DEBUGGING AND TESTING ROBOT PROGRAMS

Last Updated: 06-11-2016

Table of Contents

Testing strategies....3

How to write an easy to test robot program....4

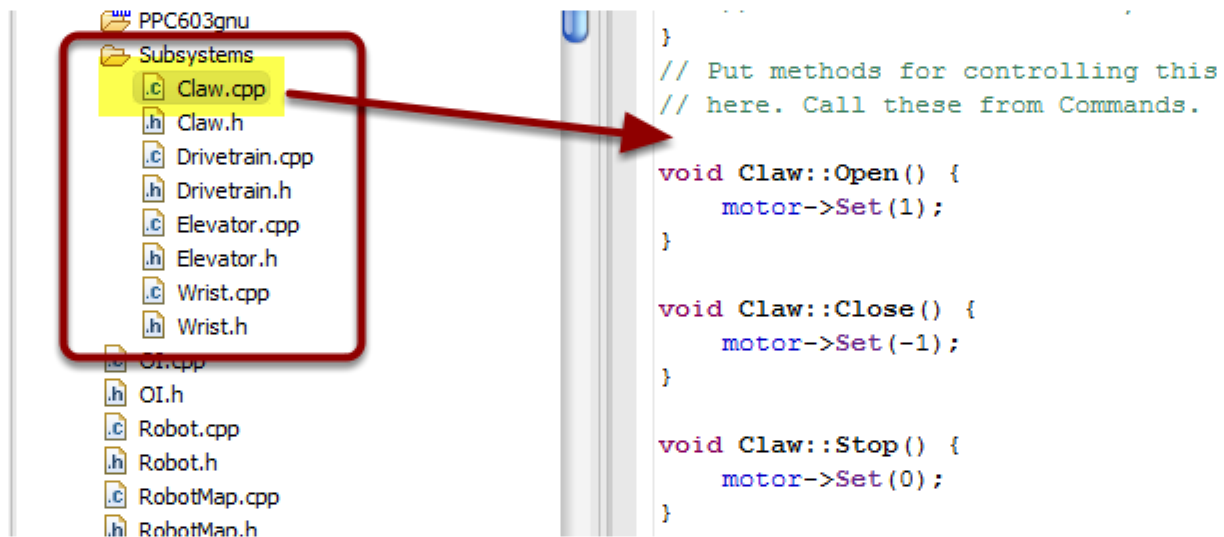
Using test mode....9

Testing strategies

How to write an easy to test robot program

The command based programming model is designed to simplify creating very easy to write and especially easy to test robot programs. All the pieces come together when it's time to see how your program works so that pieces developed by multiple programmers can be integrated and tested with the main robot program.

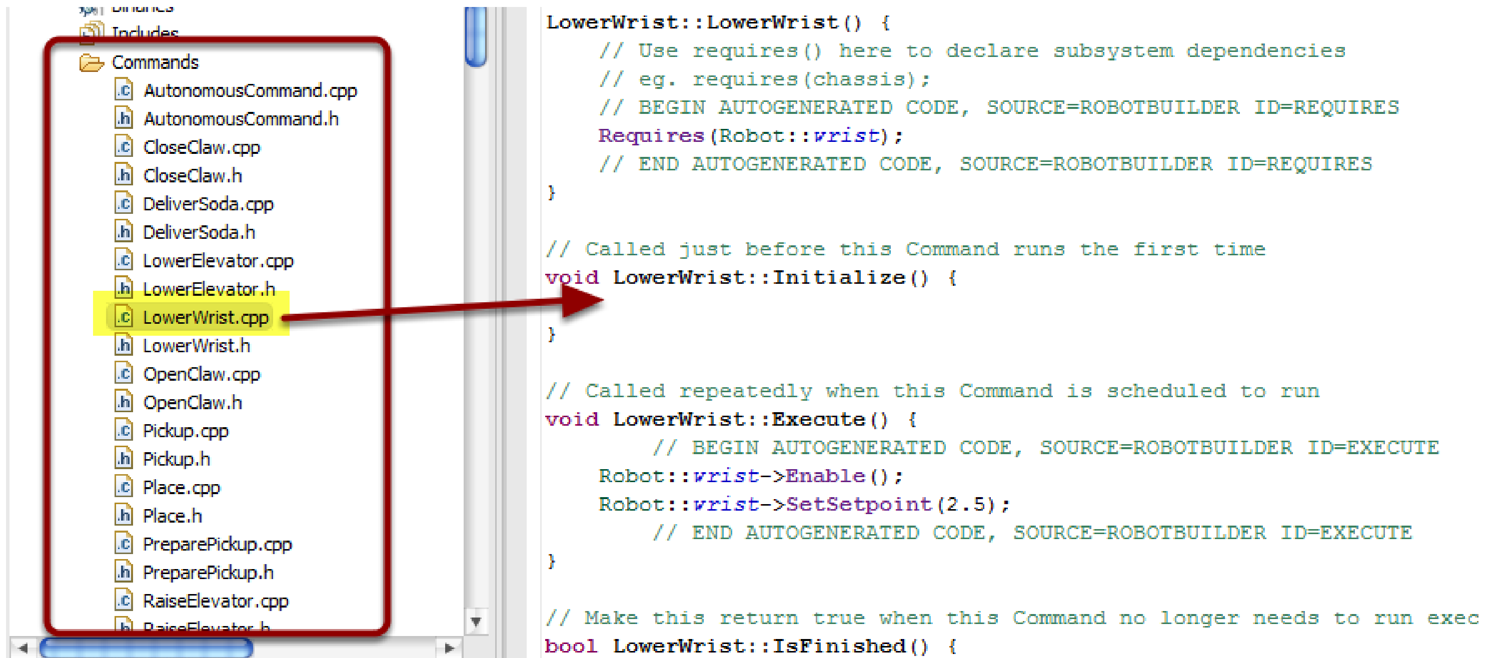
Break up the robot into subsystems



The first step it to break the robot program into subsystems that each represent a separately controlled mechanism on the robot. Each subsystem has methods that operate the mechanism in some way.

Debugging and testing robot programs

Create commands for each robot behavior



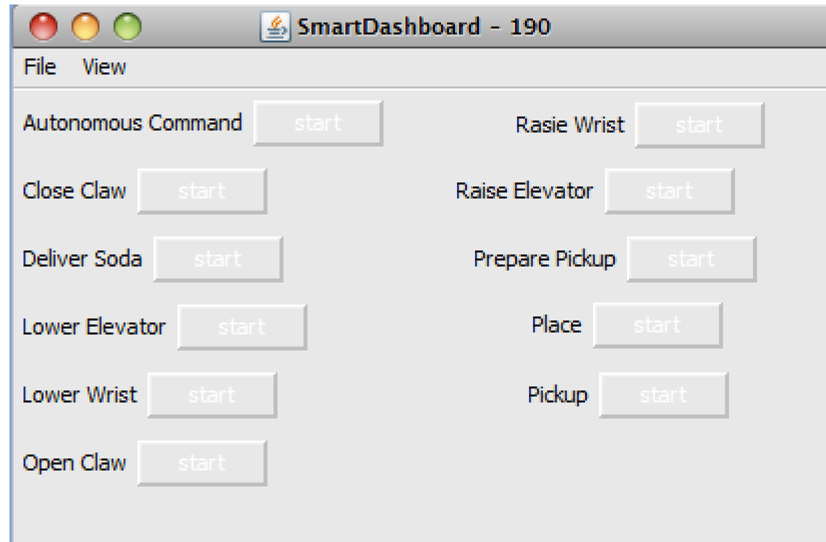
Commands define the behaviors of the robot, that is the operation of the subsystems over time. A command starts a subsystem doing something, then waits until it is finished. Upon completion, the next command can be scheduled. It is the defining of commands for each behavior that is the key to making it easy to test the code as you'll see in the next few sections of this lesson.

Add commands to the SmartDashboard

```
    // SmartDashboard Buttons
SmartDashboard::PutData("Autonomous Command", new AutonomousCommand());
SmartDashboard::PutData("Open Claw", new OpenClaw());
SmartDashboard::PutData("Close Claw", new CloseClaw());
SmartDashboard::PutData("Lower Wrist", new LowerWrist());
SmartDashboard::PutData("Rasie Wrist", new RasieWrist());
SmartDashboard::PutData("Lower Elevator", new LowerElevator());
SmartDashboard::PutData("Raise Elevator", new RaiseElevator());
SmartDashboard::PutData("Prepare Pickup", new PreparePickup());
SmartDashboard::PutData("Pickup", new Pickup());
SmartDashboard::PutData("Place", new Place());
SmartDashboard::PutData("Deliver Soda", new DeliverSoda());
```

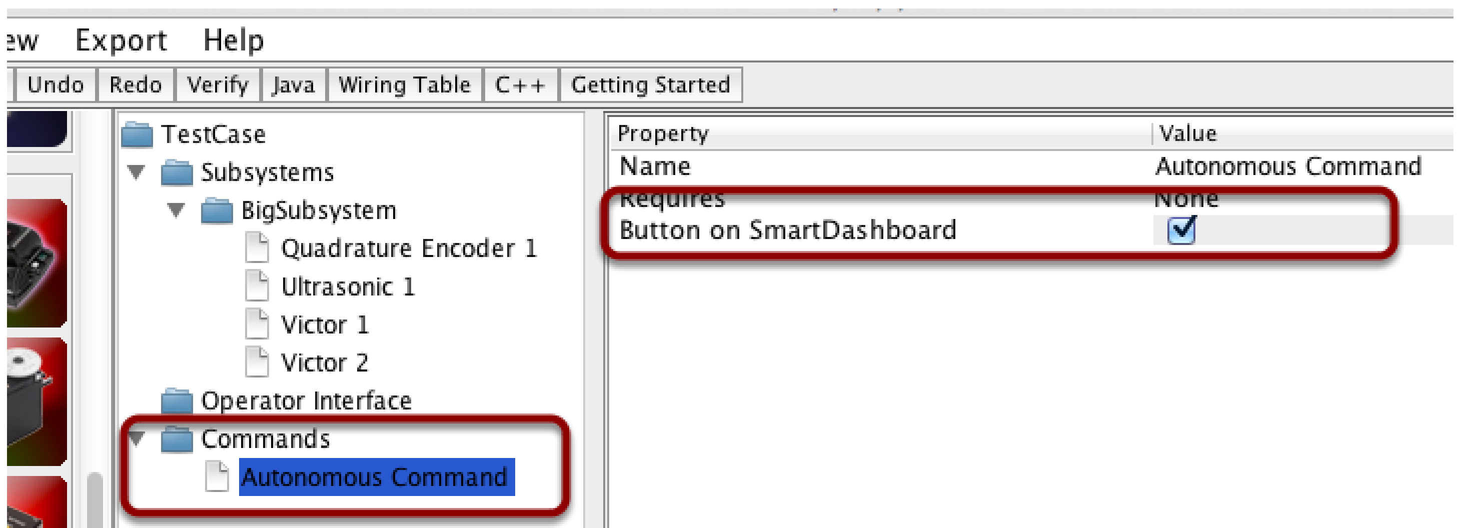
Once one more commands are developed, they can be tested by adding them to the SmartDashboard. It's easy to add commands, simply call SmartDashboard.PutData() with the command name that should be displayed and the command object itself.

Test commands indivudually using the SmartDashboard



When a command is added to the SmartDashboard a button is created on the screen that will, when pressed, schedule the command to run. By doing this you can easily test each command individually to make sure it works. This is a key principle in programming, that is unit test each part of the larger program separately, then you'll have confidence that putting the units (commands) together into a larger program will also work.

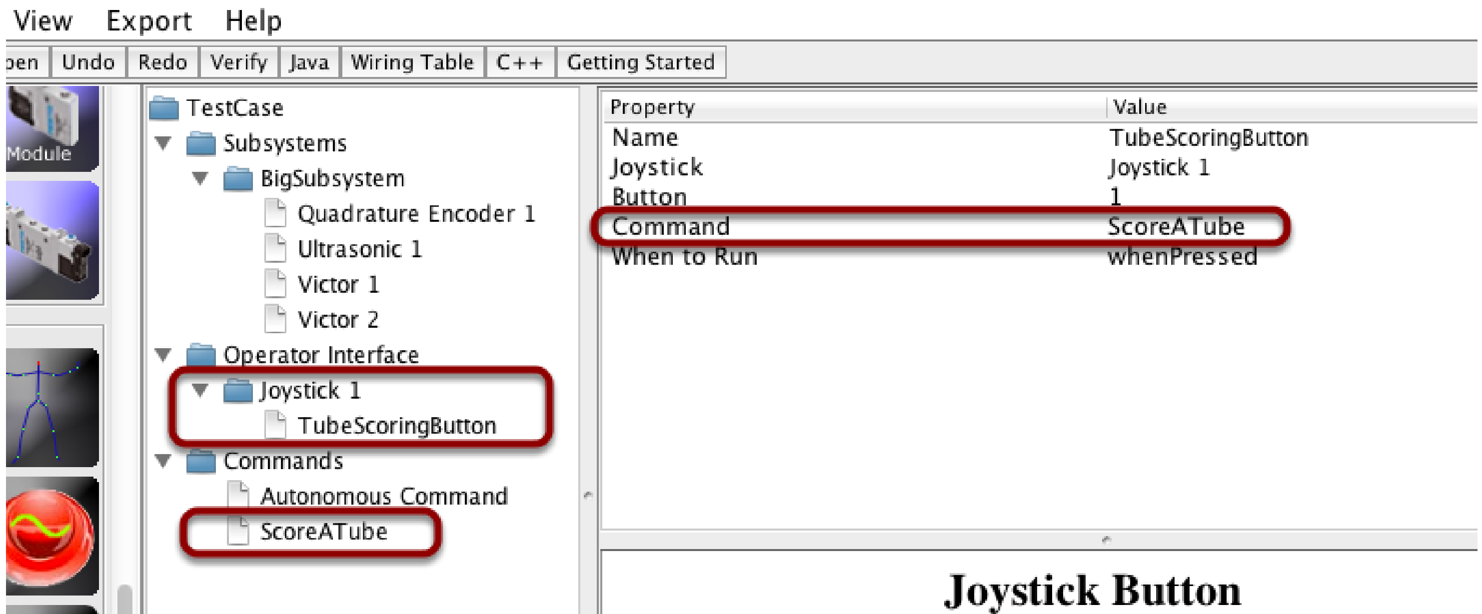
Adding commands to the SmartDashboard using RobotBuilder



The idea of adding commands to testing pieces of your robot program individually is so useful that a checkbox was added to RobotBuilder to automatically generate SmartDashboard buttons. Just

check the box for any command and it will automatically appear as a button that, when pressed, will run the command.

Test commands indivudually using a joystick button



In addition to testing commands using the SmartDashboard, you can also "wire" commands to joystick buttons. This is another easy way to verify that commands work as you expect. And, in fact, in your finished robot program, you will likely have a number of commands wired to many of the joystick buttons as a way of controlling the robot.

In this example there is a command called ScoreATube that presumably places a scoring object somewhere. Then there is a JoystickButton object called TubeScoringButton that has as its command the ScoreATube command. When the button is pressed, the command will run.

Combine simple commands into Command Groups

```
DeliverSoda::DeliverSoda() {
    AddSequential(new PreparePickup());
    AddSequential(new Pickup());
    AddSequential(new Place());
    AddSequential(new RasieWrist());
    AddSequential(new CloseClaw());
}
```

Once individual commands are working, they can be combined into Command Groups. Command Groups are commands that each consist of a list of commands to run when scheduled. This is where much of the power of the system comes into play. Suppose the robot needs to score an inner tube by driving forward for some distance, spinning rollers to release the tube for 1 second, moving a wrist joint, and backing up the robot. If there are commands for each of those operations, the can be individually tested, then a command group just lists all the individual commands, and the group is tied to a button. When the button is pressed, all the individual commands in the group run sequentially. Commands can even be scheduled to run in parallel for operations that can happen simultaneously.

Add robot status to the SmartDashboard

Often you want to see the status of the robot by displaying values on the driver station. In the past this was handled with adding lots of print statements to the program and values poured out of the program, usually so fast that it was hard to know what was going on. With the SmartDashboard, you can create a display just by sending values, and they will automatically be displayed in the graphical user interface. In fact, you can change the format of the fields to more appropriate displays. For example, for a gyro, its heading can be displayed as a compass.

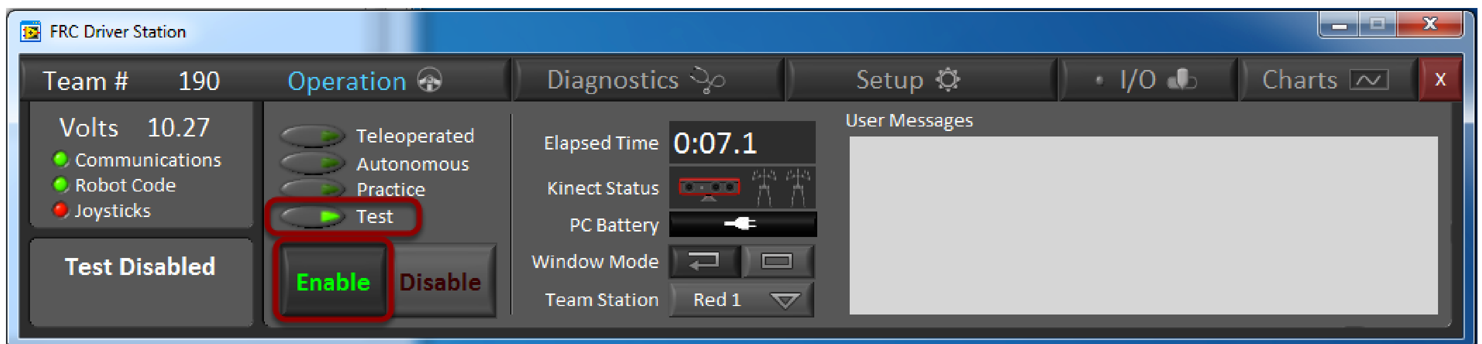
In addition display "widgets" or graphical elements can be created as standard Java code and added to the dashboard to make custom dashboards easily by programmers. For more detail see the section on the SmartDashboard.

Using test mode

It is important to verify the operation of the robot before using it for a match, demonstration or other task. So often wires can be pulled lose, circuit breakers are left out, or other issues that might cause the robot to not operate as expected. There are a number of strategies for testing your robot code. You might have a checklist where each subsystem is manually verified using the operator controls. You can also test the autonomous operation this way when the robot is off the ground can the operation can be observed.

Test mode is designed to enable programmers to have a place to put code to verify that all systems on the robot are functioning. In each of the robot program templates there is a place to add test code to the robot. If you use the IterativeRobot template, or use command-based programming you can easily see the operation of all the motors and sensors through the LiveWindow.

Enabling Test mode



Test mode on the robot can be enabled from the Driver Station just like autonomous or teleop. When in test mode, the test() methods run. To enable test mode in the Driver Station, select the "Test" button and enable the robot. The test mode code will then run.

Debugging and testing robot programs

Adding Test mode code to your SimpleRobot template

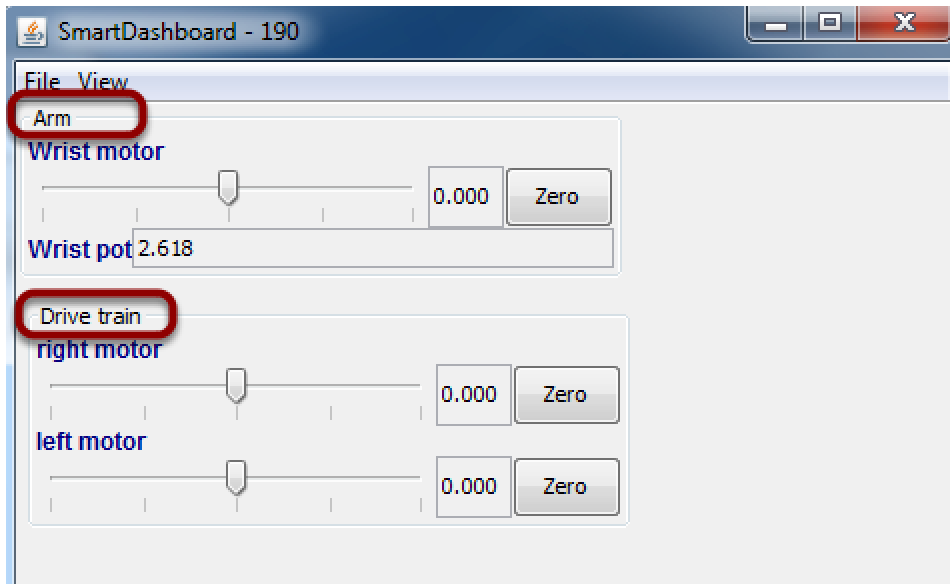
```java
public class RobotTemplate extends SimpleRobot {

    Jaguar leftMotor;
    Jaguar rightMotor;
    Jaguar wristMotor;
    AnalogChannel wristPotentiometer;

    public void robotInit() {
        leftMotor = new Jaguar(1);
        rightMotor = new Jaguar(2);
        wristMotor = new Jaguar(3);
        wristPotentiometer = new AnalogChannel(1);
        LiveWindow.addActuator("Drive train", "left motor", leftMotor);
        LiveWindow.addActuator("Drive train", "right motor", rightMotor);
        LiveWindow.addActuator("Arm", "Wrist motor", wristMotor);
        LiveWindow.addActuator("Arm", "Wrist pot", wristPotentiometer);
    }

    public void autonomous() {
        System.out.println("in autonomous");
    }

    public void operatorControl() {
        System.out.println("In operatorControl");
    }

    public void test() {
        while (isTest() && isEnabled()) {
            LiveWindow.run();
            Timer.delay(0.1);
        }
    }
}
```

The SimpleRobot template is the easiest way to get started with robot programming, but it is somewhat limited when you try to add more features to the program. To add test code to a SimpleRobot based program add a method called test() (or Test() in C++). The test method will be called by the SimpleRobot base class when the robot is put into test mode from the Driver Station. Here you can add code to automatically test sensors and motors or send LiveWindow data to the SmartDashboard. In this Java example, we are:

1. Adding all the sensors and actuators for the robot to the LiveWindow so that they can be displayed or manipulated
2. Calling LiveWindow.run() periodically when in test mode to cause the values to update.

Additional robot test code can be added in the test method as well. For example, you might want to run some motors and verify that encoders are operating. Code could be easily written to slowly turn a motor and check the encoder values. This would verify that the motor and the encoder are both working properly.

Test mode in the SmartDashboard



The above sample code produces the following output when the Driver Station is put into Test mode then enabled. YOu can operate the motors by moving the sliders and read the values of sensors such as the wrist potentiometer.

Notice that the values are grouped by the subsystem names to group related actuators and sensors for easy testing. The subsystem names are suppled in the AddActuator() and AddSensor() method calls as shown in the code examples. This grouping, while not required, makes it much easier to test one subsystem at a time and have all the values next to each other on the screen.

Debugging and testing robot programs

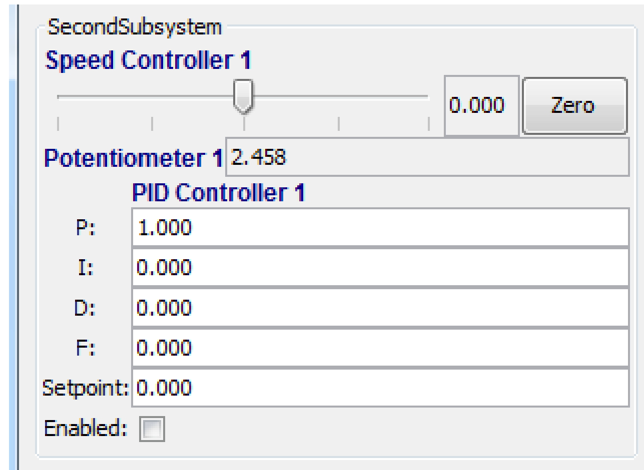
Using Test mode with the IterativeRobot template

```
public class RobotTemplate extends IterativeRobot {

    Command autonomousCommand;

    public void robotInit() {

        autonomousCommand = new ExampleCommand();

        LiveWindow.addActuator("Arm", "ArmMotor", new Jaguar(1));
        LiveWindow.addSensor("Arm", "Arm angle", new AnalogChannel(1));
        LiveWindow.addActuator("Drive base", "Left motor", new Jaguar(2));
        LiveWindow.addActuator("Drive base", "Right motor", new Jaguar(3));
    }

    public void autonomousInit() {
        autonomousCommand.start();
    }

    public void autonomousPeriodic() {
        Scheduler.getInstance().run();
    }

    public void teleopInit() {
        autonomousCommand.cancel();
    }

    public void teleopPeriodic() {
        Scheduler.getInstance().run();
    }

    public void testInit() {
    }

    public void testPeriodic() {
        LiveWindow.run();
    }
}
```

The IterativeRobot template lends itself quite nicely to testing since it will periodically call the test() method (or Test() in C++) in your robot program. The test() method will be called with each DriverStation update, about every 20ms, and it is a good place to do whatever testing commands or LiveWindow updates are desired for testing. The LiveWindow updating is built into the IterativeRobot template so there is very little that is necessary to do in the program to get LiveWindow updates. Note: this works even if you are using the IterativeRobot template and not doing Command-based programming.

In this example the sensors are registered with the LiveWindow and during the testPeriodic method, simply update all the values by calling the LiveWindow run() method. If you're program is causing too much network traffic you can call the run method less frequently by, for example, only calling it every 5 updates for a 100ms update rate.

PID Tuning in Test mode



Tuning PID loops is often a challenging prospect with frequent recompiles of the program to get the correct values. When using the command based programming model, writing PID subsystems to the robot displays and allows adjustment of PID constants with immediate feedback of the results. See more information about PID Tuning using the SmartDashboard in this section from the SmartDashboard manual.