PID Algorithm With Arduino and MPU6050 Tutorial

by Joshuar9202

I made a robot that consistently drives in a straight line as a result of a PID algorithm along with a mpu6050 sensor, Arduino mega, and Adafruit motor shield V2. This is a very useful concept especially for precise applications such as warehouse navigation or even competitions such as First Robotics Competitions (FRC).

Supplies:

Robot Chassis (4WD or 2WD you can make your own or you can buy a premade one)

Arduino Mega (UNO would work but it's hard to use the SDA and SCL pins when the shield is in use)

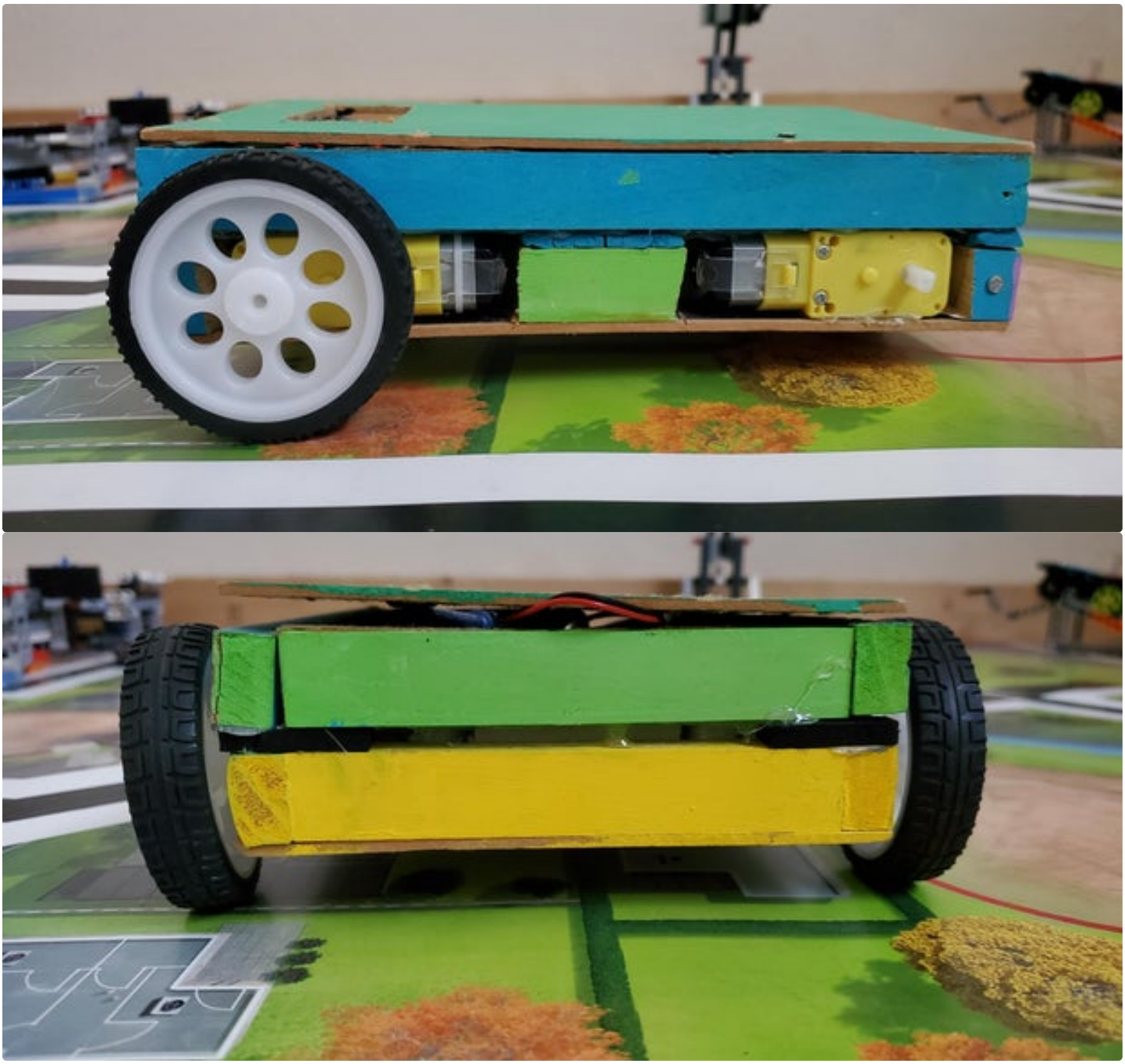
Adafruit motor shield (any motor driver can work but you have to modify the code)

MPU-6050 gyroscope and accelerometer

2 TT gear motors and wheels

Assorted jumper wires (male and female)

breadboard



Step 1: Making the Chassis

Before we go into any programming, we have to build the circuit for our robot.

MPU-6050 wiring:

VCC --> 5V (I used the servo pins on the motor shield, so I didn't have to solder any new pins)

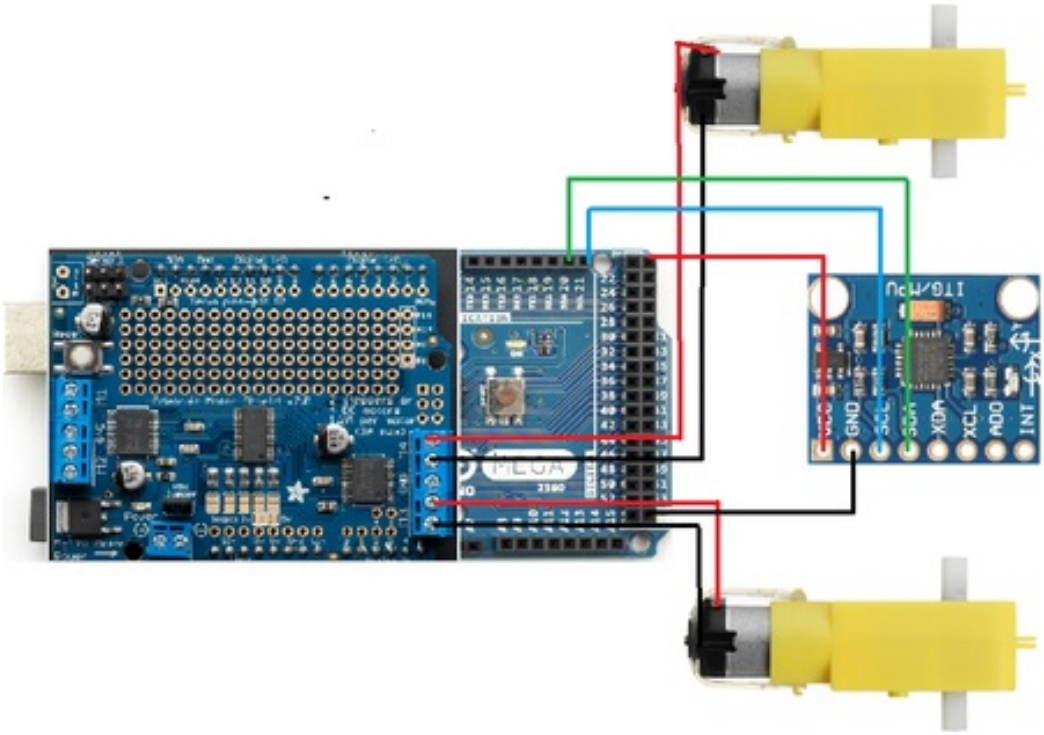
GND --> GND (Again I used the servo outputs)

SCL --> pin 21 on the mega

SDA --> pin 20 on the mega

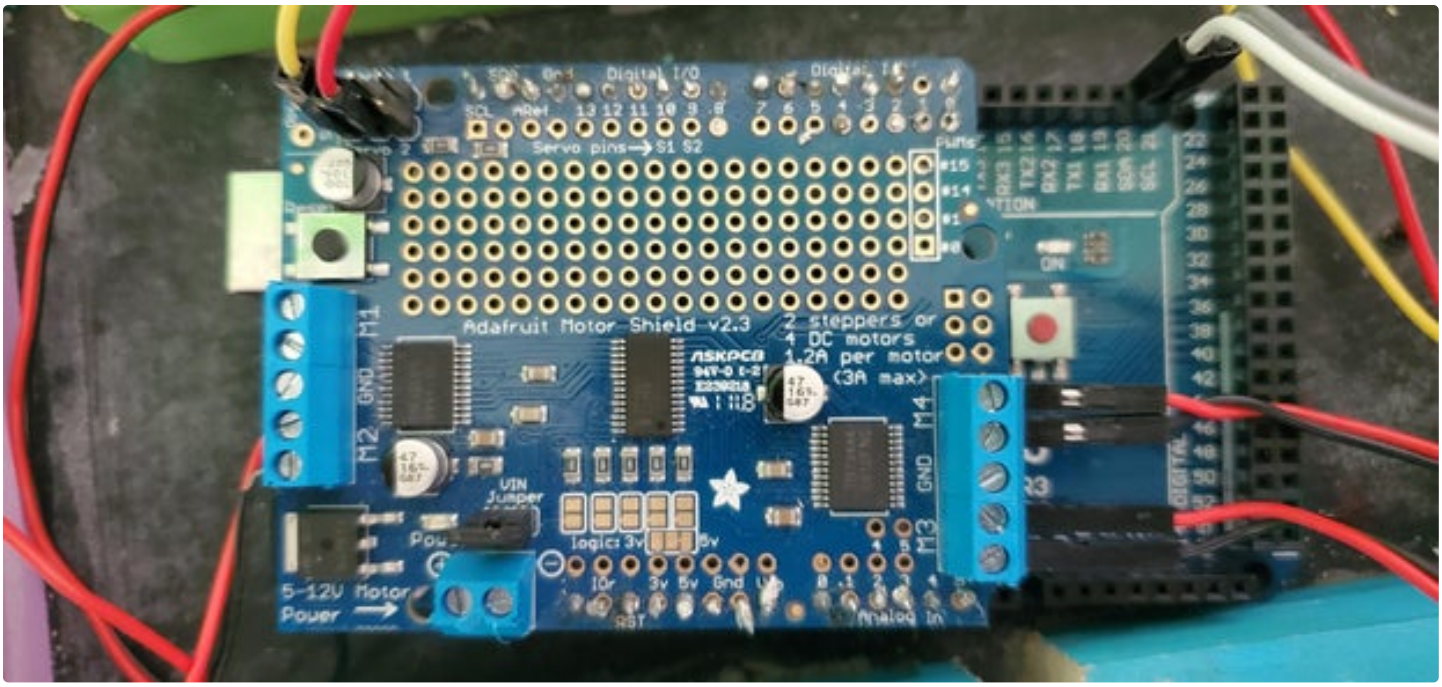
Left gearmotor --> M4 on the motor shield

Right gearmotor --> M3 on the motor shield



Step 2: Setting Up the Mpu-6050

To be able to program the sensor I used the collection of libraries from Jeff Rowberg, but since he developed the set of libraries on Github for different things, not only Arduino. The whole set of libraries can be found here https://github.com/jrowberg/i2cdevlib, but only the files for the Arduino are the I2Cdev and MPU6050 folders in the folder titled Arduino. Also, the calibration section of the program was the same one in DroneBot Workshop's tutorial on building an electronic level, so thank you to them. Now on to the PID programming.



Step 3: Understanding PID

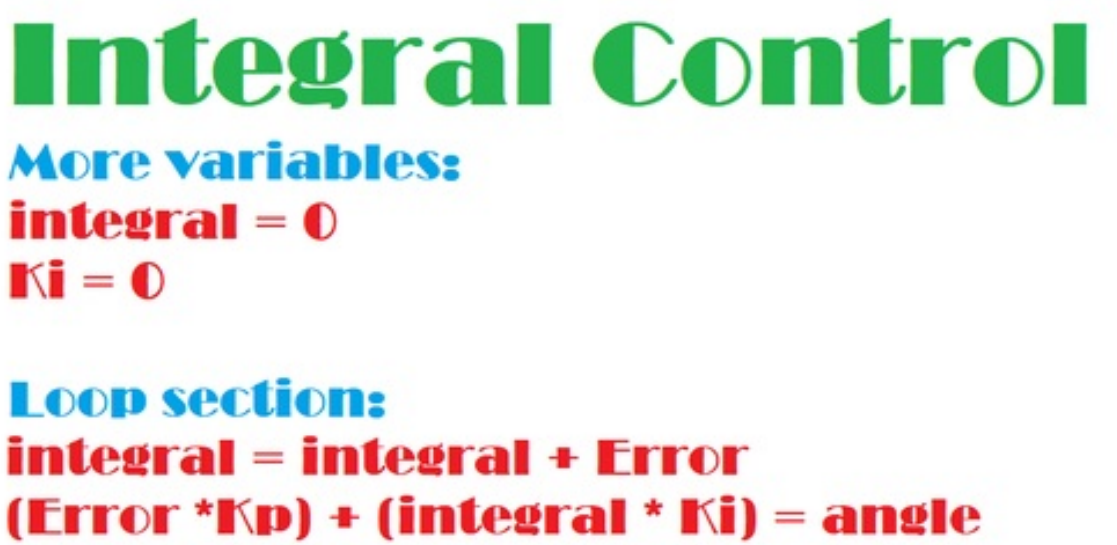
PID stands for proportional, integral, and derivative. The integral and derivative are calculus terms but considering that calculus is the mathematical study of continuous change, this almost fits our needs perfectly. PID is a control algorithm, so it could be used for everything from a gyro sensor control loop to a line follower loop. In the end the angle will be the error multiplied by the Kp added to the integral multiplied by the Ki added to the derivative multiplied by the Kd.

Step 4: Proportional Control

The proportional control is the correction stage. The way it is represented in the code is the error is equal to the target gyro value which in most cases is 0 but you can make it whatever you want, subtracted by the gyro readings. Then the error is multiplied by the Kp to give the correction output to which we will later add the integral and derivative.

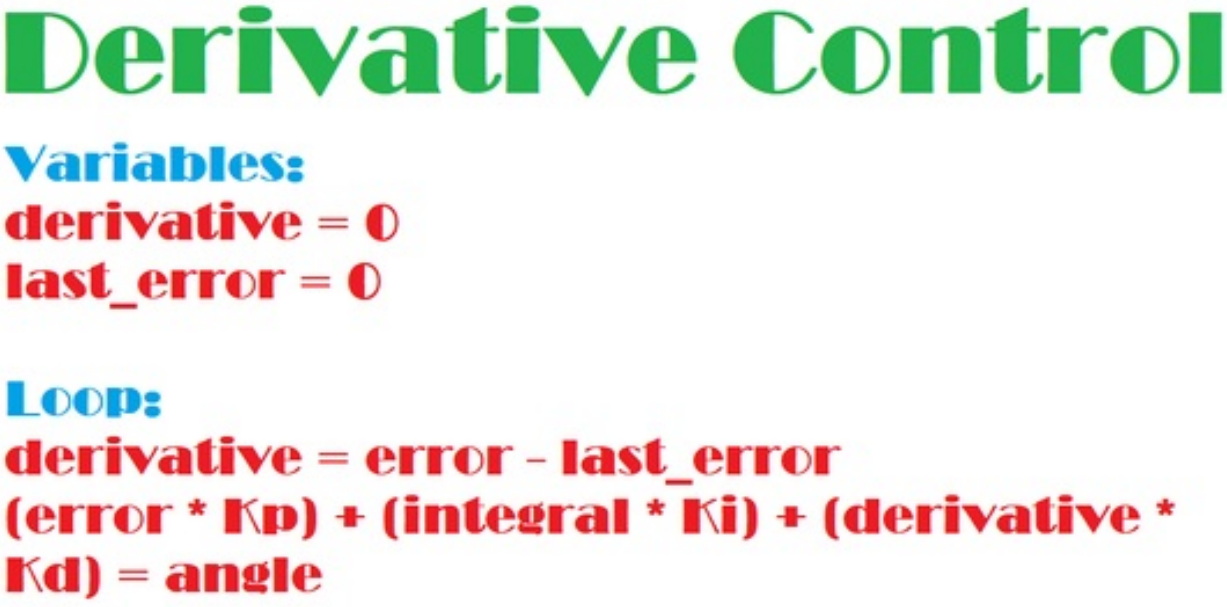
Step 5: Integral Control

The integral is supposed to keep the robot on the target path not only in terms of the angle, but in terms of the latitude too. It does this by measuring the trend that the robot strays off of the path over time. Every time the loop goes around, the error is added to the integral to make the new integral value. Then in the same section of code where we put the proportional times the Kp, multiply the integral by the Ki and add it to the proportional control that's already there. Now for the final piece of the control system, the derivative.



Step 6: Derivative Control

The derivative compares the current error, to the error from the previous time the loop went around. We do this by measuring the error and storing the error value at the end of the loop as the variable called last error. Now the next time the loop comes around, we can effectively compare the error from the proportional section by the error from the last error. This is then stored in the derivative variable and multiplied by the Kd and once again, we add it to the integral and the proportional to get our final angle output. Also note that all the variables that we use for the PID controller are floats not integers, because we need very precise values to get a smooth, straight drive. Now we actually have to use the output for controlling our motor.



Step 7: Controlling the Motors

The way we will control the motors is by adding the absolute value of the angle to the right motor and subtracting from the left motor when its veering to the right and vice versa if it is veering to the left. Again, I used the Adafruit motor shield, so if you want to use a different motor driver you have to modify the code. And now we're finally done with the code and the robot.

Step 8: Testing It Out

As you can see from the video, the robot stays on the path even with interference, however when you push it too far off the path, the angle value becomes greater than 255 (max value) or less than 0 and then the robot really doesn't know what to send to the motors since the motor driver can't send values that are in that range, but this is already much more interference than anyone would really ever see in normal robotics competition conditions so other than that it works really well. The code is down below.

https://www.youtube.com/watch?v=s8zxtvNrVA4

Download

https://www.instructables.com/ORIG/FWW/09CI/L8K7LW01/FWW09CIL8K7LW01.ino