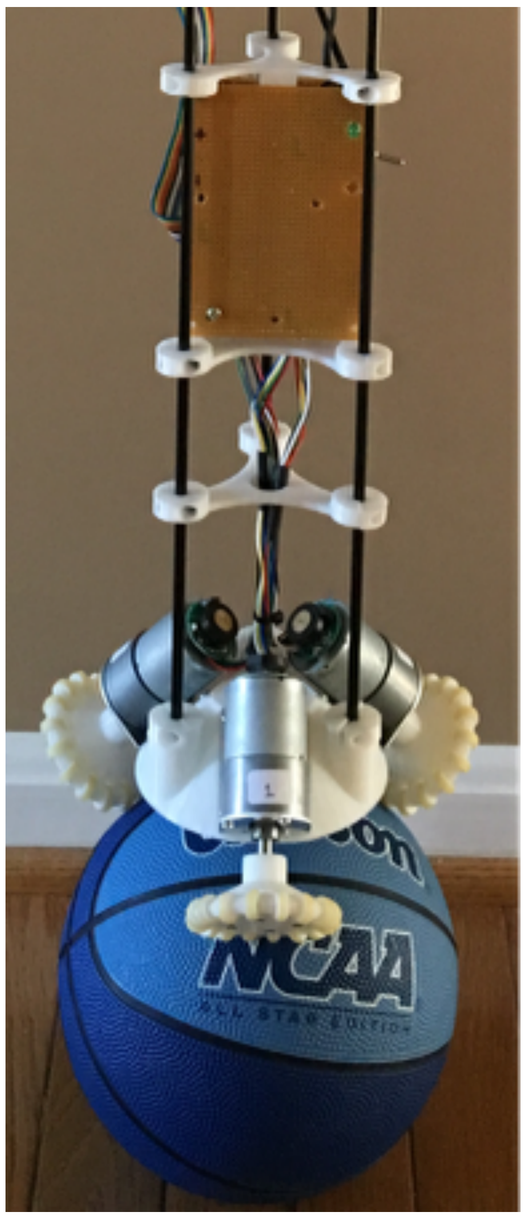
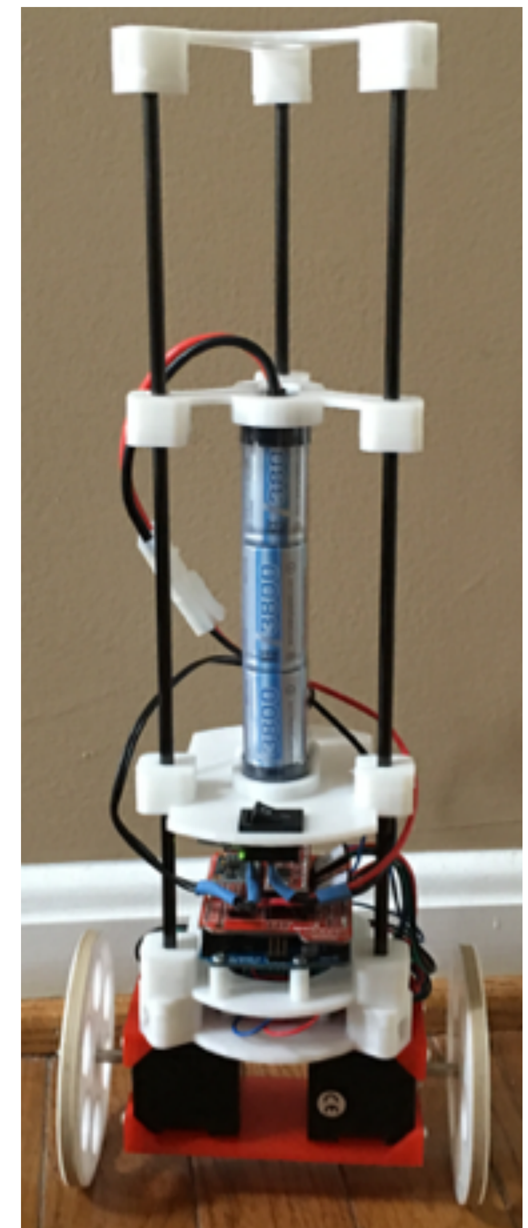
Balancing Robots





Steve Geyer

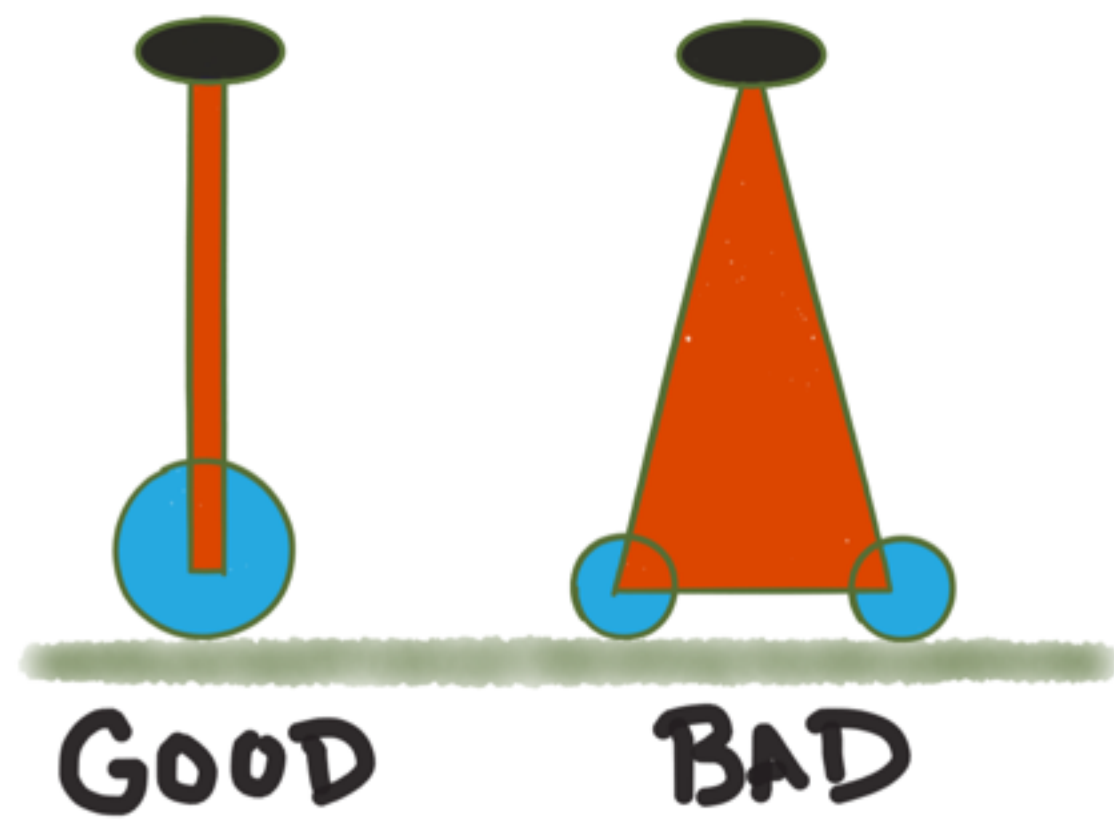
Outline

- Why build balancing robots
- Construction techniques
- Balancing algorithms (complex but important)
- Lessons learned from building several balancing robots
- Summary

- *My goal is to give you a basic background. Some of what I say today is probably wrong :(*

Why build balancing robot

- Technical challenge and cool factor
- Humans spaces assume a five foot person with a fixed size footprint top to bottom
  - ✦ Statically balanced robots generally are wider at the base
  - ✦ Balancing robots have no problem with this size
- They handle bumps from collisions, people and pets
- They move organically



Construction

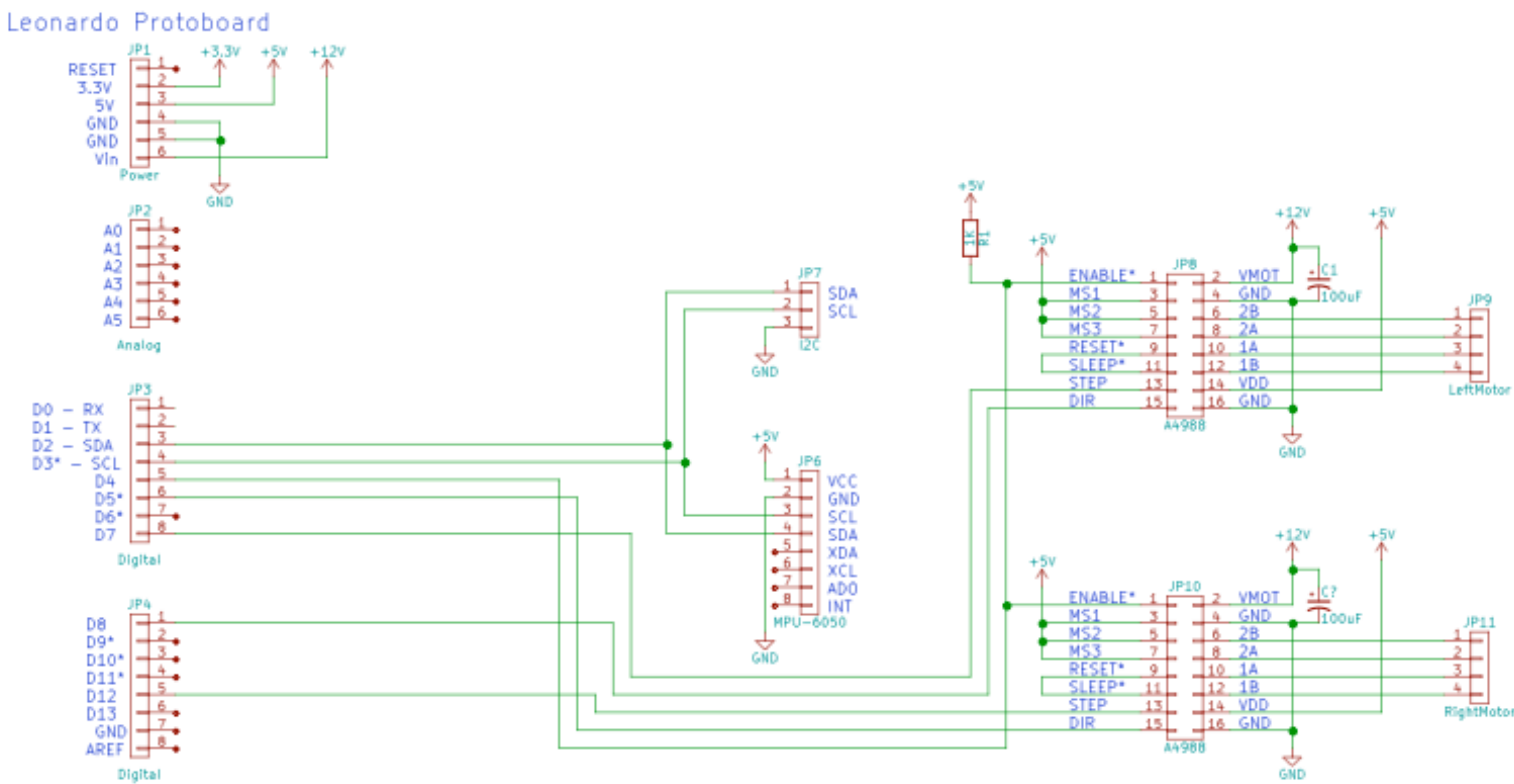
- Components are the same as other robots — batteries, sensors, computer boards, motors, etc. Sensor and motor selections have to be a little more precise.

- Construction materials and techniques the same as other robots. I have seen them built in metal and in wood. I chose 3-D printing and carbon fiber rods.

- Keep mass balanced and along vertical centerline. Moving masses, like arms, can be an issue.

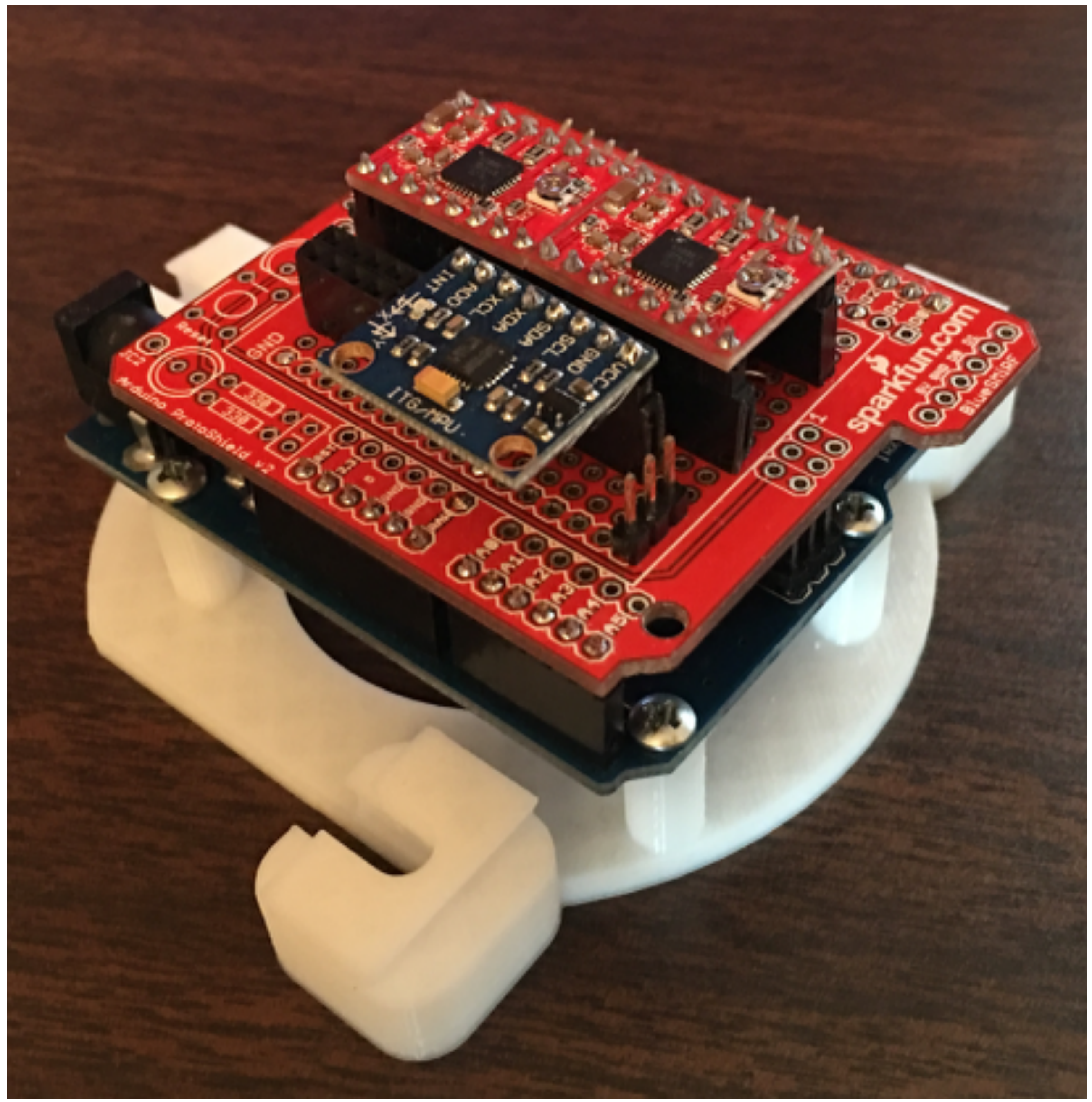
- Need real-time processing of data. Arduino boards work great. Real-time processing is more difficult with Raspberry Pis, BeagleBone Blacks or any other of the hundreds of Linux boxes.

Protoboard



Leonardo Protoboard
JP1
RESET
3.3V
5V
GND
GND
Vin
Power
+3.3V
+5V
+12V
GND
JP2
A0
A1
A2
A3
A4
A5
Analog
JP3
D0 - RX
D1 - TX
D2 - SDA
D3* - SCL
D4
D5*
D6*
D7
Digital
JP4
D8
D9*
D10*
D11*
D12
D13
GND
AREF
Digital
JP7
SDA
SCL
I2C
JP6
VCC
GND
SCL
SDA
XDA
XCL
ADO
INT
MPU-6050
R1
1K
JP8
ENABLE*
MS1
MS2
MS3
RESET*
SLEEP*
STEP
DIR
VMOT
GND
2B
2A
1A
1B
VDD
GND
A4988
C1
100uF
JP9
LeftMotor
JP10
ENABLE*
MS1
MS2
MS3
RESET*
SLEEP*
STEP
DIR
VMOT
GND
2B
2A
1A
1B
VDD
GND
A4988
C?
100uF
JP11
RightMotor

Protoboard (2)

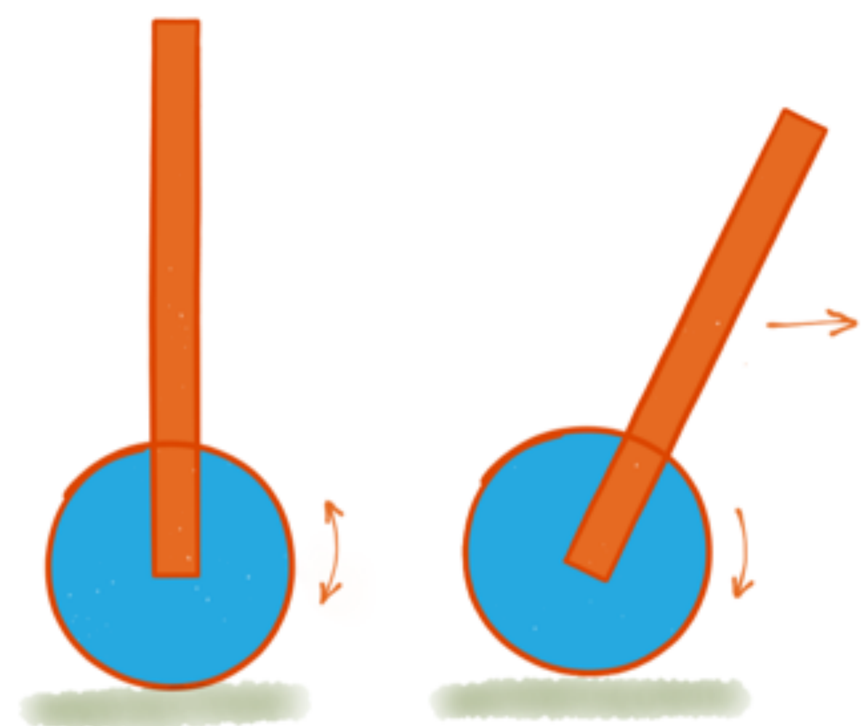


Balancing 101

- A good visualization of our task is to imagine balancing a broomstick on your finger — you move the bottom of the stick to maintain balance

- Balancing robots are either
  - ✦ Standing mostly still and balancing,
  - ✦ Moving in a controlled fall, or
  - ✦ Moving in an uncontrolled fall

- The goal is to stay in either the ***balancing*** or ***controlled fall*** states and smoothly transition between them

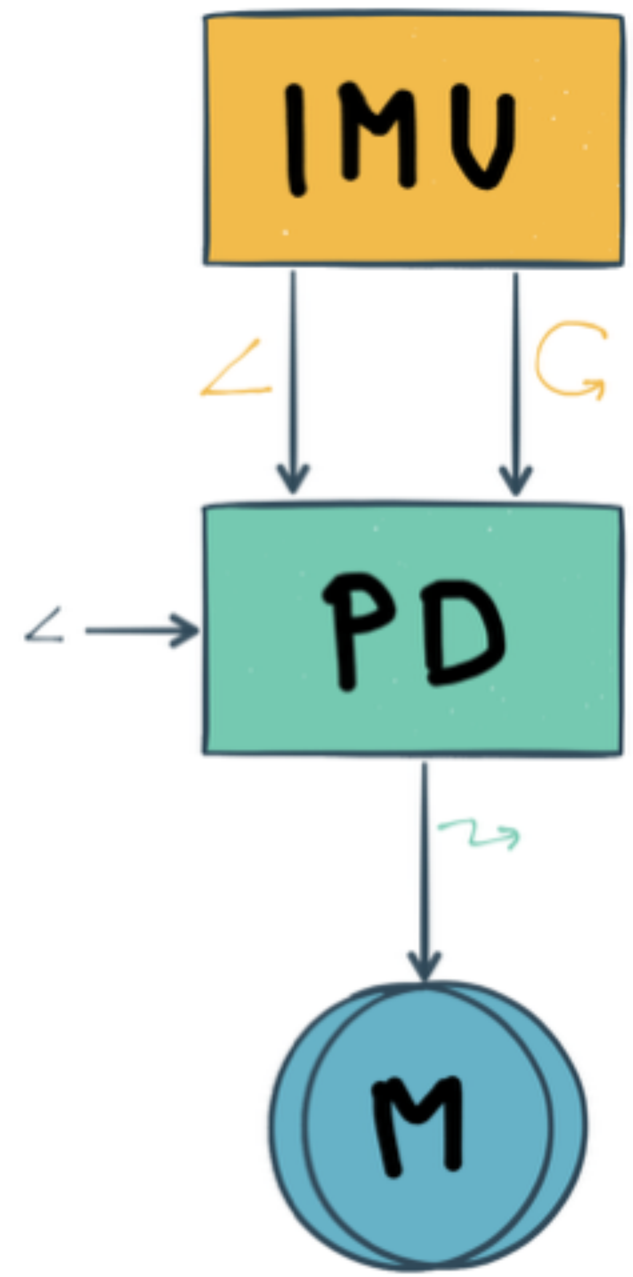


This Sounds Easy in Theory but it is Hard in Practice

- For many builders this is the first time the ***sense - analyze - act*** loop has signification consequences.
- And that loop has to act 50-200 times a second.
- Sensor noise cannot be ignored and therefore sensor fusion a serious concern.
- Motors have to react fast and precisely.

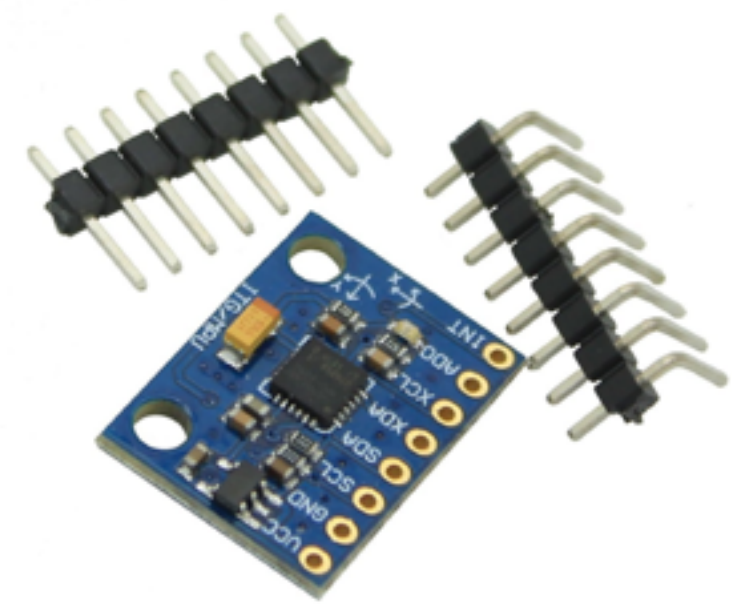
Simple Balancing Solution

- Inertial Measurement Unit measures angle (and optionally rotation)
- PD controller compares current angle against desired angle and generates correct motor speed.
- The problem is knowing what the desired angle should be.



Inertial Measurement Unit

- Measures acceleration and rotation of device. There is one accelerometer and one gyro for X, Y, and Z.
- A basic package has 6 degrees of freedom (6 DOF).
- 9 DOF packages include magnetometers to determine position relative to the earth's magnetic field.
- Accelerometers directly detect the 1G pull of the earth



Sad Caveats

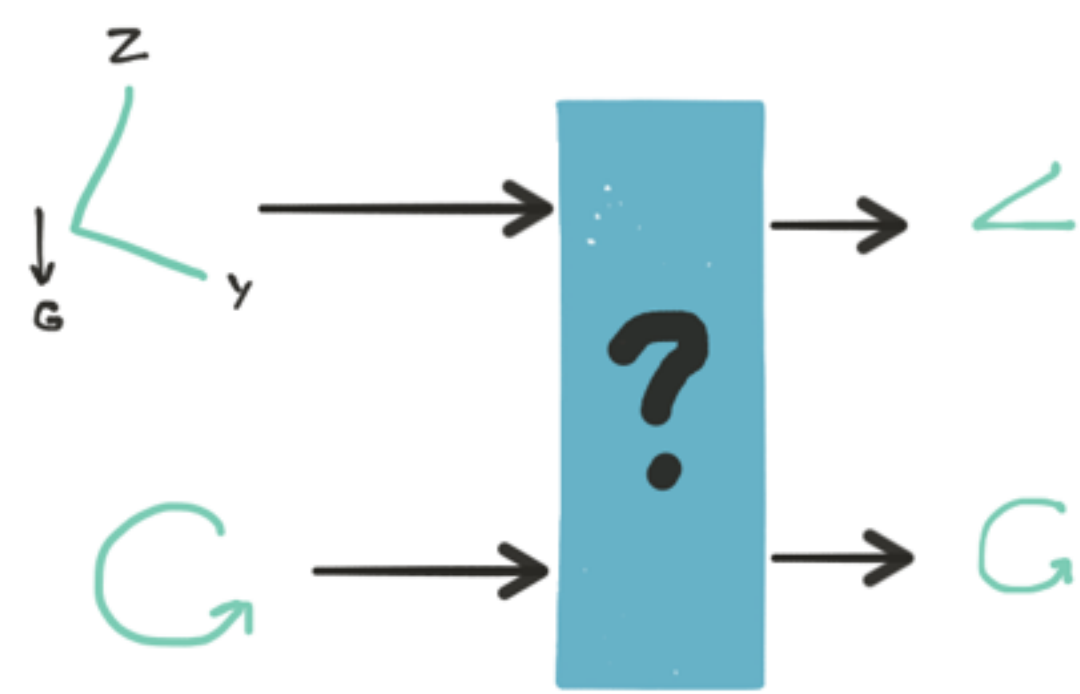
- IMU sensors are noisy and only average to a consistent value (in acceleration or rotation)
- These sensors do not give same magnitude of value for each axis for equivalent input.
- Even sadder is that they don’t give the same value for one axis when inverted.
- You should characterize your specific device and compensate
  - ✦ Statically measure error in separate pre-build step
  - ✦ Dynamically determine error as device giving answer (MPU6050).

IMU Sensor Fusion

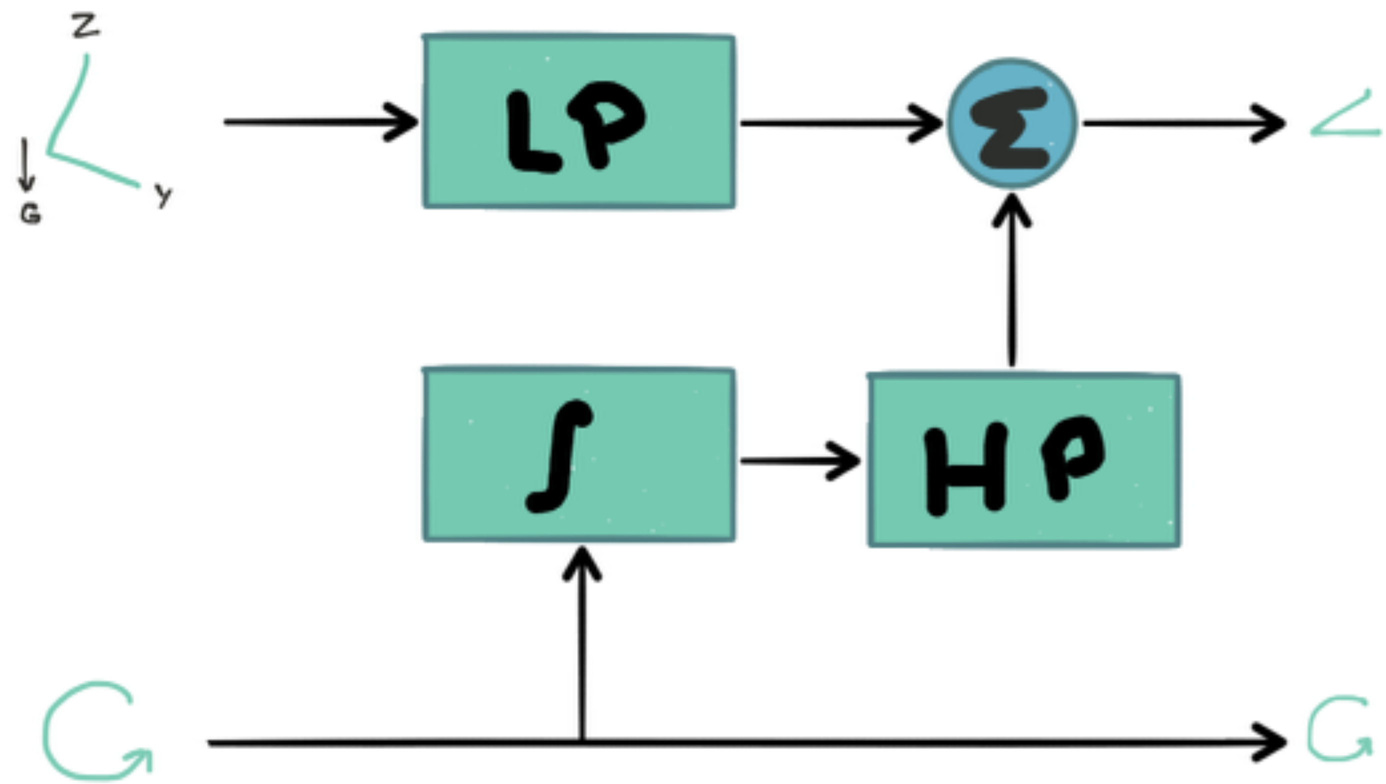
- It merges and processes data from accelerometers and gyros to generate more accurate angular and rotational information

- Useful Hacks
  - ✦ Use Y instead of Z
  - ✦ $\sin\theta \approx \theta$ for small angles

- Some sensor fusion options:
  - ✦ Hundreds of Data Cleanups
  - ✦ Kalman Filter
  - ✦ Low Pass Filter
  - ✦ Complementary Filter



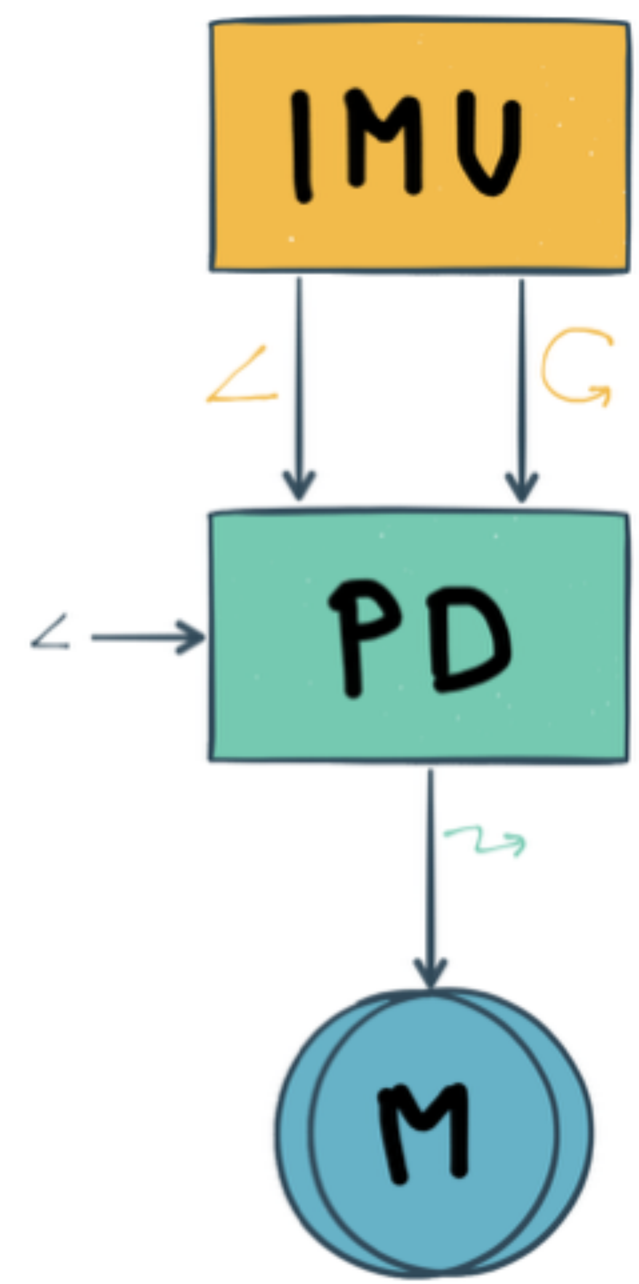
Complementary Filter (CF)



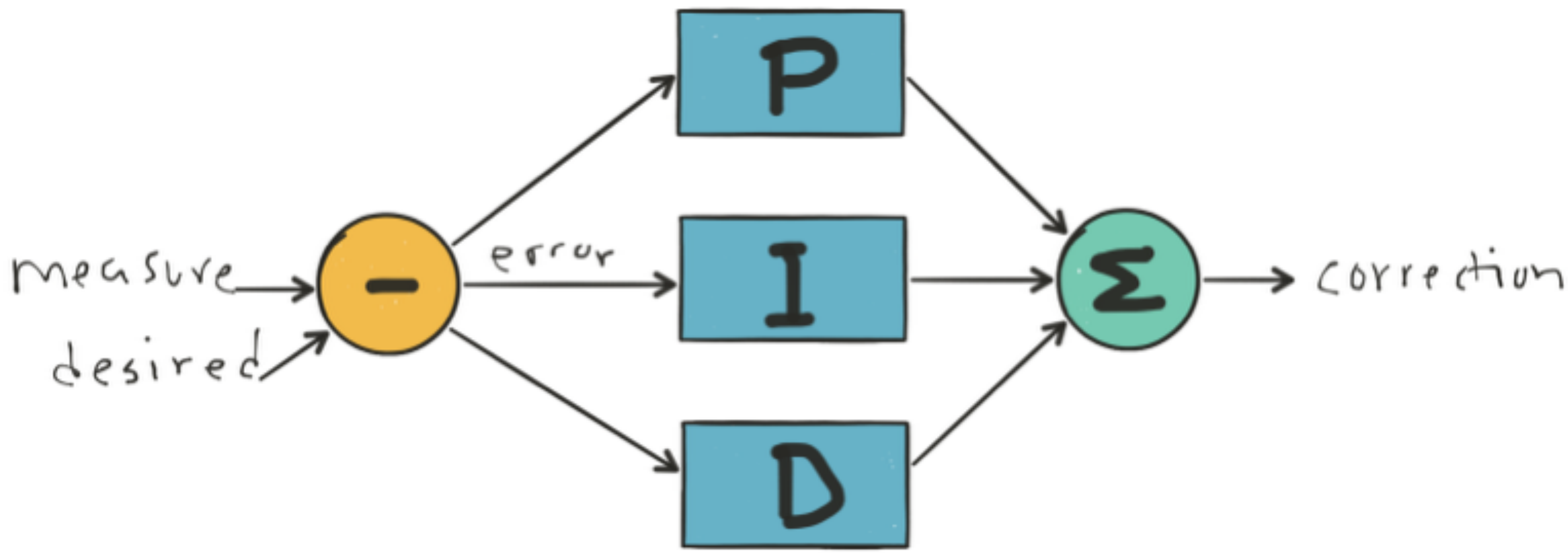

```
void getCFAngleAndRotation(float *cfAngle, float *rotation) {
    float a = mpu.getAccelerationY() * a2g;
    float r = mpu.getRotationX() * r2rs;
    cfAngleRadians = Kf*(cfAngleRadians + r/updatesPerSec) + (1.0-Kf)*a;
    *cfAngle = -cfAngleRadians*rad2deg;
    *rotation = -r*rad2deg;
}
```

Balancing Solution Review

- IMU measures angle and rotation. We will use a CF to do sensor fusion.



PID (Proportional - Integral - Derivative) controllers



- Great at calculating correction from measured and desired values.100 years old. Books written about PIDs.
- Factors in proportional, integral and derivative elements.

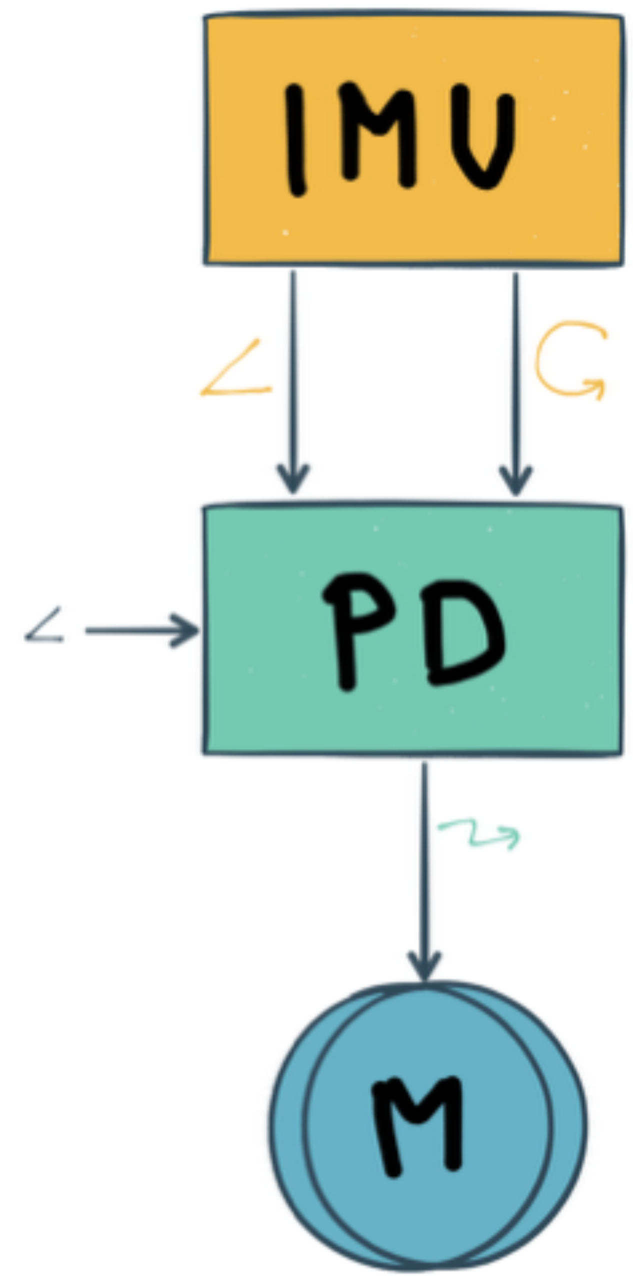
General PID Solution

error = ***setPoint*** - ***input***;
pTerm = kp * error;
iTerm = clip(iTerm + (ki * error), maxValue);
dTerm = kd * (lastInput - input);
lastInput = input;
output = clip(pTerm + iTerm + dTerm, maxValue);

This more capability than I need for balance

Balancing Solution Review

- IMU measures angle and rotation. We will use a CF to do sensor fusion.
- We will use PD controller to compare CF results against desired angle and generates correct motor speed.
- We will use the CF rotation information for D input value in the PD controller.



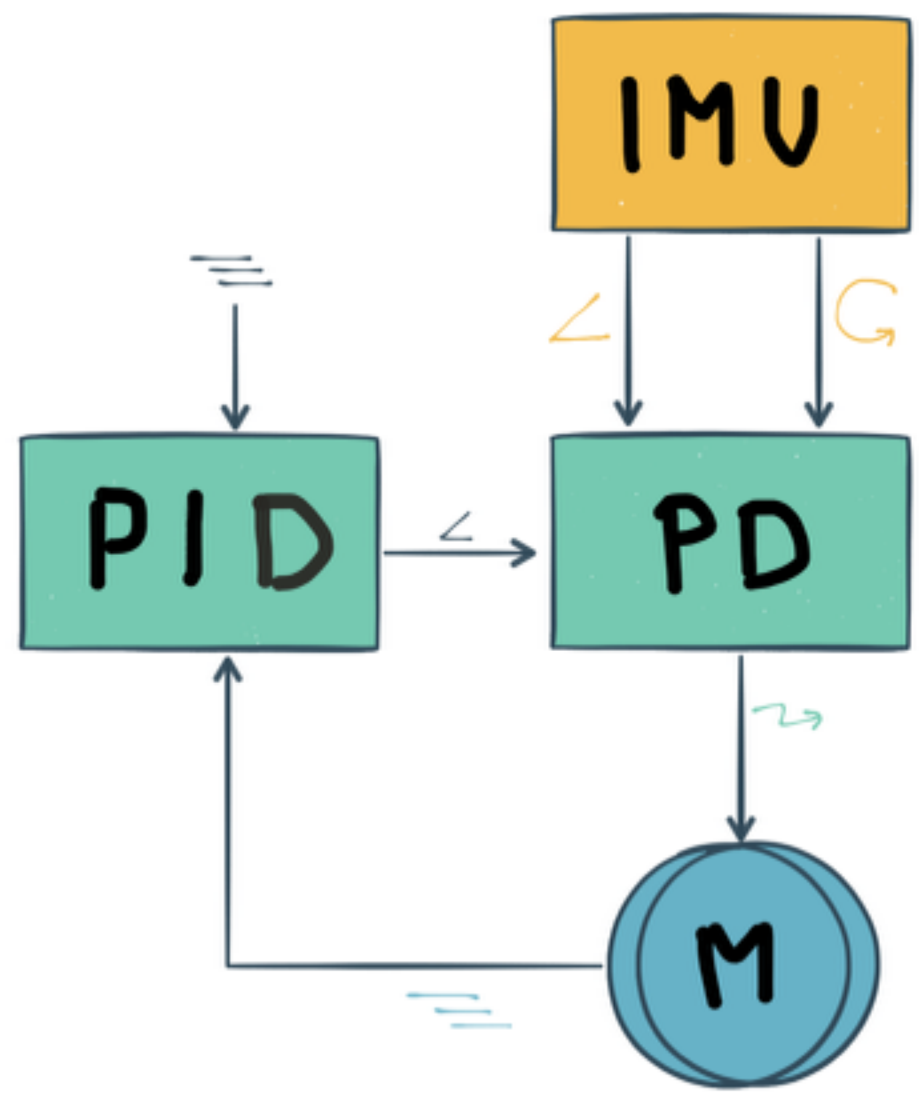
Simple Balancing Code

```
void loop() {
    if (nextSample()) {
        float cfAngle, rotation;
        getCFAngleAndRotation(&cfAngle, &rotation);
        float errorAngle = balanceOffset - cfAngle;
        motorSpeed += clip(Kbp*errorAngle - Kbd*rotation, maxMotorSpeed);
        if ((cfAngle > -stopMotorAngle) && (cfAngle < stopMotorAngle)) {
            setMotorSpeedM1(motorSpeed + direction);
            setMotorSpeedM2(motorSpeed - direction);
            digitalWrite(4, LOW);   // Enable motor
        } else {
            digitalWrite(4, HIGH);  // Disable motor
            setMotorSpeedM1(0);
            setMotorSpeedM2(0);
            motorSpeed = 0;
        }
    }
}
```

More complete solution

- Add PID controller to compare actual motor speed to desired speed and then calculate new angle (*errorAngle* in balancing code).

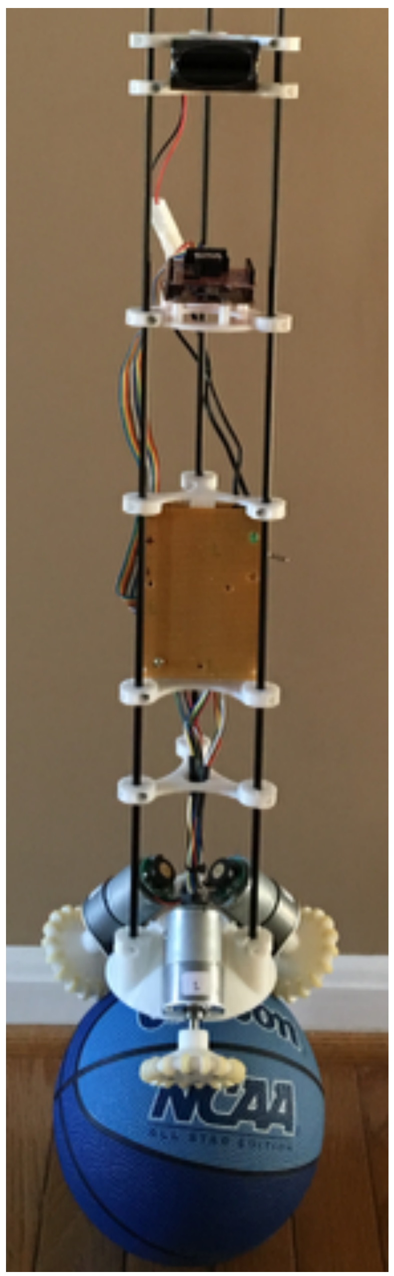
- Consider PID for directional control if using DC motors

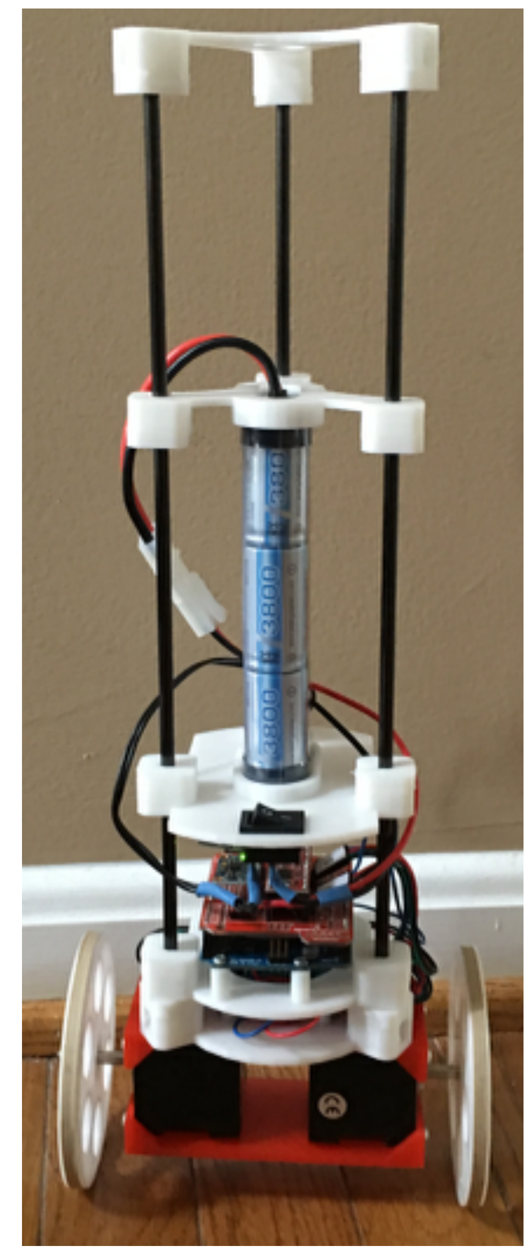


Full Implementation

```
void loop() {
  if (nextSample()) {
    float cfAngle, rotation;
    getCFAngleAndRotation(&cfAngle, &rotation);
    double speed = (stepsToRotationsSec(m1Speed)+
                    stepsToRotationsSec(m2Speed))/2;
    float errorAngle = anglePID(desiredSpeed, speed) - cfAngle;
    motorSpeed += clip(Kbp*errorAngle - Kbd*rotation, maxMotorSpeed);
    if ((cfAngle > -stopMotorAngle) && (cfAngle < stopMotorAngle)) {
      setMotorSpeedM1(motorSpeed + direction);
      setMotorSpeedM2(motorSpeed - direction);
      digitalWrite(4, LOW);
    } else {
      digitalWrite(4, HIGH);
      setMotorSpeedM1(0);
      setMotorSpeedM2(0);
      motorSpeed = 0;
      angleITerm = 0.0;
    }
  }
}
```

I Built Five Iterations Of Three Basic Designs

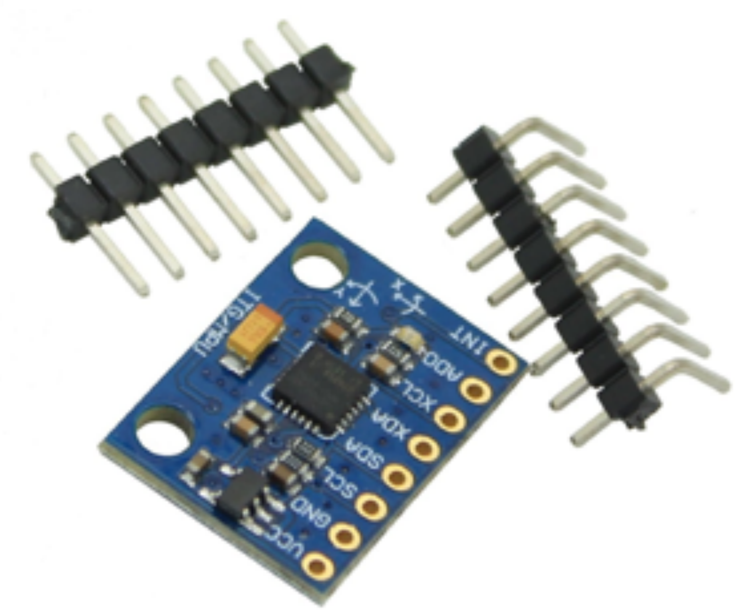




Lessons Learned (IMU)

- Sparkfun sen10121 which uses a an ITG3200 (gyro) and ADXL345 (acc) and costs $40. It works great in a Complementary Filter.

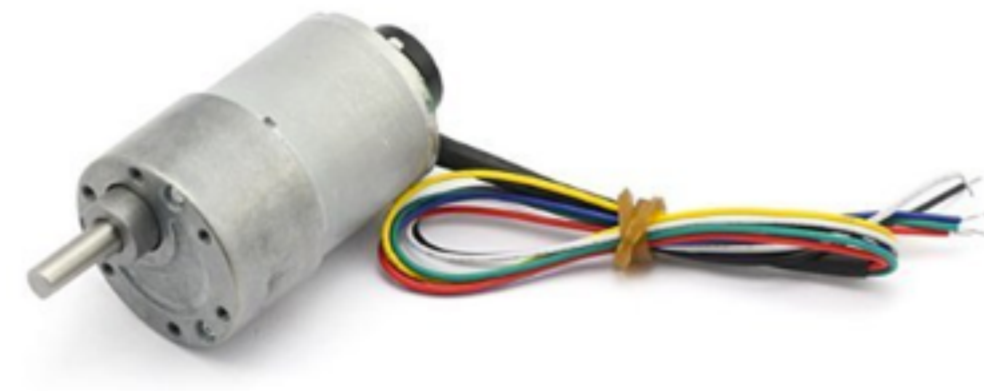
- MPU6050 single chip solution which costs $7 from Amazon. It works great in a Complementary Filter or you can use builtin Quaternion system (with warmup).



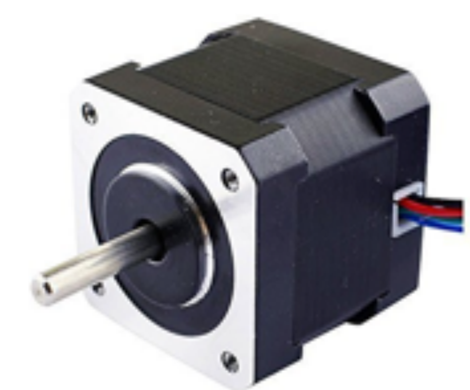
- *Choose whatever is convenient.*

Lessons Learned (motors)

- 29:1 Metal Gearmotor 37Dx52L mm with 64 CPR Encoder 12V 365rpm) — powerful, when not moving no power is used. This motor has built-in rotary encoder.



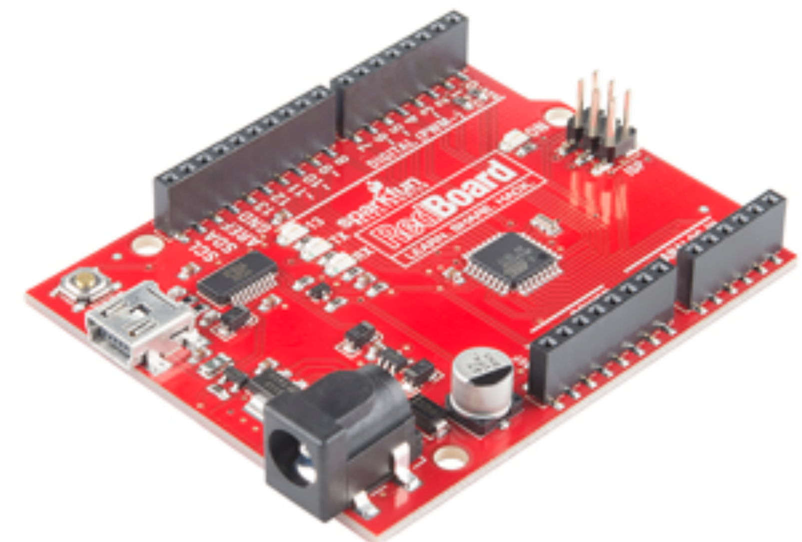
- Stepper Motor Nema 17 Bipolar 40mm 64oz.in(45Ncm) 2A 4 Lead — direct drive (quick & precise), positionally exact and power insensitive.



- *Steppers easier to balance.*

Lessons Learned (CPU)

- Sparkfun RedBoard
  - ✦ Uno Equivalent.
  - ✦ Great little board.



- Arduino Leonardo
  - ✦ Two 16-bit timers great in controlling stepper motors with minimum hardware.
  - ✦ Has booting issues.

- Either work acceptably.

Lessons Learned (4)

- Stick to dimensions you understand and can verify. I chose degrees and degrees/sec.

- Check all sensor inputs and make sure you understand their output and dimensions. Know the polarity of values (e.g. positive value on gyro means rotate forward).

- Check motor output and make sure speed is correct for value supplied. Know polarity of value (e.g. positive value means move forward).

Lessons Learned (5)

- Watch your signs — they bite. Make sure each contribution or result directionally makes sense.

- Test CF parts separately and make sure each part is doing what you think it does.

- Test each component of the PID (P - I - D) by itself and understand the range of operation before attempting to tune the PID filter — this is not normally mentioned in the tuning descriptions I found.

Lessons Learned (6)

- Get simple balancing solution working solidly before adding angle control. Very important!

- Don’t let the angle PID drive balance — it should manage robot speed and compensate for the center of gravity being a little off of center.

- Balancing works at 100 updates/sec. I have heard rumors saying 50 updates/sec is enough.

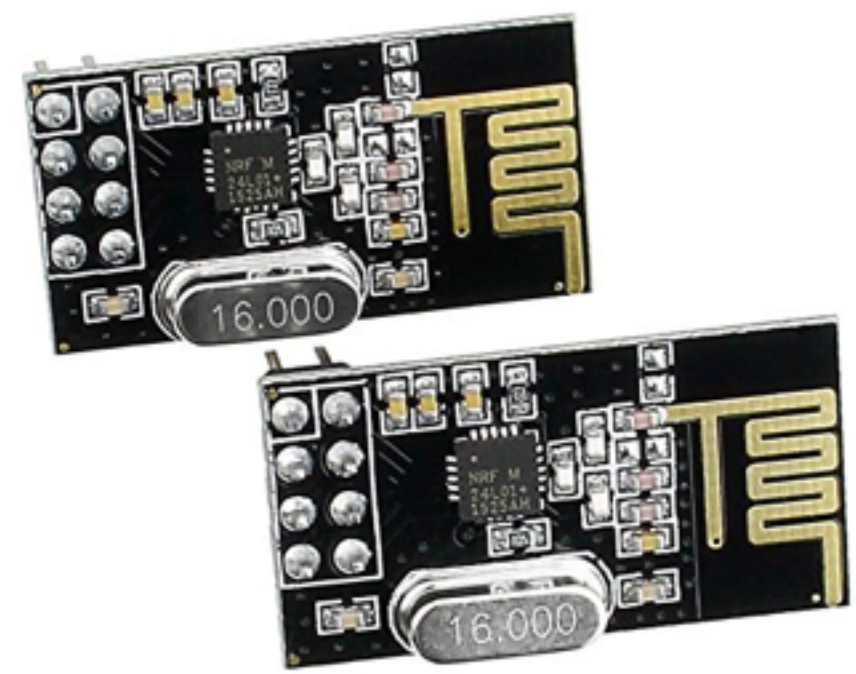
Lessons Learned (7)

- Building different versions with different technologies a great way to gain intuition and understanding on how things work and what is important.

- Build and understand a Segway design (1-d balance) before trying a Ballbot (2-d balance).

Lessons Learned (8)

- nRF24L01 is a great transceiver
- Works great with Arduinos
- Unfortunately, it has variable query time that can take more than 10ms
- This causes problems for robots running at 100 updates/ sec.



Great Resources

- ***Fast and Graceful Balancing Mobile Robots*** by *Umashankar Nagarajan*: http://www.cs.cmu.edu/~unagaraj/Umashankar_PhD_Thesis.pdf

- ***The Balance Filter*** by *Shane Colton*: http://d1.amobbs.com/bbs_upload782111/files_44/ourdev_665531S2JZG6.pdf

- ***Improving the Beginner's PID*** by *Brett Beauregard*: http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/

Summary

- It is practical to build a balancing robot, but you must focus on the many details.

- Take the time to test all input and outputs and understand their ranges. Similarly, take the time to test the parts of the algorithms before merging them into whole solution.

- May have to iterate on some of your choices.

- Well worth the time to build and study.