Adaptive Platform Stabilization Video

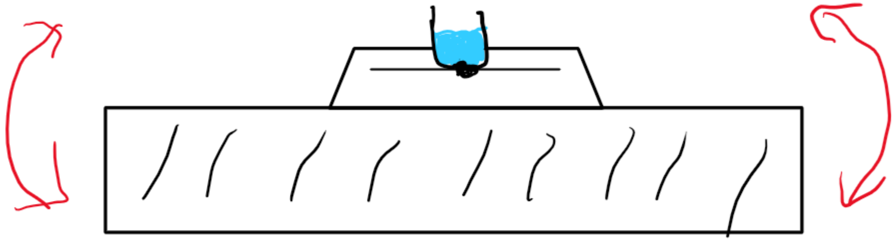
Ajay Mathur, Rohan Punamiya, Remy Bondurant, Colin Murray

ME 4012 Modeling and Control of Motion Systems

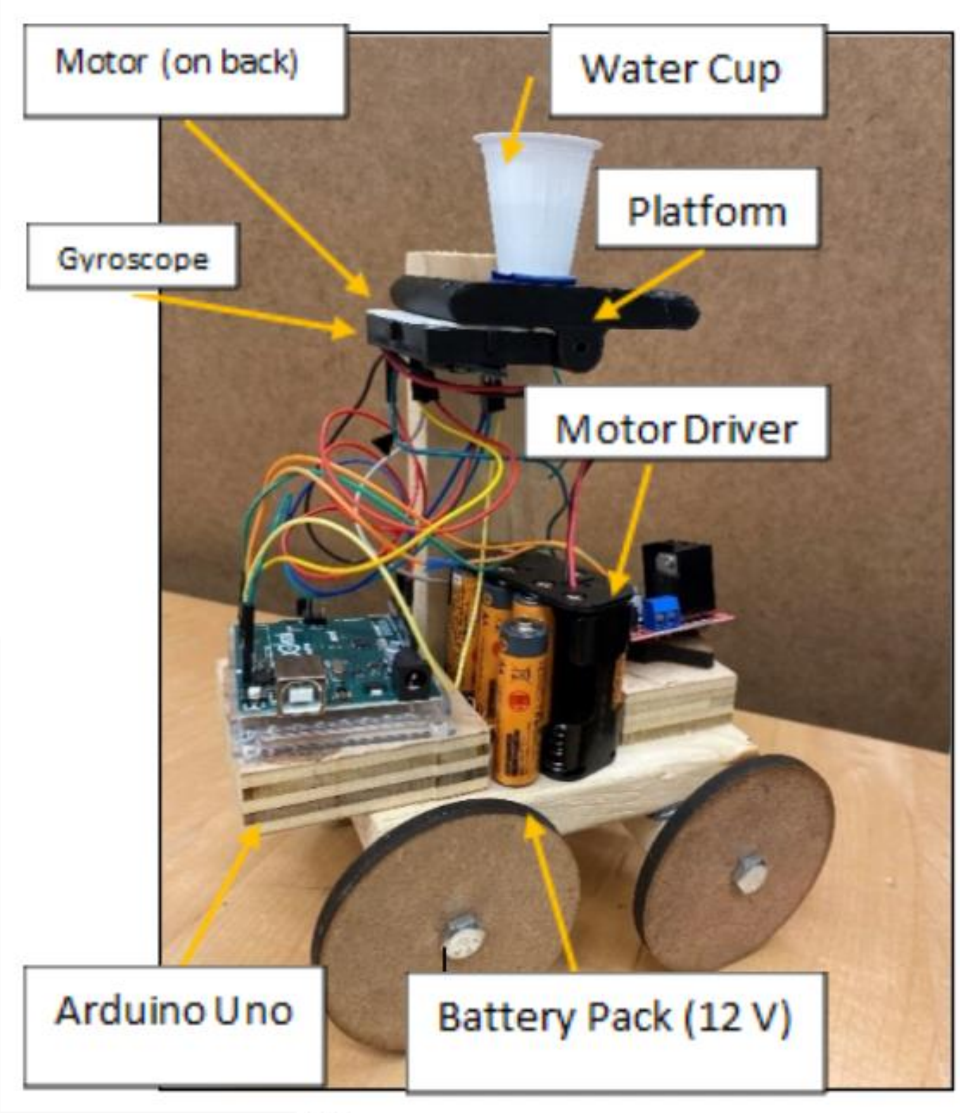
Fall 2023 Final Project Video

Motivation

- Self-balancing platform that can work on uneven terrain
- Uses in construction to improve material transport safety
- Current design can be attached to a motor to allow for movement
- Needs PID control on the gyroscope input to use the motor output to balance

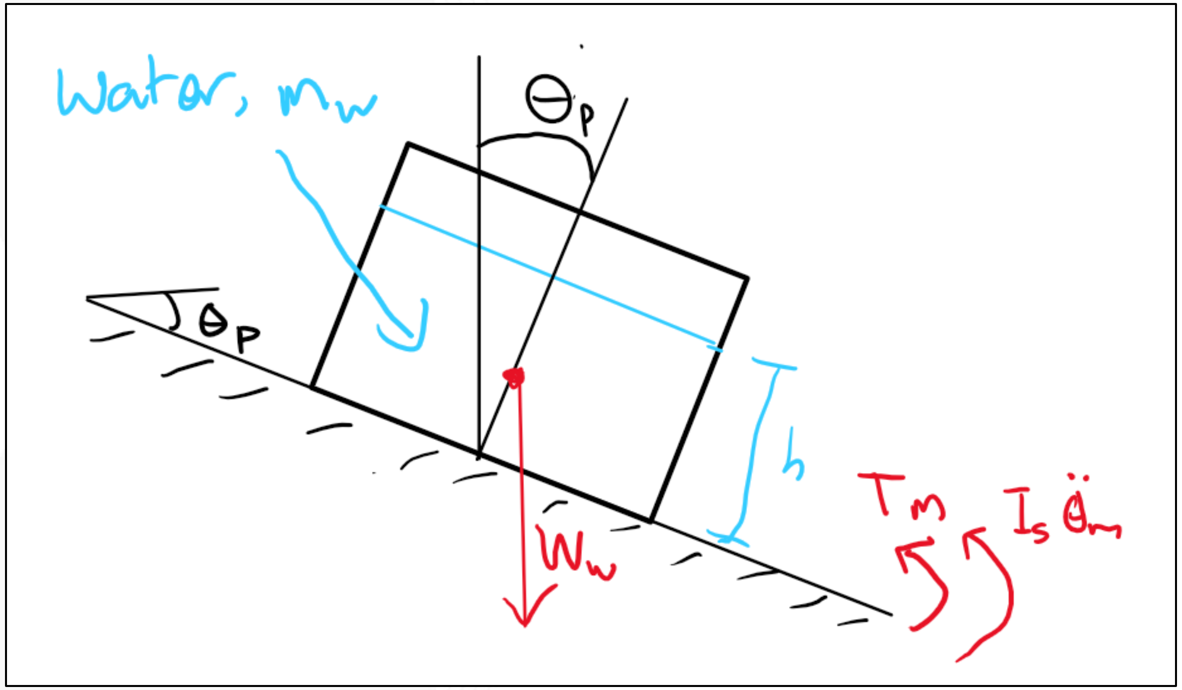


Design (Physical and Simulation Assumptions)



- Uses small geared motor as weights are not that high (could be upgraded if needed)
- Low-cost gyroscope used is noisy (could be upgraded to provide less jittery output)
- Gear ratio could be used to give more precise control of platform rotation speed

Theoretical Model

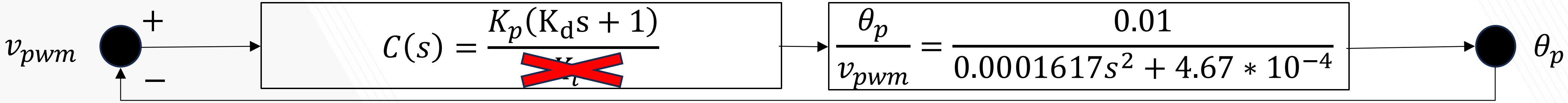


$$\boxed{\frac{\theta_p}{T_m} = \frac{1}{Is^2 + \frac{m_w g h}{2}}}$$

$$\boxed{\frac{\theta_p}{v_{pwm}} = \frac{\overset{\text{Stall Torque}}{0.001}(\overset{\text{Gear Reduction Ratio}}{10})}{Is^2 + \frac{m_w g h}{2}}}$$

Parameter	**Value**
m_w	0.01 kg
I	0.0001617 kg*m^2
g	9.81 m/s^2
h	9.525 mm

Controller Design



$$C(s) = \frac{K_p(K_d s + 1)}{\cancel{K_i}}$$

$$\frac{\theta_p}{v_{pwm}} = \frac{0.01}{0.0001617s^2 + 4.67 * 10^{-4}}$$

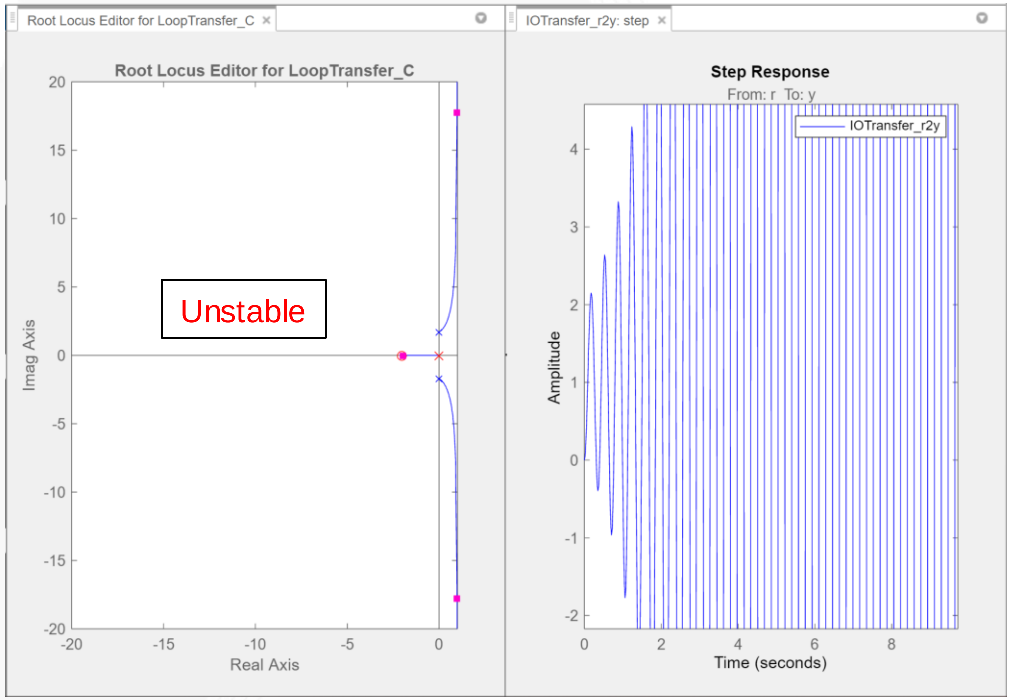
PID Controller

$OL\ Poles: s = \pm 1.7i$

Open Loop Plant

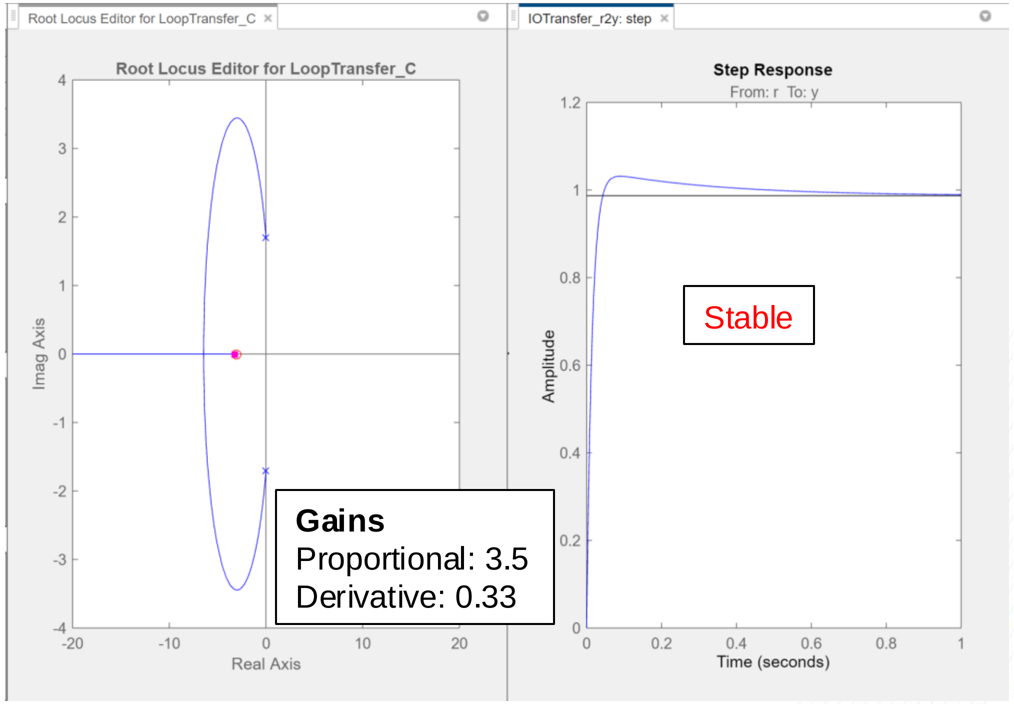
Marginally stable

PID Controller



Unstable

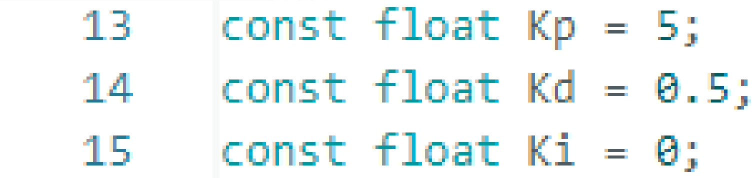
PD Controller



Stable

Gains
Proportional: 3.5
Derivative: 0.33

Experimental Results



```
const float Kp = 5;
const float Kd = 0.5;
const float Ki = 0;
```

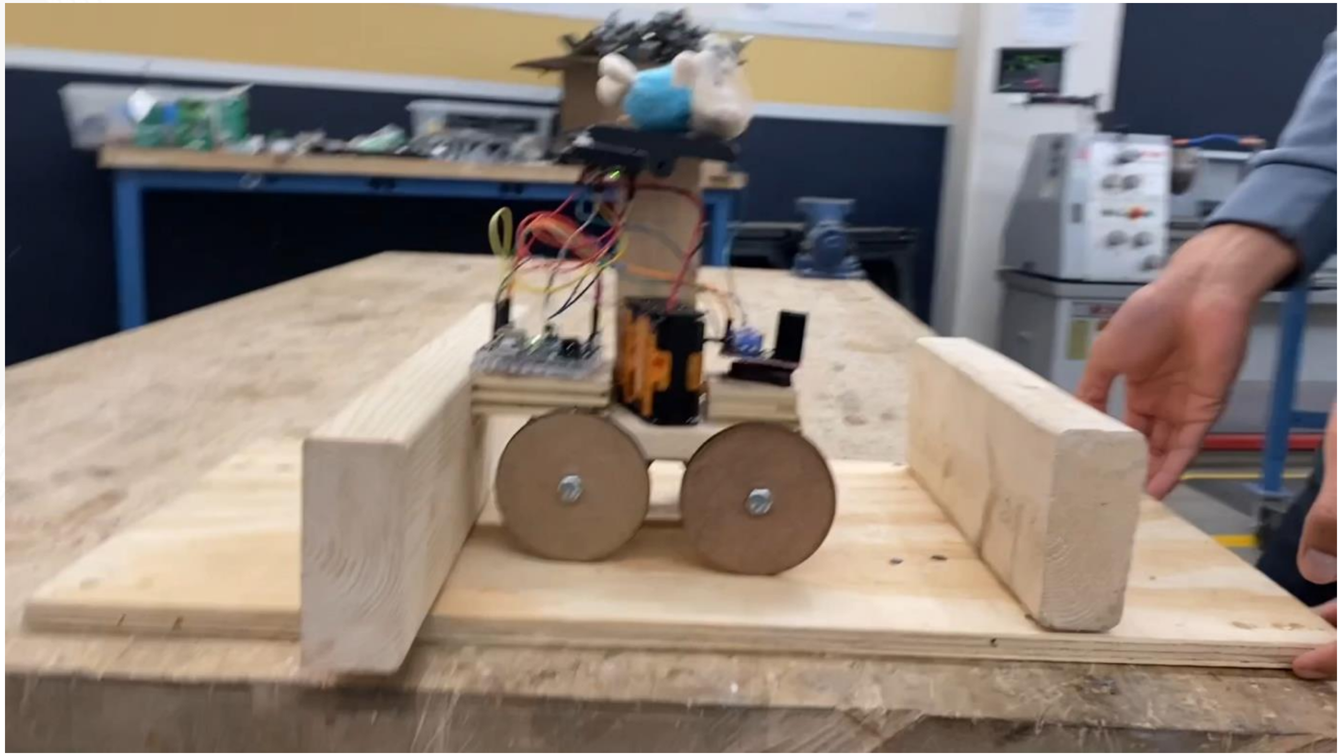
Small difference from our theoretical controller; however, still accomplishes the same goal.

```
error = y;
integral += error * timeElapsed;
derivative = (error - previousError) / timeElapsed;
motorPWM = (Kp  * error) + (Ki * integral) + (Kd * derivative);
previousError = error;

// Motor speed bound
if (motorPWM > 255) {
  motorPWM = 255;
}

//Angle limit
if (y >= 60 || y <= -60) {
  motorPWM = 0;
}

// Motor direction
if (motorPWM < 0) {
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
  delay(20);
  // Serial.println("Backward");
} else {
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  delay(20);
  // Serial.println("Forward");
}
analogWrite(enA, abs(motorPWM));
```

- Implemented a PD controller to translate gyroscope error in the y axis to motor power
- Assumes the program starts at zero relative to ground
- Speed and angle limit fail safes to prevent damage to wiring
- High derivative gain led to rapid fluctuations, so it was kept low

Video

- Two Tests: Stuffed bear and water cup
- System works but experiences overshoot upon changes in direction
- Controller gains match theory with mild error



Conclusion

- PD control is effective at balancing a platform on uneven terrain
- System can sense and react to changes with a resolution of 2°
- Drawbacks/potential improvments:
  - Response is more discrete rather than continuous as desired
  - System experiences delay and overshoot when changing direction
- Potential iterations:
  - Impulse response

Ajay Mathur

Remy Bondurant

Rohan Punamiya

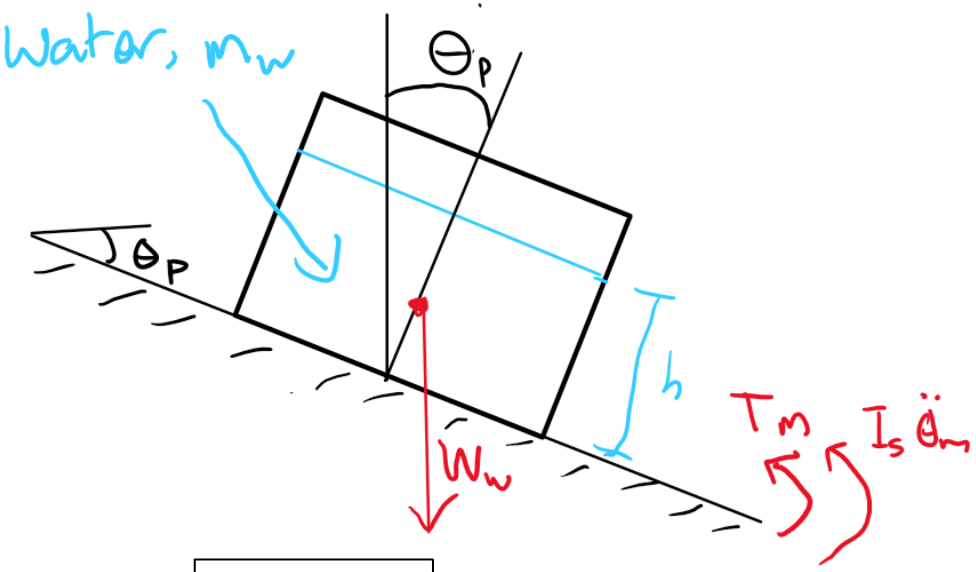
Colin Murray

References

List 3 or more references:

- [1] “Amazon.com: Greartisan DC 6V 150RPM N20 High Torque Speed Reduction Motor with Metal Gearbox Motor for DIY RC Toys : Toys & Games,” *www.amazon.com*. https://www.amazon.com/gp/product/B07FVM8YZ7/?th=1 (accessed Nov. 29, 2023).
- [2] HowToMechatronics, “Arduino and MPU6050 Accelerometer and Gyroscope Tutorial - HowToMechatronics,” *HowToMechatronics*, Apr. 09, 2019. https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/
- [3]Dejan, “Arduino DC Motor Control Tutorial - L298N | PWM | H-Bridge - HowToMechatronics,” *HowToMechatronics*, Feb. 08, 2019. https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/

Appendix A: Model Derivation



$$\Sigma M_m = I_s \ddot{\theta}_m$$

$$T_m - W_w = I_s \ddot{\theta}_m$$

$$T_m - m_w g \frac{h}{2} \sin\theta_p = I_s \ddot{\theta}_m$$

$$\theta_m = \theta_p$$

$$T_m - m_w g \frac{h}{2} \sin\theta_p = I_s \ddot{\theta}_p$$

$$T_m = I_s \ddot{\theta}_p + m_w g \frac{h}{2} \sin\theta_p$$

$\sin(\theta_p) \approx \theta_p$
(true for small angles)

$$T_m = \left[I_s + m_w g \frac{h}{2}\right] \theta_p$$

$$\frac{\theta_p}{T_m} = \frac{1}{I s^2 + m_w g \frac{h}{2}} \xrightarrow{\qquad} \boxed{\frac{\theta_p}{V_{pwm}} = \frac{0.01}{I_s s^2 + m_w g \frac{h}{2}}}$$

multiply by gear ratio of 10 *and stall torque of* 0.001 *to convert to* v_{pwm}

Appendix B: Parameter Estimation

Parameter	Value
m_w	0.01 kg
I	0.0001617 kg*m^2
g	9.81 m/s^2
h	9.525 mm

m_w Mass

Estimated by taking volume of water (~.25 fl oz) and converting to grams. We rounded the 7.087 g to 0.01 kg, knowing we would sometimes fill the cup above half way and certain other parameters could effect the center of mass of the system (gyroscope weight and uneven 3D printed platform density.

I Inertia

The main contributor to the rotational inertia of the system was the platform itself. The mass moment of inertia was found by multiplying the density of PLA (1250 kg/m^3 by the volume of the platform (18750 mm^3) then approximating our platform to a rectangular prism and finding that mass moment of inertia with the known mass and dimensions.

g Gravity

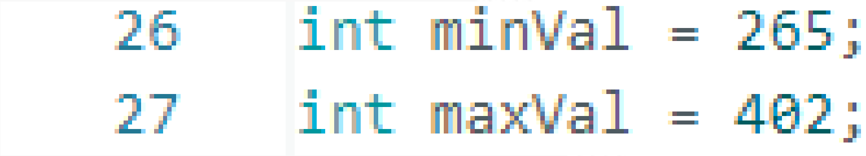
Acceleration due to gravity on earth.

h Height

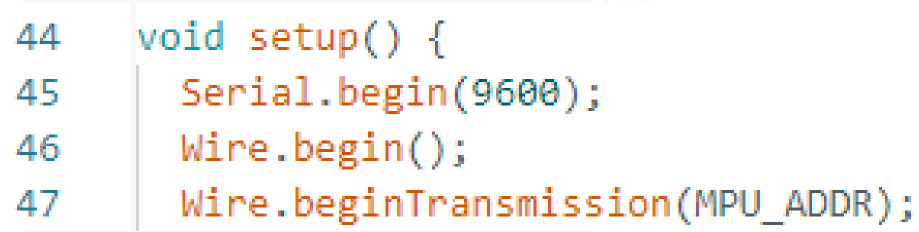
We estimated the center of mass of the system to be about 3/8” above the platform because we’d often fill the cup ¾ of the way up.

Appendix C: Reading the Gyroscope Output

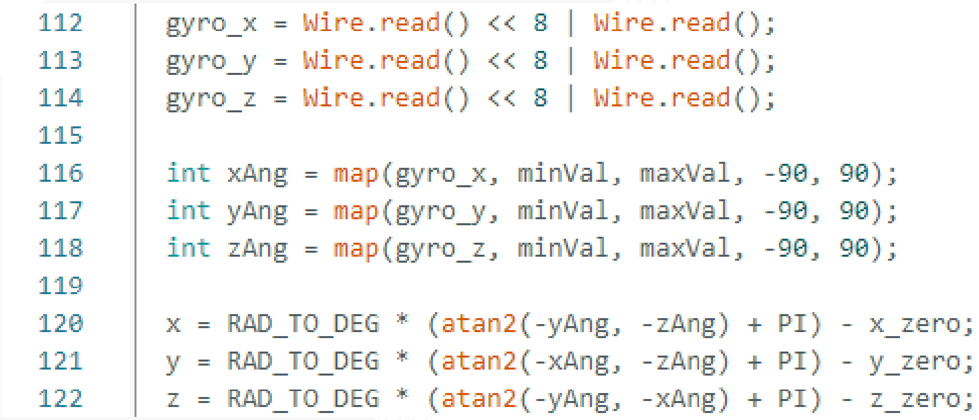
```
const int MPU_ADDR = 0x68;
```



```
int minVal = 265;
int maxVal = 402;
```



```
void setup() {
  Serial.begin(9600);
  Wire.begin();
  Wire.beginTransmission(MPU_ADDR);
```



```
gyro_x = Wire.read() << 8 | Wire.read();
gyro_y = Wire.read() << 8 | Wire.read();
gyro_z = Wire.read() << 8 | Wire.read();

int xAng = map(gyro_x, minVal, maxVal, -90, 90);
int yAng = map(gyro_y, minVal, maxVal, -90, 90);
int zAng = map(gyro_z, minVal, maxVal, -90, 90);

x = RAD_TO_DEG * (atan2(-yAng, -zAng) + PI) - x_zero;
y = RAD_TO_DEG * (atan2(-xAng, -zAng) + PI) - y_zero;
z = RAD_TO_DEG * (atan2(-yAng, -xAng) + PI) - z_zero;
```

1. **Zeroing output**
   Program reads first gyroscope input using the below method on start up from the dedicated MPU address and uses it as zero point.

2. **Measuring Raw Input**
   Gyroscope input is read and converted from the minimum and maximum values of 265 and 402 to ±90 to be used in the next step.

3. **Converting to degrees**
   The built-in atan2 function converts the previous values into an angle in radians from $-\pi$ to π for each axis, which then are converted to 0 to 2π radians, and finally converted to degrees where the zero point is subtracted.