

Gyros - Measuring rotation and controlling robot driving direction

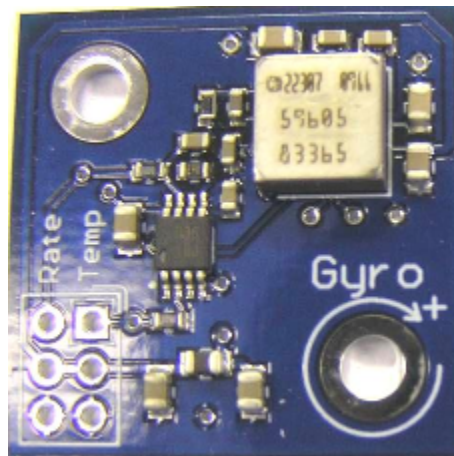
Gyros typically in the FIRST kit of parts are provided by Analog Devices, and are actually angular rate sensors. The output voltage is proportional to the rate of rotation of the axis perpendicular to the top package surface of the gyro chip. The value is expressed in $\text{mV}/^\circ/\text{second}$ (degrees/second or rotation expressed as a voltage). By integrating (summing) the rate output over time, the system can derive the relative heading of the robot.

Another important specification for the gyro is its full-scale range. Gyros with high full-scale ranges can measure fast rotation without “pinning” the output. The scale is much larger so faster rotation rates can be read, but there is less resolution due to a much larger range of values spread over the same number of bits of digital to analog input. In selecting a gyro, you would ideally pick the one that had a full-scale range that matched the fastest rate of rotation your robot would experience. This would yield the highest accuracy possible, provided the robot never exceeded that range.

Note: The AnalogGyro class in WPILib uses a hardware (implemented in the FPGA) accumulator to perform the integration. This means Gyros are supported on a specific, limited, set of channels. On the roboRIO this is currently Analog Inputs 0 and 1 on the on-board headers.

Note: The Gyro class has been renamed to AnalogGyro for FRC 2016 to better support newer gyros that are not necessarily connected through an analog input. There is now an interface, Gyro, used as the base for all gyros regardless of the connection type. Types should be declared using the interface, but initialized using the more specific device type.

Using the AnalogGyro class



Gyros - Measuring rotation and controlling robot driving direction

The Gyro object should be created in the constructor of the [RobotBase](#) derived object. When the AnalogGyro object is used, it will go through a calibration period to measure the offset of the rate output while the robot is at rest to minimize drift. This requires that the robot be stationary and the gyro is unusable until the calibration is complete.

Once initialized, the [GetAngle\(\)](#) (or [getAngle\(\)](#) in Java) method of the Gyro object will return the number of degrees of rotation (heading) as a positive or negative number relative to the robot's position during the calibration period. The zero heading can be reset at any time by calling the [Reset\(\)](#) ([reset\(\)](#) in Java) method on the AnalogGyro object.

See the code samples below for an idea of how to use the AnalogGyro objects.

Setting Gyro sensitivity

The Gyro class defaults to the settings required for the 250°/sec gyro that was delivered by FIRST in the 2012-2014 Kit of Parts (ADW22307). It is important to check the documentation included with the gyro to ensure that you have the correct sensitivity setting.

To change gyro types call the [SetSensitivity\(float sensitivity\)](#) method (or [setSensitivity\(double sensitivity\)](#) in Java) and pass it the sensitivity in volts/°/sec. Take note that the units are typically specified in mV (volts / 1000) in the spec sheets. For example, a sensitivity of 12.5 mV/°/sec would require a [SetSensitivity\(\)](#) ([setSensitivity\(\)](#) in Java) parameter value of 0.0125.

Using a gyro to drive straight

The following example programs cause the robot to drive in a straight line using the gyro sensor in combination with the [RobotDrive](#) class. The [RobotDrive.Drive](#) method takes the speed and the turn rate as arguments; where both vary from -1.0 to 1.0. The gyro returns a value indicating the number of degrees positive or negative the robot deviated from its initial heading. As long as the robot continues to go straight, the heading will be zero. This example uses the gyro to keep the robot on course by modifying the turn parameter of the Drive method.

The angle is multiplied by a proportional scaling constant (Kp) to scale it for the speed of the robot drive. This factor is called the proportional constant or loop gain. Increasing Kp will cause the robot to correct more quickly (but too high and it will oscillate). Decreasing the value will cause the robot correct more slowly (possibly never reaching the desired heading). This is known as proportional control, and is discussed further in the PID control section of the advanced programming section.

```
C++
```

Gyros - Measuring rotation and controlling robot driving direction

```
class GyroSample : public SampleRobot
{
    RobotDrive myRobot; // robot drive system
    AnalogGyro gyro;
    static const float kP = 0.03;

public:
    GyroSample():
        myRobot(1, 2), // initialize the sensors in initialization list
        gyro(1)
    {
        myRobot.SetExpiration(0.1);
    }

    void Autonomous()
    {
        gyro.Reset();
        while (IsAutonomous())
        {
            float angle = gyro.GetAngle(); // get heading
            myRobot.Drive(-1.0, -angle * kP); // turn to correct heading
            Wait(0.004);
        }
        myRobot.Drive(0.0, 0.0); // stop robot
    }
};
```

Sample Java program for driving straight

Java

```
package edu.wpi.first.wpilibj.templates;
import edu.wpi.first.wpilibj.AnalogGyro;
import edu.wpi.first.wpilibj.RobotDrive;
import edu.wpi.first.wpilibj.SampleRobot;
import edu.wpi.first.wpilibj.Timer;
```

Gyros - Measuring rotation and controlling robot driving direction

```
public class GyroSample extends SampleRobot {

    \   private RobotDrive myRobot; // robot drive system
        private Gyro gyro;

    \   double Kp = 0.03;

    public GyroSample() {
        gyro = new AnalogGyro(1); \           // Gyro on Analog Channel 1
        myRobot = new RobotDrive(1,2); \ // Drive train jaguars on PWM 1 and 2
        myRobot.setExpiration(0.1);
    \ }

    public void autonomous() {
        gyro.reset();
        while (isAutonomous()) {
            double angle = gyro.getAngle(); // get current heading
            myRobot.drive(-1.0, -angle*Kp); // drive towards heading 0
            Timer.delay(0.004);
        }
        myRobot.drive(0.0, 0.0);
    \ }
}
```

This is a sample Java program that drives in a straight line. See the comments in the C++ example (previous step) for an explanation of its operation.

Thanks to Joe Ross from FRC team 330 for help with this example.