**Sensor Fusion: Integrating Multiple Sensors to Complete a Task**

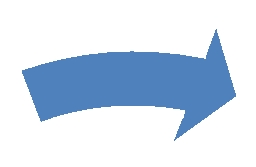
In this tutorial, you will learn how to integrate multiple sensor measurements and use those measurements to make a decision and complete a task. This tutorial builds upon material presented in previous tutorials. It is highly recommended that you complete the Sensor Basics and Getting Started with the FRC Framework Tutorials before this tutorial.

**Contents**

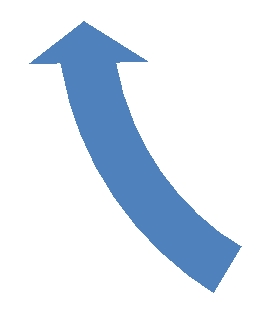
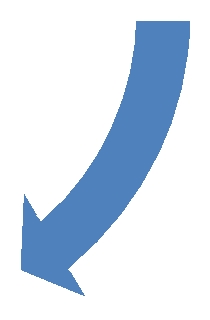
1. Introduction
2. Creating a state machine
3. The Rotate, Stop, and Go cases
4. Extending our code to complete a course
5. Conclusion

**Introduction**

Higher level functionality in robotics usually follows the Sense→Think→Act paradigm, where multiple sensors are used as inputs to make one or multiple decisions and then execute appropriate actions.



Sense Think



Act

The integration of multiple sensors into the Sense→Think→Act paradigm on your robot requires additional considerations beyond the open, get/set close paradigm including:

* What sensors are we using on our robot?
* How do we make decisions (implement logic) based on those sensor values?
* How do we return decision values to the appropriate location in our code?

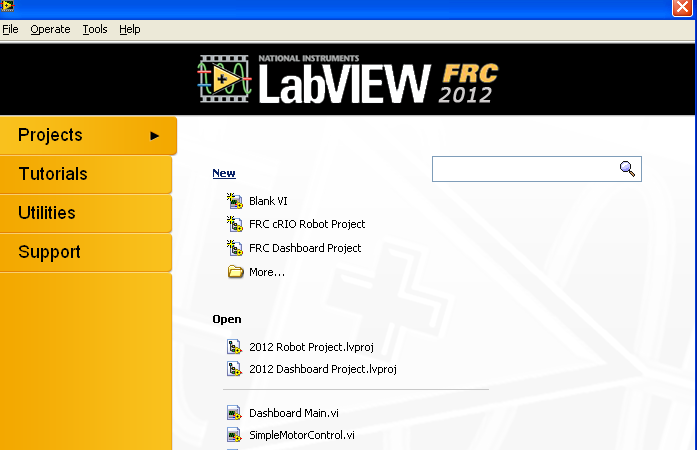
We’ll address these considerations by walking through an example of applying the Sense→Think→Act paradigm in adding autonomous functionality to a basic robot.

Let’s say that we have a robot that will drive around in response to joystick input. We want to add the ability to autonomously complete a course when given a series of moves in terms of distance and rotation. Below are detailed step by step instructions to create a simple autonomous program for this robot.

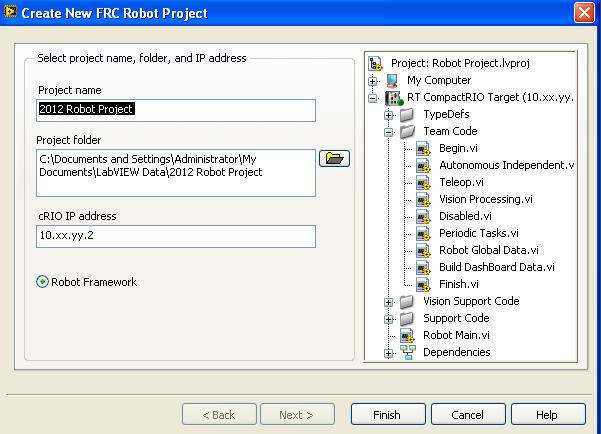
We will be placing our code in the Autonomous Independent VI in the Default Robot Framework.

**Creating a State Machine**

1. From the LabVIEW Getting Started window select **FRC cRIO Robot Project**



1. Name the project and select a location to save the project and set the IP address of the cRIO according to your team number.

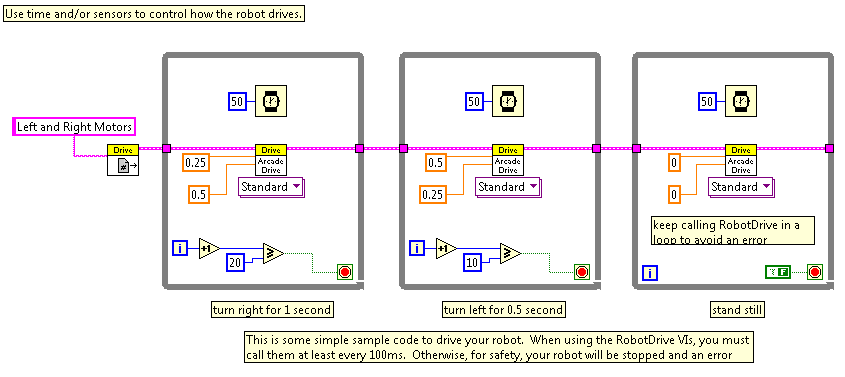


1. Click **Finish**

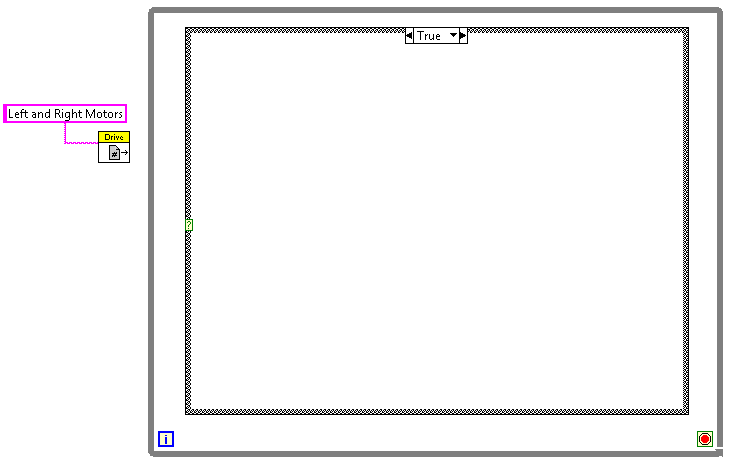
Because we are going to be building autonomous code for our robot, we will be writing all of our code in the Autonomous Independent VI. This VI is included in the cRIO Robot Project and is called by reference by the Default Robot Framework. The state of our robot, autonomous or teleop, is determined either by the FMS or by the competition mode button on the driver station. This modularity is very beneficial because we can keep all of our autonomous code in a single VI and not have to edit anything else in the framework.

To open the Autonomous Independent VI, double-click on the Team Code virtual folder in the project explorer window, double click on Autonomous Independent VI, and select **Window » Show Block Diagram**. You will see there is already some sample autonomous code in this VI.This code will move a robot forward and backwards briefly, but does not incorporate any sensors. We will be removing most of this code in the next few steps. We will also be creating a state machine to hold the three basic steps of our autonomous program. For more information on state machines please see the  [State Machines Tutorial.](http://zone.ni.com/devzone/cda/tut/p/id/7532) In our state machine we will have three states that correspond to the actions that we want to robot to take while in autonomous mode: Rotate, Go, and Stop.

5. Select everything in the VI except the RobotDriveRefNum Registry Get VI and delete the selected items.

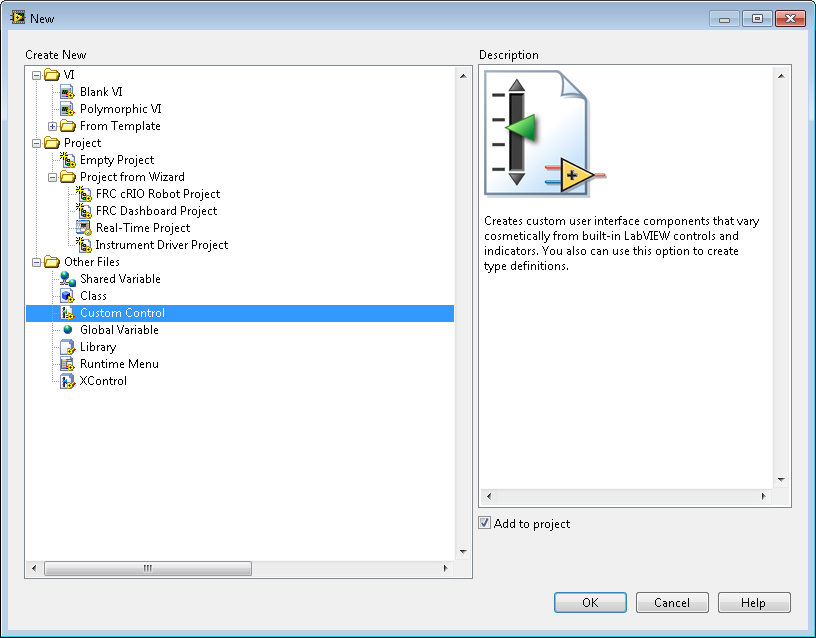


1. To create a state machine, first add a large While Loop from the functions palette **Programming»Structures»While Loop**
2. Now add a case structure within the While Loop from the functions palette **Programming»Structures»Case Structure**

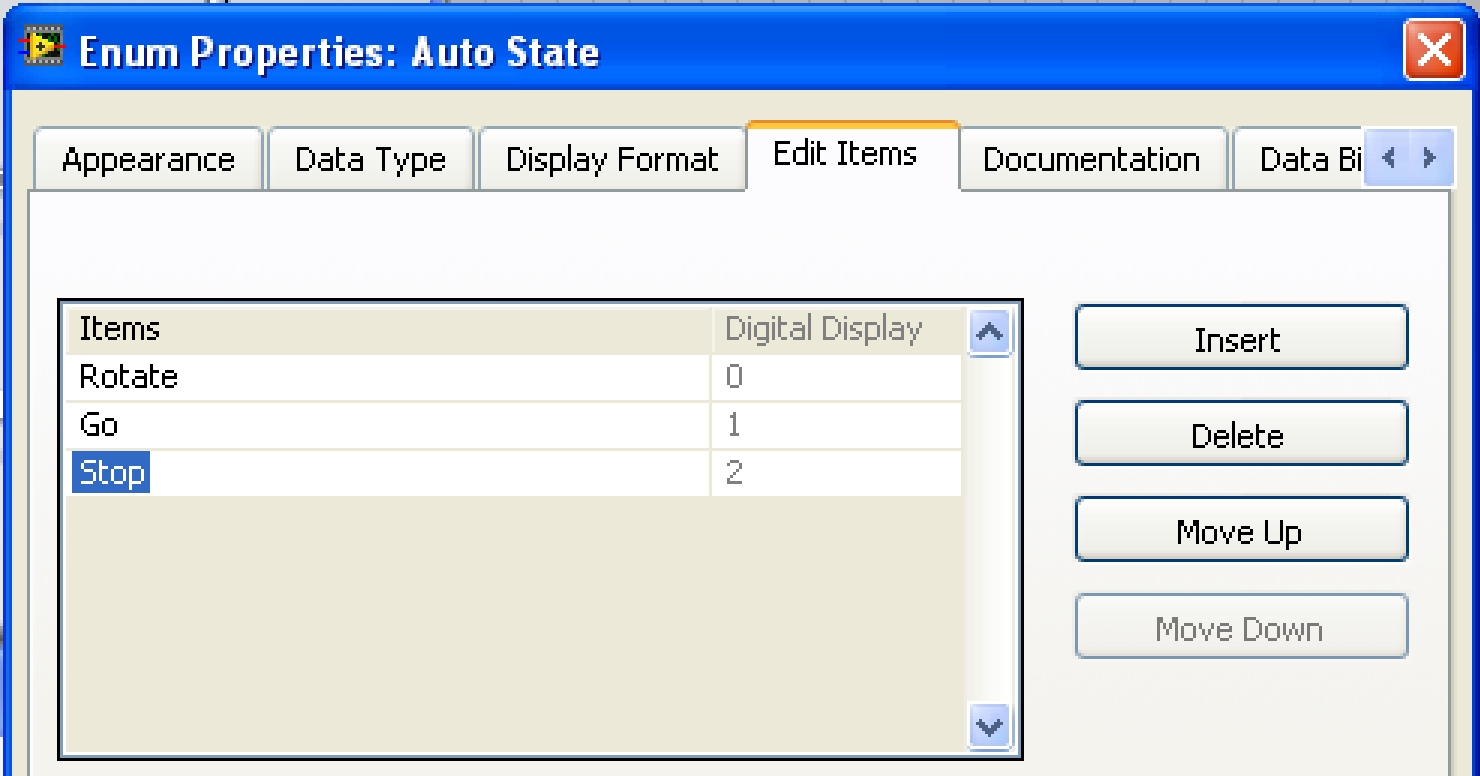


The current state of the state machine is determined by the value of an enum (enumerated list). For this program we will be using a type defined enum so we can easily add more states to our state machine in the future. Rather than using the enum as a front panel control which would allow the user to select the transition states, we will be using it as a constant in our code to programmatically select the next state.

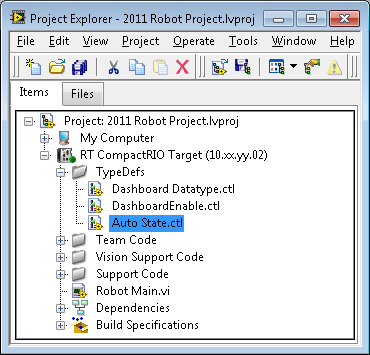
1. From the project explorer windows select **File»New…**
2. In the Create New window select **Other Files»Custom Control** and click **OK.**



1. On the Custom Control window that opens right-click on the front panel and select **Modern»Ring & Enum»Enum.**
2. Name the Enum *Auto State*.
3. From the **Control Type** drop down select **Type Def**.
4. Select **File»Save** and save and name the control *Auto State.ctl*
5. Right-click on the enum and select **Edit Items…**
6. Add **Rotate**, **Go**, and **Stop** to the Items list

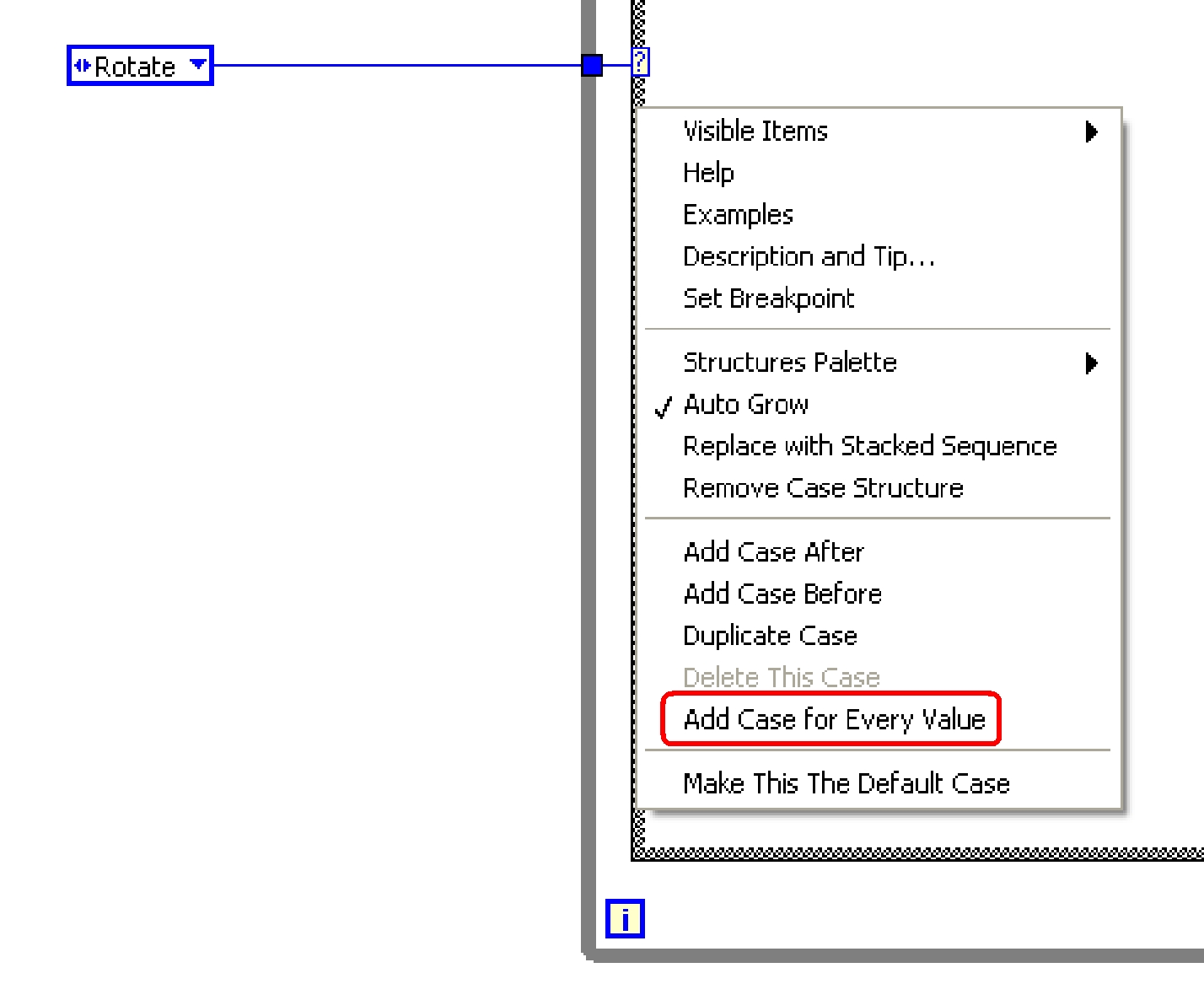


1. Save and close *Auto State.ctl*
2. To add this Type Def to your project, navigate to its location on disk. From Windows explorer and drop the file into the project explorer window or right-click the TypeDefs virtual folder and select **Add»File…**

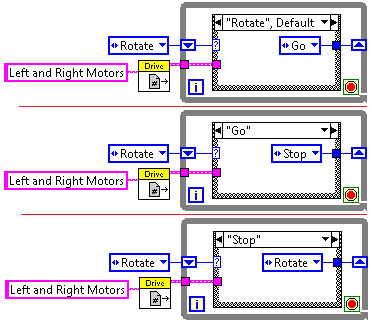


1. Open the block diagram for the Autonomous Independent VI
2. Drag *Auto State.ctl* from the project explorer window to the block diagram and place it to the left of the While Loop.
3. Wire the enum constant to the case selector of the case structure. Ensure **Rotate** is the selected item.
4. Wire the RobotDriveDevRef to the left side of the case structure.

22. Right-click the edge of the case structure and select **Add Case for Every Value**.



1. Right-click the blue tunnel on the While loop and select **Replace with shift register.**
2. Add three more instances of the *Auto State.ctl* to the block diagram and place one in each state of the state machine. Each case of the case structure represents a state.
3. In the Rotate case set the enum to **Go** and wire it to the shift register and the right side of the While Loop. Do this again for the other two states, but set the enum in the Go case to **Stop** and the enum in the Stop Case to **Rotate**. This will ensure our state machine progresses from one state to another correctly.



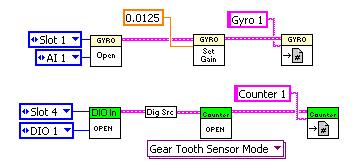
Now that we have created our state machine let’s break down our code according to the Sense→Think→Act paradigm.

**The Rotate, Stop, and Go Cases**

In the following steps we will add code to the three states in our state machine. The two main cases, rotate and go, contain sensor VIs (sense), logic functions and algorithms (think), and motor drive VIs (act).

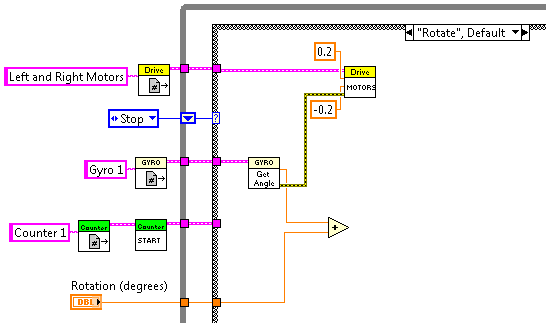
**Begin VI**

1. Open the Begin.vi from the Project Explorer Window and select   
   **Window»Show Block Diagram**.
2. Add the Gyro SetGain, Open Vis, and RefnumSet from the   
   **WPI Robotics Library»Sensors»Gyro** palette
3. Add the DIO Open and ToDigSource VIs from the   
   **WPI Robotics Library»IO»DigitalInput** palette.
4. Add the counter Open and RefnumSet VIs from the   
   **WPI Robotics Library»Sensors»Counter** palette.
5. For the Gyro and DIOin Open VIs, create constants for the slots and channels. We will be using the defaults for each of these.
6. Right-click the gain input on SetGain.vi and select **Create Constant.** Make the value for this constant **0.0125**.
7. From the polymorphic selector on the counter Open VI select **Gear Tooth Sensor Mode**.
8. Right-click the Name input on the Gyro and Counter RefnumSet Vis and give them meaningful names (Ex. Gyro1, Counter1)
9. Wire the Gyro device references through the Open, SetGain, and RefnumSet VIs.
10. Wire the DIOin Reference to the ToDigitalSource VI, the output of the ToDigitalSource VI to the input of the Counter Open VI, and the output of the Counter Open VI to the RefnumSet VI.

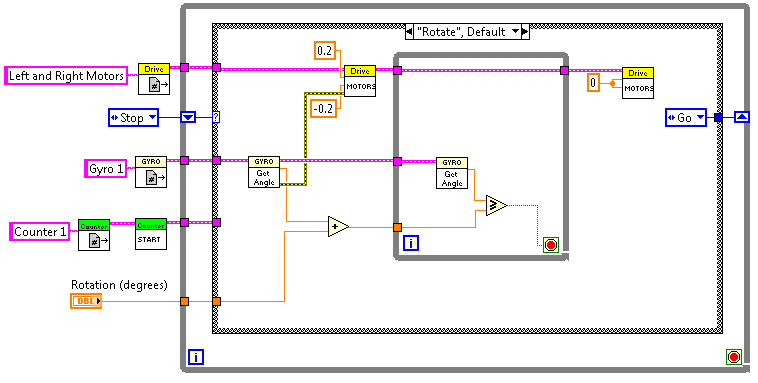


**Rotate Case**

1. Switch back to the Autonomous Independent VI and open up the Rotate Case.
2. Add a Gyro RefnumGet from the **WPI Robotics Library»Sensors»Gyro** palette to the left of the Case Structure
3. Right-Click the Name input of the Gyro RefnumGet, select **Create»Constant**, and enter the name that you gave the Gyro in the Begin VI
4. Add a Counter RefnumGet from the **WPI Robotics Library»Sensors»Counter** palette to the left of the Case Structure
5. Right-Click the Name input of the Counter RefnumGet, select **Create»Constant**, and enter the name that you gave the Counter in the Begin VI
6. Add a Counter Start VI from the **WPI Robotics Library»Sensors»Counter** palette to the right of the RefnumGet VI
7. Wire the Device Reference output of the Counter RefnumGet VI to the Counter Start VI and the Output of the Counter Start VI to the right side of the case structure.
8. Wire the GyroDevRef output of the Gyro GetRefnum VI to left side of the case structure.
9. In the **Rotate** case of the case structure, add the gyro GetAngle VI and wire the GyroDevRef to the appropriate terminal on the left edge of the case structure.
10. On the front panel add a numeric control. Name the control *Rotation (degrees)*. Change the value of the control to **90**. Right-click on the control and select **Data Operations»Make** **Current Value Default**. This ensures the value of 90 will be the default value of this control eachtime we launch the VI.
11. On the block diagram place the control you just created to the left of the While Loops and wire it to the left edge of the case structure.
12. Add two instances of the **Motors** VI from the **WPI Robotics Library » RobotDrive » Advanced » Motors** palette.
13. Wire the error out of the **Get Angle** VI to the error in input of the leftmost Motors VI.
14. Create two constants for the left motor and right motor inputs of the Motors VI and give them the value of **0.2** and **-0.2** respectively
15. Wire the RobotDevRef input from the left edge of the case structure to the RobotDevRef input terminal of the Motors VI.
16. Add an **Add** function from the **Programming » Numeric** palette.
17. Wire the angle output of the Get Angle VI and the control you named *Rotation (Degrees)* to the two inputs of the Add function.



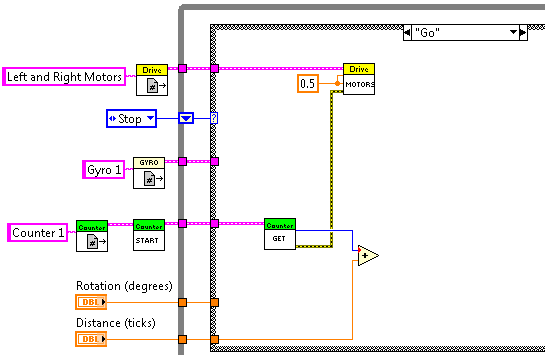
1. Add a small While Loop inside the Rotate case from the **Programming»Structures** palette.
2. Place another gyro Get Angle VI in the While Loop from the   
   **WPI Robotics Library»Sensors »** **Gyro** palette.
3. Place a Greater or Equal? function inside the small While Loop from the **Programming»Comparison** palette.
4. Wire the output of the Get Angle VI to the X input of the Greater or Equal? function and wire the output of the Add function to the Y input of the Greater or Equal? function.
5. Wire the output of the Greater of Equal? function to the input of the loop conditional terminal.
6. Wire the RobotDriveDevRef from the output of the first Motors VI through the While Loop to the RobotDriveDevRef input of the second Motors VI**.** This will ensure the second Motors VI does not execute until after the While Loop is finished executing.
7. Wire a constant of **0** to the Left Speed and Right Speed inputs of the second Motors VI.



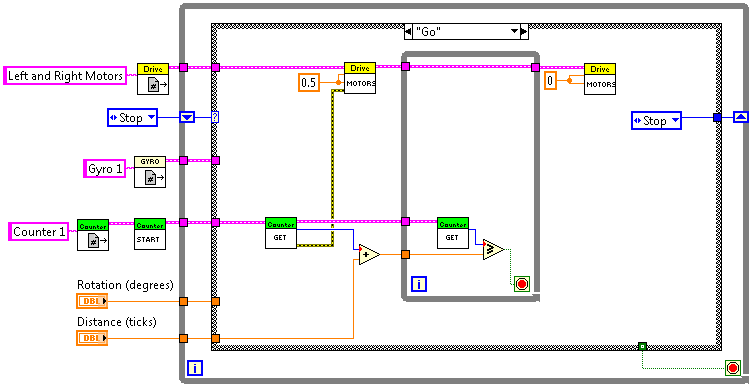
This completes the Rotate case.

**“Go” Case**

1. On the front panel add a numeric control. Name the control *Distance (ticks).* Change the value of the control to **20**. Right-click on the control and select **Data Operations»Make Current** **Value Default**. This ensures the value of 20 will be the default value of this control each timewe launch the VI.
2. Use the case selector to select the **Go** case of the case structure
3. Add a Counter Get VI and wire its device reference input to the appropriate terminal on the left edge of the case structure.
4. Add two instances of the **Drive** VI from the   
   **WPI Robotics Library»RobotDrive»Advanced » Motors** palette.
5. Wire the error out of the counter Get VI to the error in input of the first Motors VI.
6. Create a numeric constant of **0.5** and wire it to both the LeftSpeed and RightSpeed inputs of the Motors VI.
7. Wire the RobotDevRef from the left edge of the case structure to the RobotDevRef input of the Motors VI
8. Add an **Add** function from the **Programming»Numeric** palette.
9. Wire the angle output of the counter Get VI and the numeric control you named *Distance (ticks)* to the two inputs of the Add function.



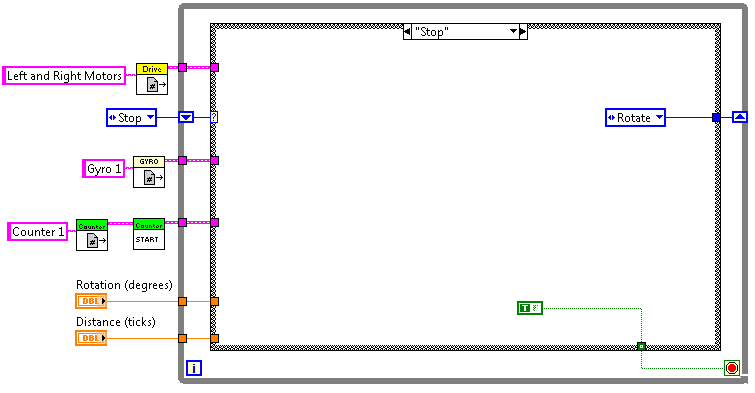
1. Add a small While Loop inside the Rotate case from the **Programming»Structures** palette.
2. Place another counter Get VI in the While Loop from the   
   **WPI Robotics Library»Sensors»Counter** palette.
3. Place a **Greater or Equal?** function inside the small While Loop from the **Programming»Comparison** palette.
4. Wire the output of the counter Get VI to the X input of the Greater or Equal? function and wire the output of the **Add** function to the Y input of the Greater or Equal? function.
5. Wire the output of the Greater of Equal? function to the input of the loop conditional terminal.
6. Wire the RobotDriveDevRef from the output of the first Motors VI through the While Loop to the RobotDriveDevRef input of the second Motors VI. This will ensure that the second Motors VI does not execute until after the While Loop is finished executing.
7. Wire a constant of **0** to the Left Speed and Right Speed inputs of the second Motors VI.



Finally we will create the Stop case of our state machine to end the execution of the While Loop.

**Stop Case**

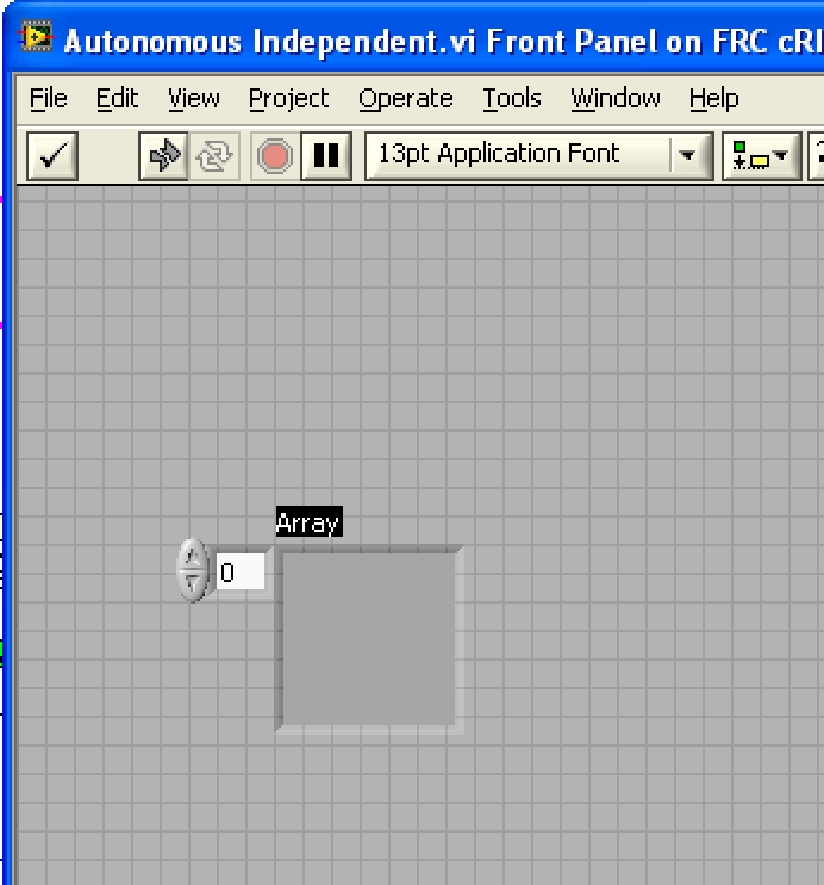
1. Use the case selector to select the **Stop** case of the case structure
2. Select a True constant from the functions palette **Programming»Boolean»True Constant** and place the constant inside the Stop case of the case structure
3. Wire the Boolean constant to the loop conditional terminal.
4. Right-click the tunnel created by the Boolean constant on the case structure and select **Use** **Default Terminal if Unwired**.

****

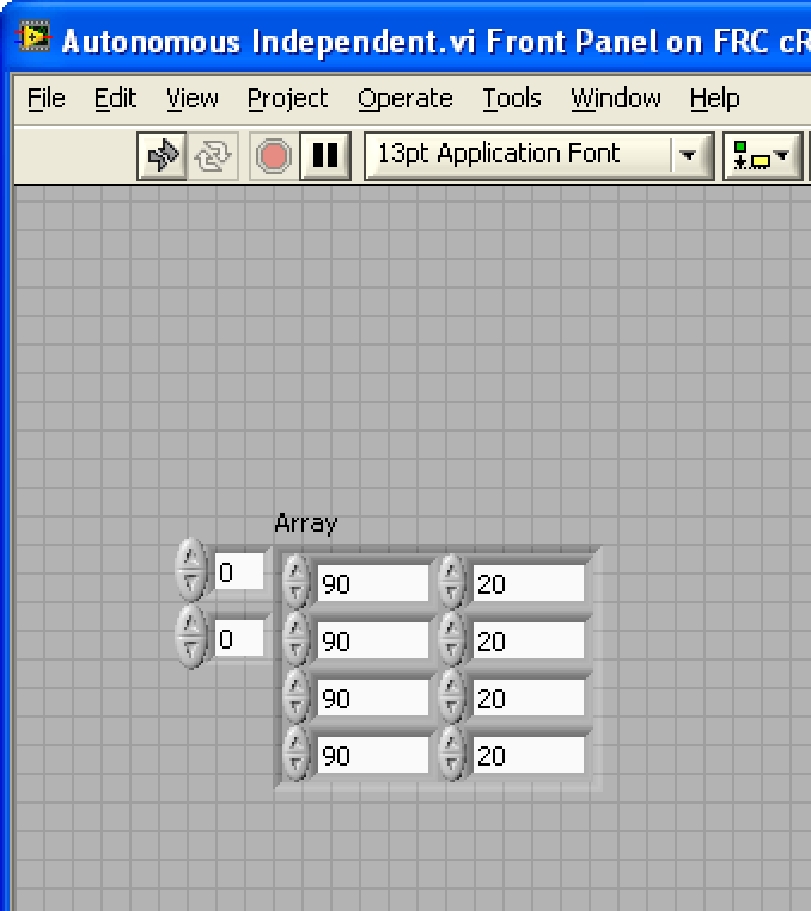
**Extending our code to complete a course**

Now let’s make some simple changes to this VI so that our robot can complete a square course. We will do this by adding an array of two parameters: degrees to rotate and distance to travel. We will index through this array to pass the correct value into the correct case.

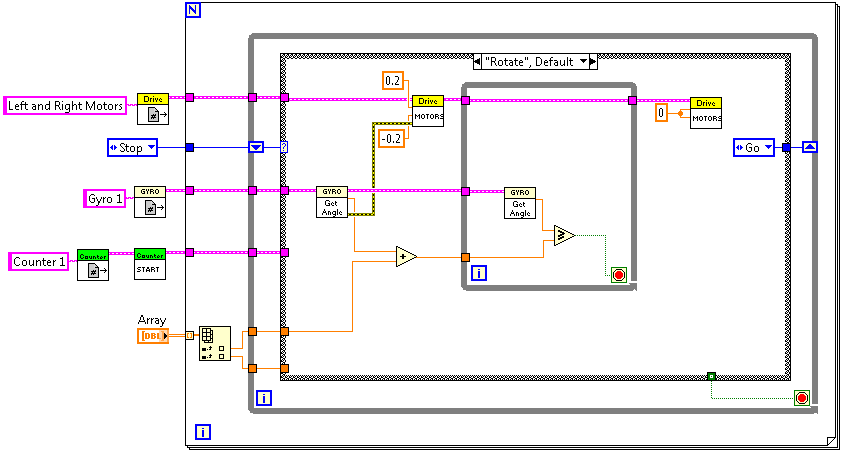
1. Remove the controls you had named *Rotation (degrees)* and *Distance (ticks)* from your code. Remove the broken wires from outside the While Loop but leave all of the other broken wire since we will be reconnecting the loose ends later.
2. Place a For Loop around your While Loop by selecting   
   **Programming»Structures»For Loop** and lassoing it around your While Loop.
3. On the front panel add an empty array shell from the control palette   
   **Modern»Array, Matrix, &** **Cluster»Array**



1. Place a numeric control inside the empty array shell from the controls palette **Modern»Numeric»Numeric Control**.
2. Right-click the array index selector and select **Add Dimension** and drag out the edge of the array to display both dimensions.
3. Drag down the bottom edge of the array to display a 2x4 array
4. The first column of the array will be our rotation in degrees and the second column will be our distance in ticks. We will execute each row each time we run our state machine, and our autonomous program will end when the end of the array is reached.
5. To make our robot travel in a simple square, set the all the values in the first column to **90** and all of the values in the second column to **20**.



1. Switch to the block diagram. Make sure the array control we created is to the left of the For Loop and wire it to the edge of the For Loop. Notice that an auto-indexing terminal is automatically created on the For Loop. This will index a single row of the 2D array each iteration of the For loop.
2. Add an Index Array function between the For and While Loops from the **Programming»Array** palette.
3. Drag down the bottom of the Index Array so two index input terminal and two element outputs.
4. Wire the array input terminal to the auto-index array tunnel on the For loop. Wire the first element output to the tunnel passing values into the Rotate case and wire the second tunnel passing value to the Go case.



Your robot is now ready to complete a simple course autonomously!

**Conclusion**

Congratulations! You now know how to structure and implement tasks for your robot using the Sense→Think→Act paradigm. This paradigm, and the concepts you learned in implementing it in this tutorial, can be used to give your robot some truly advanced autonomous behavior!