



# LabVIEW Introduction

---

*By: Kevin Phan*

*This handout will cover the basics of LabVIEW, tips for beginning programmers in LabView, ideas of how to use LabVIEW effectively and examples of code that your team could implement for the FRC season.*

1/4/2014

---

# LabVIEW Introduction

By: Kevin Phan

---

## Table of Contents

Introductions.....	(2-3)
Installation and Set-up.....	(4-6)
The Vis.....	(6-9)
The Basics.....	(9-12)
Controls.....	(12-17)
Sensors.....	(18-20)
Dashboard.....	21
Sub Vis.....	22
Conclusion.....	23

## **What is LabVIEW?**

LabVIEW is a graphical programming language, used throughout the technical industry and is widely used in applications in robotics, automation and data acquisition. An example of this would be at the Boeing facility in Ridley Park where Boeing uses LabVIEW to help them gather data from the tests they perform on the aircrafts they produce for their clients.

## **Why should I learn LabVIEW?**

LabVIEW nowadays are expanding its reach and has a strong influence on the industry today. National Instruments have expanded their hardware and software to do more than data acquisition, which was one of its focus points, and now have created software that can be used to operate machines, robots and space shuttles.

Many colleges and universities use LabVIEW to perform their research and applications, such as Drexel University which used LabVIEW to code HUBO, a robot created in South Korea in response to DARPA's robotics challenge.

One such example would be SpaceX's space shuttle, Dragon, which is unmanned space shuttle with the purpose of restocking the international space station with supplies.

With today's world becoming more technologically advance every day, it's imperative now for students, professionals, professors, teachers and the average person to become more aware of such advances.

## How does it work?

LabVIEW works by using **Vis**, also known as virtual instruments, that are graphical representation of code. LabVIEW employs a method known as **Data flow programming** to run Vis in conjunctions with one another.

Data flow programming means that the Vis work by having data “wired” to entered into or from them, that come or connect to other Vis, constants, an array, clusters, etc. Furthermore, Data flow programming means that LabVIEW will work by seeing the code from the top left down to the bottom right. This is something you shouldn’t worry about while programming, since everything will be running at the same time on a FRC level.

LabVIEW consists of two places where code can be seen., the **Front panel** and the **Block Diagram**. The Front panel is the where you can have your code display visual feedback; via indicators, graphs, images and much more. The Block Diagram is where you, as the programmer, will create your code in. As mentioned earlier, your code will be in the form of Vis that appear as blocks in the Block Diagram, hence the name.

## Installation of LabVIEW

For those of you have issues installing LabVIEW on your computer, this handout will cover over how to correctly install LabVIEW and guide you over the starting menu. This guide will be only for windows 7 users and also for the 2013-2014 LabVIEW FRC version. Please check on the FIRST national instruments website to get any updated changes to LabVIEW installation or major changes. The website address is:

[https://decibel.ni.com/content/community/academic/student\\_competitions/frc](https://decibel.ni.com/content/community/academic/student_competitions/frc)

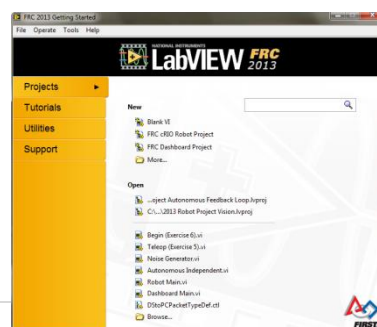
On the website there should be a link to download the 2014 software, use the national instruments downloader unless you want to download the files directly to your computer then use the direct download method. After you have downloaded it and follow the installation instructions, there are updates that you must download and install for the FRC season.

More documentation of how to install is on the national instruments website. If you need more assistance don't hesitate to email me at [team357royalassaultlabviewhelp@gmail.com](mailto:team357royalassaultlabviewhelp@gmail.com). There will be further instructions on the team 357's website: [http://team357.org/RA\\_iWeb/index.html](http://team357.org/RA_iWeb/index.html)

## **The Getting Started Window**

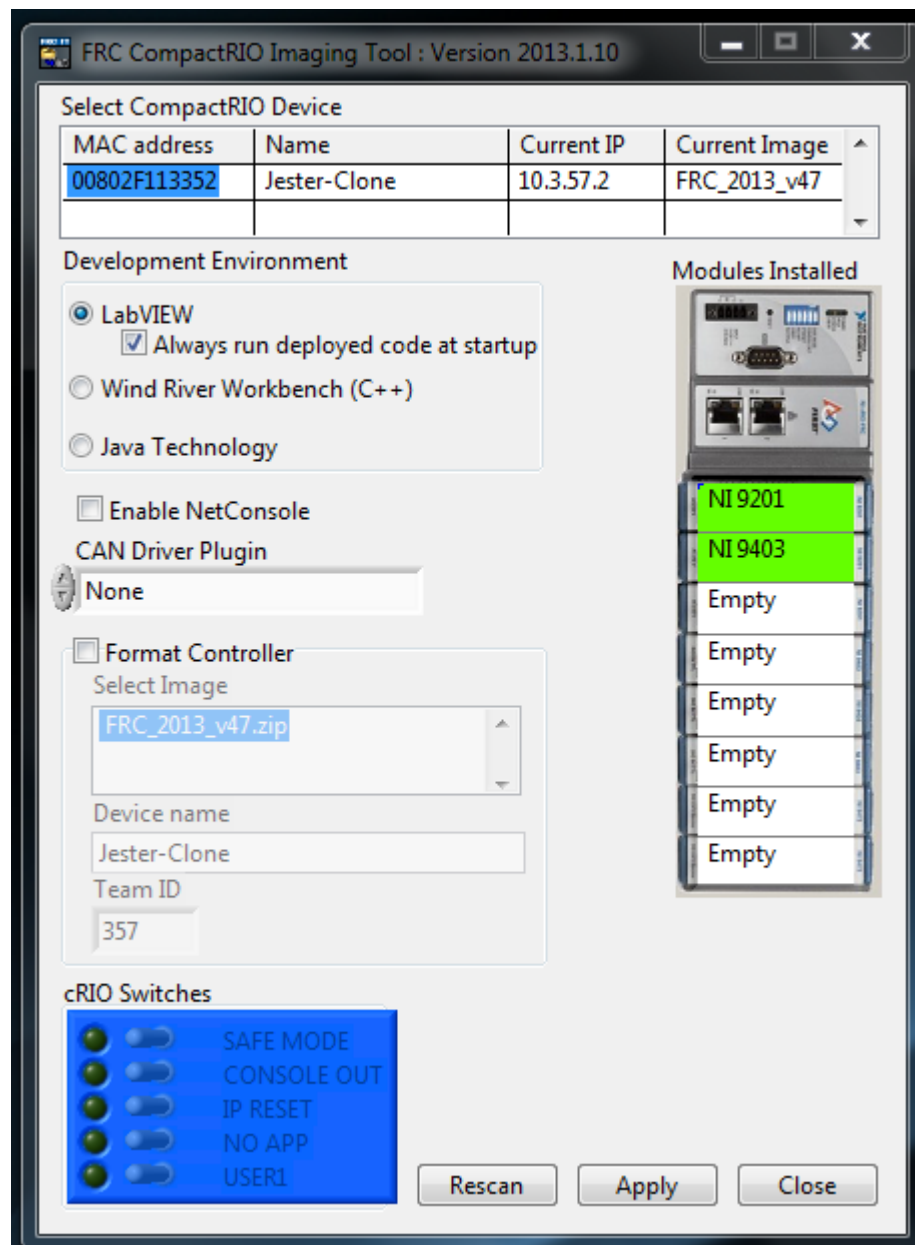
This handout will use the 2013 season Ultimate Ascent version of LabVIEW to show the functionality the window has. National instruments has never made major changes to the window, but if there are then more complete documentation of how to use it will be again on team 357's website. (Link above) The Getting started window is simple and quick to use to get working. On the projects tab you will find your recent projects and the ability to create either a robot or dashboard project. On the tutorials tab you find examples that can help you get on your feet quickly to develop the robot's program and get an understanding of how to use LabVIEW's features. The utilities tab will let you access the various other tools provided by national instruments. These tools are the cRIO reimaging tool, setting up the axis camera tool and BDC-Comm. The cRIO reimaging tool is vital to setting up your robot to compete.

When using the cRIO reimaging tool there some things you want to keep in mind. You want to make sure that your cRIO is being formatted for LabVIEW if you're using LabVIEW to develop your robot's program. That you have the current up to date version of the cRIO image and your cRIO ip address is set to your team name. The cRIO ip address will look like this 10.xx.yy.2. The two xs are for the first two numbers of your team number and the two ys are the last two numbers of your team number. An example is 10.45.75.2. If you are in the triple digits, you want to set up your cRIO ip address like this 10.3.57.2.



# LabVIEW Introduction

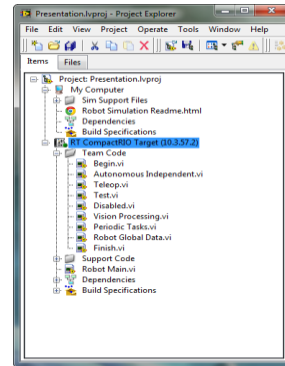
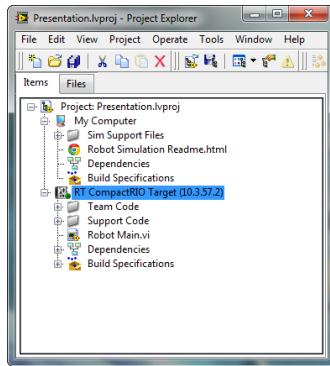
By: Kevin Phan



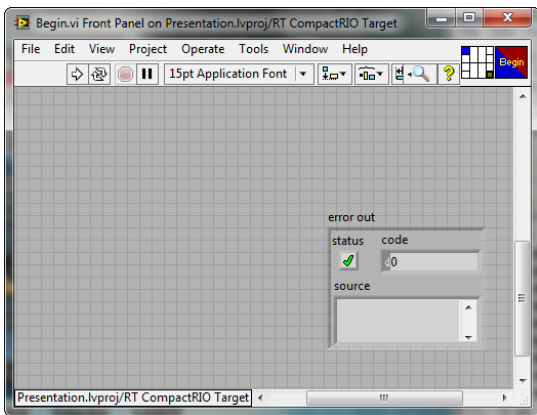
The image above shows how you should reimage your cRIO. The soft switches are what you want off or your cRIO won't be able to process as fast, especially console out because the cRIO will try to output what it is doing and start slowing down the processes.

## Understanding the Project Menu

The project menu is where you can access the various parts of your code. The project menu has a folder for team code and support code and the cRIO. In the team code folder, you can access your tele-op, autonomous, disabled, global variables, etc. From this menu you will be able to download code to your robot. There are instructions later on this handout that goes over how to perform said operation. The project menu is pretty much where you go to access your code.



## The Begin Vi



The Begin Vi is where you initialize everything on your robots which are your motors and sensors. In the WPI library there are Vis that are called open for the different sensors and motors. (E.g. An ultrasonic sensor has an open VI that is different from an infrared sensor's open Vi.) In the Begin Vi you can declare all of your sensors and motors with a Vi called set refnum. This Vi will allow you to call on it through the various parts of

your program. The Begin Vi is an important part of your program to start everything on your robot and the program executes this first before anything this in your code. Examples of how the Begin Vi works will be later on in this handout.

## **The Tele-op Vi**

The Tele-op VI is where your controls for your robot during tele-op begins. Here is where you put your joystick controls for your robot. The joystick can send a value from -1 to 1 and you can wire that to your motors that you want to run. You could use a case statement wired to a button from the joystick that sends a constant value to the motor.

There are many ways to manipulate the values of your controls, but for the time being we will cover the basic controls. The tele-op Vi will need to have get Vis that will allow you to then to write the controls for the robot.

## **The Autonomous Vi**

The Autonomous Vi is where the robot runs on its own without having a human operator controlling it. When trying to develop your autonomous vi use a flat sequential structure to make your robot perform your code in the right order you want it to. Since, autonomous is somewhat predictable in where your robot will be and the opposing robots are in spots you know, it is okay for you to code a linear autonomous.

This does not mean to not develop multiple autonomous modes, because although you know where you perform the best, doesn't mean that you will always be able to have the desired location. Therefore, develop multiple modes for your autonomous to make the most out of the time you get. Develop functions that will be able to work in autonomous and tele-op vis.

In autonomous, the robot can receive input from the joystick. This is legal, as long as the joystick is not touched by your human drivers. An example to develop a method to have multiple autonomous routines is having a toggle switch on your driver station that and using its output to control which autonomous to be executed.



## **The Robot Global Vi**

The robot global vi is where the global variables of your program will be located. This is where you would go to set up the global variables. This is only a front panel, but whatever the name of the control or indicator will be the name of the global variable.

Global variables can help immensely when you have the same variables you need to use in tele-op, autonomous, begin, finish, or any vi. Using the global variables will make it quicker to call the variables you need between the vis and sub-vis. When using a constant global variable or when saving your values in the global vi, it is important for you to use the “Make Current Values Default” command to save the values.

## **The Finish Vi**

This vi is for the program to end and your references to close. This is usually forgotten by many teams since the robot is automatically disabled when the match stops. This vi is where you close vi for all of your components of your robot. The finish vi will then move into the disabled vi.

## **The Disable Vi**

This vi is for when the robot is disabled. Here is where you will put functions and tasks what the robot should do in this vi. The tasks can be resetting your robot in its neutral state or a last minute execution of task such as lifting up your robot at the last second. (Please remember though every time at the end of the match this will execute and the robot does not care about its surroundings at this time. Use this vi very carefully.) Usually teams leave these vis empty, because well you do not want the robot to do anything during this time or else it could harm people coming on the field.

## **The Periodic Tasks Vi**

This vi is used for, you guess it, periodic tasks. The periodic tasks vi executes when your robot is enabled and runs alongside of your robots main code. (E.g. Tele-op, Autonomous) This can be used to sample data of your robot without letting it taking up the entire processing power of your cRIO.

This is crucial when you have multiple sensors that you are relying on outputs from. If you were to check their outputs in the tele-op and autonomous vi you will end up slowing your robot's processes down. The reason being is that you have a continuous while loop that is executing your code you have placed in the tele-op and autonomous vi.

## **The Basics**

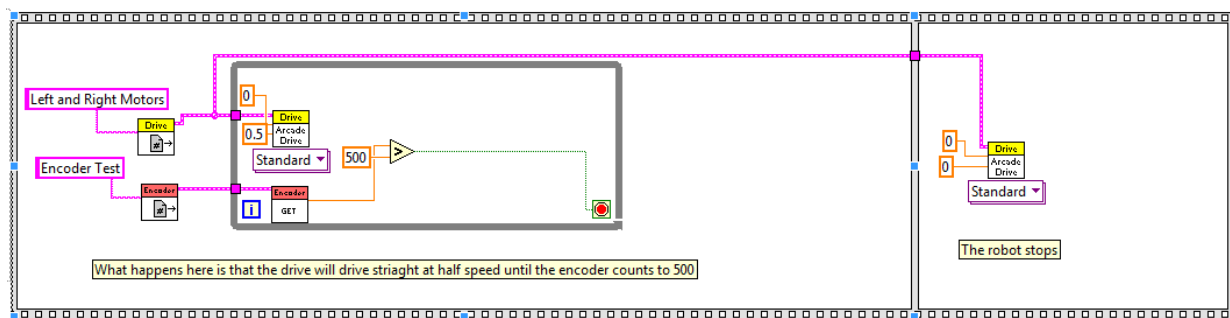
The basics for LabVIEW are essentially figuring out how to use the structures of LabVIEW, general programming practices, understanding the vis of LabVIEW, how to create code in LabView, how to create an executable . The structures you want to get familiar with is with case structures, while loops and flat sequence structures. The case structures are similar to your text base case statements. The case structure on default is set to a true or false Boolean case. The case structure though can adjust itself to take inputs from arrays and use the elements in the array.

The while loop structure is the same as a text base while loop. In LabVIEW though it has an added feature called state machines. State machines are pretty much the ability for a loop to go through steps of a task. Along with state machines are shift registers which are the values of the last loop iteration that you are using again in the loop at the beginning. State machines and shift registers will not be covered in this handout, but there will be resources later on this handout if you want to learn more about them.

# LabVIEW Introduction

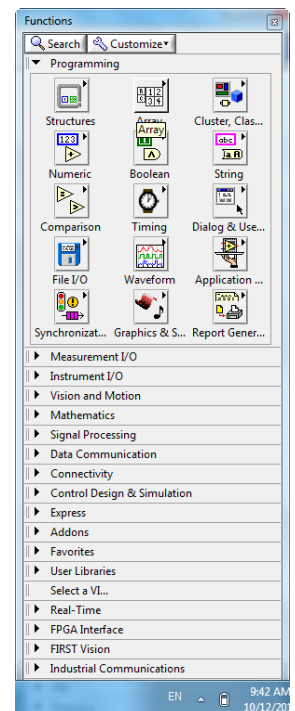
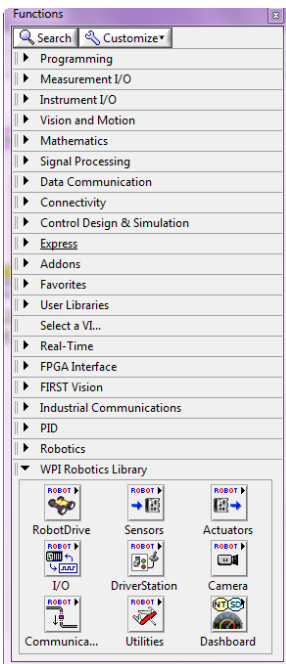
By: Kevin Phan

The flat sequence structure is something close to the linear style of the text vase programming languages. This works by using “frames”. The frames are where you place the code that you want to run during that frame. A frame executes when the program encounters it and the frame will run until it reaches the end of the frame. The frame structure can hold multiple frames that are next to each other and it runs left to right. Here’s an example of a flat sequence structure.



## Understanding the function pallet

LabVIEW can be very overbearing at first for the rookie programmer and people first exploring it. The main tools you will be using to create the program to run your robot can be located in the WPI robotics library and the programming library. In the WPI robotics library you can find all of the major vis to use for your robot. The programming libraries will have the majority of the tools you need to create the program you want. The WPI libraries are the basic materials of setting up the robot program and the programming libraries will be your tool in crafting those materials into something usable.



## **The Robot Main Vi**

The robot main vi is where all the vis are executed by the field. You can see in the block diagram that there is the while loop mention that executes your tele-op and autonomous program. You can also see that the begin, periodic tasks and other vis are outside the loop. The way this is constructed shows that the code that we put in the tele-op and autonomous will continually run until we tell it to stop. This is why we do not need a while loop for the controls of the robot, because there is already a while loop set up for us. If we were to have a while loop doing the same controls within another while loop, you start eating a lot of processing power that your robot really needs. Another thing to know is why the periodic tasks and begin are not in the while loop where the tele-op and autonomous code is written.

The periodic tasks vi is outside the loop, because if it was inside the loop you would not be actually checking the sensors data every specified time that you had set. Instead you start checking it every time since the bigger while loop overrides it and again processing speed is now at crawling speed. The begin vi has a different reason for not being in the while loop where the tele-op and autonomous code is located. The reason is that if the begin vi was in the while loop, you will end up initializing the same motors and sensors over and over again, which will prevent you from actually controlling the robot.

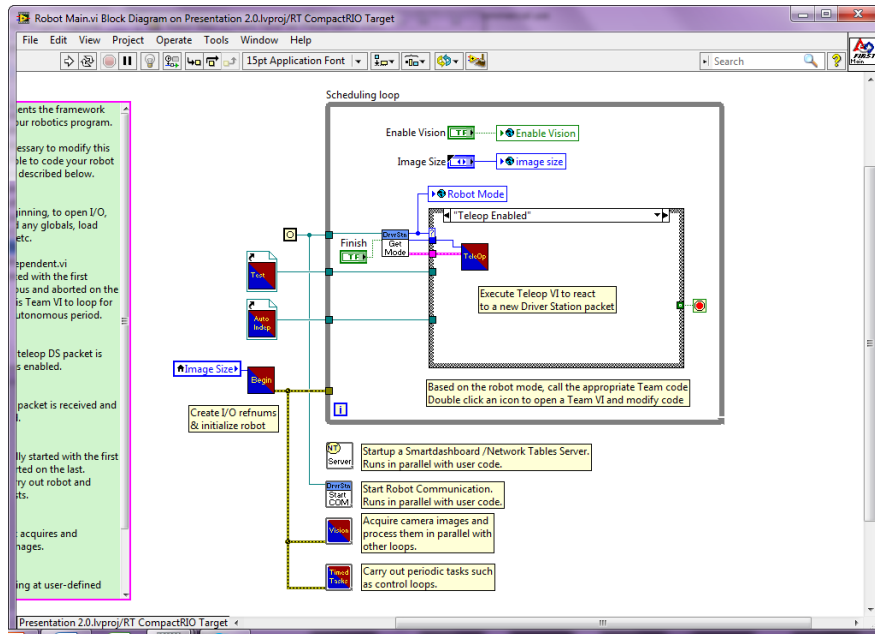
You might have noticed that the autonomous vi is located outside the loop and something else is there. Autonomous can only run once so it can not be in a loop, but if you do that then the robot will start running it after it turns on and initializes the features of the robot. The something else is a vi that sets which vi can be enabled and at what time depending on the field.

There are lot more to discuss about the robot main vi, but in the future there will be documentation located again on the team website and in future seminars. To give you a basic overview through the other vis though, the test vi is for testing

# LabVIEW Introduction

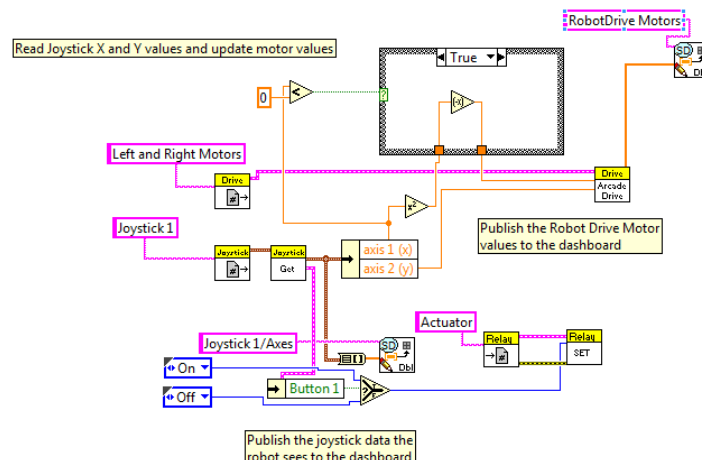
By: Kevin Phan

purposes, but not many teams use this vi, the vision vi is used for vision for on the robot and this makes it convenient to place all of your vision code.



## Simple Controls

Simple controls for FRC consist of press and hold and moving the joystick. This type of control is useful for testing but during competition it could use adjustments to make it just right for the operator and driver. These adjustments can go a long way for rookie teams just getting started and off their feet to create code that can help them during competition. These adjustments can also be useful for teams using joysticks that they really don't want to use but have to due to budget constraints. Here's an example of simple controls:



# LabVIEW Introduction

By: Kevin Phan

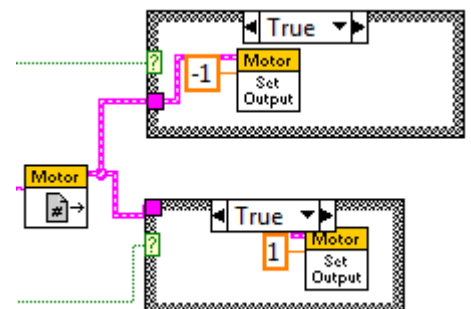
---

As you can see in the picture above, the code for driving has the joystick output being manipulated in the case statement. This is an example of the adjustment that I mentioned. Above in the picture you can see that there is a small vi that holds on and off values. That vi is called an express vi and can simplify Boolean controls when you need a constant change on a button press.

I want to point out a mistake a lot of rookie teams assume with the case structure. The case structure is exactly like a case statement in a linear program. If you try to use two different case statements to perform a task such as moving an arm up or down by button presses, you run into a problem that you probably never encountered before, the other case statement does not work. The only way I can describe what is happening is that LabVIEW sees these two cases statements and depending which one becomes true, LabVIEW will only run that statement. Here's an example of one:

The false cases do not contain any vis, so there should be no issues with operating the robot, but you might want to have a 0 constant assigned to the motors in the false case to prevent any movement of an arm. The way you should create this two button control is to use an array and convert the Boolean array to a number array. This will allow you to have that type of control now, but since it is a case structure, you need to figure out which case number it needs to be in binary. Truth tables are what you're going to use to figure out the case number. To do truth table you first need to figure out how many things are being checked. Once you figure that out you can start creating your truth tables. If you haven't used truth tables before, don't worry this handout will cover over this.

To use as an example, let's use the picture on the right. This picture shows two case structures that are going to have an issue. The issue is that the arm will only run up or down depending which button is pressed. To resolve this issue, we're going to use the case structure with the truth tables. Here's how to set up a truth table.



# LabVIEW Introduction

By: Kevin Phan

---

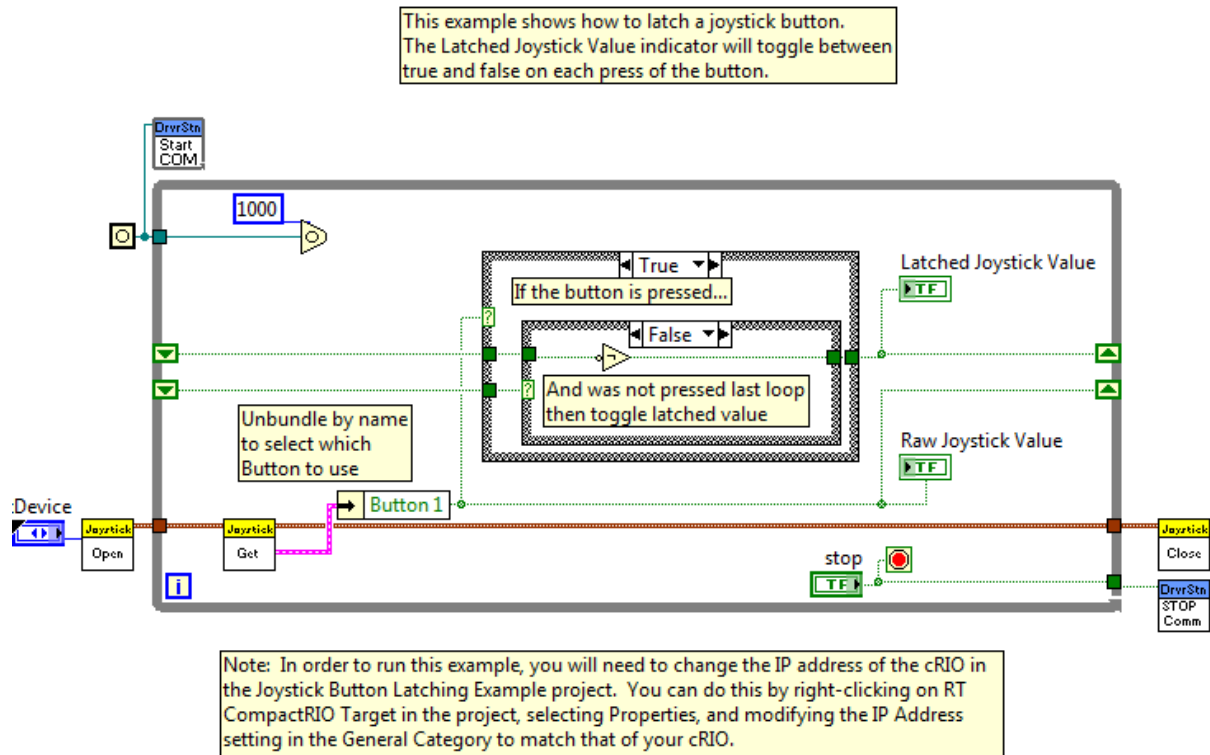
Button 1	Button 2
1	1
1	0
0	1
0	0

These 1's and 0's mean true or false. The truth tables are very useful for this, because this table will help you determine what you want done during each case. To figure out which case number each case is, the number of columns will determine which exponent of the base two will be. For example, the left column is  $2^0$  and the right column is  $2^1$ . When doing binary addition you evaluate the exponent and add the product. It depends on which one is true or false also, if there is a zero then the column is false and does not contribute to the binary addition. For example the second case shows that button 1 is true and button 2 is false. The case number for this will be then. For the next case that number is 2, because the second button is true. That means that only the second button is pressed and the case number evaluated is 2. The top row shows when the both buttons pressed. A case like this is something that you probably don't want to happen and do not want to do anything during this time for the arm. The case number evaluated is 3 because both buttons are pressed and they are both evaluated and added together. You can place a zero constant to the motor output to prevent anything from happening. For the next two statements, you place what constant value you want to assign to the arm. The last case is 0 by default, because both buttons are not pressed which makes it false.

# LabVIEW Introduction

By: Kevin Phan

If you want to create a toggle switch with a press button, LabVIEW has an example that you should look at. Here's the example:



In this picture you can see that the while loop have boxes at the beginning and end of the loop. These boxes are called shift registers. Shift registers have the ability to record the value of the output at the end of the loop and you can use that value in the beginning of the loop for something else. So essentially what it is doing here is that the loop is checking for each loop iteration for the last value of the joystick and what the value of the latch joystick. This control can be useful for toggling pneumatic controls. You do not need the occurrence vi to do this, what you can use if time. You can have the loop execute every 10-50ms, depending on what you prefer, to run this example. You will want to play with this example first though.



## **Pneumatic controls**

If your team is considering pneumatic controls, please make sure that if it is going to be a vital part of your robot that the design prevents any possible puncturing of the tubing to the cylinder. To use the pneumatics you can use either the relay or solenoid vis to operate the pneumatics. The pneumatics are wired to the digital module of the cRIO. They need spikes to wire to the digital module or digital sidecar.

When using pneumatics it is important to know if they are double action or single action. That means if the pneumatic is going to use compress air backwards and/or forwards. It is important to know this, because you only have a finite amount of compressed air on your robot at any given time so it is important to know how many times it is going to activate and design around it.

Keeping this mind when you are initializing your pneumatics, they follow the same rule as anything else. You need to use the open vi and the set vi in begin to get it started. For the compressor it works the same way, but you want to develop a control system for it. The reason being is that when you start the compressor it won't stop until you specify it too. There is one vi that you use to help monitor the pressure in the cylinder and tanks.

This vi is the control loop vi for the compressor. That loop will help dictate when the compressor should stop running. You have other ways to control the compressor though, there are the start and stop vis for the compressor to manually start or stop the compressor. This control vi should be placed in the periodic tasks vi.

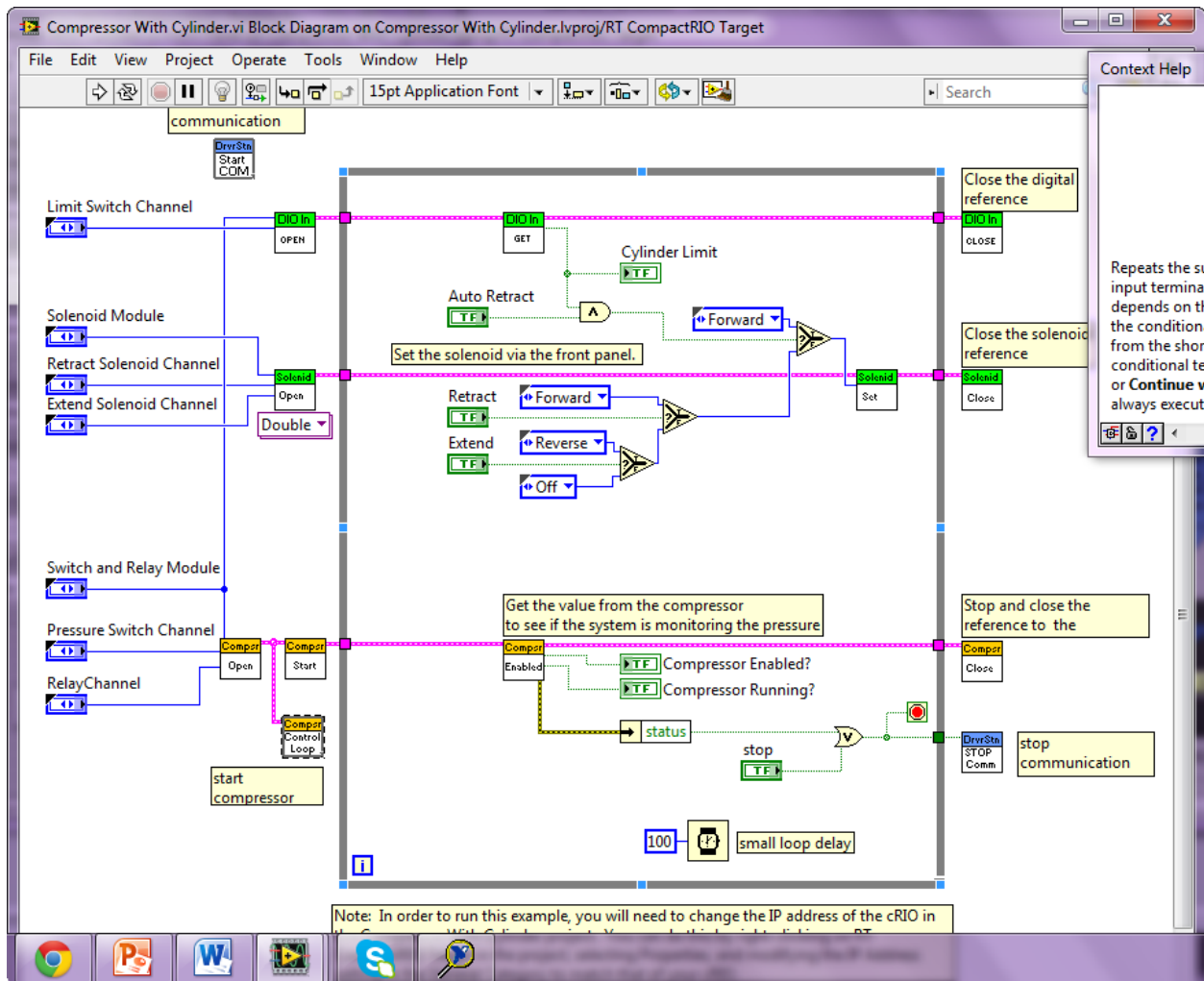
When the robot is disabled, it is very important for you to check the default state of your pneumatic cylinder. Knowing this will help you disable your robot properly. If the game requires hanging and at the last second you need to get your robot a quick hang, but you failed to press the button in time, your program in the disabled vi can have your robot hang at the last second. You just want to be careful

# LabVIEW Introduction

By: Kevin Phan

about where your robot is at during this time or else you might get red carded for damaging an opposing robot.

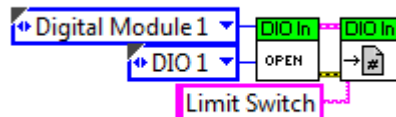
Here's an example of pneumatic controls:



## Sensors

The use of sensors on your robot is vital for your team to do well in competition. The human error is too great to not take it into account. To minimize the human error sensors can help immensely with real-time data that the human can miss. The common sensors are limit switches, gyros, encoders, potentiometers, and light sensors. All of these sensors can work together to get your robot to simplify the tasks the human operator has to do.

The limit switch is a simple Boolean control you can have on the robot to help determine stages of the robot or if the robot has something or anything that involves the sense of touch. The limit switches though are fragile, the metal bar that is attached to the sensor is the part that can take a beating. The placement of this switch has to be at a location where the bar touches the object and is not going to be smashed over and over again. To program a limit switch is not really difficult and only outputs true or false. To set it up you just need the basic open vi and set vi in the begin vi to use it. Here's an example:



The gyro is quite the sensor to have on the robot. A gyro can help you determine which direction your robot should face and can help you stop guessing how much time to spin in autonomous. A gyro is a sensor that your team should take a look at for any competition. A thing to keep in mind about gyros is that gyros will output in voltage and you need to figure out how to interpret that voltage in degrees. Thankfully, that work is already done thanks to the wpi libraries. The get angle vi in the wpi libraries can output both the angular rate and angle the gyro is currently at.

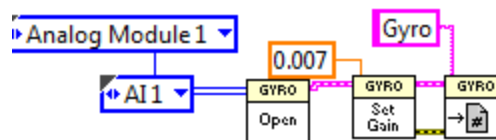
A gyro is set up with the usual open and set vi. There is something else that a gyro can do though. A gyro can be calibrated to a voltage to be its “zero”. This can be done with the open vi and you can designate the amount of time it should calibrate also. You do not necessarily need to do this, unless you feel like it is

# LabVIEW Introduction

By: Kevin Phan

something that your team would like to do. Gyros are wired to the analog module of the cRIO.

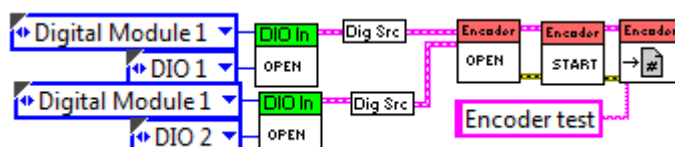
There is one more thing in the begin vi that you must do, and that is to set a gain for the gyro. A gain is the conversation factor from volts to degrees. You want to refer to the first technical manual about details or your product's manual to find out the gain of the gyro. Without the gain your readings can be off. When using gyros it is important to test it regularly and make sure to have the gyro as close to the center of the robot for you to be able to read the angles correctly for your robot. Gyros have some other quirks about them, but this handout will not cover them and will go over basic controls with gyros. Here's an example of setting up a gyro:



The gyro has a vi that resets the gyro to zero. This can be used as a start method of zeroing the gyro at the spot you want it to. An example will be having the gyro rotated away from its designed zero and manually zeroing the gyro using this vi.

The encoders are sensors that are best suited for something that will rotate 360 degrees. The encoders measure these movements by “clicks”. Clicks are the encoders' measurement and they can vary from sensor to sensor. It is important again for you to read the manuals provided for the encoders. The encoders are set up the same way as any other sensor, but they have digital ports that you must wire to the digital sidecar or digital module. The encoders have start, stop and reset vis that you can use to help control your robot. The encoder also has a get vi for you to be able to figure out what direction your moving in and the amount of distance the encoder measures.

Here's an example for setting up your encoder:

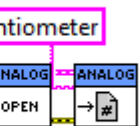


# LabVIEW Introduction

By: Kevin Phan

---

The potentiometer is a fragile sensor to use. The reason for this is the limited amount of degrees that a potentiometer can move. The potentiometer is used for mechanisms that have limited movement. Potentiometers have more precision than the encoders since the potentiometer is made for more precise movement generally. The difference comes down to how many degrees the potentiometer is made for. Accuracy can vary from brand to brand so always refer back to the product manual about it. Another thing to keep in mind about potentiometer is the effect the environment has on it. (E.g. Vibration, debris, temperature, etc.) In general, use potentiometer for more precision and in locations least affected by environment. Here's an example to set up a potentiometer:



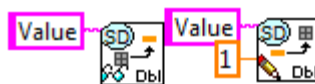
A light sensor can do a couple of things for your robot. It can tell what color it sees and/or a distance between it and the object. This optical sensor can be very useful for competitions that involve light and they can be used to help determine if the object it sees is yours or not. A light sensor can use all ends of the light spectrum and you should find out what your sensor detects. To set a light sensor up, it will vary for different kinds. I will be posting documents about light sensors on the team's website.

There are many more sensors out there for your team to experiment and use and I can't cover them all in this handout without further research. More details and sensors will be posted on the team's website later on for you to look at.

## How to use the dashboard

The FRC dashboard is the only way during matches for your drivers to get data about what is happening out on the field to your robot. The dashboard in the earlier versions of LabVIEW was difficult to set up, but now it has become much simpler. To use the dashboard effectively your team needs to figure out what kind of response from robot they want. The dashboard also has a finite window space since your screen on the classmate or laptop running the driver station is limited. The dashboard can help the robot with its processes by doing the process for the robot, freeing up space. When designing your dashboard those are the key things you want to keep in mind.

To hold output values of your robot you need to make sure that in the robot main that you have the nt server running in parallel. Without this you cannot store any values to be sent to the dashboard. The vis you will be using is located in WPI libraries and in the dashboard section. There are two types of vis for use in the dashboard. To store a value to the dashboard, you want to use the write vi for the dashboard to store it. You need a name attached to it for you to be able to read it on the dashboard. To read the value you need to use the read vi for the dashboard. You can write and read arrays and individual values. Here's an example of both of them:



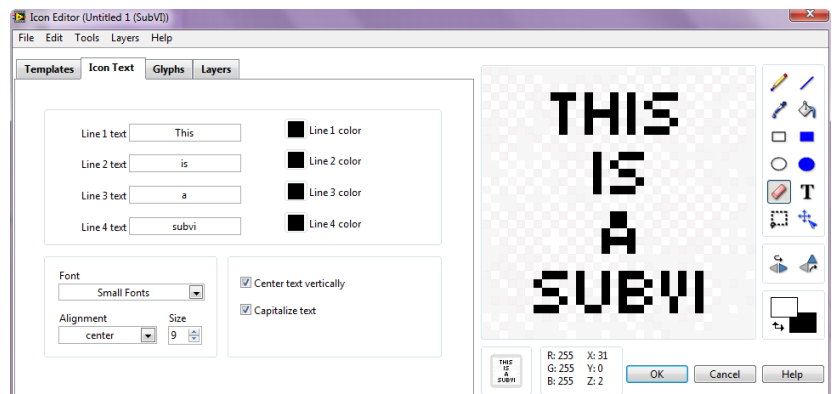
The dashboard can be useful for vision too, but I do not know enough vision code for me to explain to how to set-up vision properly, but it will be covered in the future. Possibly after build season to the next year.

## Creating SubVis

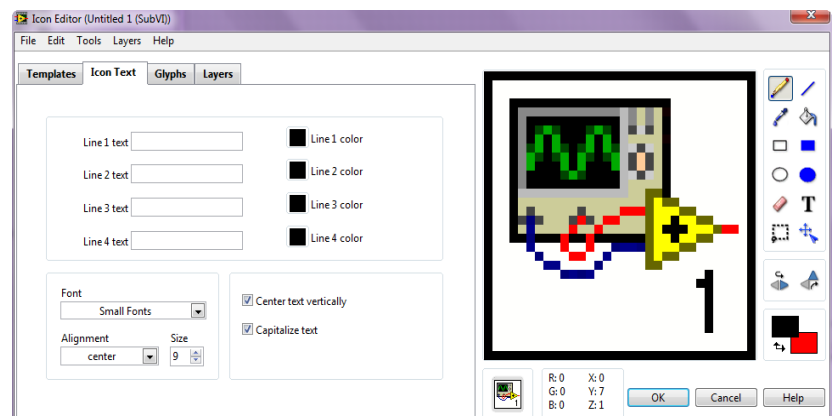
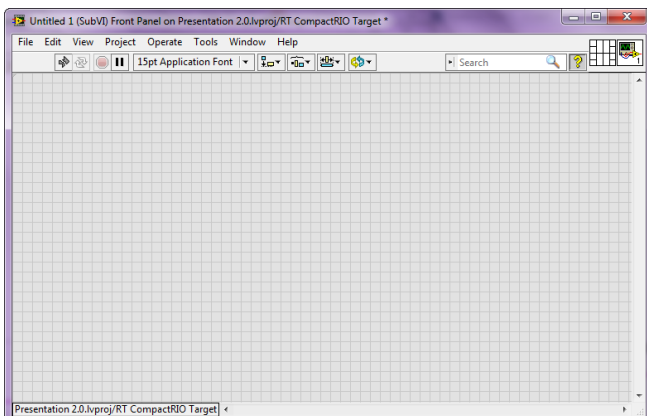
When you writing code in LabVIEW, things could get cluttered quickly and it becomes longer and longer to find what you need. A sub vi can solve that solution by packing what you have programed into another vi that you can access.

Sub Vis will look the same as a regular vi, but it will contain the code that you have created in it. You can have the subvi take inputs and output it something else. Subvis has patterns that you can use to help dictate what an input is and what is an output. You want to have the pattern to have inputs on the left and outputs on the right or you might have trouble remembering which terminal is which. SubVis can also have documentation along with it to help you remember what that vi is used for.

Subvis has the ability to have a custom image for it. Usually you want the vi to be descriptive visually also to help you figure out what vi is quicker. Here are some examples of creating a sub vi:



Patterns can be changed by clicking the box next to the image of the vi.



## **Conclusion**

This concludes the LabVIEW Introduction for FRC 2014. I hope you have learned a lot from this handout and attend another seminar next year. This handout does not cover everything about LabVIEW. More documentation will be created later on during the build season. Please if you could; take the survey for this handout at:

[https://docs.google.com/forms/d/1IResKCwyGsDzO3Pw5hRydKE8xeLW68itUjjE\\_X2Nes/viewform](https://docs.google.com/forms/d/1IResKCwyGsDzO3Pw5hRydKE8xeLW68itUjjE_X2Nes/viewform)

Thank you for reading this handout and if you have questions feel free to email me at [team357royalassaultlabviewhelp@gmail.com](mailto:team357royalassaultlabviewhelp@gmail.com). Priority for questions are for rookie teams and teams with no programming resources available nearby. I hope you have a great season and good luck at the competitions.