How To Code an Awesome Robot (in LabVIEW)

So you're new to the team this year? New to LabVIEW? New to programming in general? Well, whatever the case, your team is relying on you to take all their amazing engineering and make it into a honest-to-goodness *robot*. That *drives around* and *wins*. But how? That's what this guide is for. (This guide assumes a basic understanding of LabVIEW and programming concepts.)

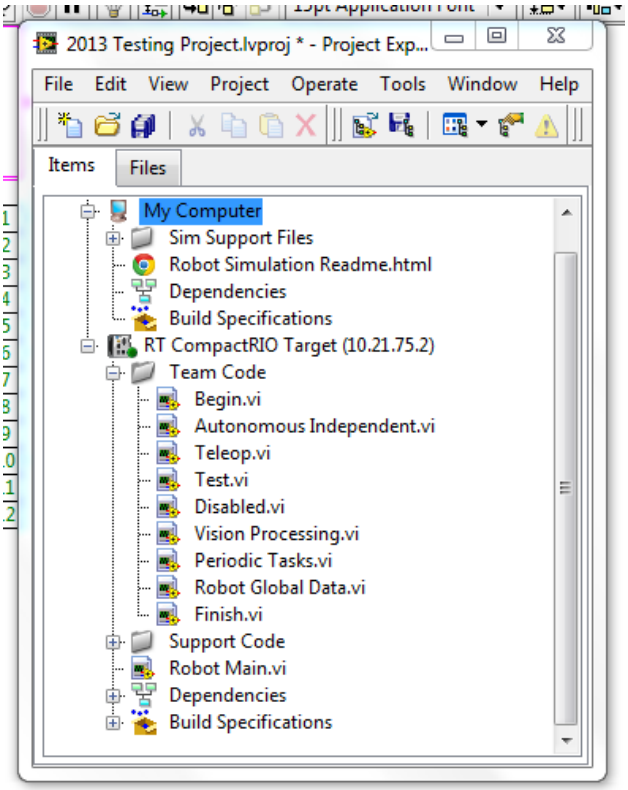
Table of Contents

The Very Basics

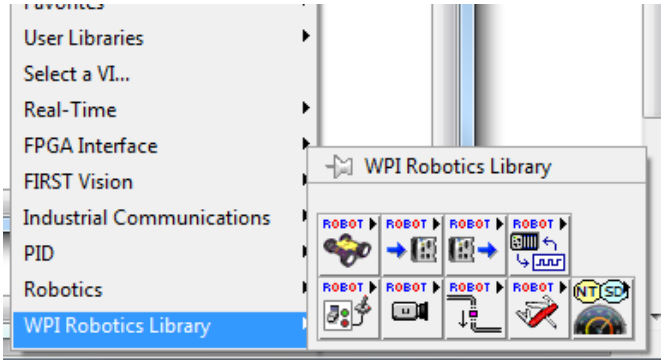
I'm assuming that you have a default project set up at this point. In case you hadn't found it yet, all the important stuff is hidden underneath the RT CompactRIO Target menu. Expand that.

The only VI you see there, Robot Main.vi, is the main program you'll deploy to the robot. It references all the other VI's you'll be modifying.

Expand Team Code. It contains all the good stuff – autonomous, teleop, and the others. "But what are all the others?" you might be saying. You usually only have to worry about three others – Begin.vi, Periodic Tasks.vi, and Robot Global Data.vi. Begin.vi is where you set the references to all the physical stuff on your robot. Periodic Tasks.vi has loops that run at various speeds; you can use it to run background tasks that should always be running. Robot Global Data.vi is where you set your global variables (which can be accessed from anywhere in your project.)



So that's where you'll find all the VI's you'll be working with. All the FRC-specific LabVIEW blocks are in the palette labeled WPI Robotics Library. It's at the bottom of the list of palettes when you right-click on the block diagram.

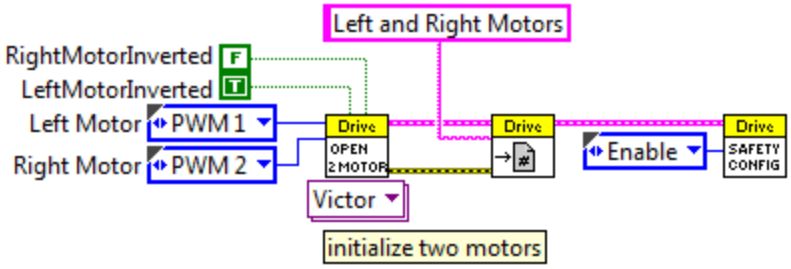


Start in Begin.vi. You can work on Teleop and Autonomous after you get Begin set up.

Begin.vi

Drive Configurations

Your robot can probably drive around with the default code provided in Begin.vi, although you may have to tweak it a bit.



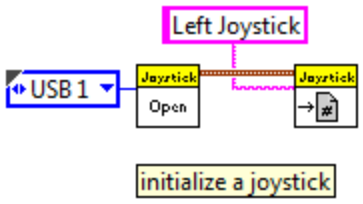
Their default setup assumes that you are driving your robot with two motors. You can replace that block with the Open 4 Motor block from the RobotDrive palette if you are using four motors to drive (but you probably are not.) (NOTE: If your robot has two motors on each side acting together [e.g. in a gearbox], then you should stick to 2 motor. You can use one PWM output for both motors.)

Make sure that you change the dropdown to the type of motor controller you are using. You can ask the people who built the robot what kind of controllers they installed.

The string of text (“Left and Right Motors”) is what you will use to reference the drive configuration elsewhere in the code. You can make it whatever you want, or leave it as is.

Joysticks

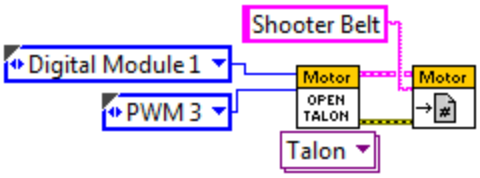
Super easy. Just copy and paste what they have if you want to add another joystick or gamepad (but be sure to change the USB input.)



And in case you're wondering, the USB order is defined in the Setup tab of the Driver Station program. It has nothing to do with the actual USB ports on your PC.

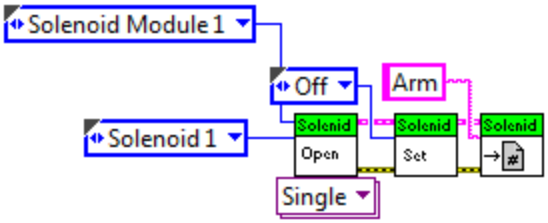
Other Motors (Shooter, etc.)

About as easy as adding a joystick. These blocks are underneath Actuators → MotorControl. If your motor is not controlled with a Talon, switch the dropdown to the appropriate controller.



Pneumatic Cylinders

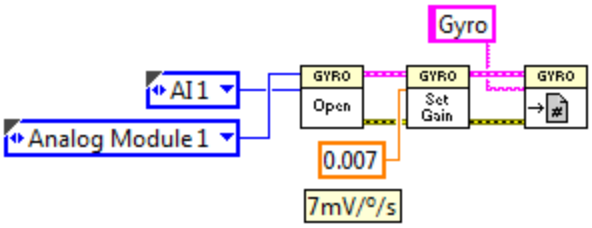
You'll need to find out if the cylinder is the robot is single- or double-action. If it's double-action, you'll need to switch the dropdown to Double and put in two Solenoid #'s. (Again, ask whoever built the robot.) These blocks are underneath Actuators → Solenoid.



The other important distinction is this: single solenoids use the On and Off commands to determine whether or not they are extended. Double solenoids use Forward and Reverse. Make sure you use the appropriate commands.

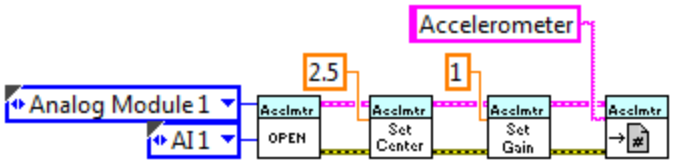
Gyro

Are you sensing a pattern yet? The gyro blocks are under Sensors → Gyro. You may have to tweak the Gain constant based on your model of gyro.



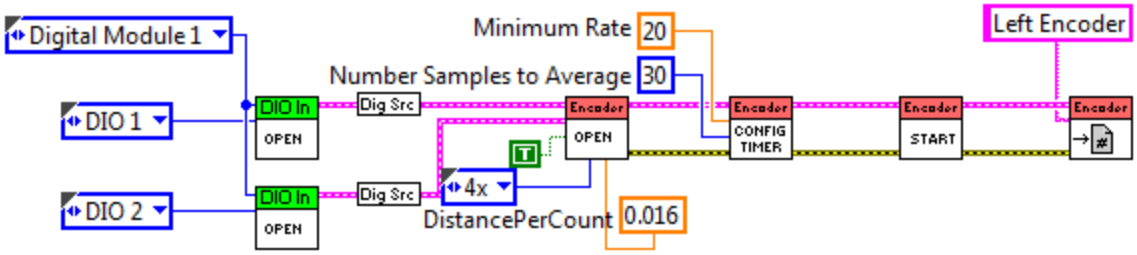
Accelerometer

The Center and Gain constants may have to be tweaked. Also, this is only for one axis of the accelerometer, so if you have to get another axis you will have to add it as another accelerometer.



Encoder

Uh oh. Brace yourself.



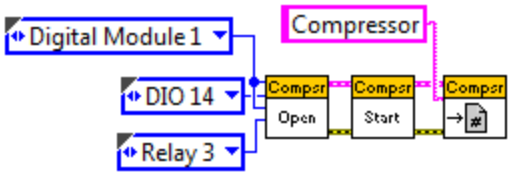
The DIO blocks are under IO → DigitalInput. The rest are under Sensors → Encoder.

The Minimum Rate constant is the threshold where, when the encoder value is less than that constant, the encoder will consider the robot stopped. Number of samples to average can help reduce signal noise, if you increase this number.

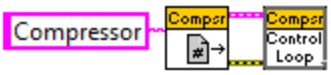
DistancePerCount is a very important number – it specifies how far (in a unit you choose) the robot has traveled for every “tick” the encoder measures. You can get this value using a simple formula: (Wheel Diameter * π)/Ticks Per Rotation. So, if you had 6-inch wheels and your encoder ticked 64 times per rotation, your value should be 0.2945.

Compressor

Easy again. Whew. It’s under Actuators → Compressor.



That's not all though! You also need to go into Periodic Tasks and put this code in:



That will start up the loop that actually makes the compressor run when you are low on air. It's kind of necessary. Those blocks may already be in your code, but disabled – to enable them, right-click the frame that says "Disabled" and click "Remove Diagram Disable Structure".

Teleop.vi

All the references are set, so now it's time to make them do stuff! We'll start with driving. It's a nice place to start.

Different Kinds of Driving

I should start by saying this: there are lots of different ways to make your robot drive. The main two you'll see in the code are Arcade Drive and Tank Drive. The difference between the two is how they handle joystick inputs.

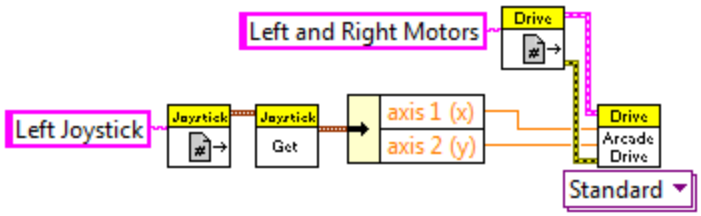
In Arcade Drive, one joystick axis controls the forward/back speed of the robot (the "gas pedal") and another axis controls the left/right spin speed (the "steering wheel"). To drive forward, you just push forward on the joystick. To spin in place, you just push the joystick to the left or right. If you want to drive forward and to the right, you push forward on the joystick and a little bit to the right. You get the idea.

In Tank Drive, you need two joysticks. One joystick's vertical axis controls the speed of the left wheels, while the other joystick's vertical axis controls the speed of the right wheels. (Think of each joystick as controlling the treads of a tank.) To drive forward, you push both joysticks forward. To spin in place, you push the joysticks in opposite directions. This may be a little weird at first, but it becomes easier with practice.

Keep in mind that you can rearrange the inputs however you want – our driver's preferred control scheme is to use two joysticks with arcade drive: the left joystick controls forward-back motion, while the right joystick controls the steering. It's really up to you.

Arcade Drive

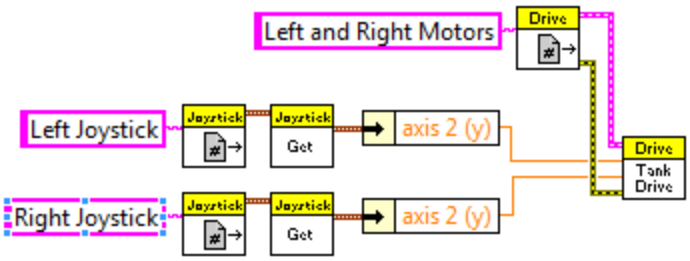
The explanation was the hard part. This is how you set it up:



Ta-da! Now your robot can drive.

Tank Drive

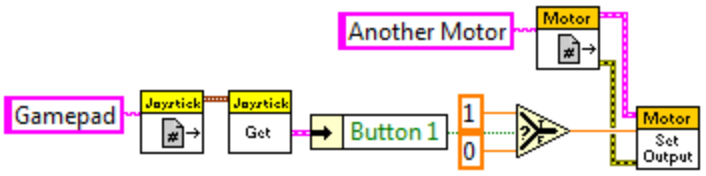
Much the same, but you just need to grab two joysticks instead of one:



Make sure that you're looking at the Y axes of the joysticks though! It doesn't work very well to push the joysticks to the side.

Everything That's Not Driving

Allow me to introduce you to Mr. Select Block:

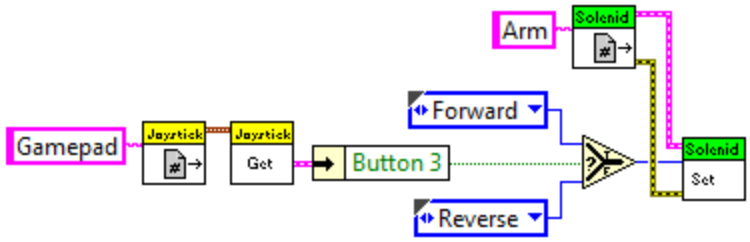


That block with the ?/T/F is the Select block. It feeds through the top (T) value if the input is True, and the bottom (F) value if the input is False.

All joystick buttons read as Boolean (True/False) signals – true if pressed, false if not. The code up above would send a signal of 1 (maximum) to the motor if the button was pressed; 0 if the button was not. Those two values can be set to whatever you want,

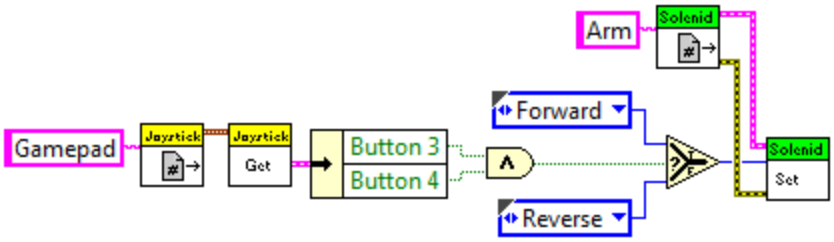
depending on your setup.

This approach works for other stuff on the robot too! You can use a Select block for basically any button-driven Teleop task.



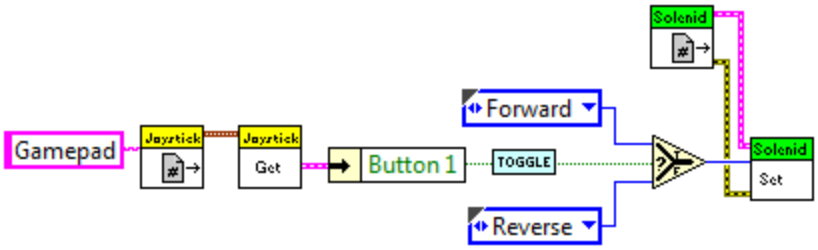
What About Two Buttons?

AND or OR or XOR or whatever else you want. Observe! All logic blocks are in the Boolean section of the Programming palette.



What About Toggling?

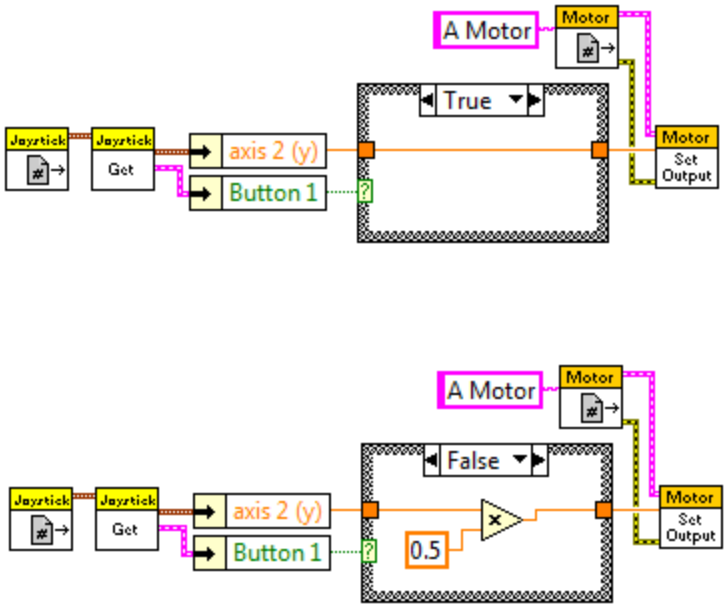
Sometimes you want to press a button to turn something on, then press it again later to turn something off. Toggling isn't built into the joystick VI's, but our team wrote a reeeeaallly simple vi that gives any boolean input a toggling action. You can download it here.



What About Case Structures?

I've found that about 80% of the time you can use a Select block instead of a case structure and your code will be much clearer and more concise because you can see both

cases at once. On the other hand, certain tasks (like our robot's Precision Mode) work better with a case structure because they make the solution look more linear. It's really up to you.



What About a Sequence of Events?

Coming soon...

Autonomous Independent.vi

You have just stumbled into the realm of the very difficult, rather unreliable, often hilarious Autonomous Mode. Thankfully, FIRST was very nice to the teams this year and provided an awesome solution to make your Autonomous work.

Their Solution (A Good One)

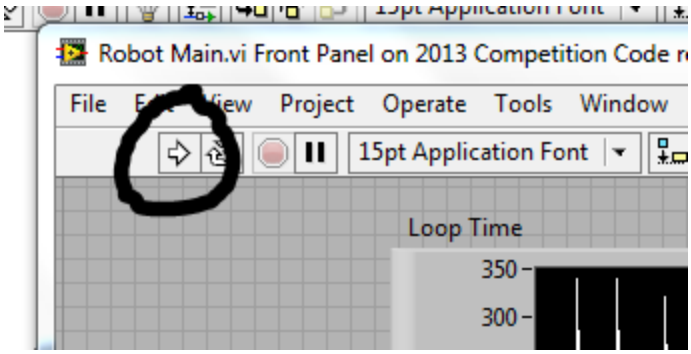
Don't worry! I'll write this eventually.

Deploying Code

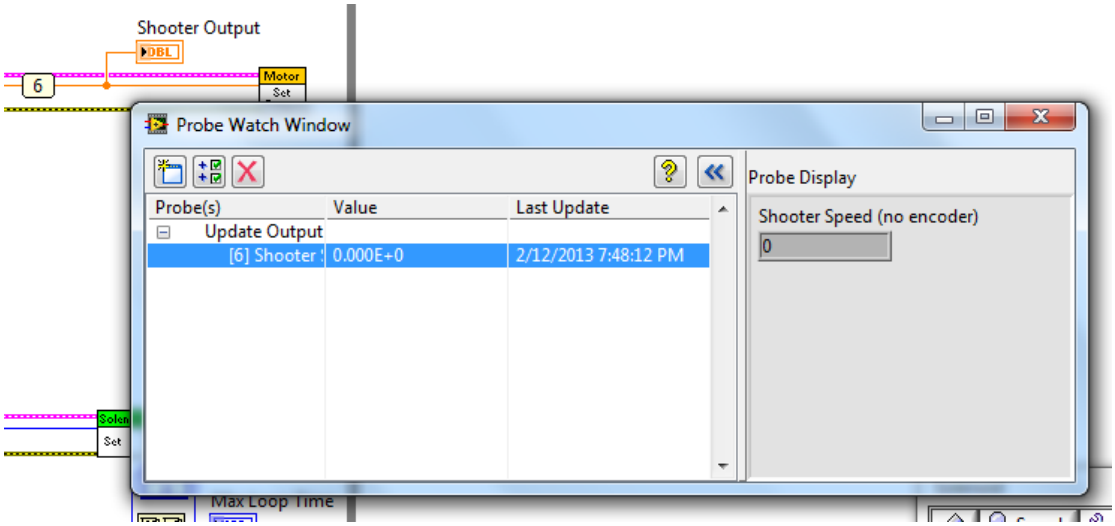
There's no point to writing code if you can't actually make it run on your robot. There are basically two different ways to deploy code, and they're both equally important to understand.

Live Deploy

If you open Robot Main.vi, you'll see an arrow up in the top left of the window. That's the Run button. If you have communications with your robot (see Driver Station below), then you can click Run to deploy code to the robot.



If you deploy code this way, the code will *not* remain on the robot permanently. What's useful about a live deploy, though, is that you can probe the values of all the wires in your code, letting you see exactly what values are being passed through your program. This is very useful for debugging! Just click on any wire when your program is running to probe it.

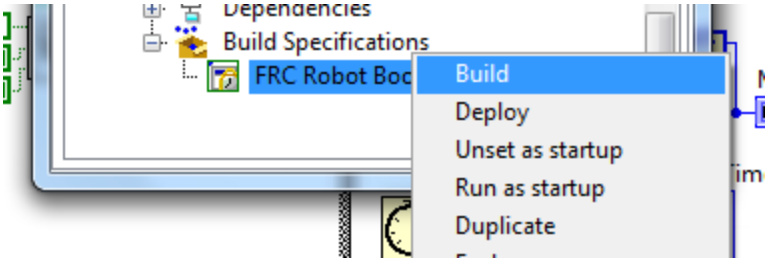


Be warned, though! Deploying this way takes a *very* long time! Deploying can take upwards of five minutes the first time. Thankfully, consecutive deploys go much more quickly.

Run As Startup

This is ESSENTIAL if you want to compete with your code! This method puts the code permanently on the robot, making it the first thing that runs when you turn on the robot. And while you can't probe wires when you run this way, it takes much less time than a live deploy.

Start by scrolling down to the bottom of your Project Explorer window and expand Build Specifications. Right-click FRC Robot Boot-Up Deployment and click Build.



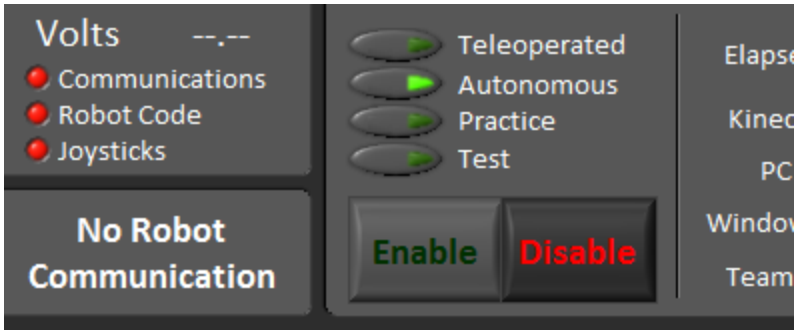
Once the build process is complete, right-click FRC Robot Boot-Up Deployment again and click Run as Startup. Again, you'll need communications with your robot for this step. (By the way, the Deploy button may tempt you, but it's not the option you want to use.) Once the progress bar is done making its merry way across the screen, LabVIEW will ask you if you want to reboot the cRIO. Go ahead and confirm.

Driver Station

The Driver Station program is the program that lets you enable and disable your robot. It also shows you if you have communications with the robot, shows diagnostic info, and lets you configure your joysticks.

Start it up by going to your start menu and searching for "FRC Driver Station". Clicking that should open both the driver station and the Dashboard, which you don't necessarily have to modify (I won't be covering it in this guide.)

Basically, if the three lights along the left side are green, then you can enable your robot! If one of them is red, you'll have to fix the issue before your robot will actually drive.



"Communications" lights up if you have a connection to the robot. "Robot Code" lights up when code is deployed and ready to run on the cRIO. If this is red, you'll probably have to deploy code to make it light up. "Joysticks" lights up if you have joysticks plugged into your computer.