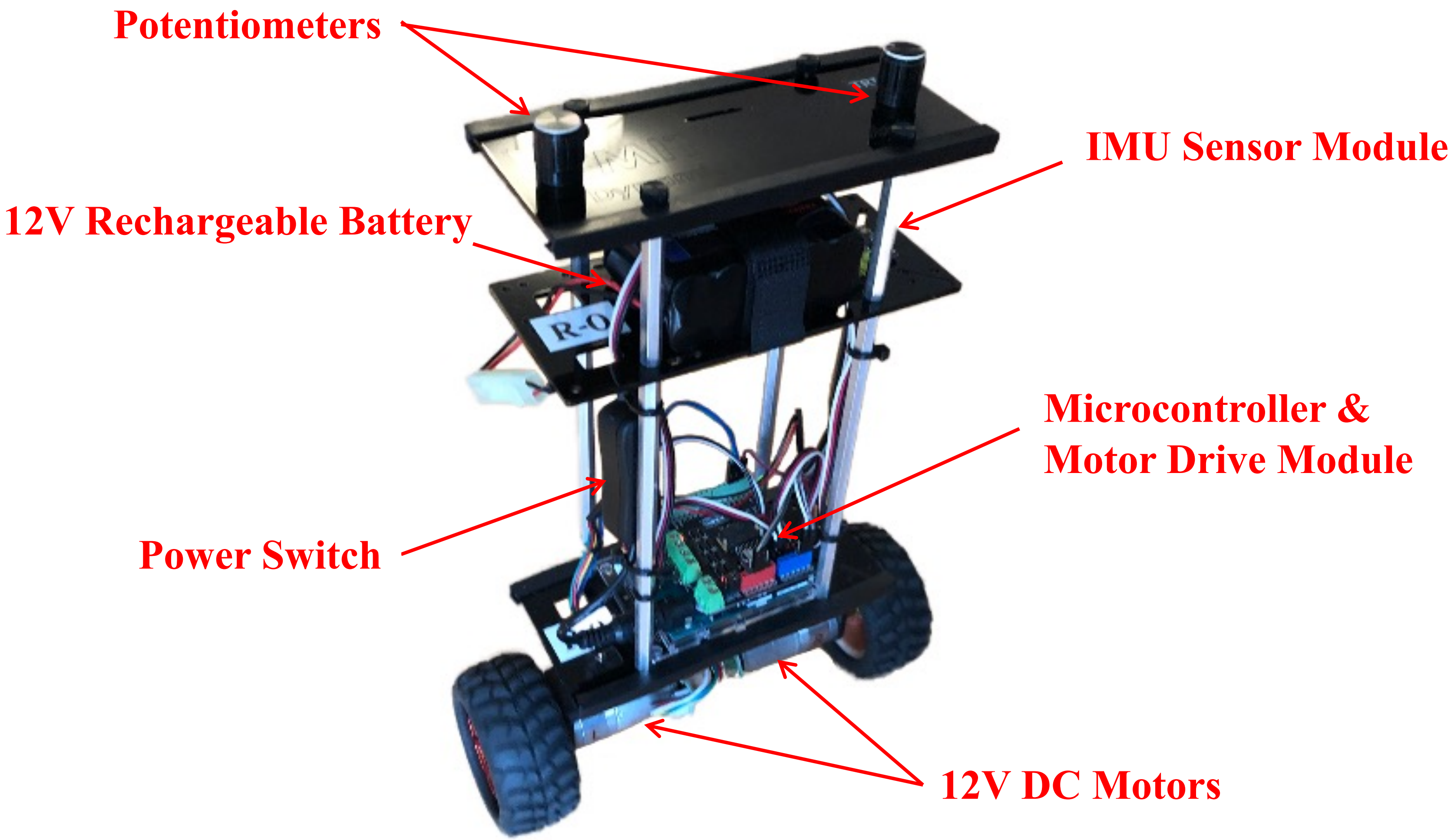
Segway Control

http://www.segway.com/

- Need control system to prevent crashes (open loop unstable)
- Stabilization control similar to balancing an inverted pendulum



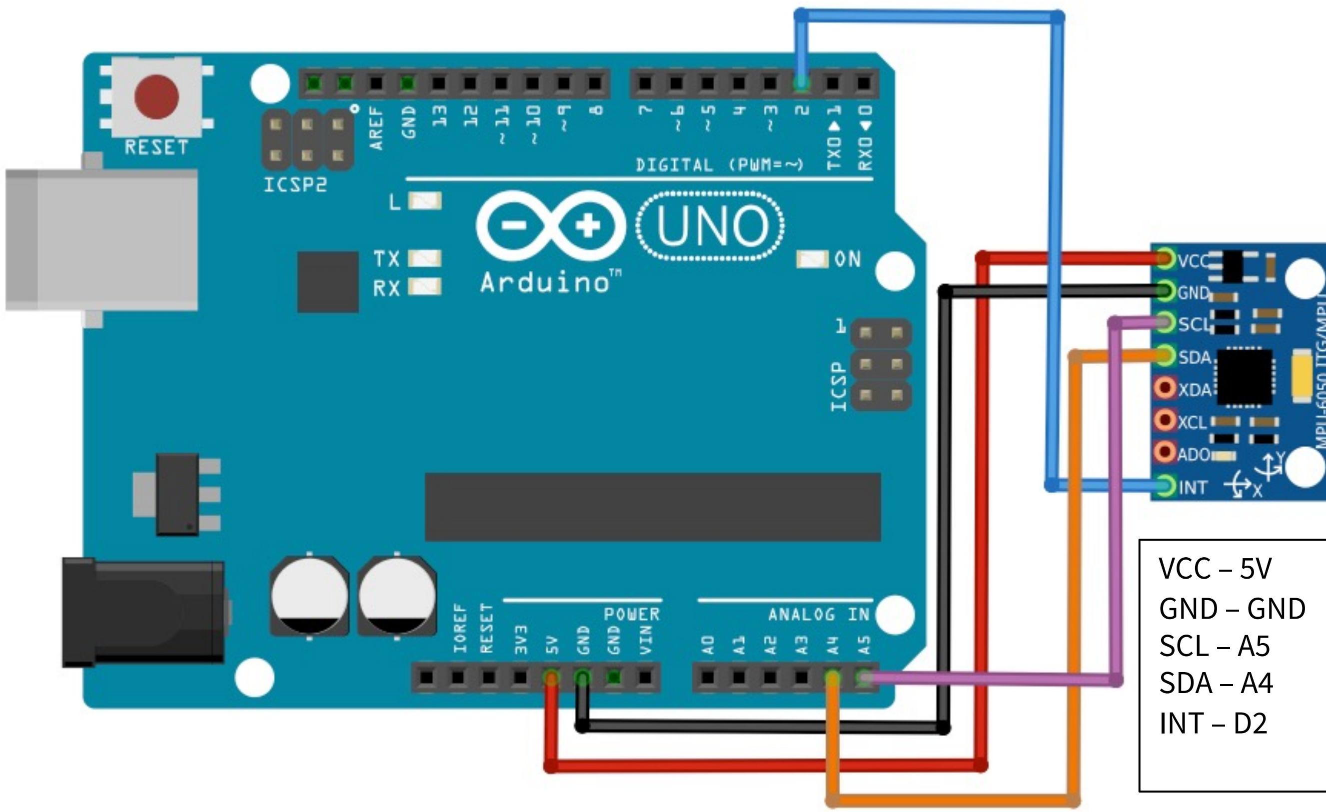
2.004 Self-Balancing Robot



IMU: MPU-6050 (by TDK InvenSense)

- The MPU-6050 is the world's first integrated 6-axis Motion Tracking device.
- It combines a 3-axis gyroscope, 3-axis accelerometer, and a Digital Motion Processor™ (DMP) all in a small 4x4x0.9mm package.
- It uses a standard I2C bus for data transmission.
  - With it's I2C bus, it can accepts inputs from an external 3-axis compass to provide a complete 9-axis Motion Fusion output.
- A number of different breakout boards are available containing the MPU-6050 chip.

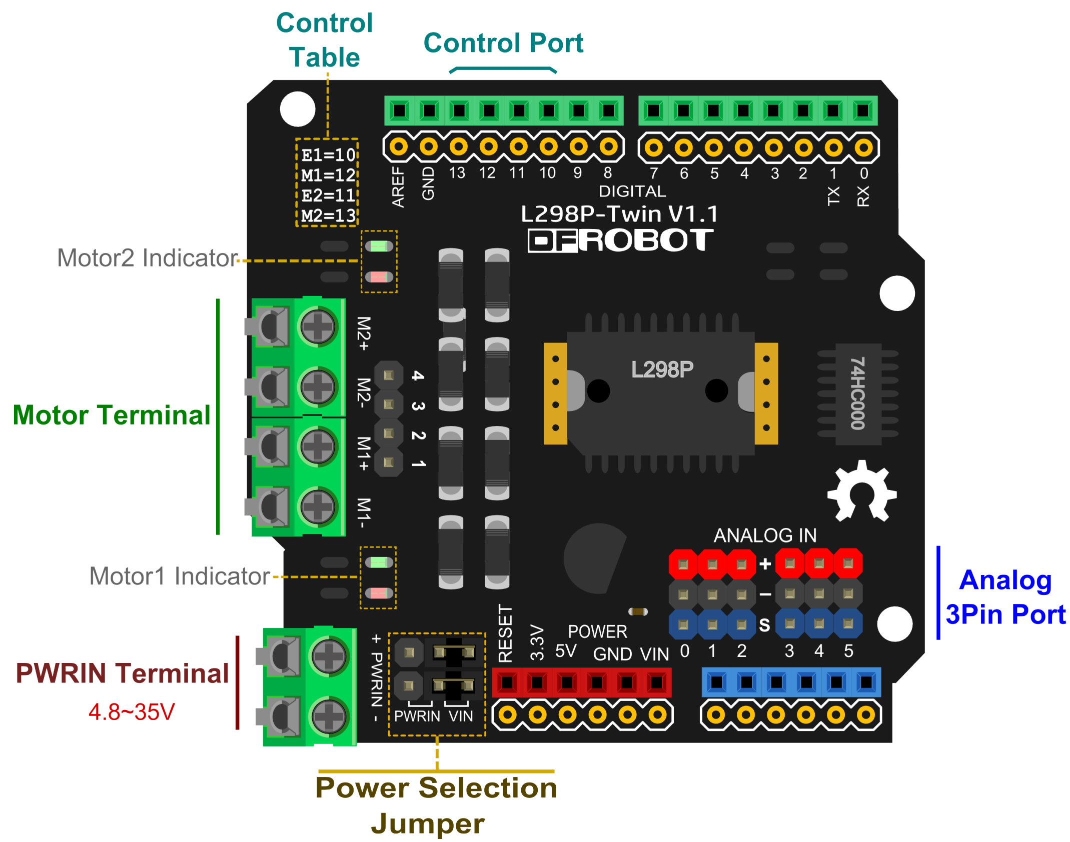
MPU-6050 Arduino Connection Diagram



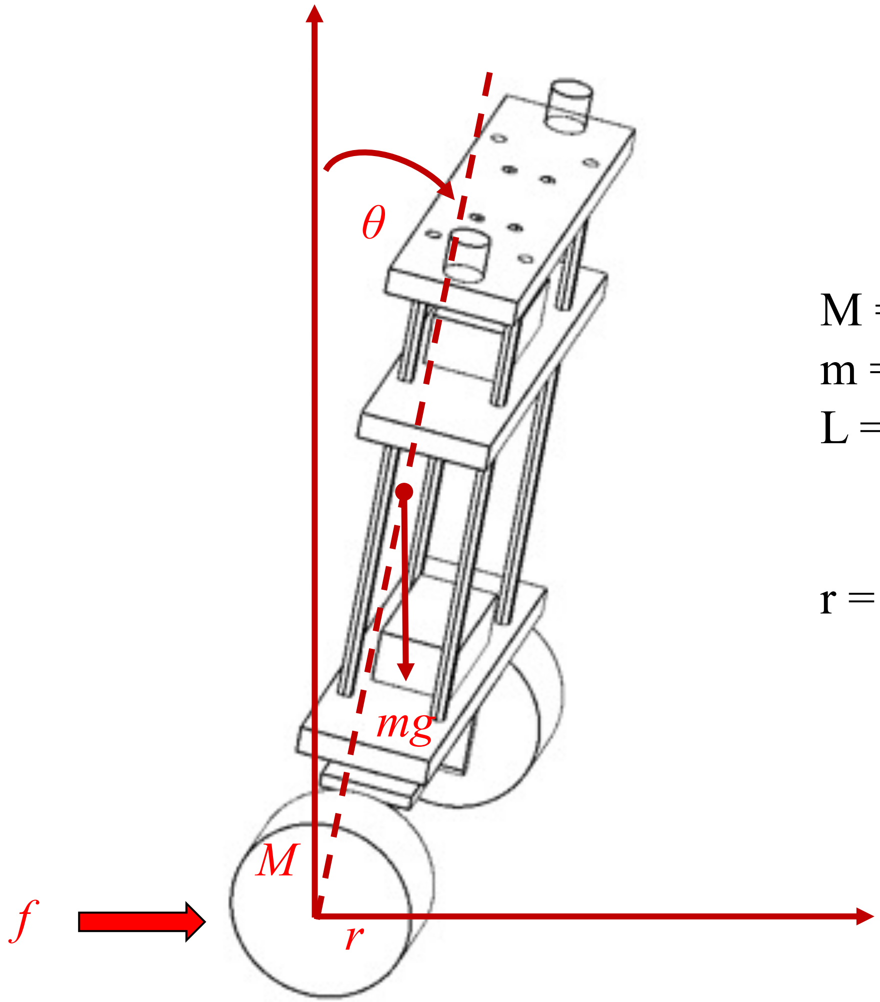
VCC – 5V
GND – GND
SCL – A5
SDA – A4
INT – D2

Dual Channel 2A Motor Shield

- The motor shield allows Arduino to drive two channel DC motors, using an L298N chip which deliveries output current up to 2A each channel.
  - Motor Driven Voltage: 4.8V to 35V
  - Output Current: up to 2A/channel
  - Total Power Dissipation: 25W (T=75℃)
  - Driven Structure: Dual full-bridge driver

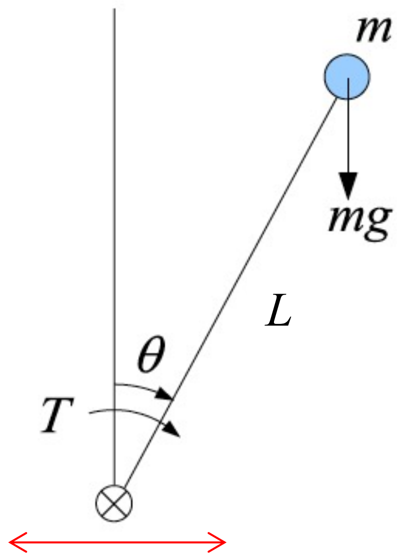


Mechanical Properties



M = 0.34; % [Kg] mass of wheels
m = 0.31; % [Kg] mass of body
L = 0.18; % [m] length of body from pivot point to CG, total length of body is 0.26 m
r = 0.035; % [m] wheel radius

Inverted Pendulum Model



$$T_{inertial}(t) = T_{gravity}(t) + T_{applied}(t)$$

$$\Rightarrow mL^2\ddot{\theta}(t) = mgL\sin\theta(t) + T_{applied}(t)$$

$$\Rightarrow \ddot{\theta}(t) - \frac{g}{L}\sin\theta(t) = \frac{T_{applied}(t)}{mL^2}$$

When $T_{applied}(t) = 0$

$$\Rightarrow \ddot{\theta}(t) = \frac{g}{L}\sin\theta(t)$$

A longer stick produces a slower angular acceleration so it is easier to balance.

$$\frac{\Theta(s)}{T(s)} = \frac{\left(1/mL^2\right)}{s^2 - \left(g/L\right)}$$

By increasing the length L, the unstable pole $+\sqrt{g/L}$ would move closer to the origin, *i.e.,* the time constant would increase. This means that the transient response become slower and easier to control (based on the characteristic equation only).

Relating PWM to Motor Torque

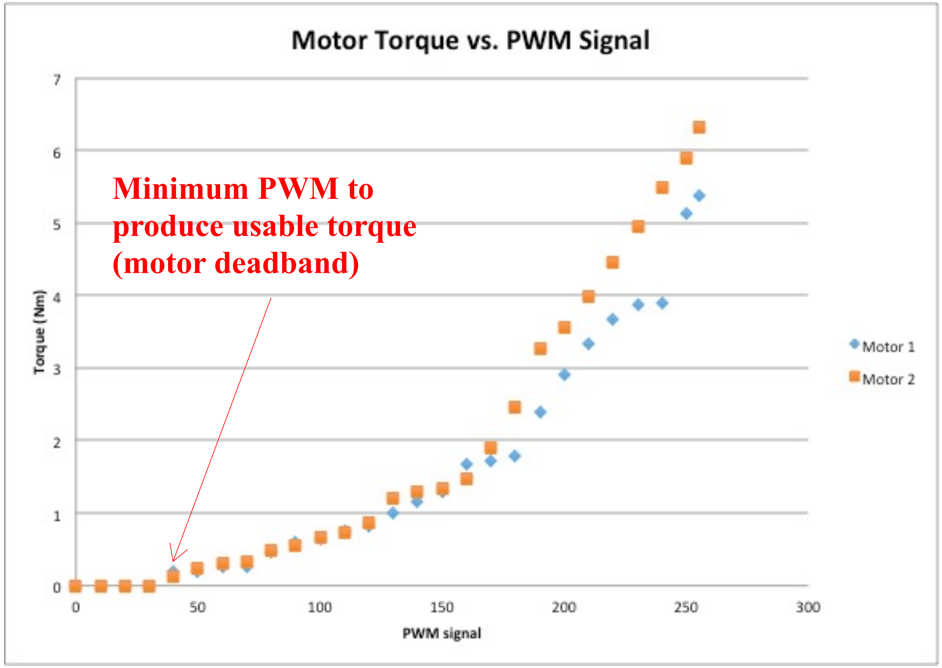
- **PWM signal is used in most microcontroller to approximate an analog output voltage. For Arduino the PWM range is between 0 and 255 (8-bit) which corresponds to 0 – 5 V.**
- **Motor torque can be expressed as:**

$$T_m = K_m \cdot i = K_m \left(\frac{V}{R}\right) = \frac{K_m}{R} K_{pwm} \cdot PWM$$

$$\Rightarrow \frac{T_m}{PWM} = \frac{K_m K_{pwm}}{R} = K_q$$

The gain was experimentally determined as:

$$K_q \approx 0.0015 \quad [\frac{Nm}{PWM}]$$



State-Space Model of the Robot

$$\dot{x} = Ax + Bu$$

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ (M+m)g/Ml & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -mg/M & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ -1/Ml \\ 0 \\ 1/M \end{bmatrix} f$$

A more realistic model uses a rod with moment of inertia J instead of a point mass, as well as a friction force with damping coefficient b between ground and wheels.

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{mgl(M+m)}{q} & 0 & 0 & \frac{-mlb}{q} \\ 0 & 0 & 0 & 1 \\ \frac{-m^2gl^2}{q} & 0 & 0 & \frac{-(J+ml^2)b}{q} \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{-ml}{q} \\ 0 \\ \frac{J+ml^2}{q} \end{bmatrix} f$$

where $q = J(M+m) + Mml^2$

Simplified State-Space Model

Ignoring the friction (i.e., $b = 0$) then we can further simplify the model as:

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \dfrac{mgl(M+m)}{q} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \dfrac{-m^2 g l^2}{q} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \dfrac{-ml}{q} \\ 0 \\ \dfrac{J+ml^2}{q} \end{bmatrix} f$$

where $q = J(M+m) + Mml^2$

Transfer function relating the force to tilt angle:

$$\boxed{\frac{\Theta(s)}{F(s)} = \frac{ml}{qs^2 - mgl\left(M+m\right)}}$$

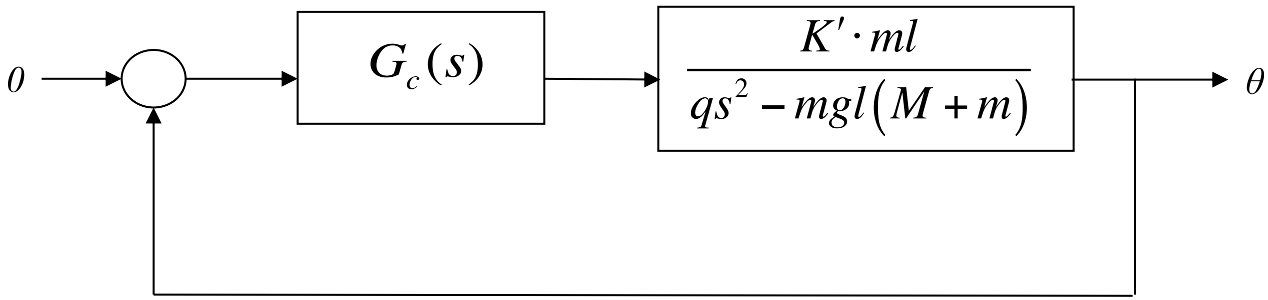
Here the transfer function is made positive by defining the force f and the angle θ in opposite directions so we can work with standard negative feedback control.

Controller Design

- **All relevant files are in 2.004 Course Locker under “Labs\Lab8” folder:**
  - Seqway_robot_template folder – Arduino code for controlling the robot
  - segway_model.m – Matlab script with model parameter values and the transfer function of interest
  - Segway Robot Control Report Guidelines.docx
- **Tasks:**
  1. Design a PD controller by placing the controller zero at around -10 and have a closed-loop damping ratio of around 0.7. Iteratively fine-tune the gains to observe the behavior of the robot.
  2. Design a PID controller to improve the performance of the robot. Try to maintain K_p and K_d values from your PD control design. Fine-tune the gains to obtain the best possible response.

Controller Design

Control the tilt angle using the transfer function and a PD or PID controller:



K' represents the overall system gain

<u>*PD controller transfer function*</u>   A real zero

$$G_c(s) = K_p + K_d s = K_d(s+z) = K_d\left(s + \frac{K_p}{K_d}\right) = K_p\left(1 + \frac{K_d}{K_p}s\right)$$

<u>*PID controller transfer function*</u>   An integrator and a pair of (complex) zeros

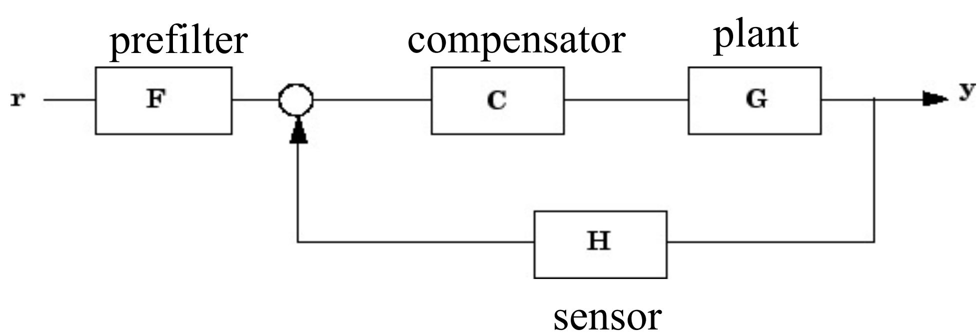
$$G_c(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s} = K_d\left(\frac{s^2 + \left(K_p/K_d\right)s + \left(K_i/K_d\right)}{s}\right)$$

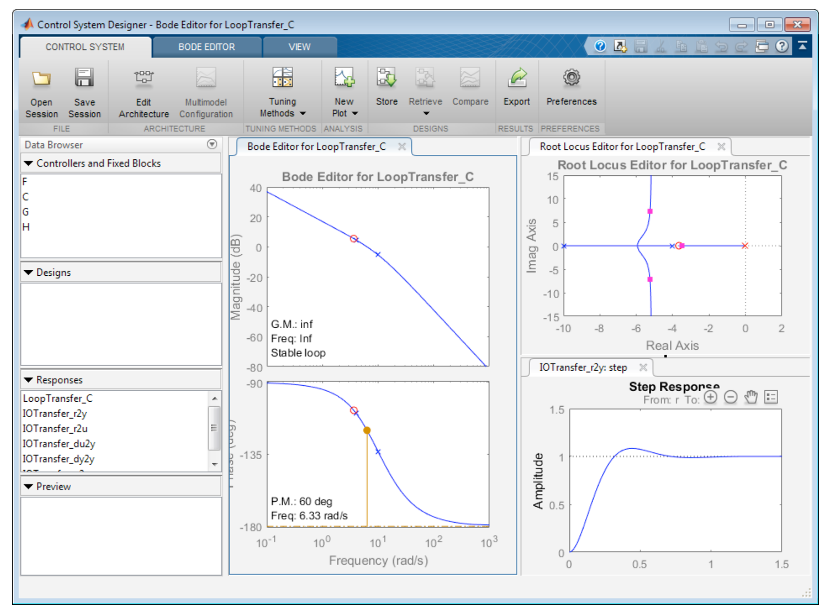
Matlab SISOTOOL

- **Interactively design and tune a SISO (Single Input Single Output) feedback system**

>> sisotool(plant,comp,sensor,prefilt)

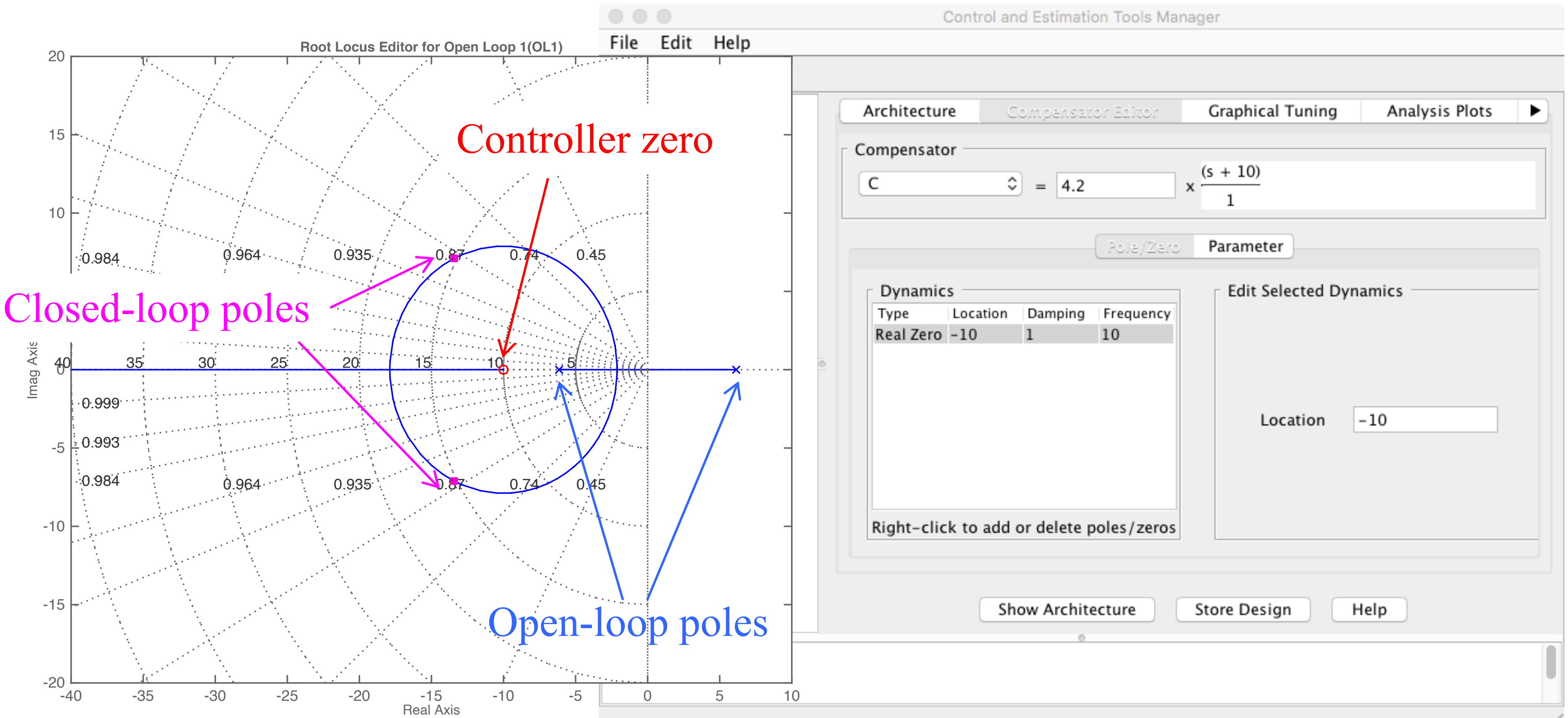
(The *sisotool* command has been updated to *controlSystemDesigner* in current version of Matlab)



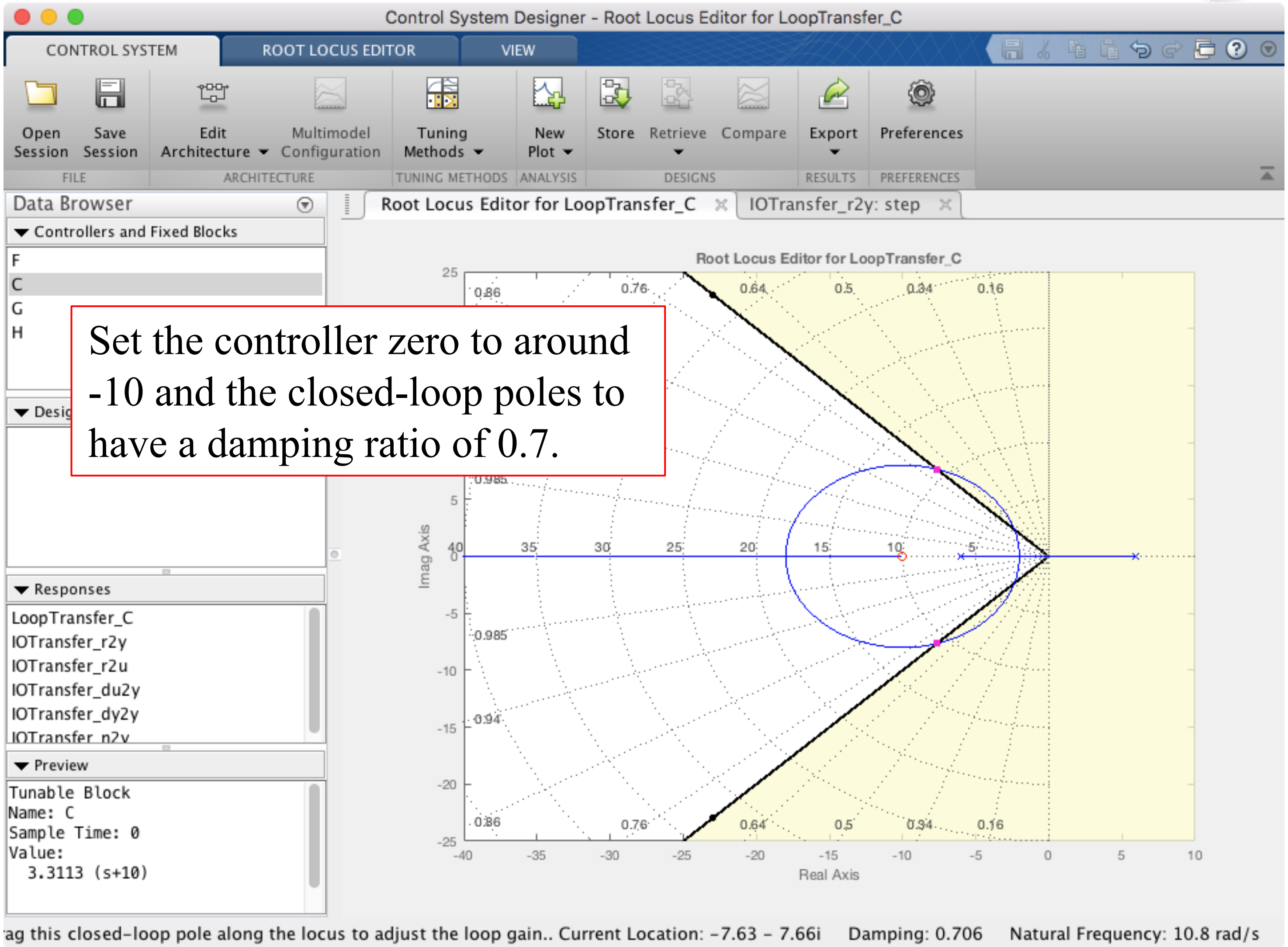


Stabilization Control (PD Controller)

A PD controller is used to “pull” the unstable pole to the L.H. plane.



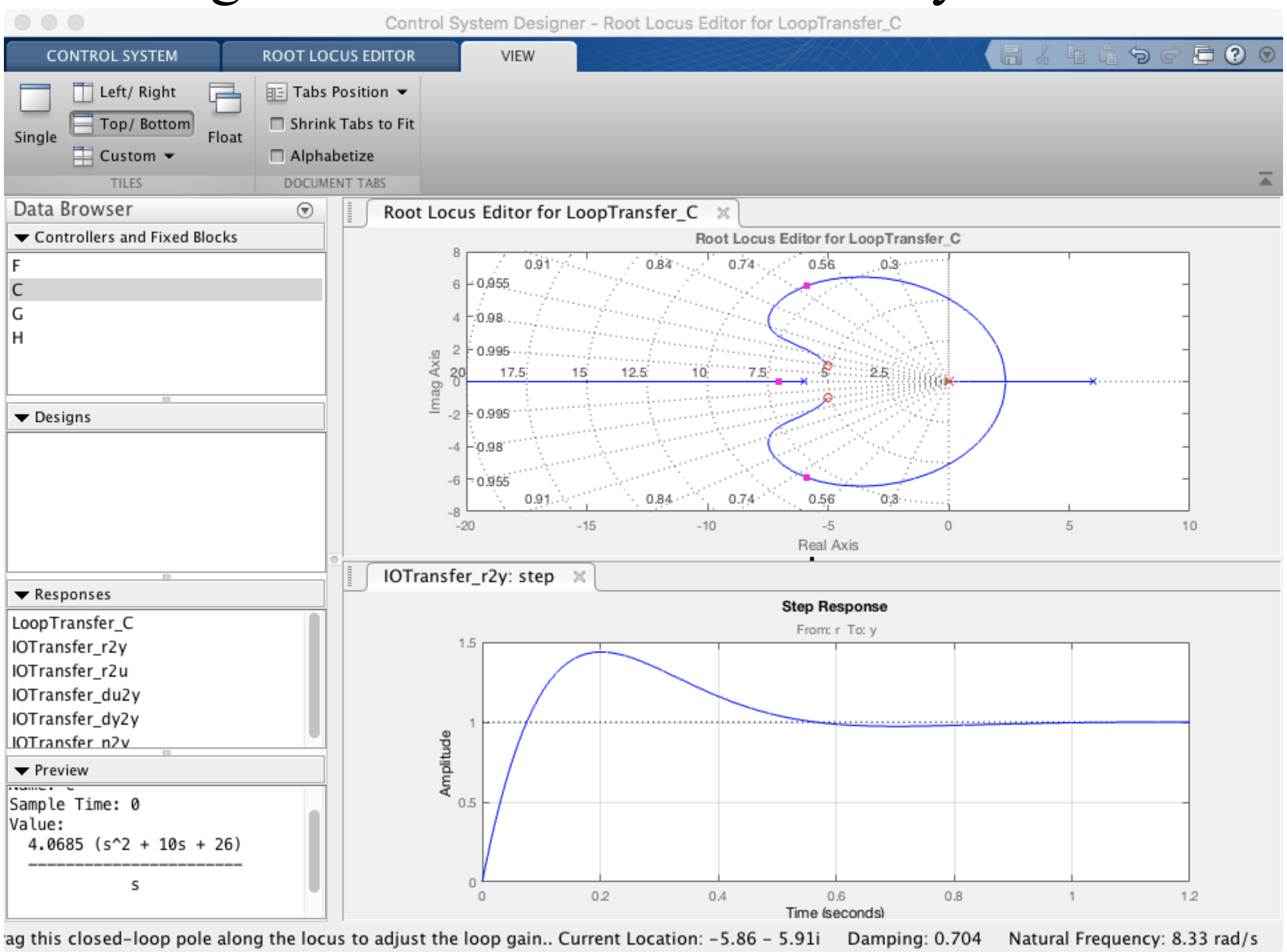
PD Controller Design



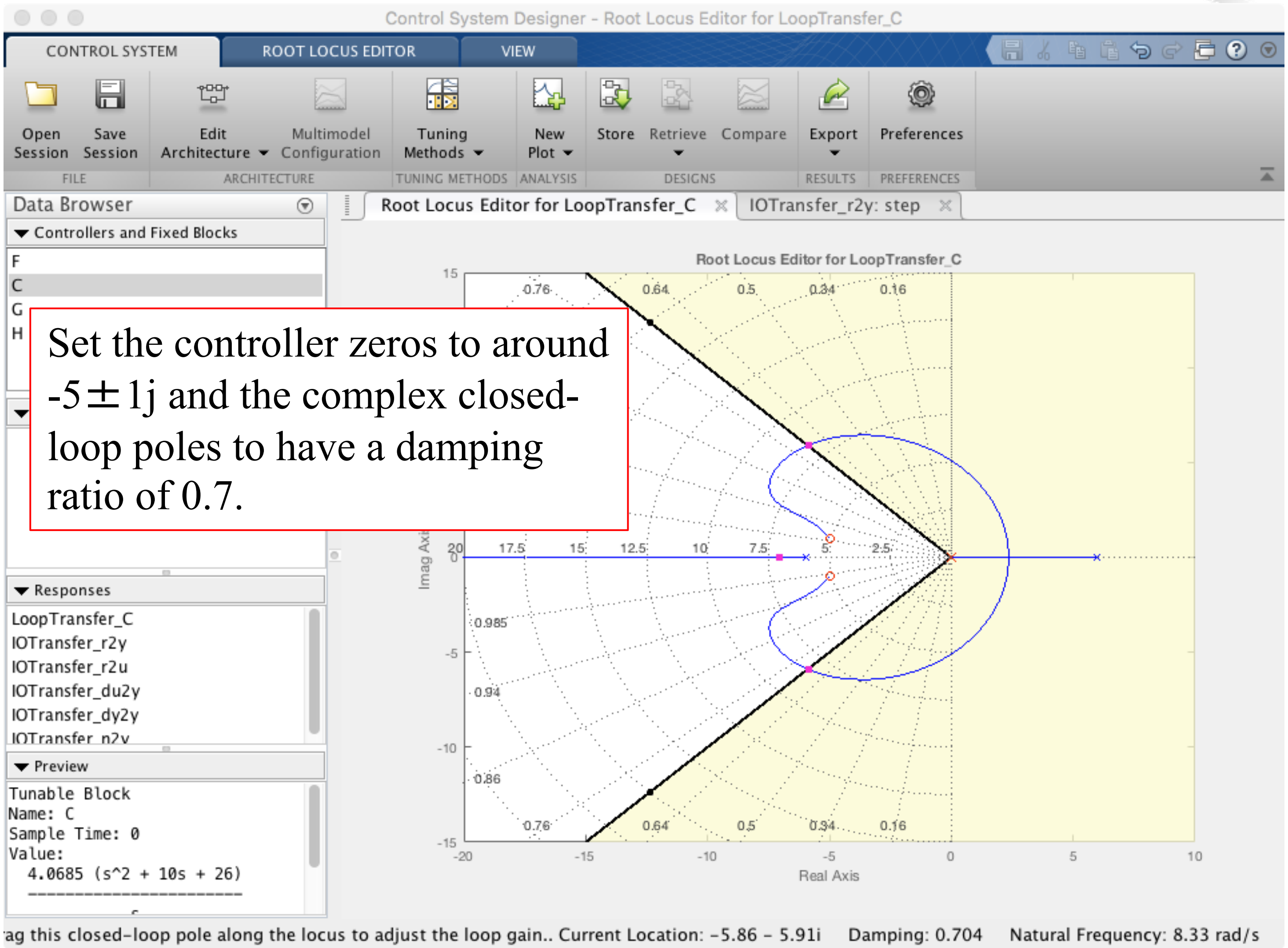
Set the controller zero to around -10 and the closed-loop poles to have a damping ratio of 0.7.

Stabilization Control (PID Controller)

Add integral action to eliminate steady-state error.



PID Controller Design



Set the controller zeros to around -5±1j and the complex closed-loop poles to have a damping ratio of 0.7.

Arduino Code Template

segway_robot_template | Arduino 1.8.13

segway_robot_template | EnableInterrupt.h | I2Cdev.cpp | I2Cdev.h | MPU6050.cpp | MPU6050.h | MPU6050_6Axis_MotionApps20.h | MPU6050_9Axis_M

```
// ===                    OPTIONS                    ===

// CONTROL SCHEME
#define PID
//#define FULL_STATE

// SERIAL OUTPUT CONTROL
#define arduinoSerialPrint
// #define MATLABSerialPrint

// SETPOINT SELECTION
#define setPointFn setpoint_constant
// #define setPointFn setpoint_square_wave
// #define setPointFn setpoint_sin_wave
// #define setPointFn setpoint_circle
// #define setPointFn setpoint_yaw_turn
// #define setPointFn setpoint_figure_eight
// #define setPointFn setpoint_sonar_turn
// #define setPointFn setpoint_MY_TRACK

// Variables for setpoints:
// change tilt_offset to make the robot move forward or backward
// change lr_offset to make the robot turn right or turn left
float tilt_offset = 1.5;   // tilt angle in degrees, nominal = 0 deg
float lr_offset = 0.8;   // adjust left/right motor pwm ratio, nominal = 1

// ENABLE OR DISABLE THE TWO KNOBS FOR REAL-TIME ADJUSTMENTS
#define TRIM_POT
#define BALANCE_POT
// Adjusment values if trim and/or balance knob is disabled
float trim_val = 0.0;   // adjust this value to center the robot (in degrees) (DEFAULT = 0)
float bal_val = 1.0;    // adjust this value to make the robot move straight (DEFAULT = 1)

// ===              CONTROL PARAMETERS              ===
#ifdef PID
// Provide PID controller gains here
float Kp = 0.0;      // P control gain
float Kd = 0.0;      // D control gain
float Ki = 0.0;      // I control gain
#endif

#ifdef FULL_STATE
// Provide FULL STATE controller gains here
float K1 = 0.0;      // theta
float K2 = 0.0;      // theta dot
float K3 = 0.0;      // x
float K4 = 0.0;      // x dot
#endif
```

Lab 8: PID

Lab 9: Full-state feedback

Setpoint selection

PID gains

Full-state feedback gains

Done Saving.

```
In file included from /Users/hchin/Dropbox (MIT)/2.004 Fall 2021/Labs/Lab 8/Files to be posted/segway_robot_template/segway_robot_template.ino:16:0:
sketch/EnableInterrupt.h:22:121: note: #pragma message: NOTICE: *** EnableInterrupt library version 0.9.7. This is not a problem. Keep calm, and carry on
 #pragma message("NOTICE: *** EnableInterrupt library version 0.9.7. This is not a problem. Keep calm, and carry on. ***")
```

56 — Arduino Uno on /dev/cu.usbmodem144101

Fine Tuning of Controller Gains

- **Fine-tuning of controller gains may be necessary due to model uncertainty.**
- **In general:**
  - Increase K_p to improve speed of response (decrease rise time).
  - Increase K_d to reduce overshoot and settling time.
  - Increase K_i to eliminate steady-state error.
  - High gains may lead to saturation and instability.
  - Avoid making large adjustments once the system is close to having desirable response. Try making ±10~20% incremental adjustment one gain at a time.

Stability Competition

- **Control your Segway robot so it can stand on a 1’ x 1’ wooden square.**
- **The robot with the longest standing time without rolling out of the square is the winner. Both wheels must be on the square.**
- **Extra credits:**
  - 1st place: 2 points + prize
  - 2nd place: 1 point