2 Wheel Self-Balancing Robot

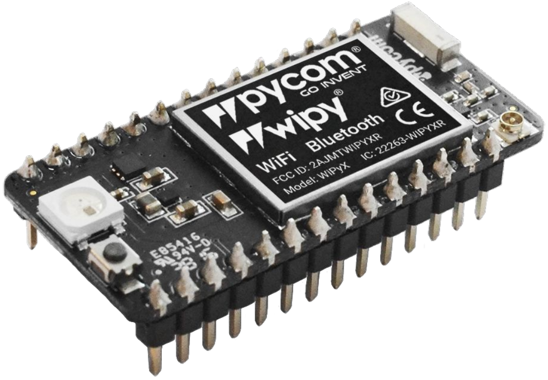
Eric Wang

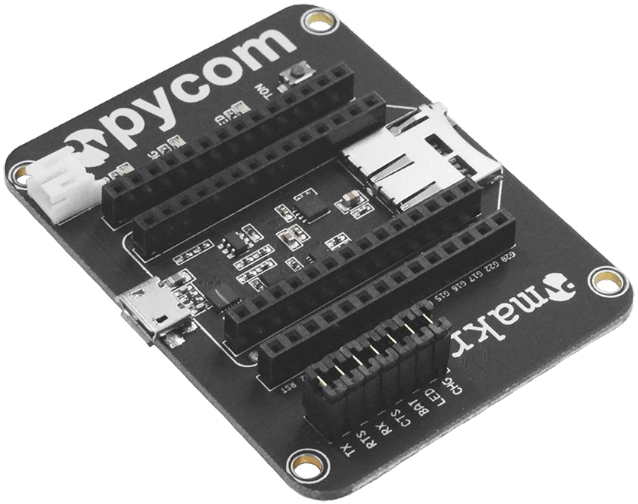
Segway

Handle (Boston Dynamics)

Development Board: WiPy 2.0

- ESP32 dual core microcontroller
- Wifi
- MicroPython





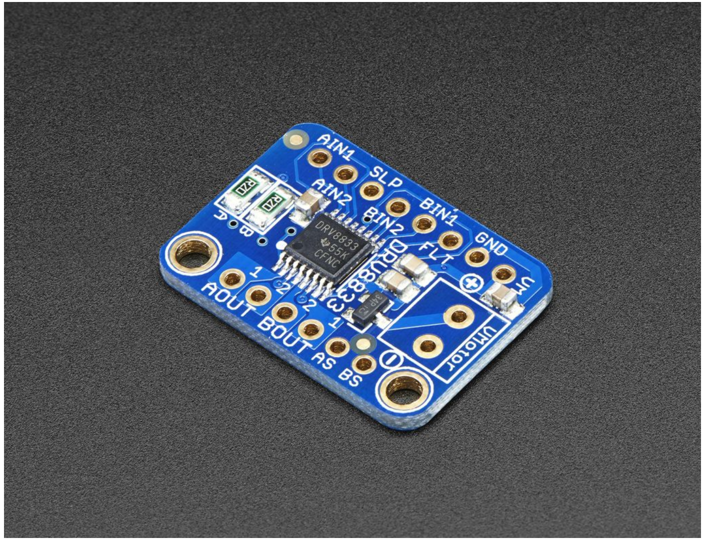
Sensors: MPU9250

- Accelerometer
- Gyroscope
- Magnetometer

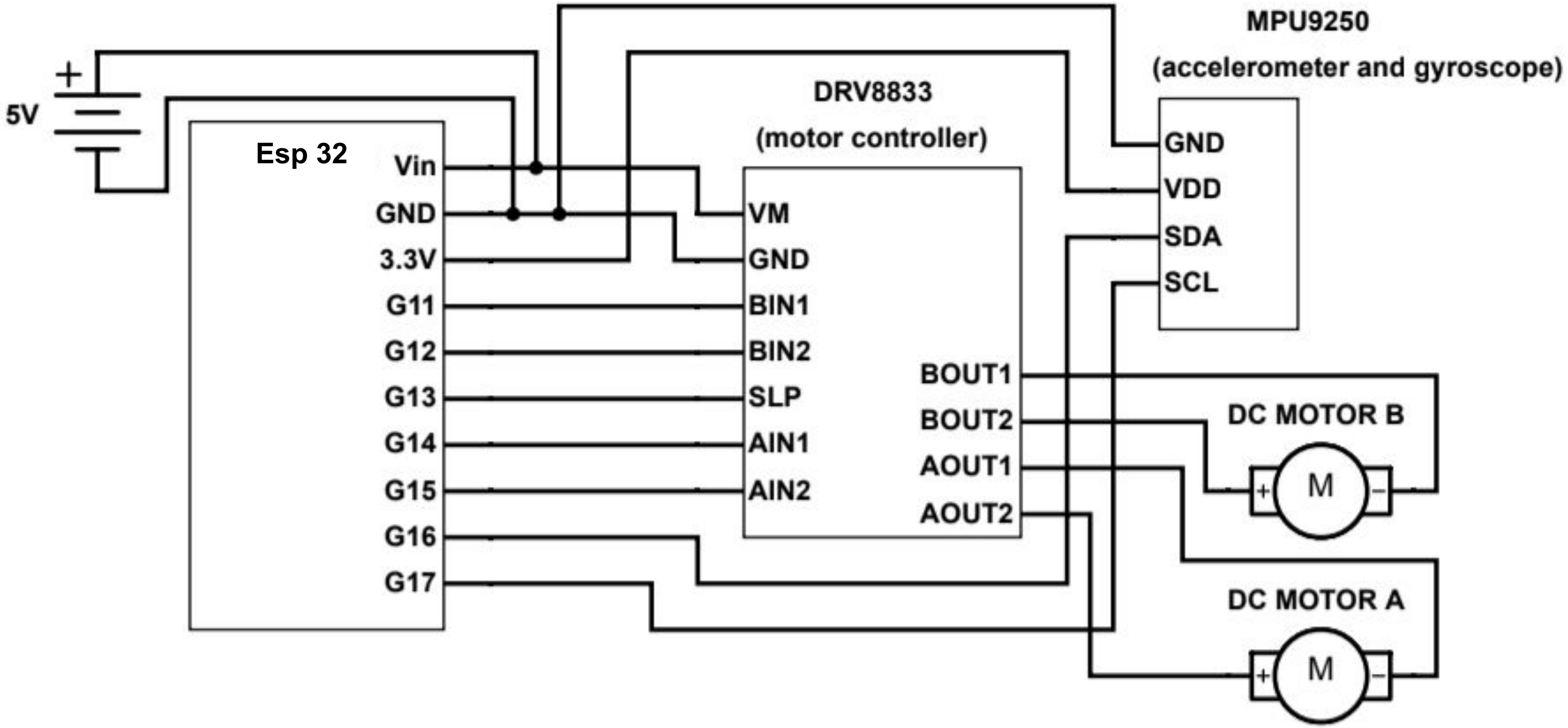
Motor Controller: DRV8833

2 H-Bridges - can drive 2 DC motors or

1 stepper motor

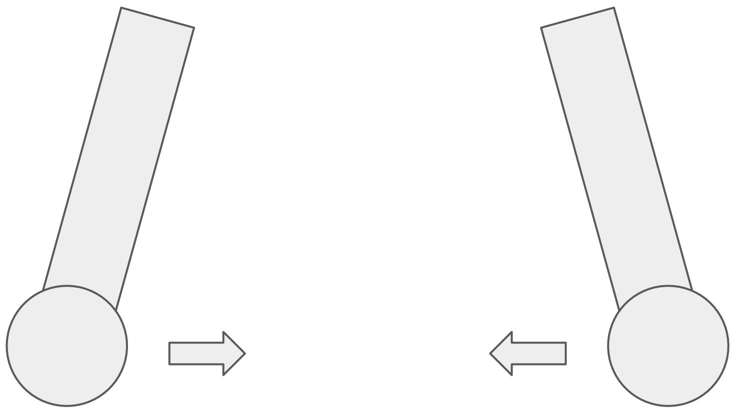


Circuit Diagram



Balancing

Move motors in direction of tilt



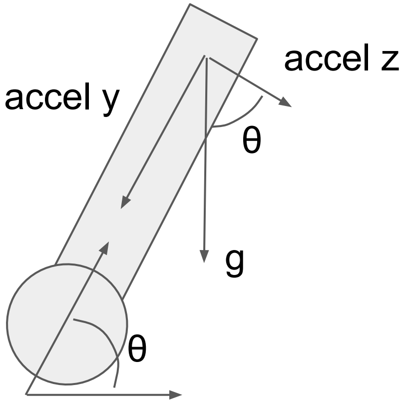
1. Determine tilt angle
2. Balance by correcting tilt angle

Determining Tilt Angle With Gyroscope

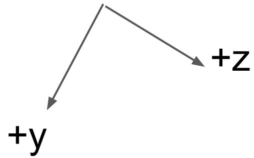
$$\theta = \int \omega dt + \theta_0$$

- Gyroscope has noise and θ will drift over long periods of time

Determining Tilt Angle With Accelerometer



Accelerometer coord system




```
angle = math.atan2(y,z)
```

Inaccurate when other forces are present

Complementary Filter

- Combines accelerometer and gyroscope data to give a good estimate for angle

```python
#complementary filter
while True:
    t = time.ticks_ms()
    #accelerometer data
    y = imu.accel.y
    z = imu.accel.z
    #angle from accelerometer in degrees
    tempAng = math.atan2(y,z) * 180 / math.pi
    #angular velocity from gyroscope
    gyrox = imu.gyro.x
    #pass data through filter
    angle = 0.1*tempAng + 0.9*(0.02*gyrox+angle)
    #make sure dt = 20ms
    elapsed = time.ticks_ms() - t
    time.sleep_ms(20-elapsed)
```

Modified Complementary Filter

Change the weight of accelerometer data based on how close it is to 1g

```
#complementary filter
while True:
    t = time.ticks_ms()
    #accelerometer data
    x = imu.accel.x
    y = imu.accel.y
    z = imu.accel.z
    #angle from accelerometer in degrees
    tempAng = math.atan2(y,z) * 180 / math.pi
    #determine weight for accelerometer data
    mag = math.sqrt(x*x + y*y + z*z)
    weight = 1 - 5 * math.fabs(1 - mag)
    if weight < 0:
        weight = 0
    weight /= 10
    #angular velocity from gyroscope
    gyrox = imu.gyro.x
    #pass data through filter
    angle = weight*tempAng + (1-weight)*(0.02*gyrox+angle)
    #make sure dt = 20ms
    elapsed = time.ticks_ms() - t
    time.sleep_ms(20-elapsed)
```

PID Controller

Control loop feedback mechanism

Proportional, integral, and derivative terms

```python
#PID Controller
balance = 86.7
errI = 0
Kp = 27
Kd = 0.12
Ki = 2.5

while True:
    #calculate P I D terms
    err = angle - balance
    errI += err
    errD = imu.gyro.x
    pid = (Kp * err) + (Kd * errD) + (Ki * errI)
    if pid > 100:
        pid = 100
    elif pid <- 100:
        pid = -100
    #move motors using slow decay
    if (pid > 0):
        forward(100 - pid)
    elif (pid < 0):
        backward(100 - (pid * -1))
```

PID constants

1. Make Kp, Ki, and Kd equal to zero.
2. Adjust Kp. Too little Kp will make the robot fall over, because there's not enough correction. Too much Kp will make the robot go back and forth wildly. A good enough Kp will make the robot go slightly back and forth (or oscillate a little).
3. Once the Kp is set, adjust Kd. A good Kd value will lessen the oscillations until the robot is almost steady. Also, the right amount of Kd will keep the robot standing, even if pushed.
4. Lastly, set the Ki. The robot will oscillate when turned on, even if the Kp and Kd are set, but will stabilize in time. The correct Ki value will shorten the time it takes for the robot to stabilize.

https://maker.pro/projects/arduino/build-arduino-self-balancing-robot

Placement of Parts

- Heaviest part on top
  - Reduces angular acceleration due to gravity - slower fall
  - PID controller can make adjustments before robot has tilted too far

Demo Video