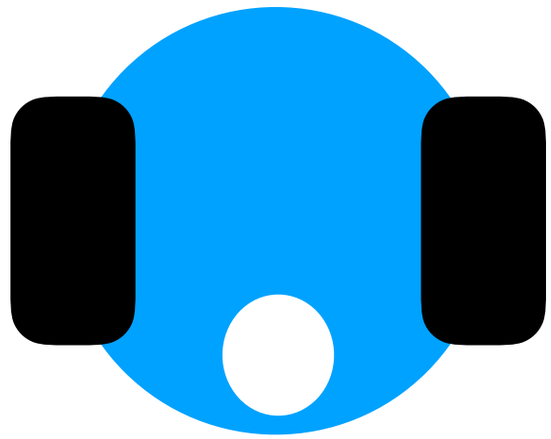
Driving Straight and Turning with the MPU6050

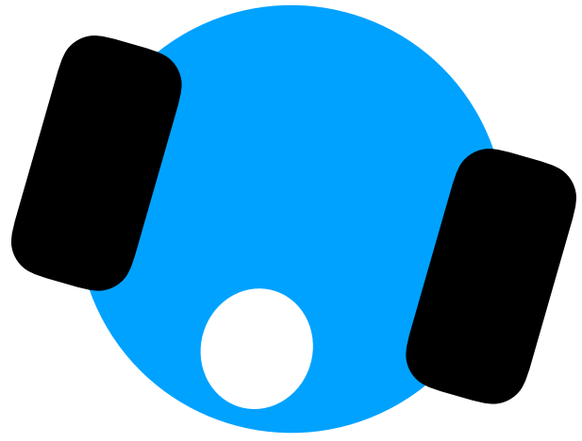
What’s the Problem?

- Can’t I just send the same PWM to each motor and have it drive straight?



What’s the Problem?

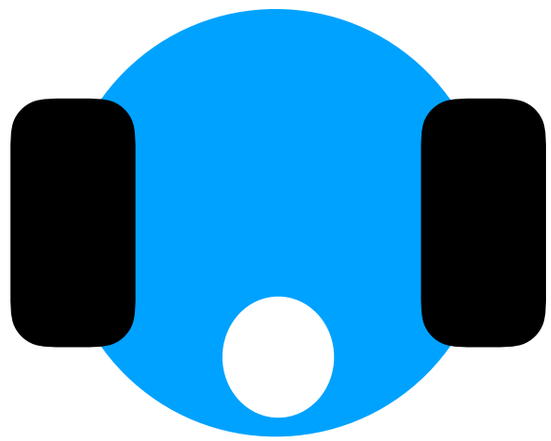
- Can’t I just send the same PWM to each motor and have it drive straight?



More than likely, the two motors will not spin at exactly the same speed even at the same voltage.

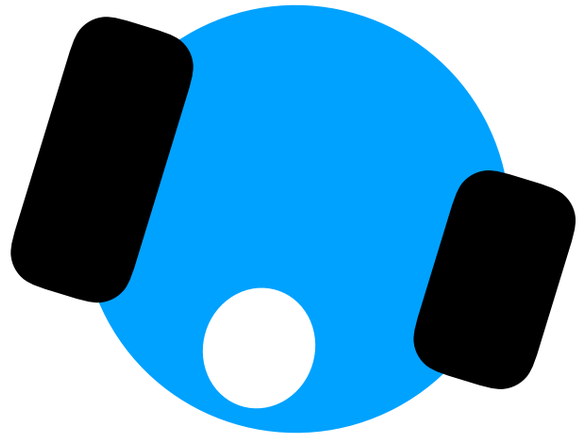
What’s the Problem?

- So I should add encoders and a PID loop to keep the motor speeds the same, right?



What’s the Problem?

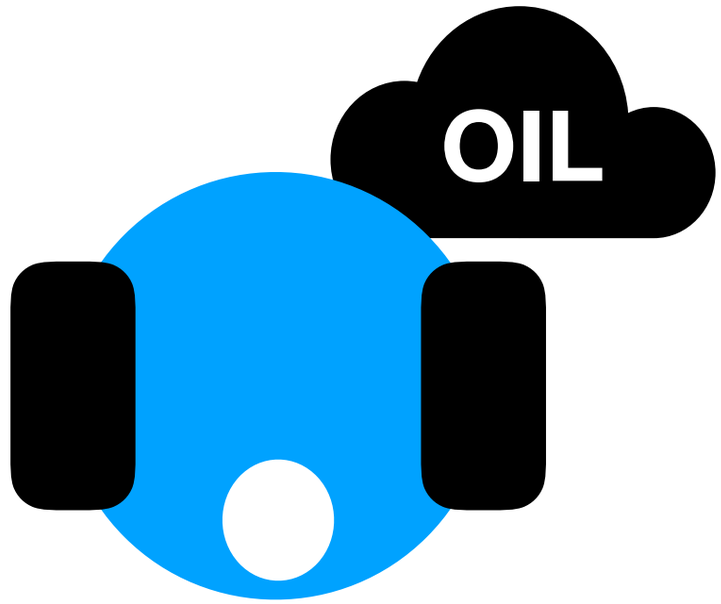
- So I should add encoders and a PID loop to keep the motor speeds the same, right?



Yes, that’s a good idea, but what if your wheels aren’t exactly the same size, or they aren’t perfectly aligned?

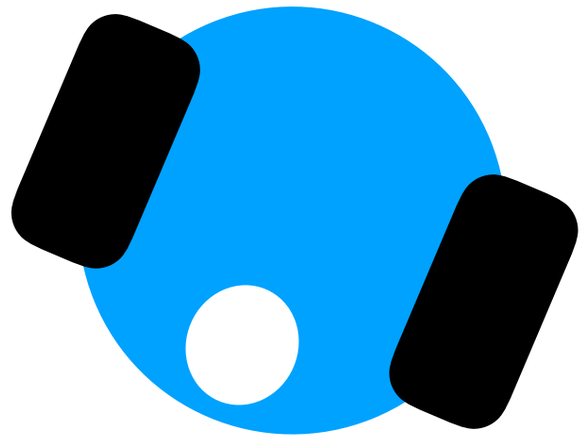
What’s the Problem?

- But I’ve designed and built a perfect robot, and tested motors so that they are matched exactly. Take that!



What's the Problem?

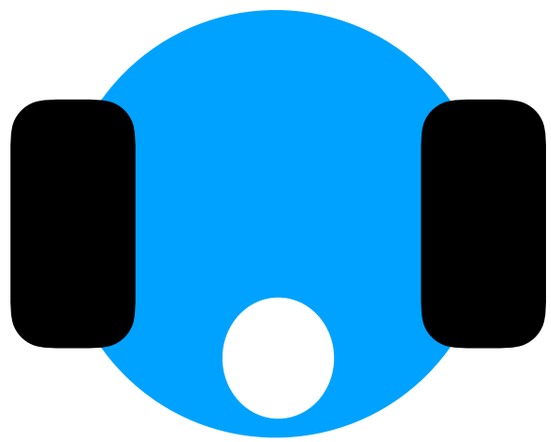
- But I've designed and built a perfect robot, and tested motors so that they are matched exactly. Take that!



Sure, but what happens when the environment causes uneven slippage?

Solutions

- Build the most mechanically accurate robot you can
- Use encoders to control speed
- Use other sensors



Solutions

- Use other sensors

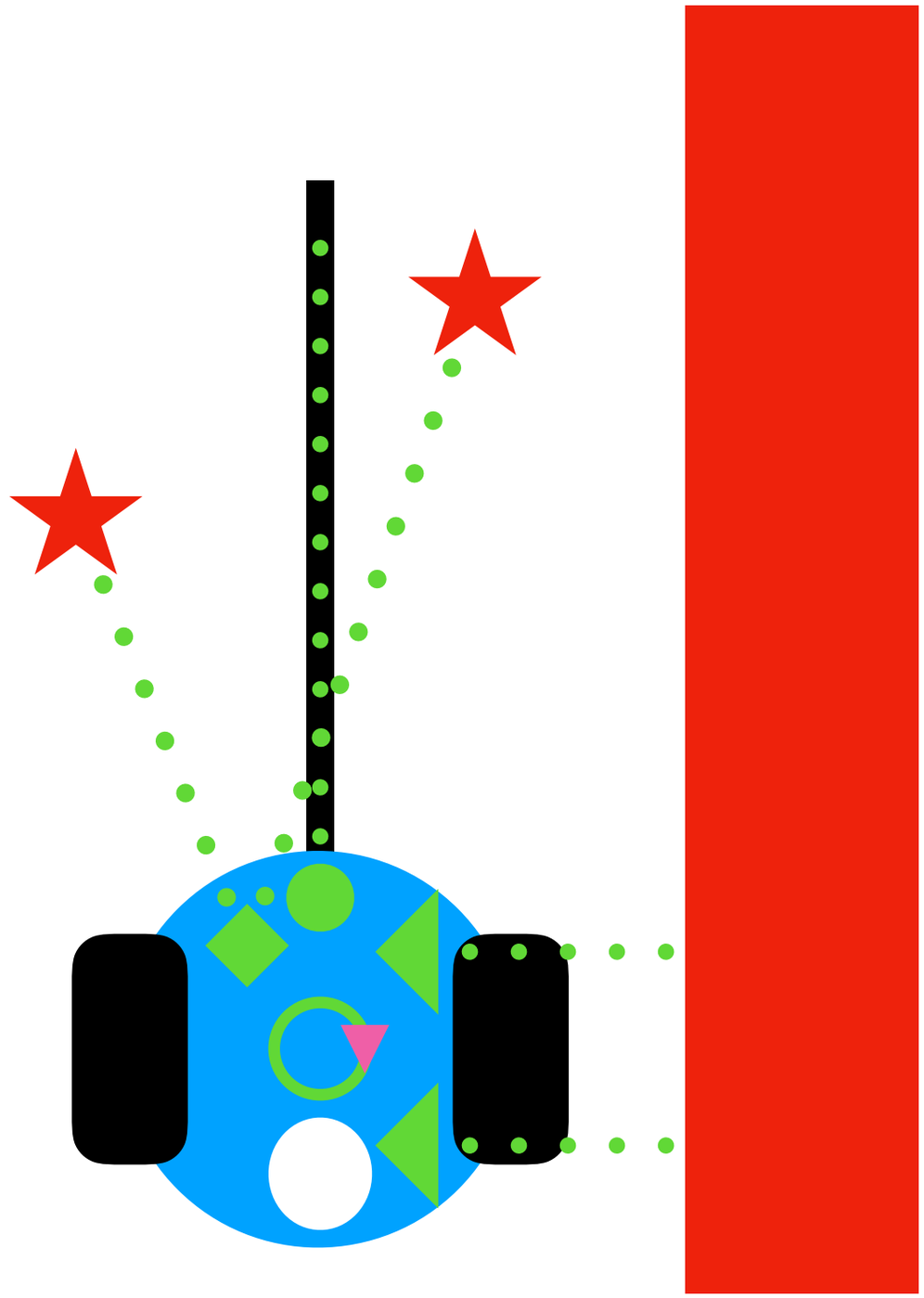
For example:

Follow walls

Follow lines

Watch interesting objects

Gyroscope



Gyroscope

- Measures angular rates (i.e. degrees or radians per second)
- Can integrate to obtain angular position
- Integration in 3 dimensions gives you Yaw, Pitch and Roll
- Attitude Heading Reference System (AHRS)
- See Skye's presentation "What is an IMU"

http://fll-freak.com/NRB/imu2.pdf

MPU6050

https://www.amazon.com/gp/product/B00NH8Z6BU
$6.95
Can be found cheaper via usual alternatives.

- 3D Accelerometer
- 3D Gyroscope
- “Digital Motion Processor” - separate processor that can handle integration of sensors
- I2C interface (simple to use with Arduino)
- I2C Dev Library plus MPU6050 library:

https://github.com/jrowberg/i2cdevlib

MPU6050

Demo:

MPU6050_DMP6

Example program from the MPU6050 Library - uses the built in Motion Processor

Outputs Yaw, Pitch, and Roll

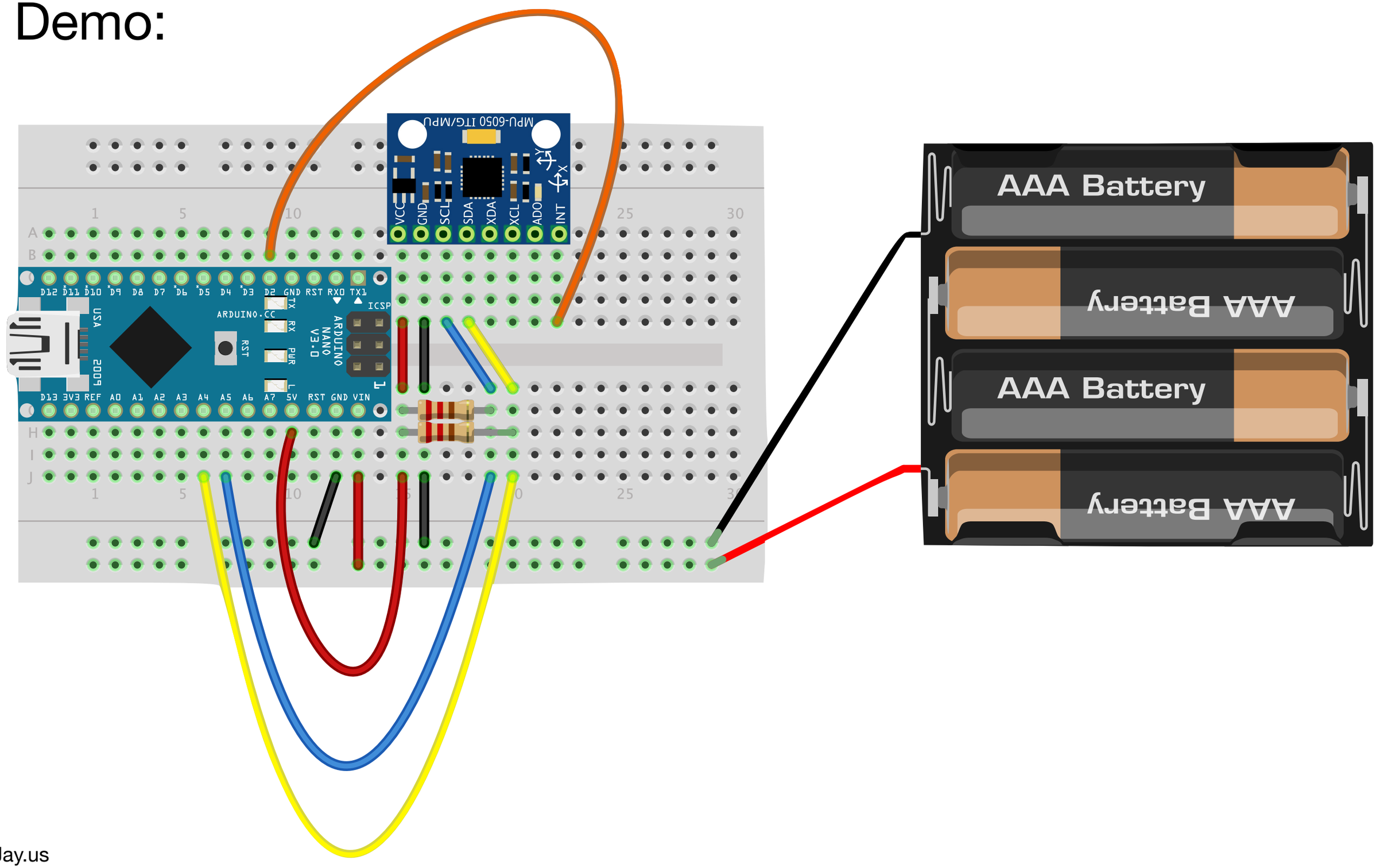
(uncomment #define statements for other outputs)

Things to notice…

Settling time, range of angle measures, drift.

MPU6050

Demo:



MPU6050

- Settling time

  It seems to take the Motion Processor about 15 seconds to stabilize (no matter if the device has been turned on already).

- Range of angle measures

  +/- 180 degrees (actually comes from MPU6050 in radians).

- Drift

  All gyros have drift (see Skye’s talk).

MPU6050

- Settling time

This is the most troublesome item, if you need your robot to start moving right away.

My current solution is to wait 15 seconds before starting to use the data.

A better solution would be to watch the data dynamically and let the code decide when the device is stable.

MPU6050

- Range of angle measures

Because the range is +/- 180 degrees, there is a discontinuity right at the +/- 180 degree point.

Any math needs to appropriately handle the possibility of crossing from +180 to -180

This may take a little head scratching at times :-)

MPU6050

- Drift

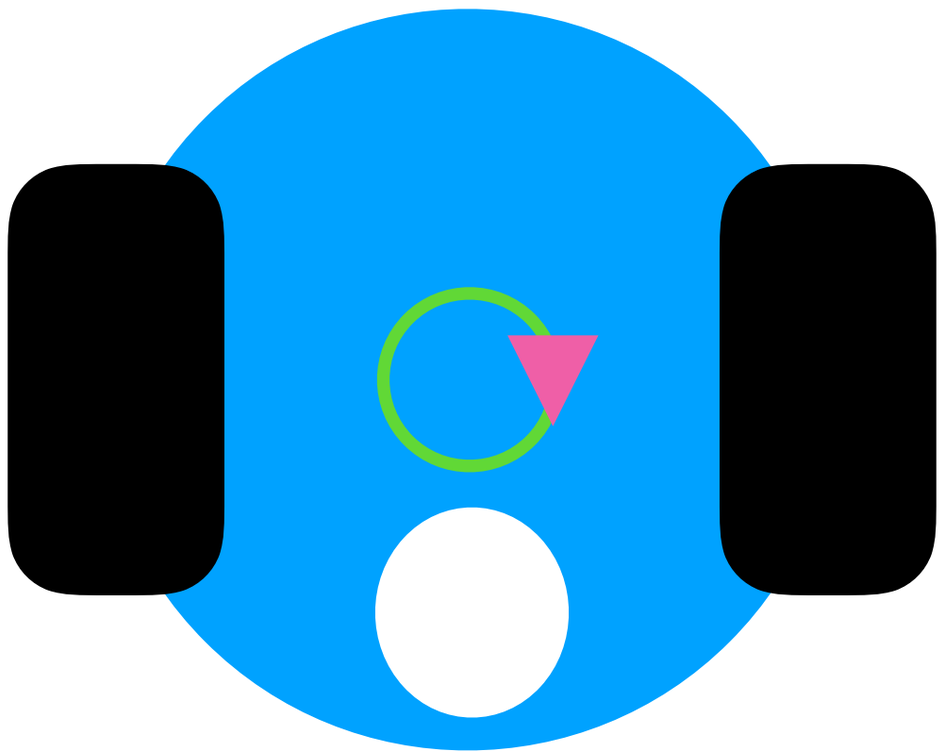
Over short distances, drift is less of a problem. If you can “reset” your absolute position every now and then, using other features (walls, lines, etc.), you can reduce the effect of drift.

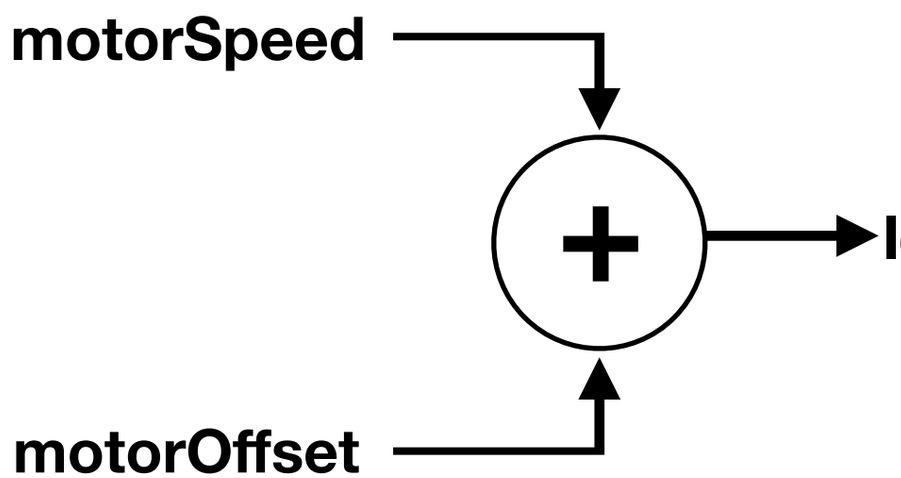
Although not easy to see, there could also be fixed offset errors in measurements. Use the IMU_Zero example code to determine YOUR DEVICE’s offsets.

Robot

Differential Drive with the MPU-6050 along the wheel axis (close to center, but not perfect).

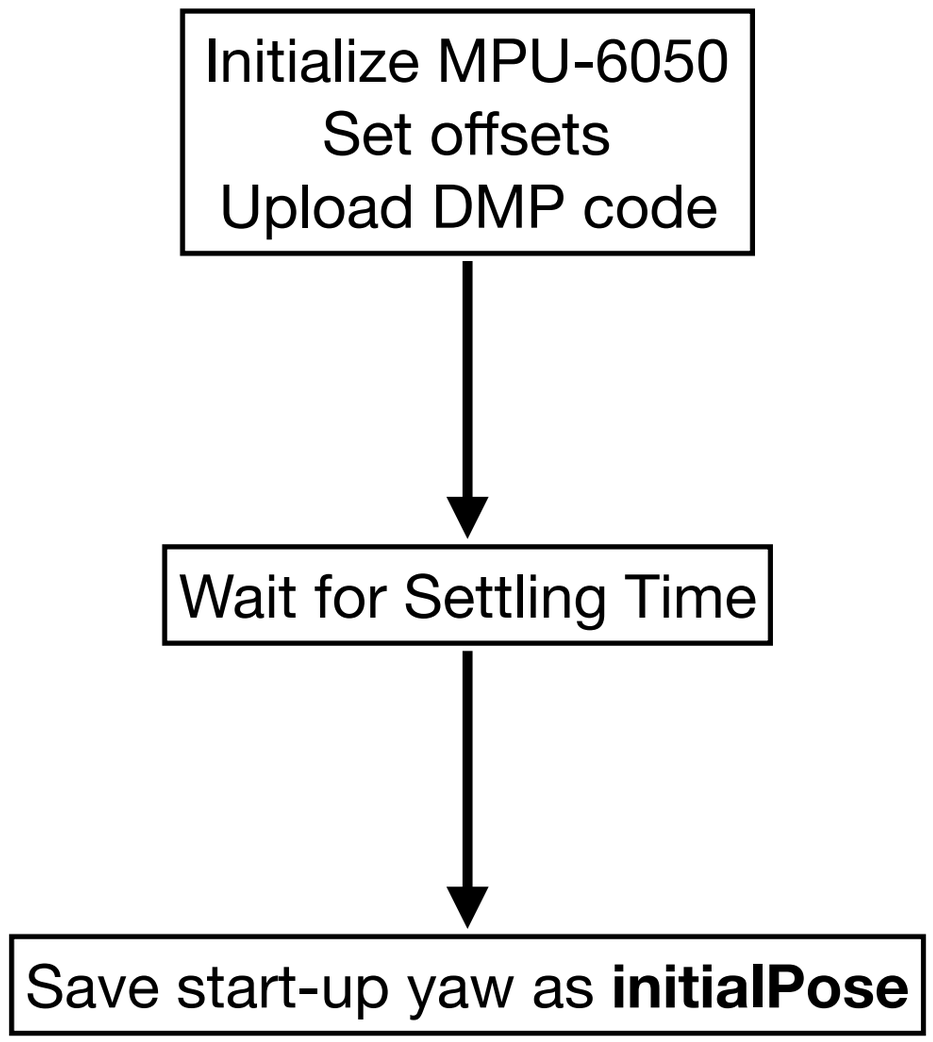
Yaw indicates the angle about the center of the robot as viewed from above.





Drive Forward, motorSpeed > 0

Turn in Place, motorSpeed = 0

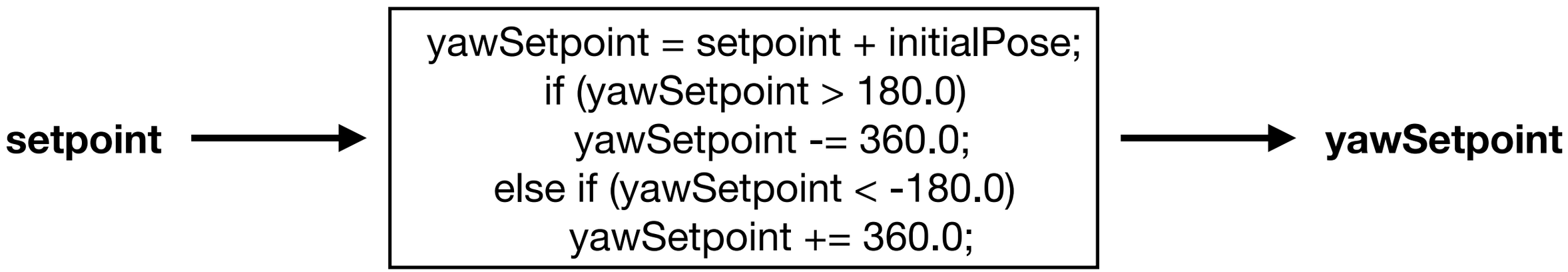

Initialize MPU-6050
Set offsets
Upload DMP code
Wait for Settling Time
Save start-up yaw as **initialPose**

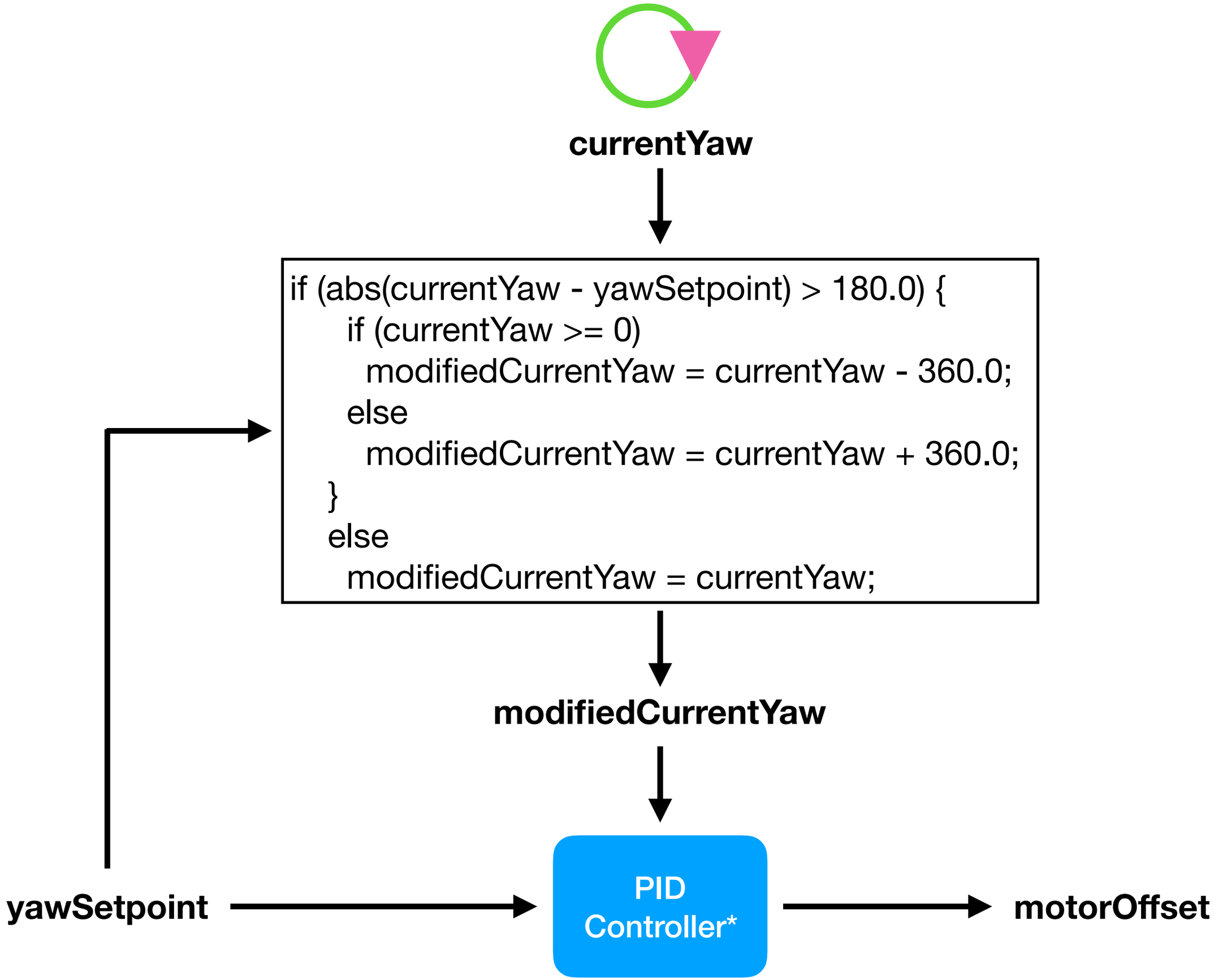
setpoint is our desired yaw relative to our **initialPose**

yawSetpoint is the absolute yaw used to compare with the readings from the MPU-6050

The code shown also takes care of the discontinuity at +/- 180 degrees

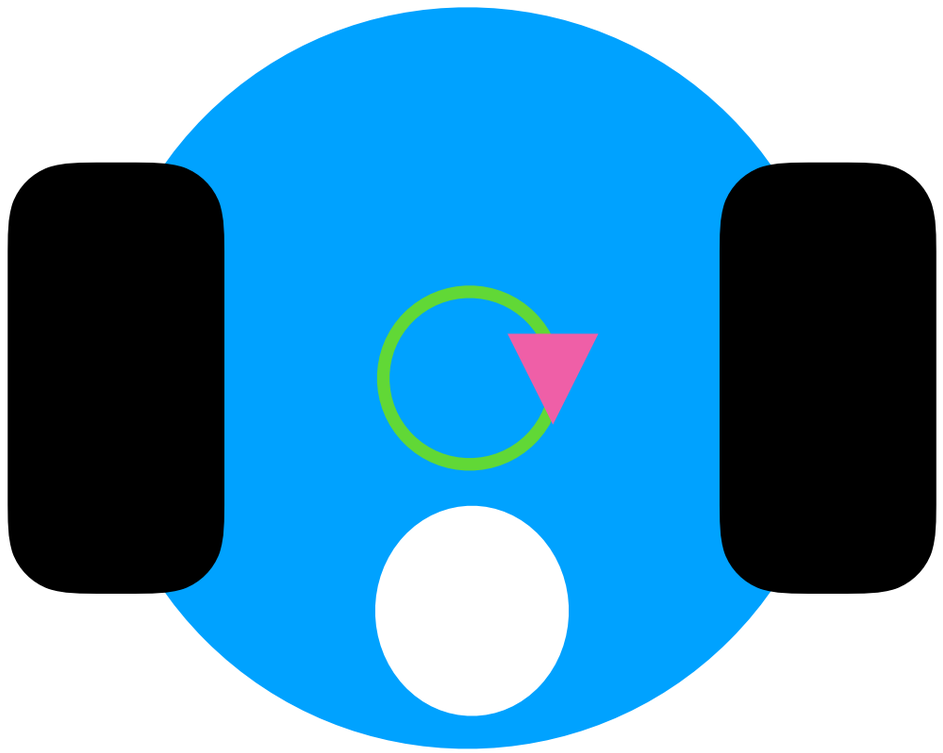
Assumes that our **setpoint** is constrained between +/- 180 degrees





*See Skye's "Understanding PID Loops" at http://fll-freak.com/NRB/PID.pdf

Demo



Questions?

Code will be posted along with the presentation.

I’m working on library extensions for the 2020 Bot.