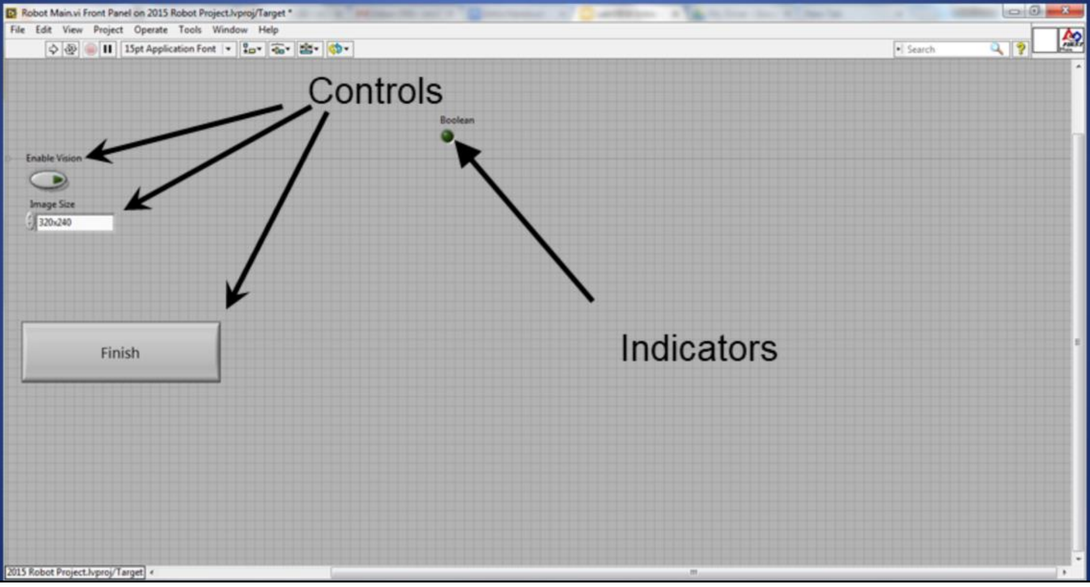
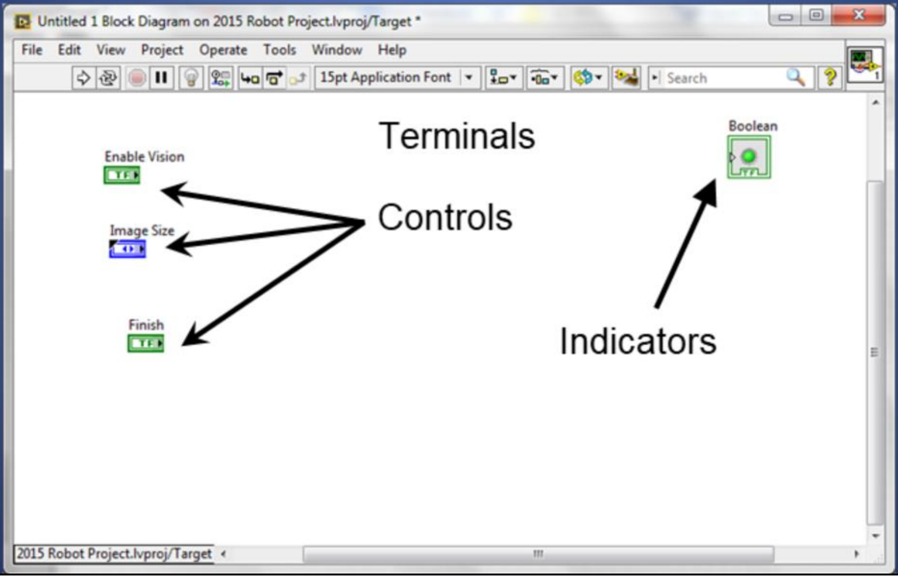
# Intro to LabVIEW

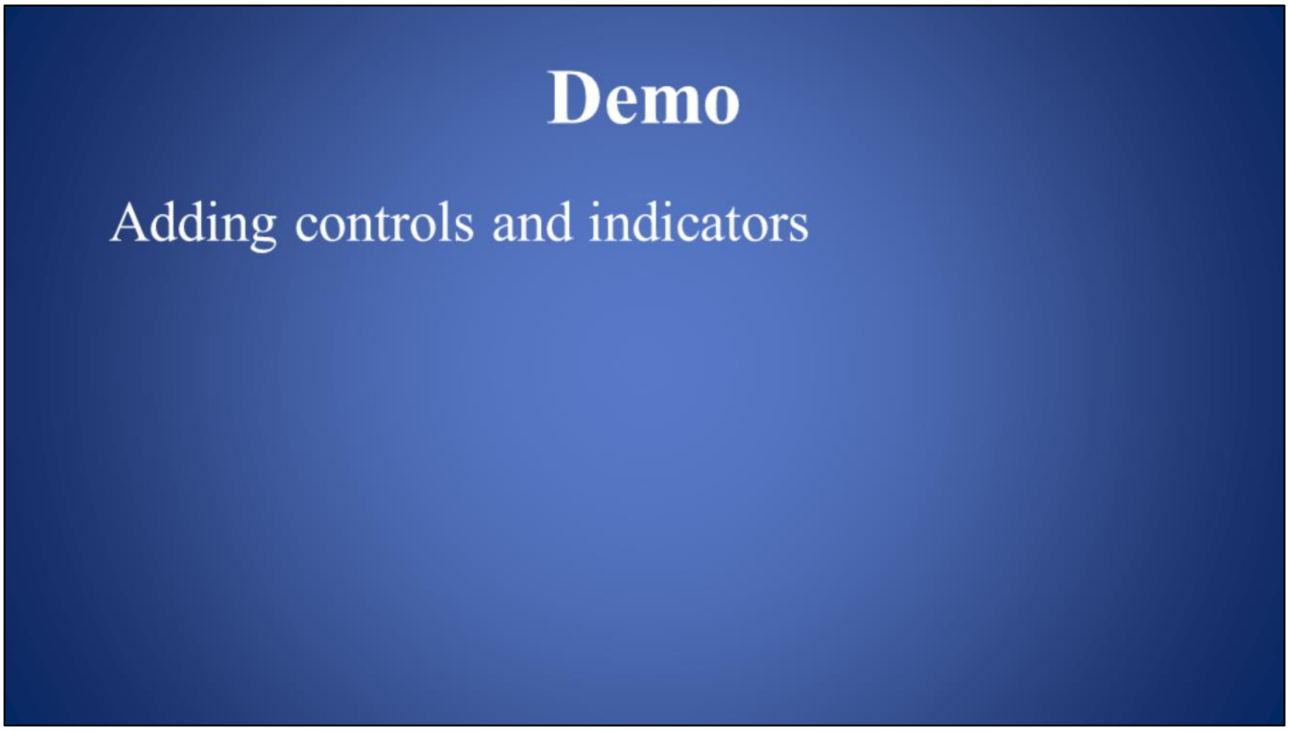
frclabviewtutorials.com/workshop

## Front Panel


Robot Main.vi Front Panel on 2015 Robot Project.lvproj/Target *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Controls
Enable Vision
Image Size
320x240
Finish
Boolean
Indicators
2015 Robot Project.lvproj/Target

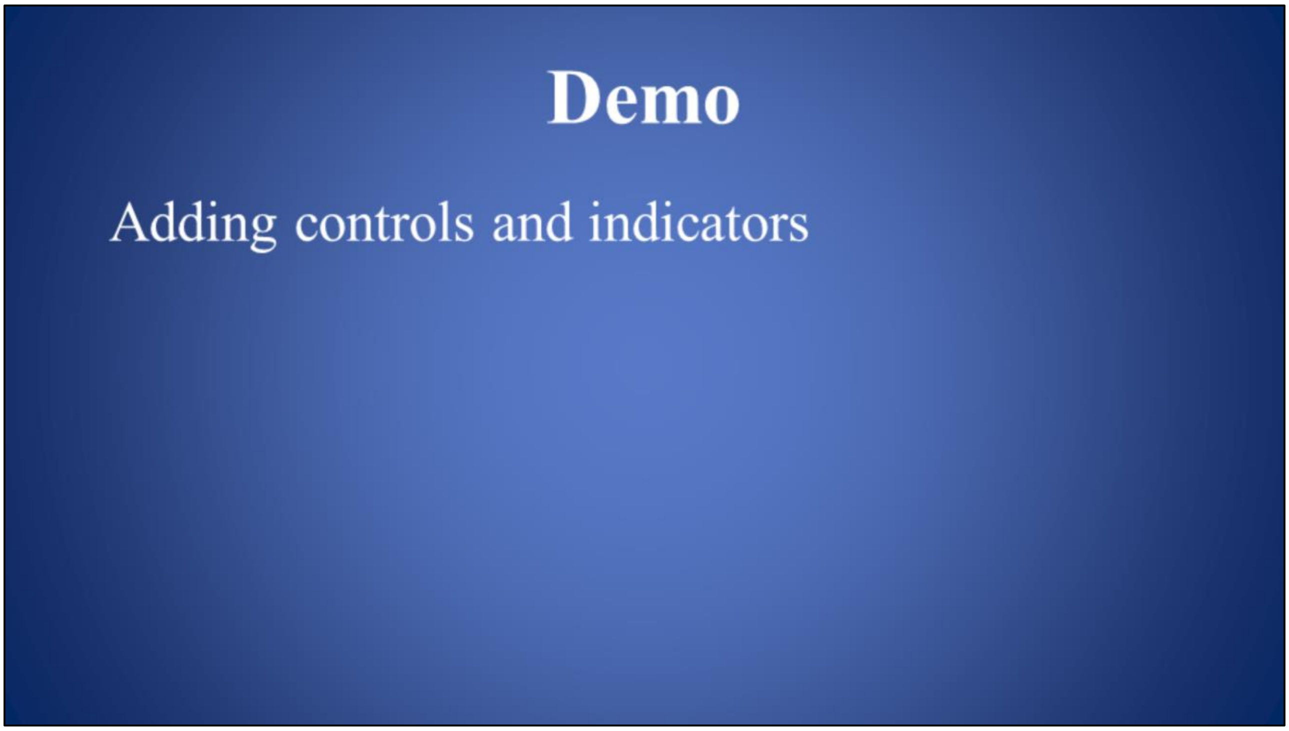
## Block Diagram


Untitled 1 Block Diagram on 2015 Robot Project.lvproj/Target *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Terminals
Enable Vision
Image Size
Finish
Controls
Boolean
Indicators
2015 Robot Project.lvproj/Target



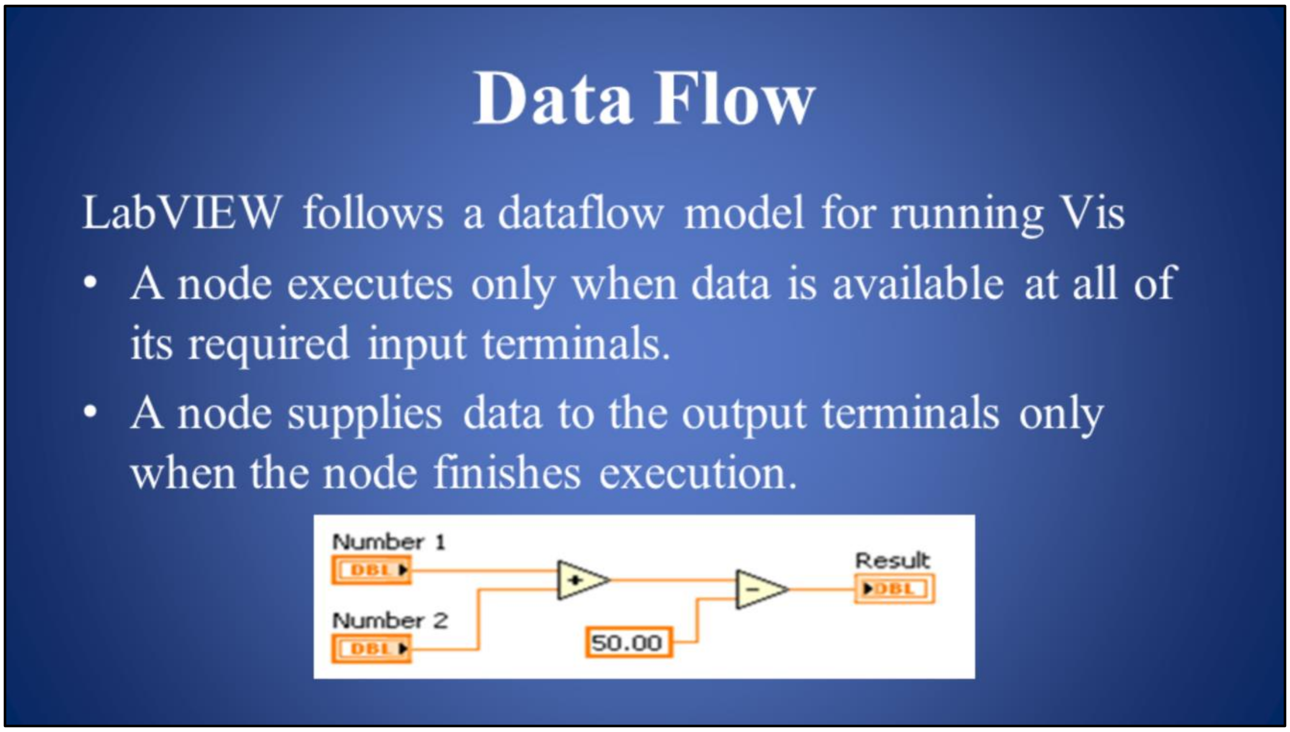
## Demo

Adding controls and indicators



## Demo

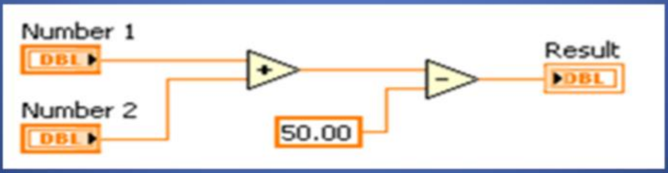
Adding controls and indicators



## Data Flow

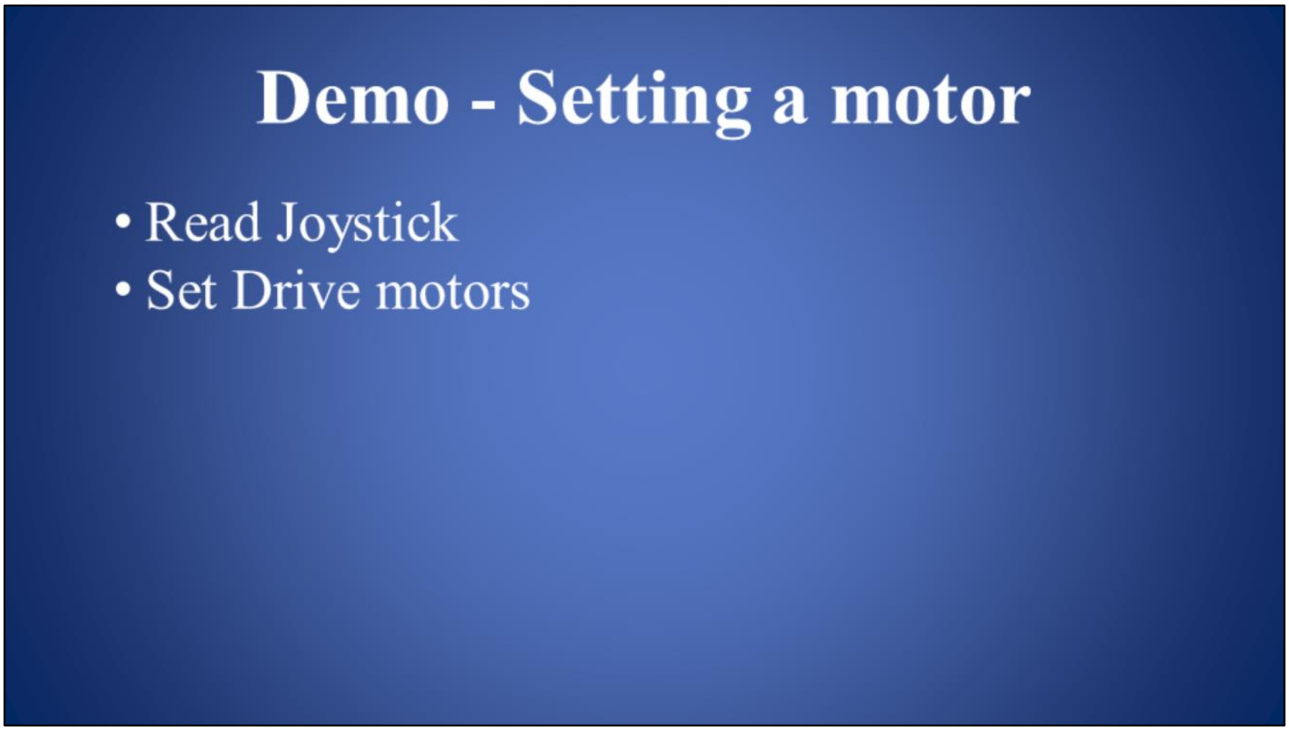
LabVIEW follows a dataflow model for running Vis

- A node executes only when data is available at all of its required input terminals.
- A node supplies data to the output terminals only when the node finishes execution.



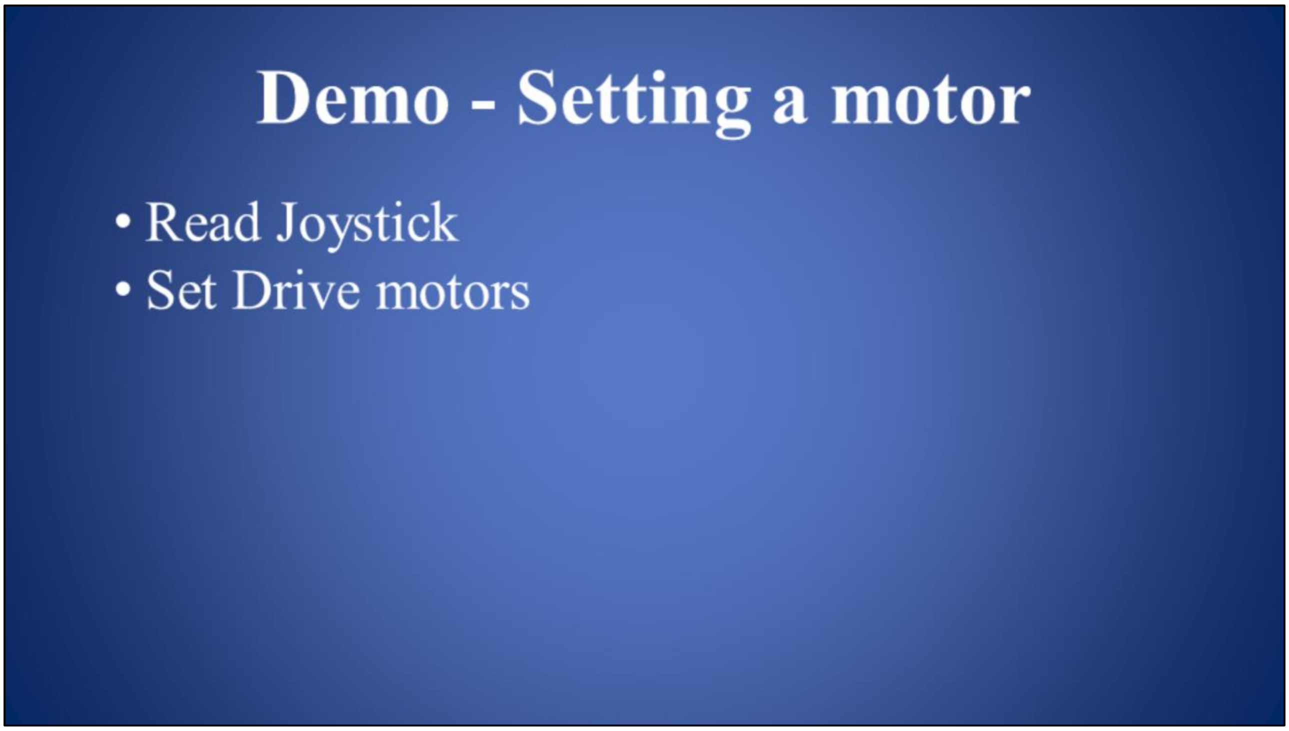
When a node executes, it produces output data and passes the data to the next node in the dataflow path.
The movement of data through the nodes determines the execution order of the VIs and functions on the block diagram.

LabVIEW does NOT use a control flow program execution model like Visual Basic, C++, JAVA, and most other text-based programming languages. In a control flow model, the sequential order of program elements determines the execution order of a program.



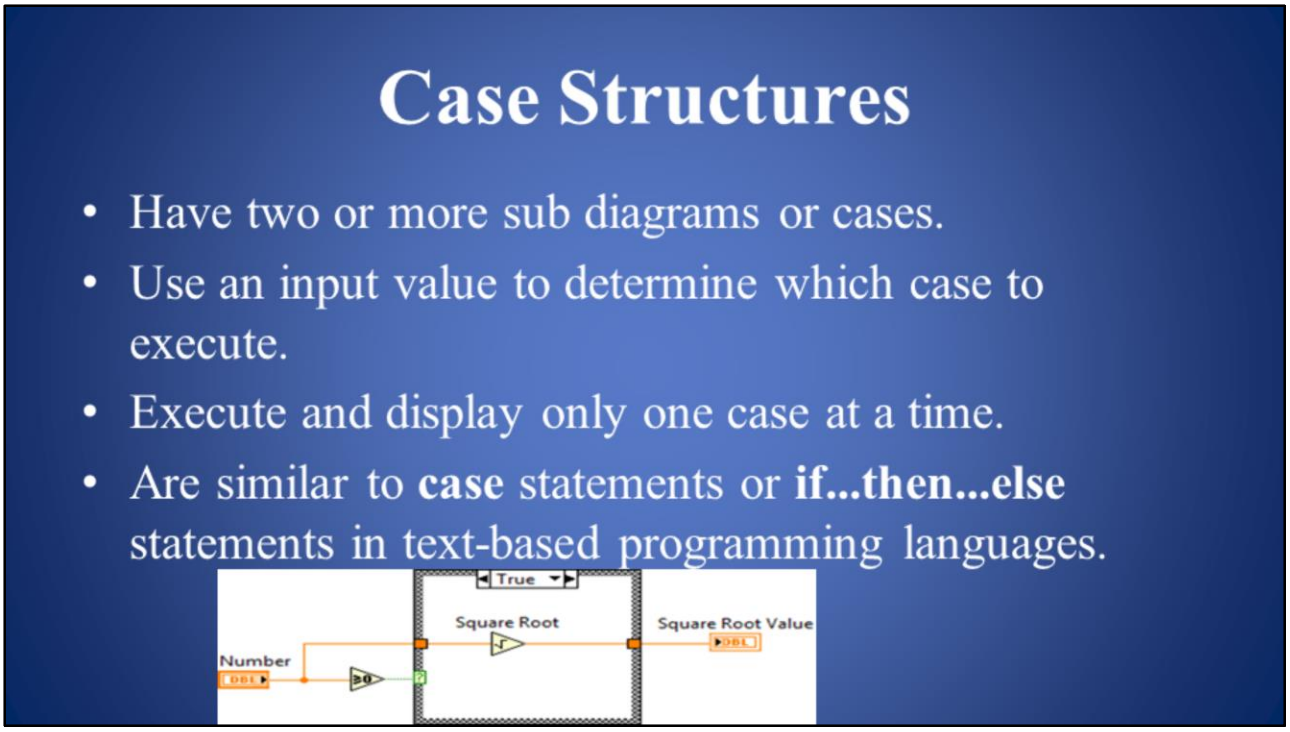
## Demo - Setting a motor

- Read Joystick
- Set Drive motors



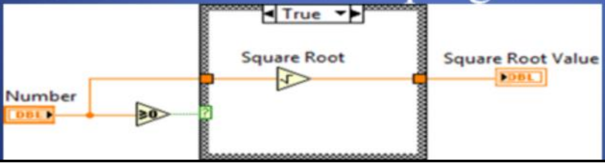
## Demo - Setting a motor

- Read Joystick
- Set Drive motors



## Case Structures

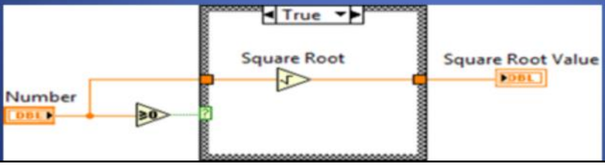
- Have two or more sub diagrams or cases.
- Use an input value to determine which case to execute.
- Execute and display only one case at a time.
- Are similar to **case** statements or **if...then...else** statements in text-based programming languages.



Casestructure.vi and select.vi

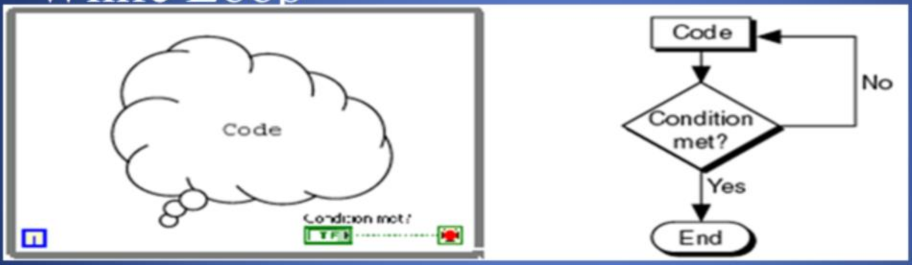
## Case Structures

- Input and Output Tunnels
  - You can create multiple input and output tunnels.
  - Input tunnels are available to all cases if needed.
  - You must define each output tunnel for each case.*



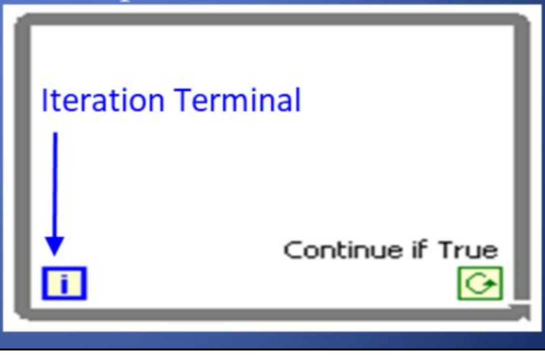
## Repetition

- While Loop



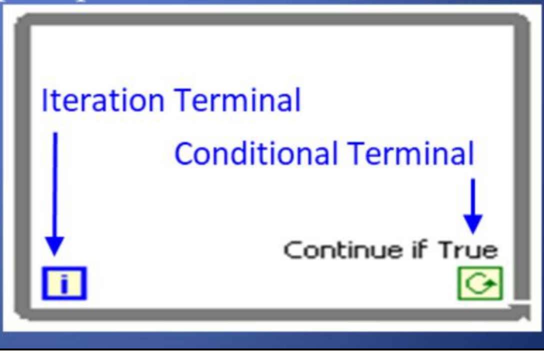
## Repetition

- While Loop
  - Iteration terminal
    - Returns number of times loop has executed.
    - Is zero-indexed.



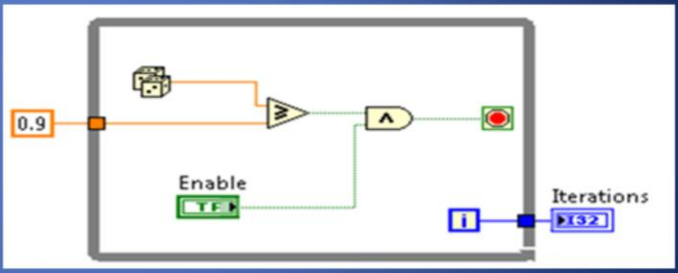
## Repetition

- While Loop
  - Conditional terminal
    - Defines when the loop stops.
    - Has two options.
      - Stop if True
      - Continue if True



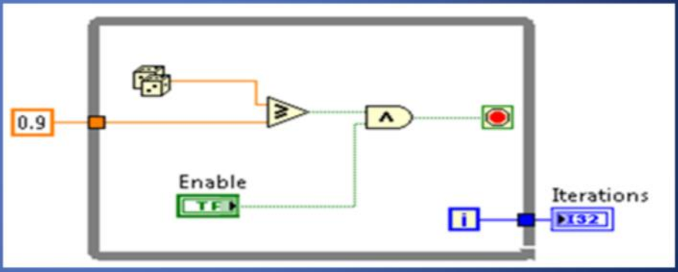
## Repetition

- While Loop
  - Tunnels transfer data into and out of structures.



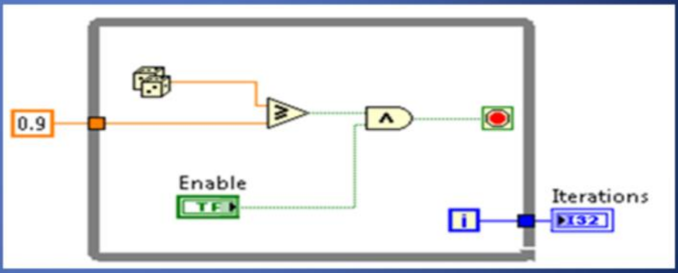
## Repetition

- While Loop
  - Tunnels transfer data into and out of structures.
  - Data pass out of a loop after the loop terminates.



## Repetition

- While Loop
  - Tunnels transfer data into and out of structures.
  - Data pass out of a loop after the loop terminates.
  - When a tunnel passes data into a loop, the loop executes only after data arrives at the tunnel.

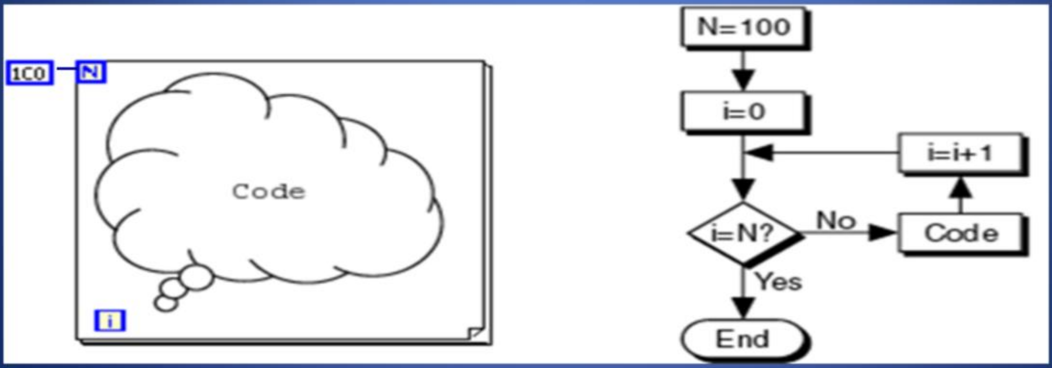


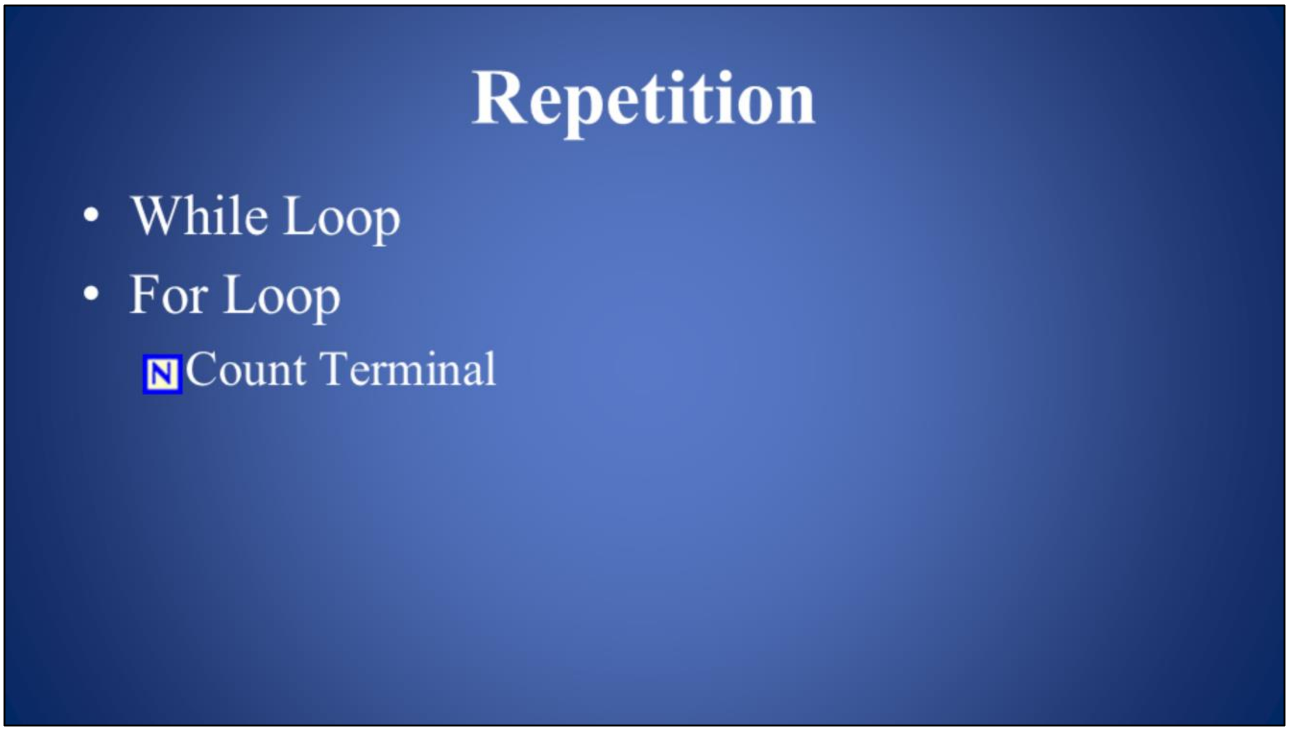
## Repetition

- While Loop  - Demo

## Repetition

- While Loop
- For Loop



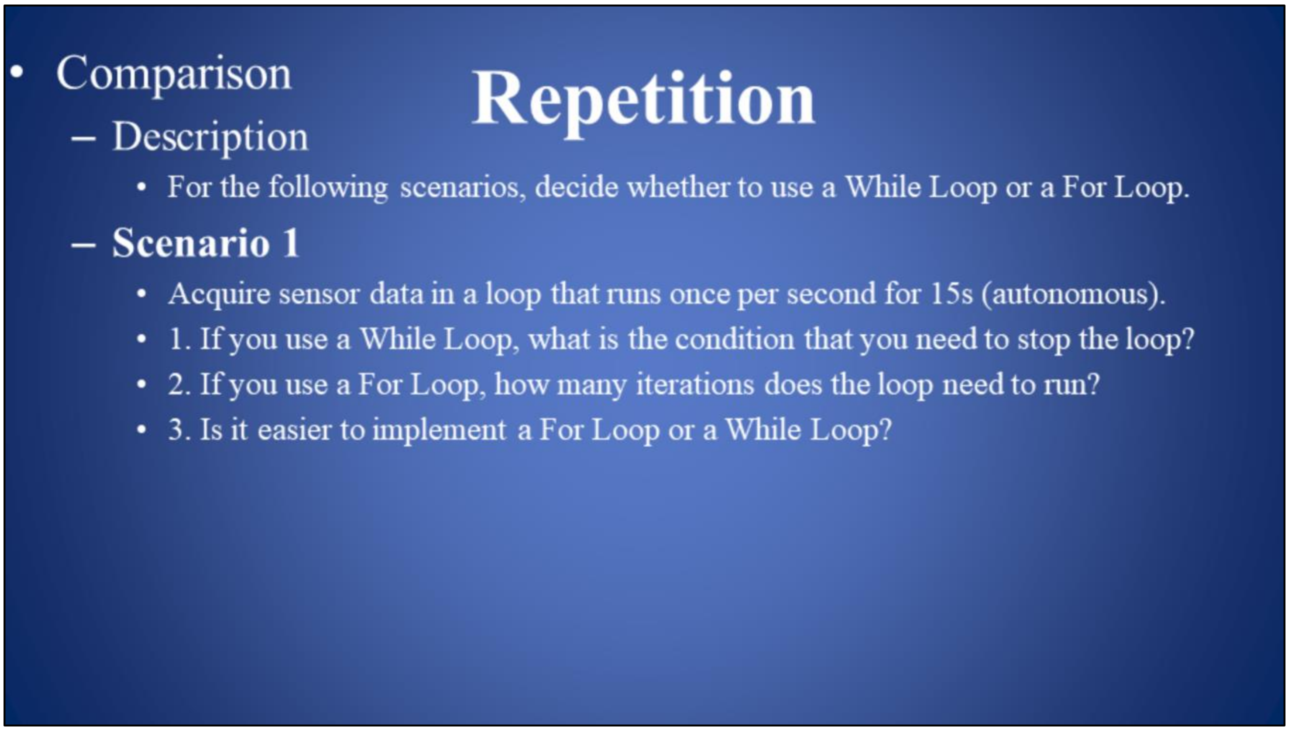


## Repetition

- While Loop
- For Loop
  - N Count Terminal

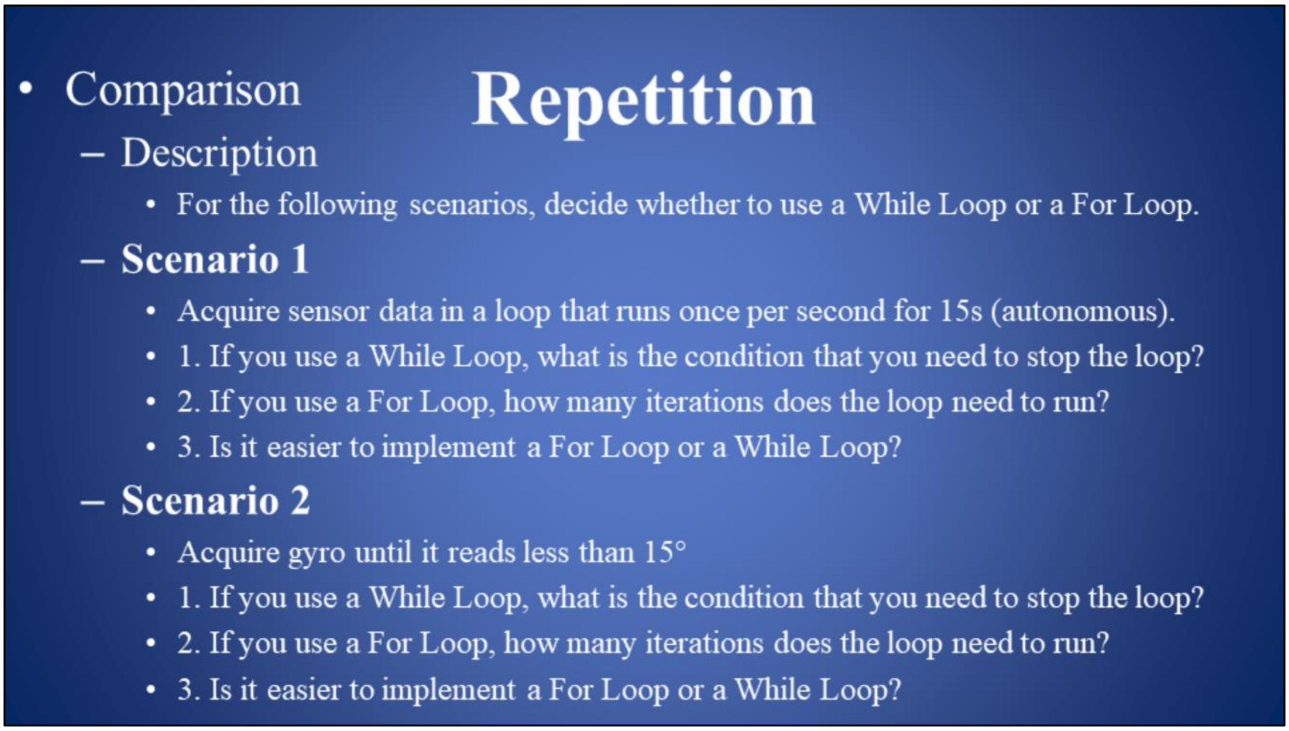
## Repetition

- Comparison
  - Description
    - For the following scenarios, decide whether to use a While Loop or a For Loop.



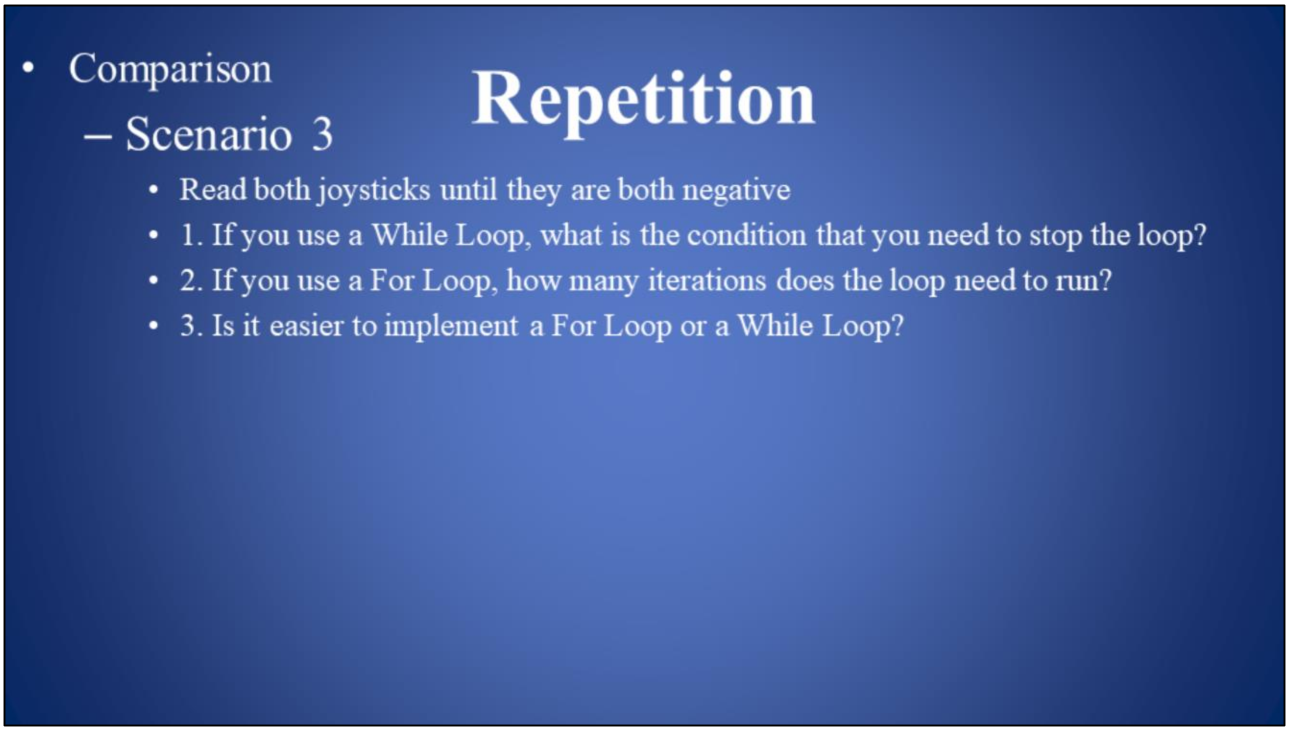
## Repetition

- Comparison
  - Description
    - For the following scenarios, decide whether to use a While Loop or a For Loop.
  - **Scenario 1**
    - Acquire sensor data in a loop that runs once per second for 15s (autonomous).
    - 1. If you use a While Loop, what is the condition that you need to stop the loop?
    - 2. If you use a For Loop, how many iterations does the loop need to run?
    - 3. Is it easier to implement a For Loop or a While Loop?



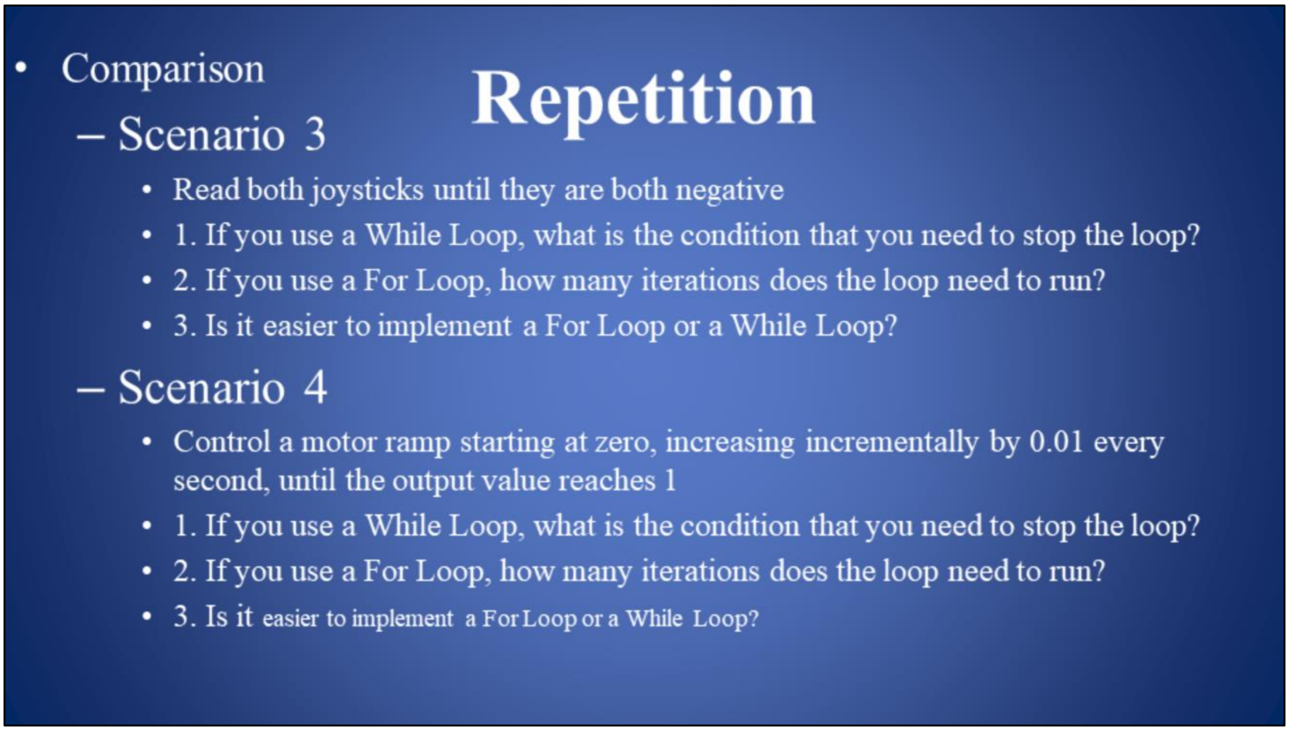
## Repetition

- Comparison
  - Description
    - For the following scenarios, decide whether to use a While Loop or a For Loop.
  - **Scenario 1**
    - Acquire sensor data in a loop that runs once per second for 15s (autonomous).
    - 1. If you use a While Loop, what is the condition that you need to stop the loop?
    - 2. If you use a For Loop, how many iterations does the loop need to run?
    - 3. Is it easier to implement a For Loop or a While Loop?
  - **Scenario 2**
    - Acquire gyro until it reads less than 15°
    - 1. If you use a While Loop, what is the condition that you need to stop the loop?
    - 2. If you use a For Loop, how many iterations does the loop need to run?
    - 3. Is it easier to implement a For Loop or a While Loop?



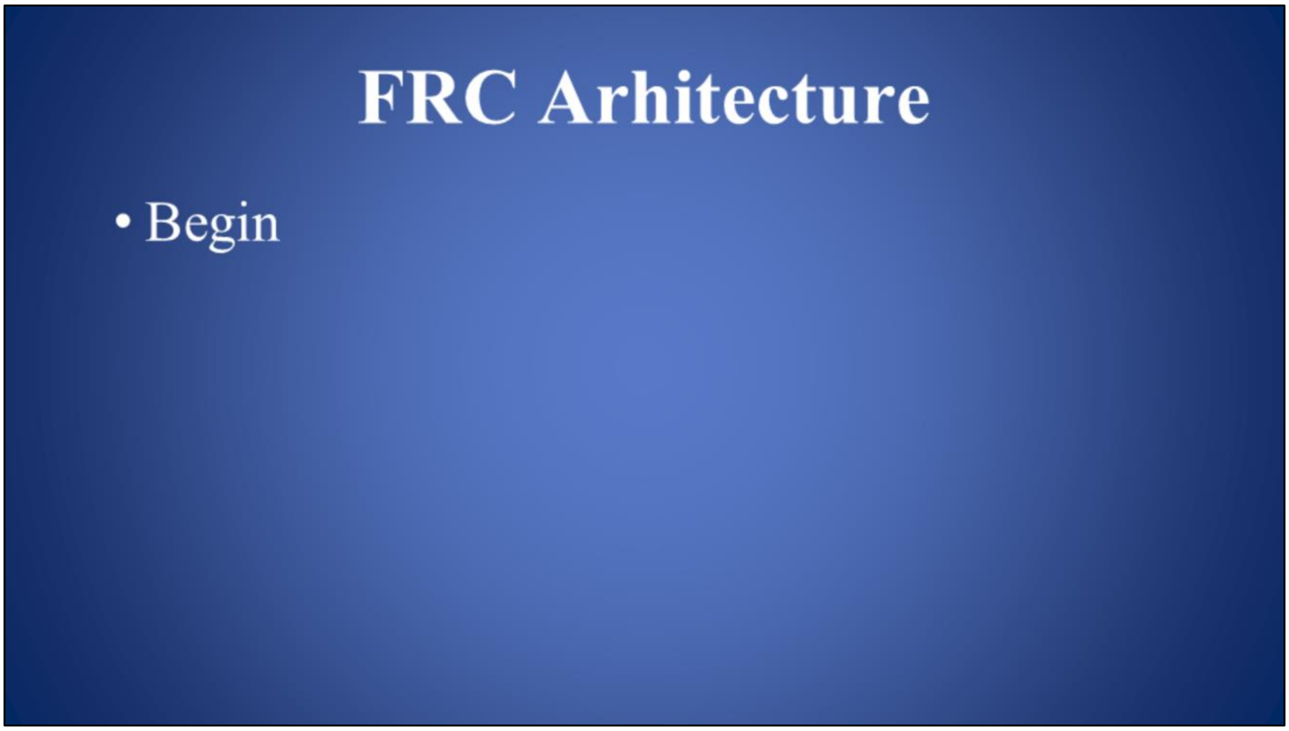
## Repetition

- Comparison
  - Scenario 3
    - Read both joysticks until they are both negative
    - 1. If you use a While Loop, what is the condition that you need to stop the loop?
    - 2. If you use a For Loop, how many iterations does the loop need to run?
    - 3. Is it easier to implement a For Loop or a While Loop?



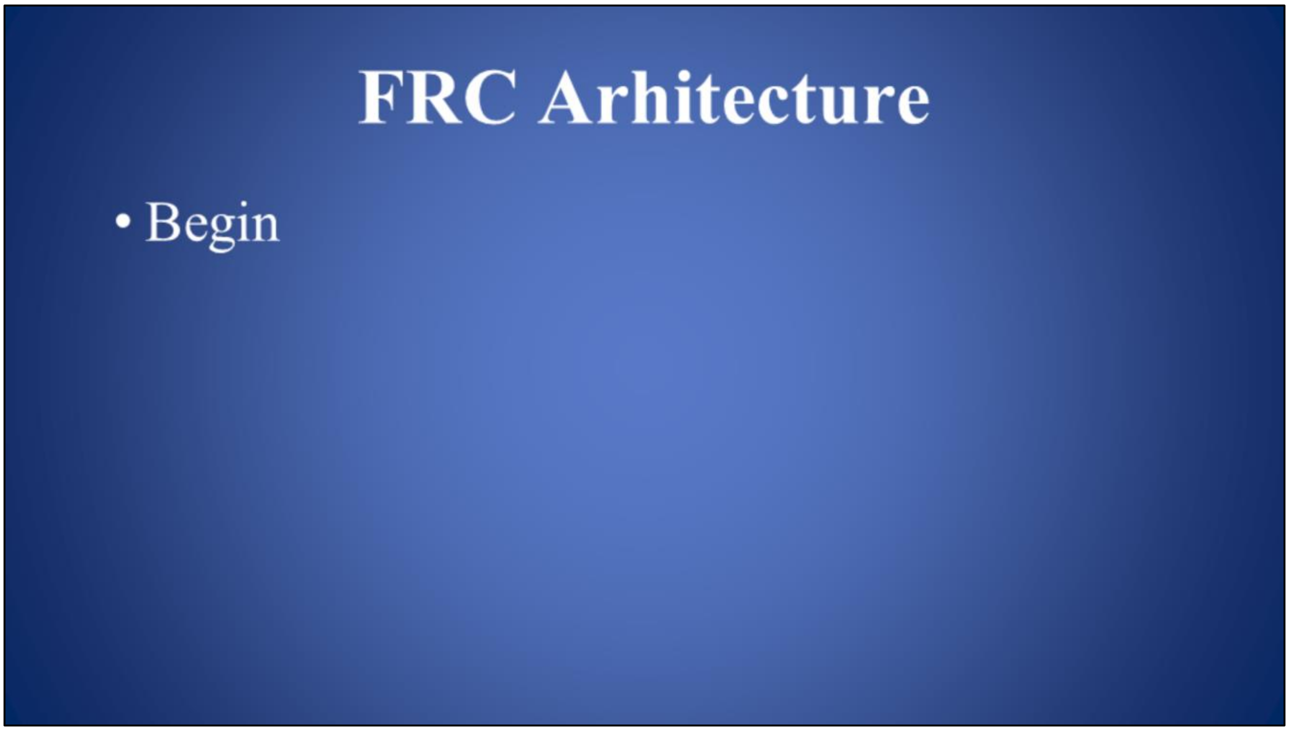
## Repetition

- Comparison
  - Scenario 3
    - Read both joysticks until they are both negative
    - 1. If you use a While Loop, what is the condition that you need to stop the loop?
    - 2. If you use a For Loop, how many iterations does the loop need to run?
    - 3. Is it easier to implement a For Loop or a While Loop?
  - Scenario 4
    - Control a motor ramp starting at zero, increasing incrementally by 0.01 every second, until the output value reaches 1
    - 1. If you use a While Loop, what is the condition that you need to stop the loop?
    - 2. If you use a For Loop, how many iterations does the loop need to run?
    - 3. Is it easier to implement a For Loop or a While Loop?



## FRC Arhitecture

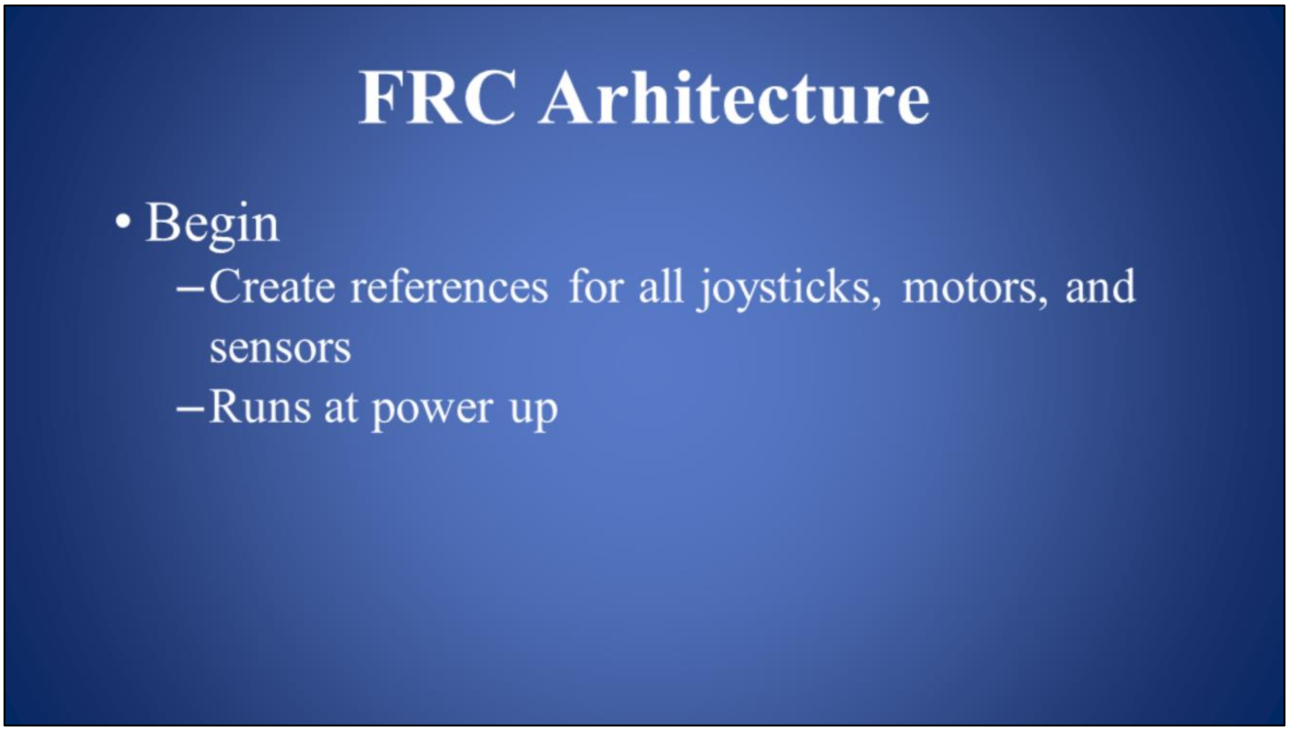
- Begin



## FRC Arhitecture

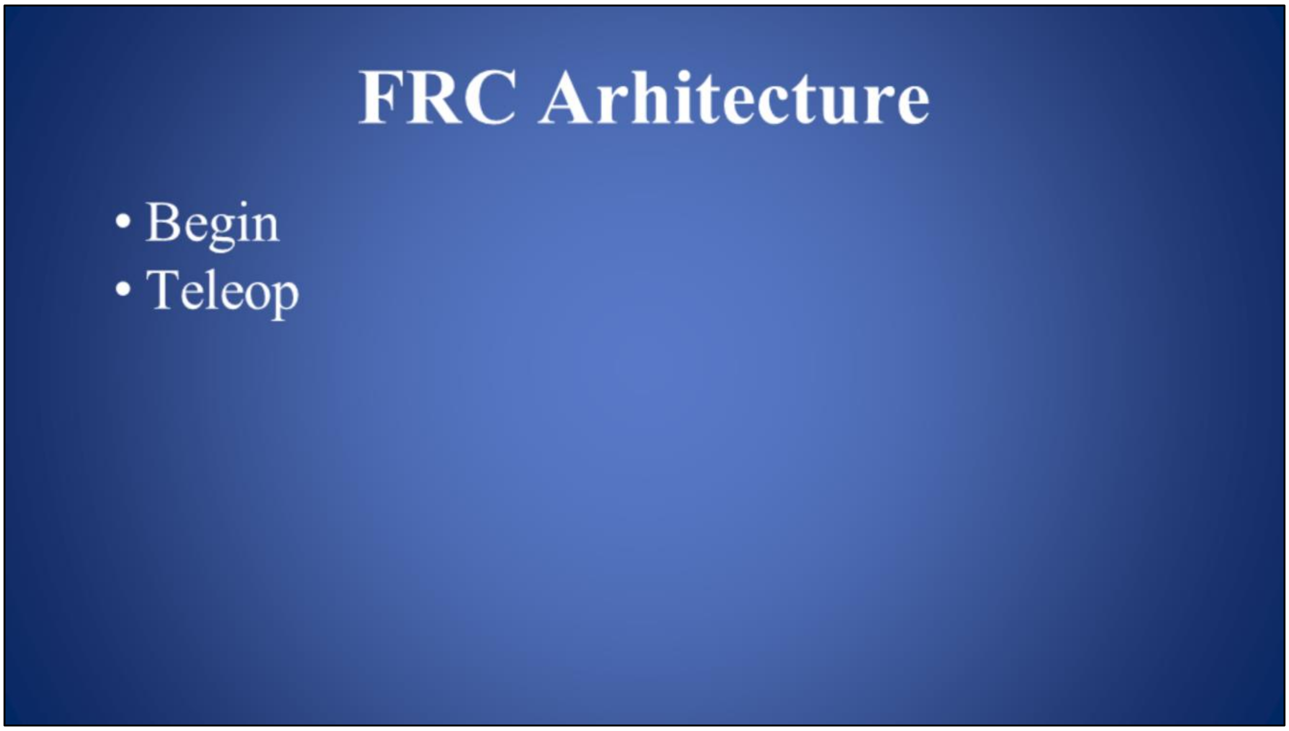
- Begin

Show it



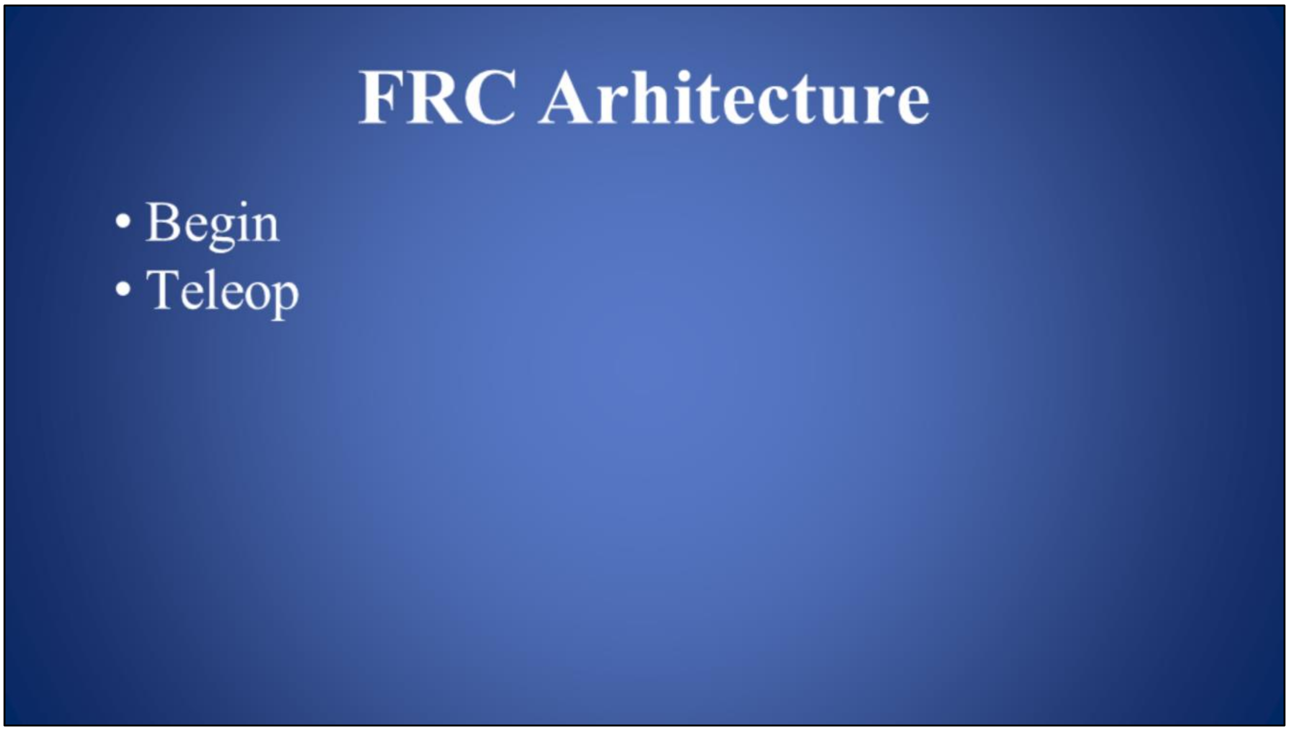
## FRC Arhitecture

- Begin
  - Create references for all joysticks, motors, and sensors
  - Runs at power up



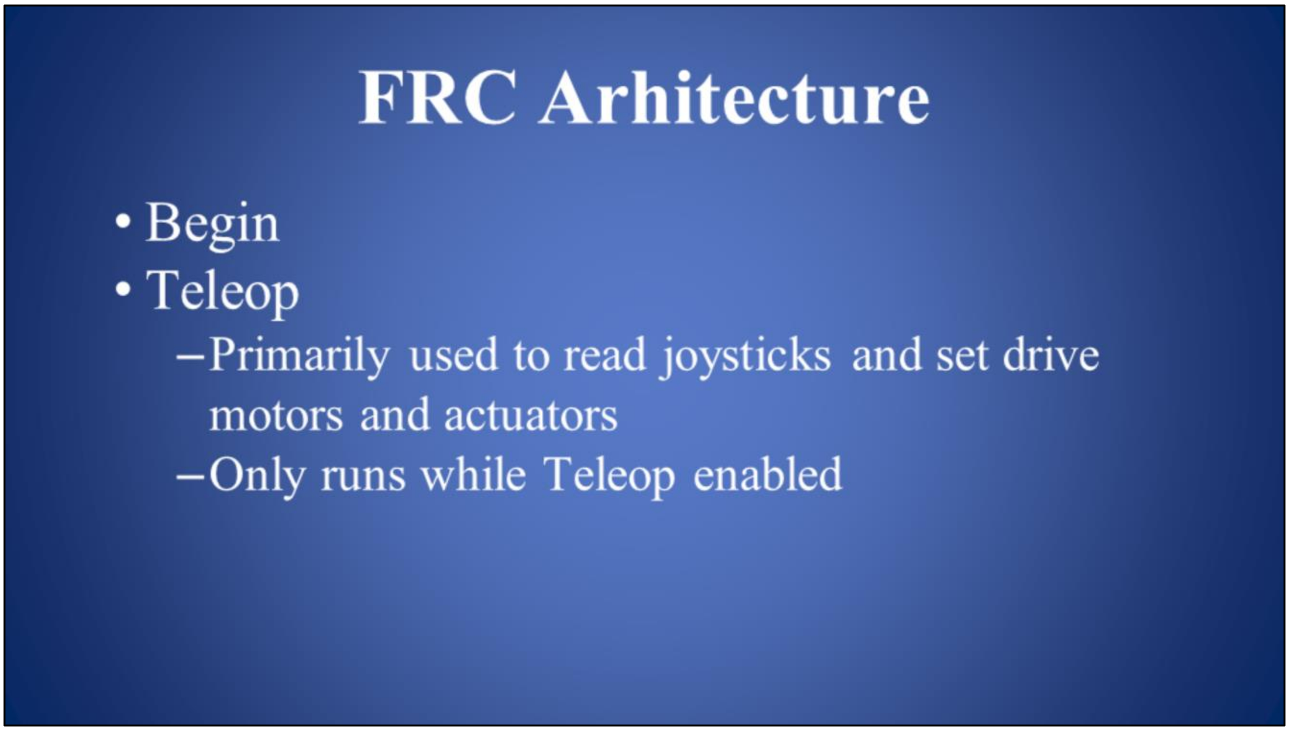
## FRC Arhitecture

- Begin
- Teleop



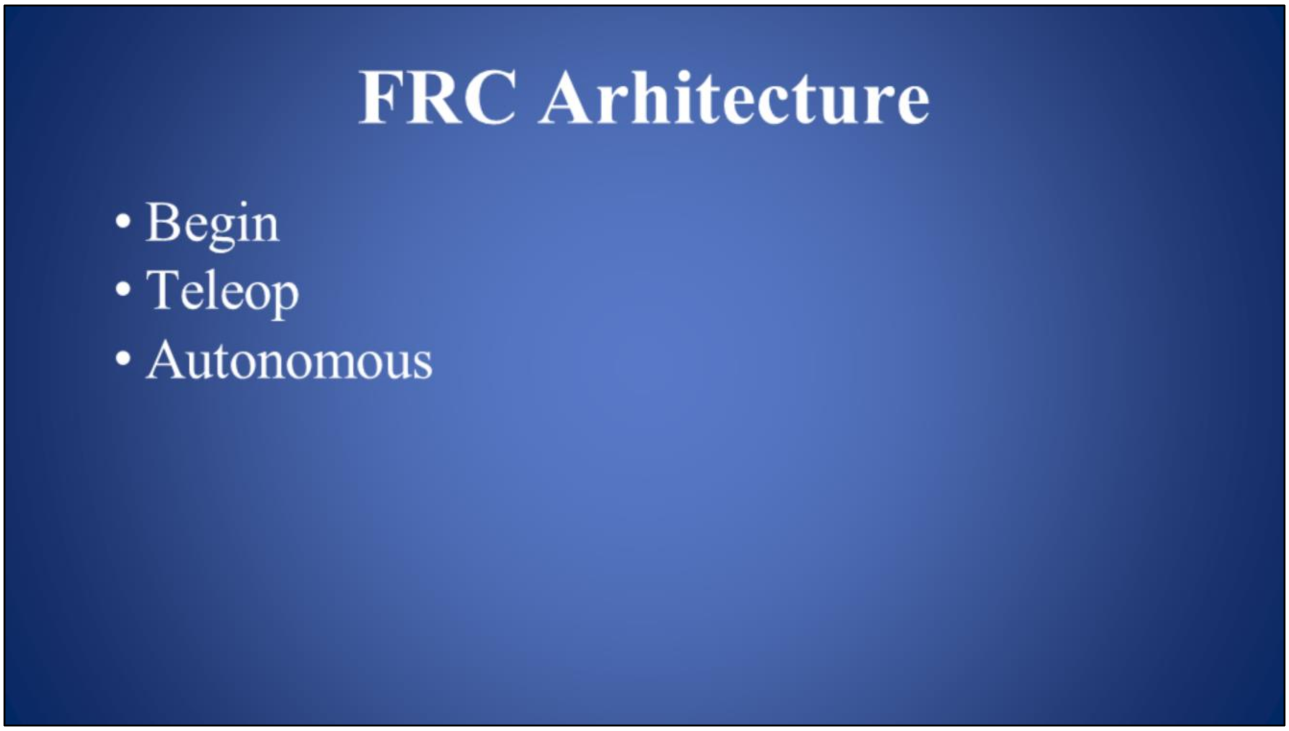
## FRC Arhitecture

- Begin
- Teleop



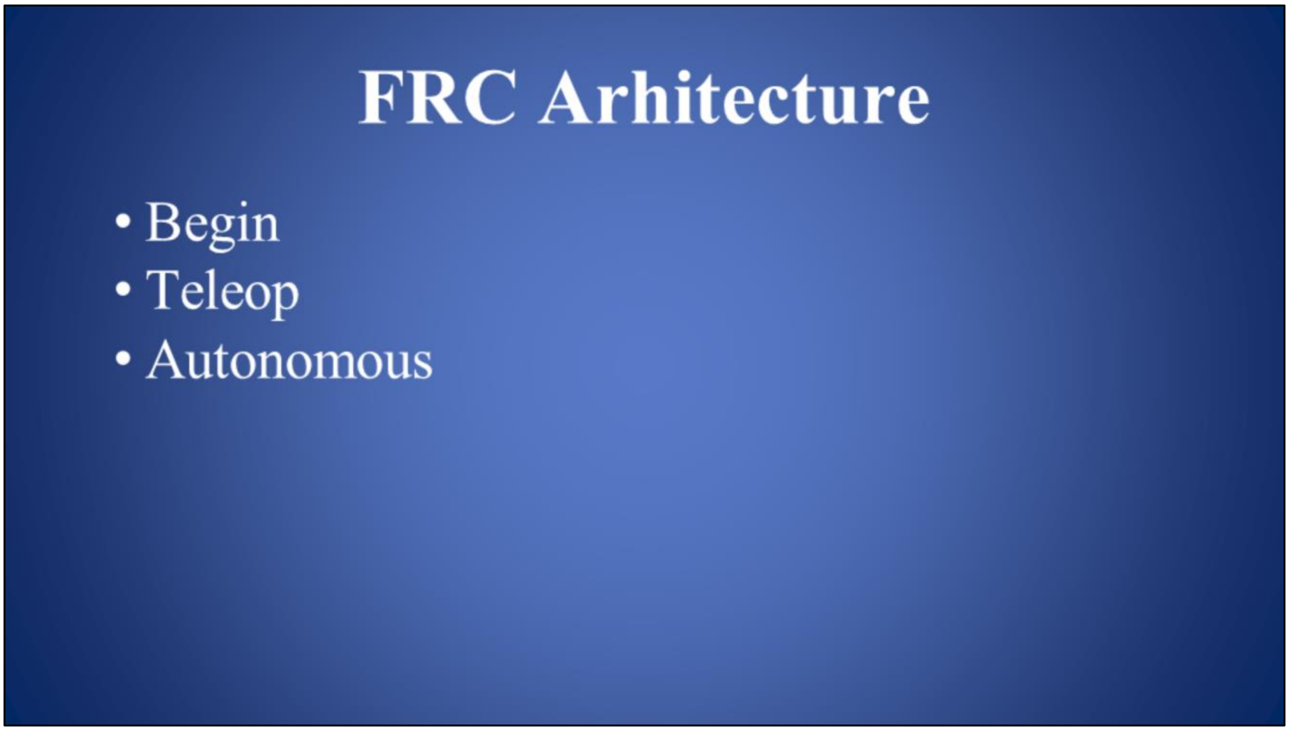
## FRC Arhitecture

- Begin
- Teleop
  - Primarily used to read joysticks and set drive motors and actuators
  - Only runs while Teleop enabled



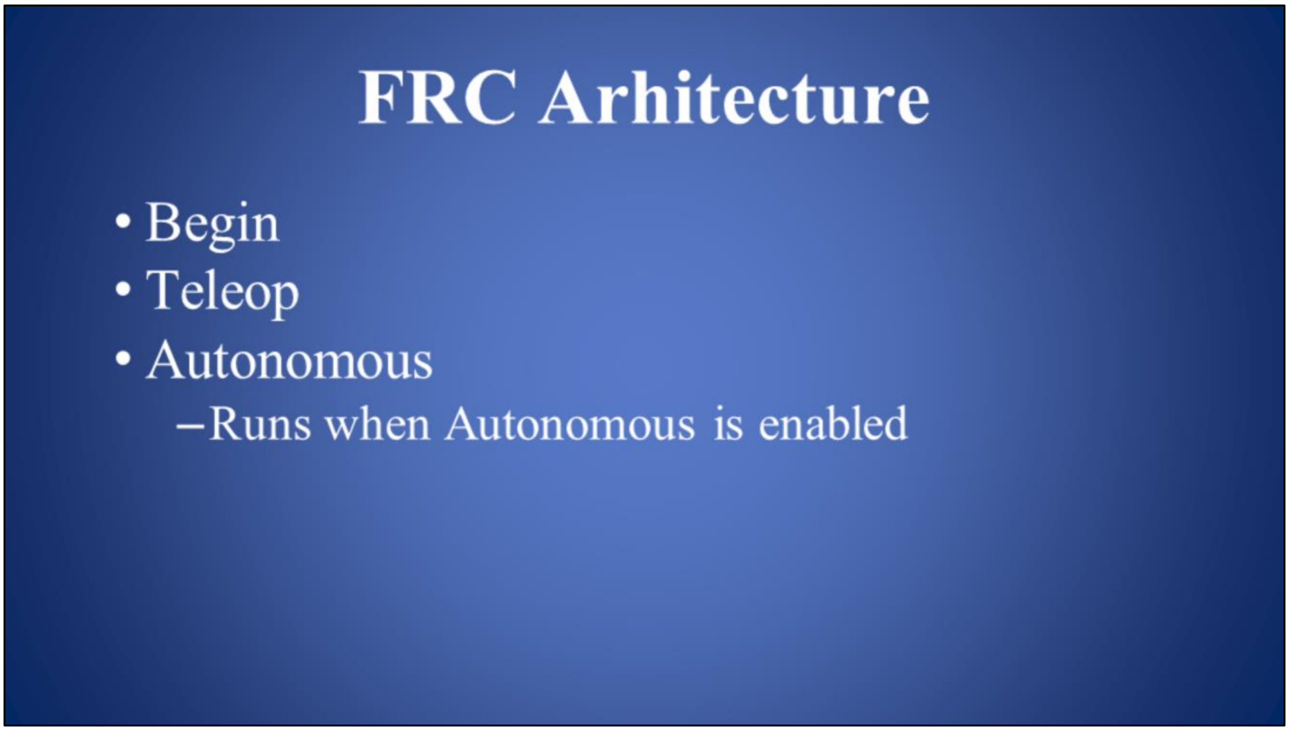
## FRC Arhitecture

- Begin
- Teleop
- Autonomous



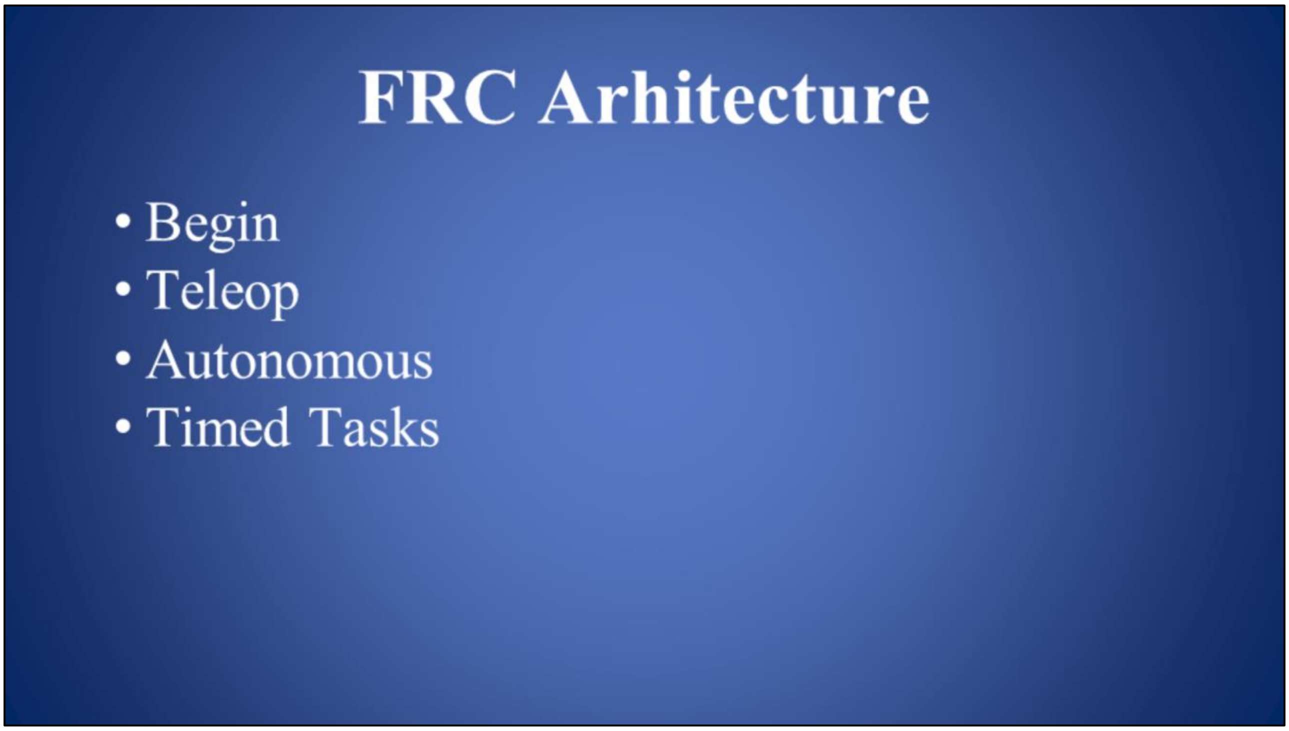
## FRC Arhitecture

- Begin
- Teleop
- Autonomous



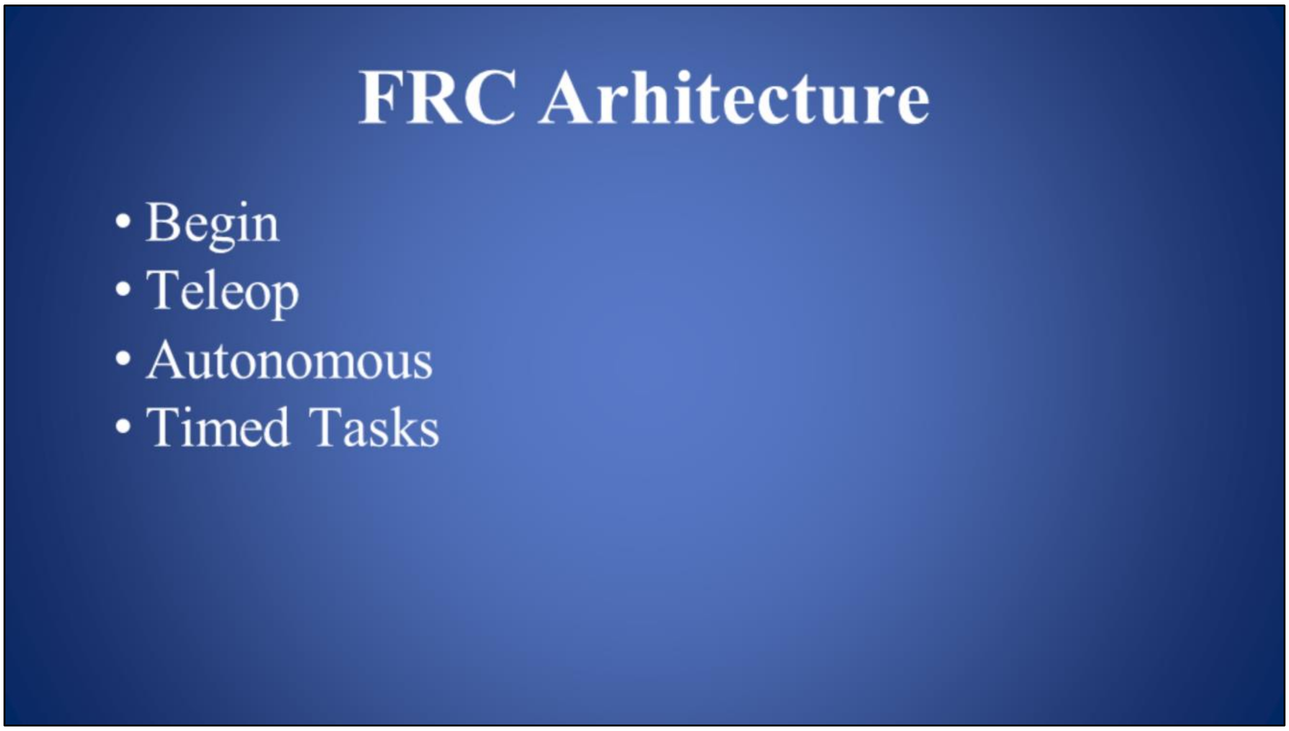
## FRC Arhitecture

- Begin
- Teleop
- Autonomous
  - Runs when Autonomous is enabled



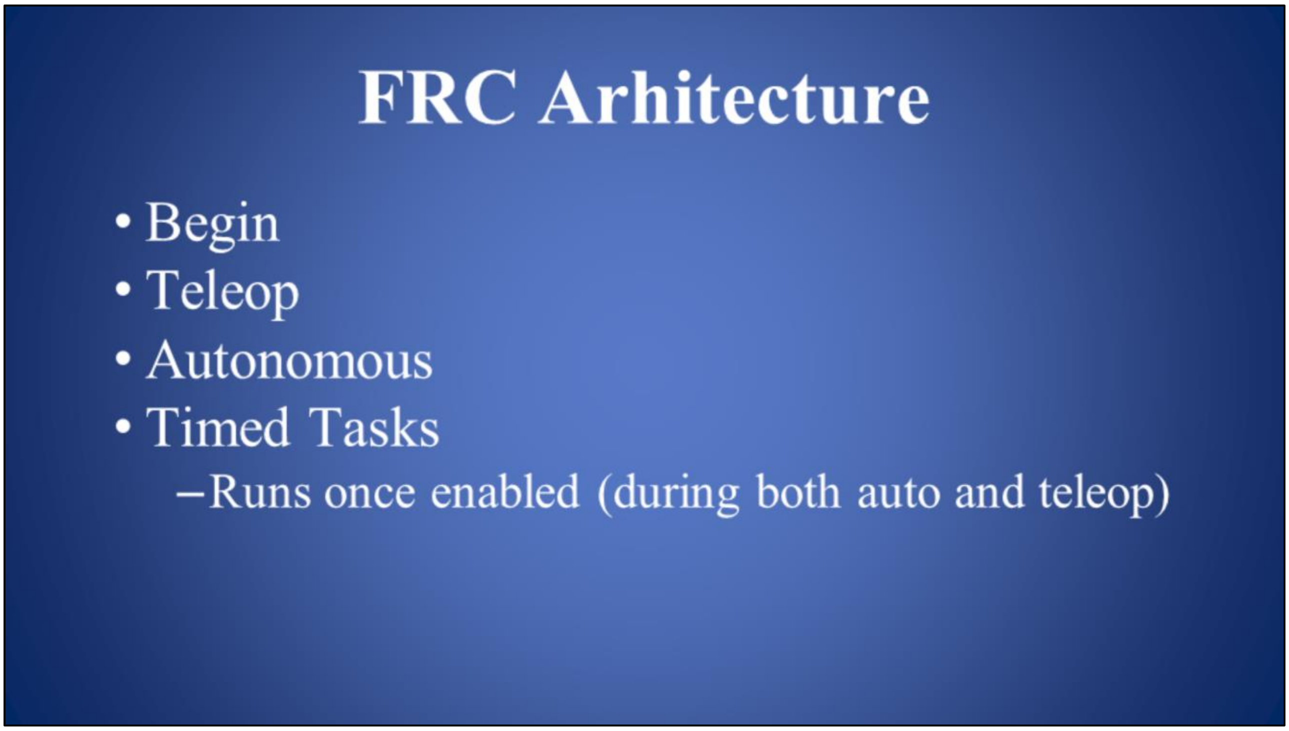
## FRC Arhitecture

- Begin
- Teleop
- Autonomous
- Timed Tasks



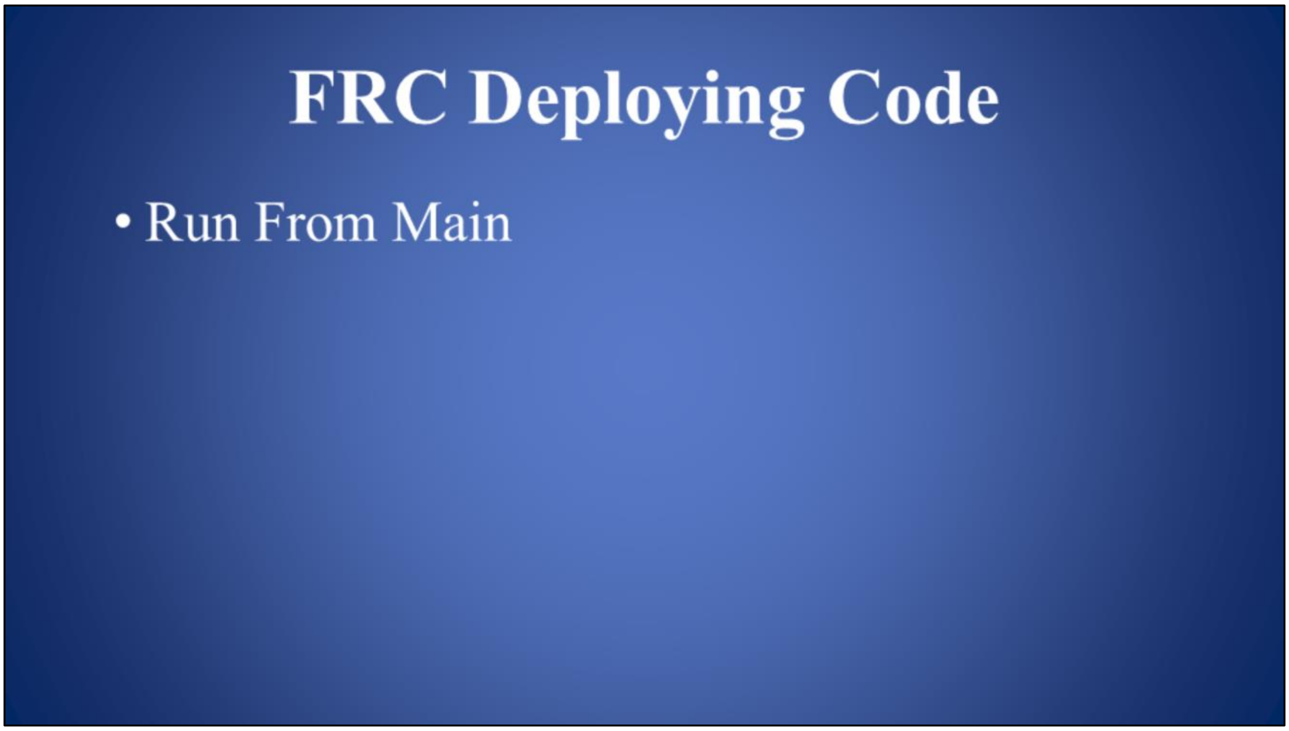
## FRC Arhitecture

- Begin
- Teleop
- Autonomous
- Timed Tasks



## FRC Arhitecture

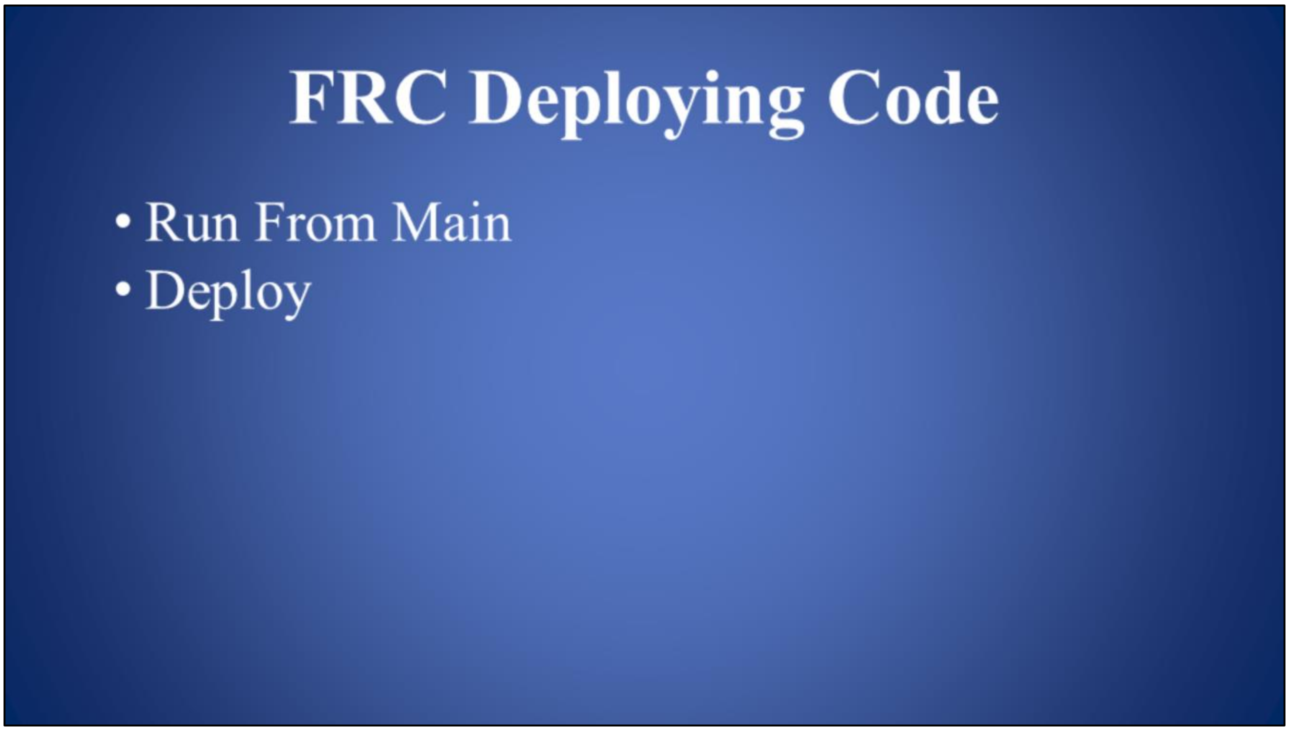
- Begin
- Teleop
- Autonomous
- Timed Tasks
  - Runs once enabled (during both auto and teleop)



## FRC Deploying Code

- Run From Main

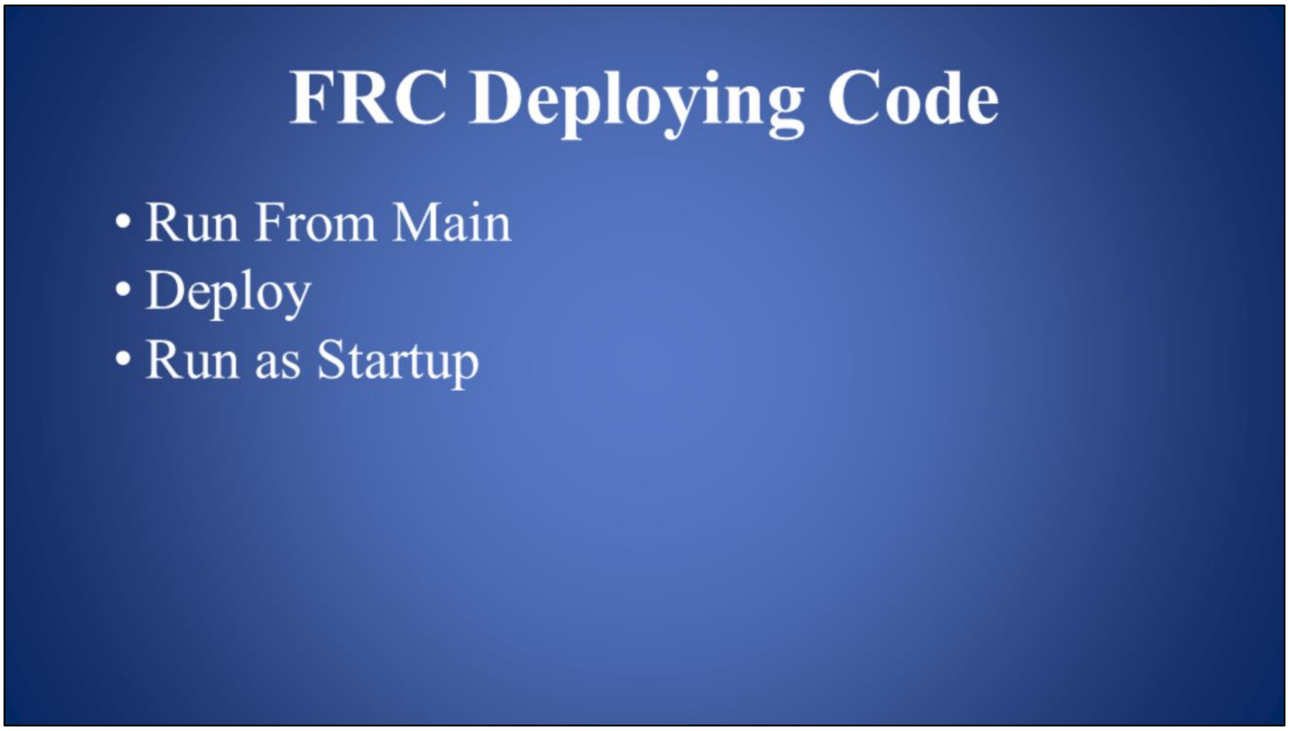
Open Main.vi (from project) – click the run buton



## FRC Deploying Code

- Run From Main
- Deploy

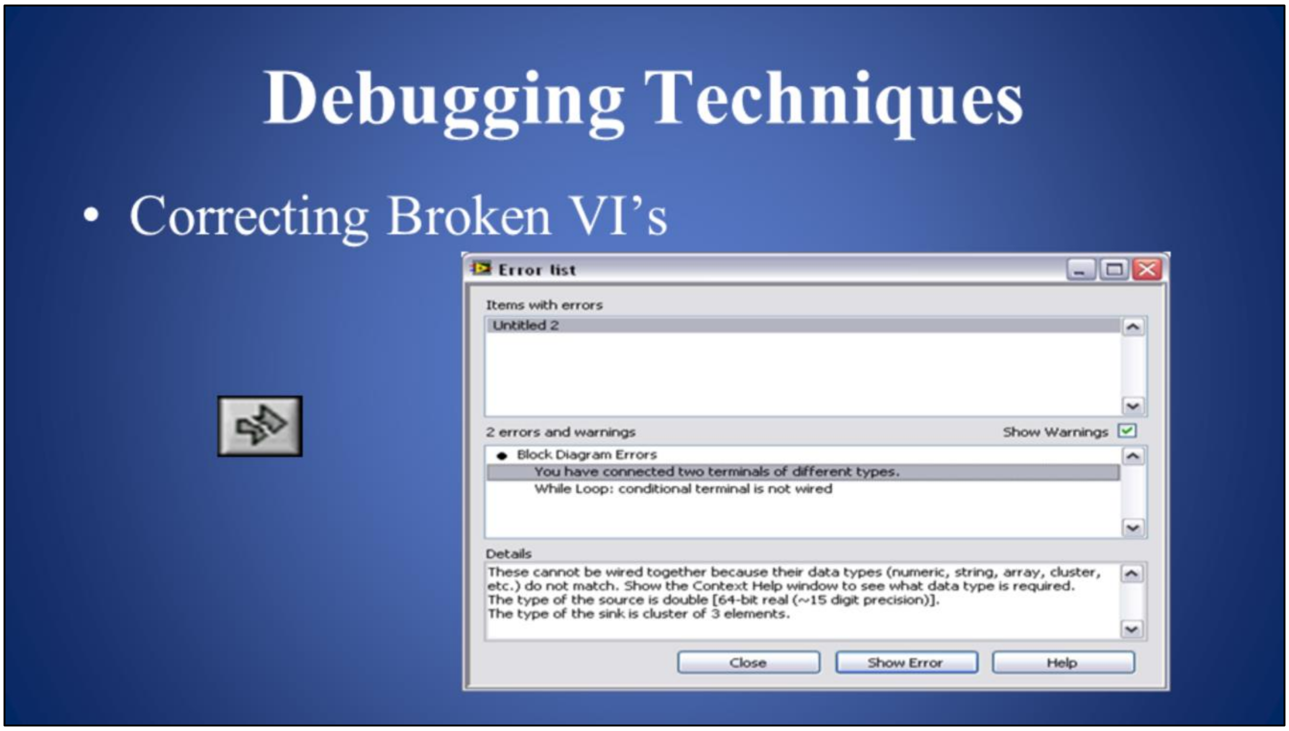
Deploying the robot code puts it on the robot – causing it to take effect on next reboot (or after Run as startup)



## FRC Deploying Code

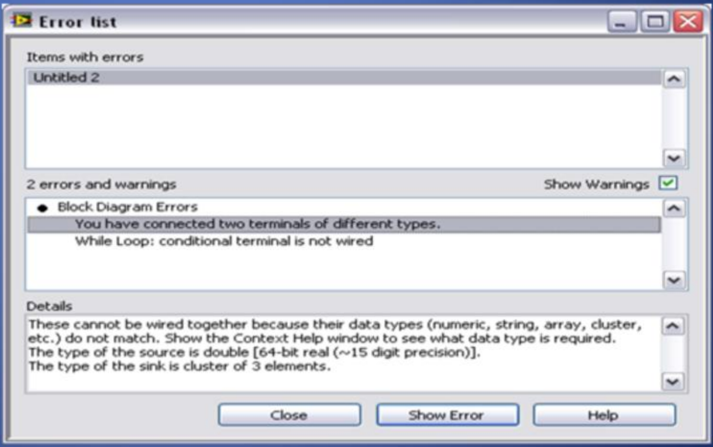
- Run From Main
- Deploy
- Run as Startup

Run as startup puts the code on the roborio in a temporary location and starts it running (does not persist across reboots).



## Debugging Techniques

- Correcting Broken VI's



Error list

Items with errors

Untitled 2

2 errors and warnings

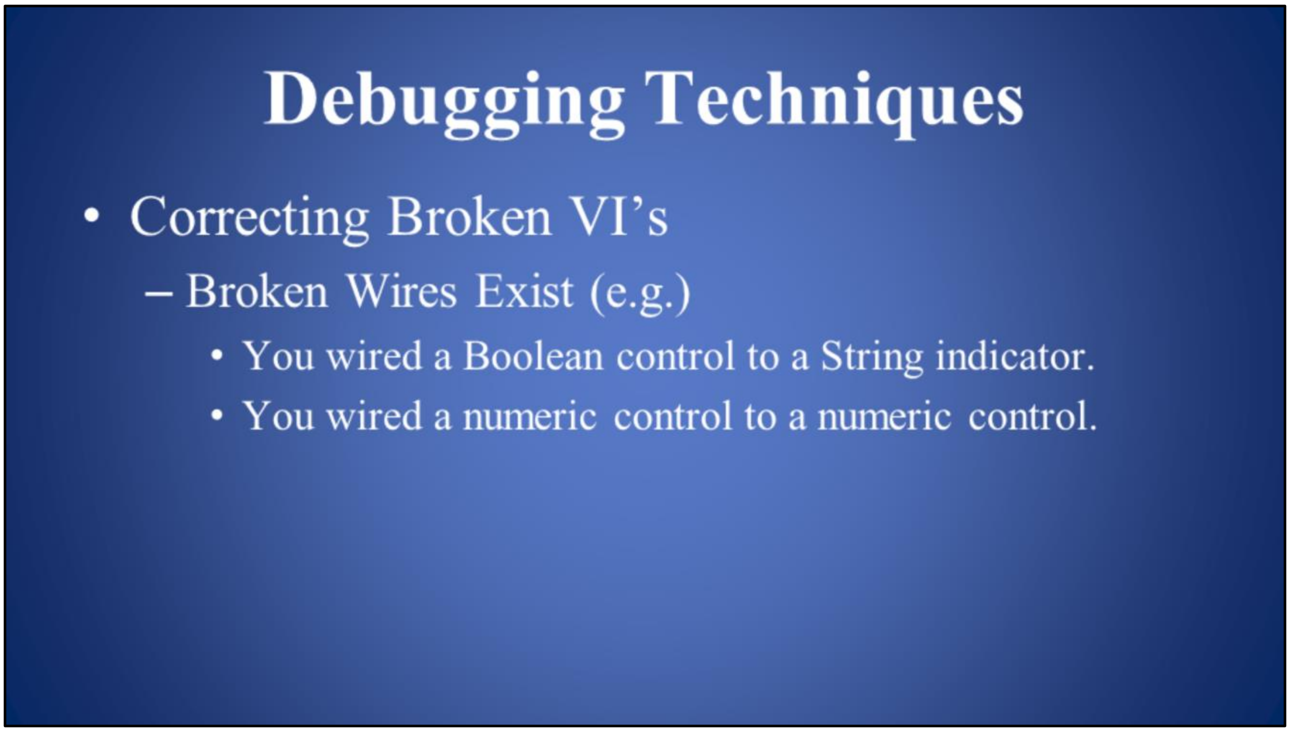
● Block Diagram Errors

You have connected two terminals of different types.

While Loop: conditional terminal is not wired

Details

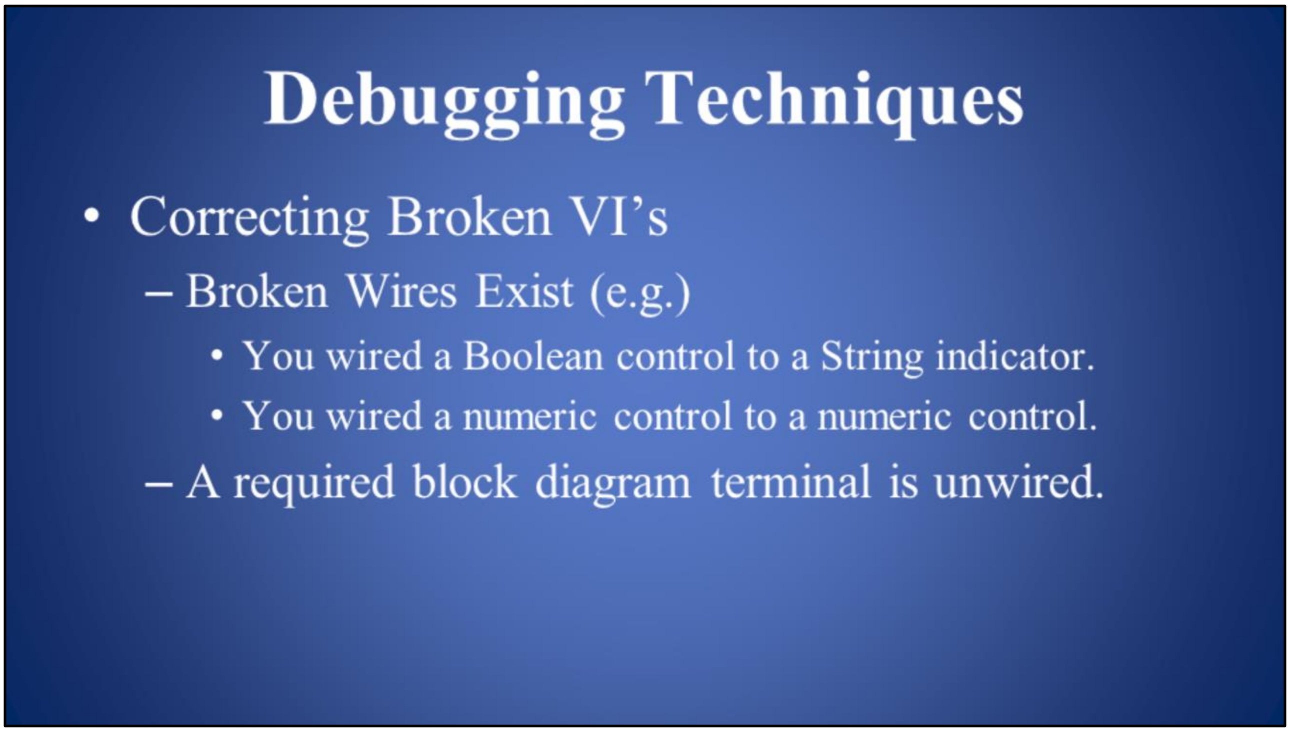
These cannot be wired together because their data types (numeric, string, array, cluster, etc.) do not match. Show the Context Help window to see what data type is required.
The type of the source is double [64-bit real (~15 digit precision)].
The type of the sink is cluster of 3 elements.



## Debugging Techniques

- Correcting Broken VI's
  - Broken Wires Exist (e.g.)
    - You wired a Boolean control to a String indicator.
    - You wired a numeric control to a numeric control.

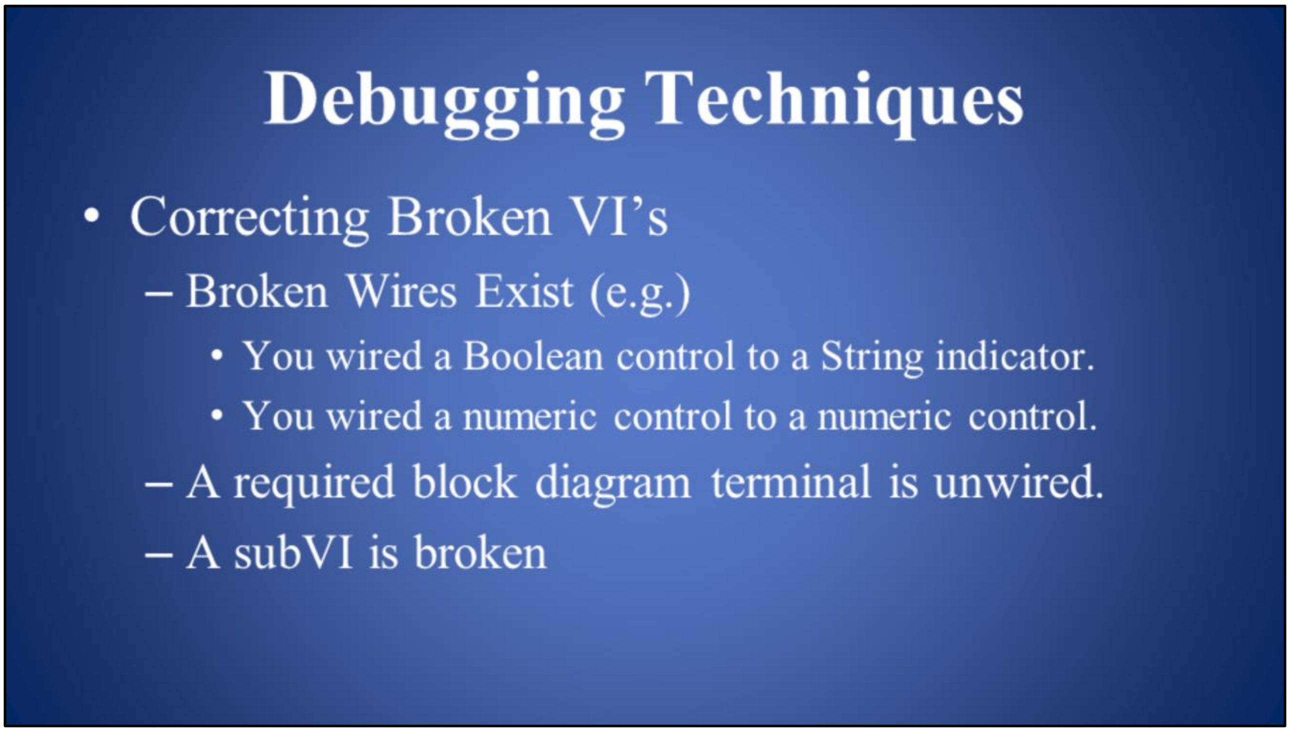
Untitled1.vi



## Debugging Techniques

- Correcting Broken VI's
  - Broken Wires Exist (e.g.)
    - You wired a Boolean control to a String indicator.
    - You wired a numeric control to a numeric control.
  - A required block diagram terminal is unwired.

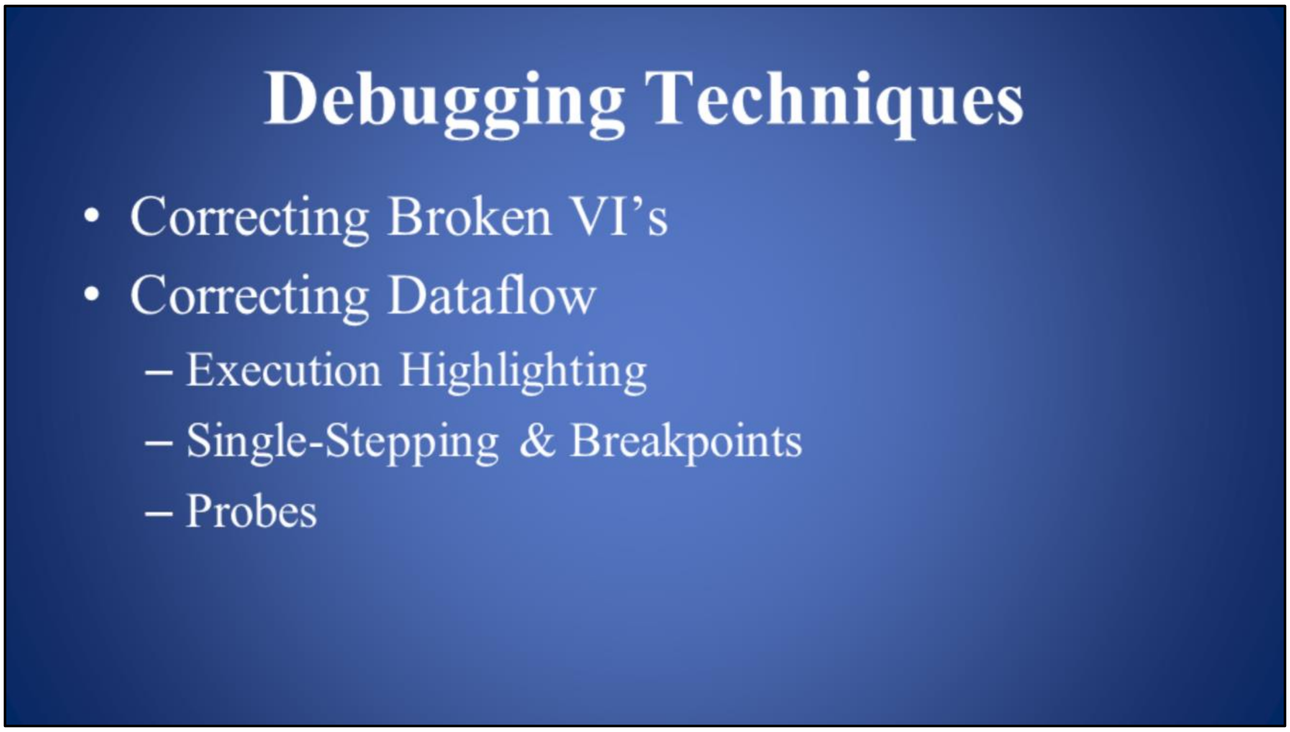
Untitled1.vi



## Debugging Techniques

- Correcting Broken VI's
  - Broken Wires Exist (e.g.)
    - You wired a Boolean control to a String indicator.
    - You wired a numeric control to a numeric control.
  - A required block diagram terminal is unwired.
  - A subVI is broken

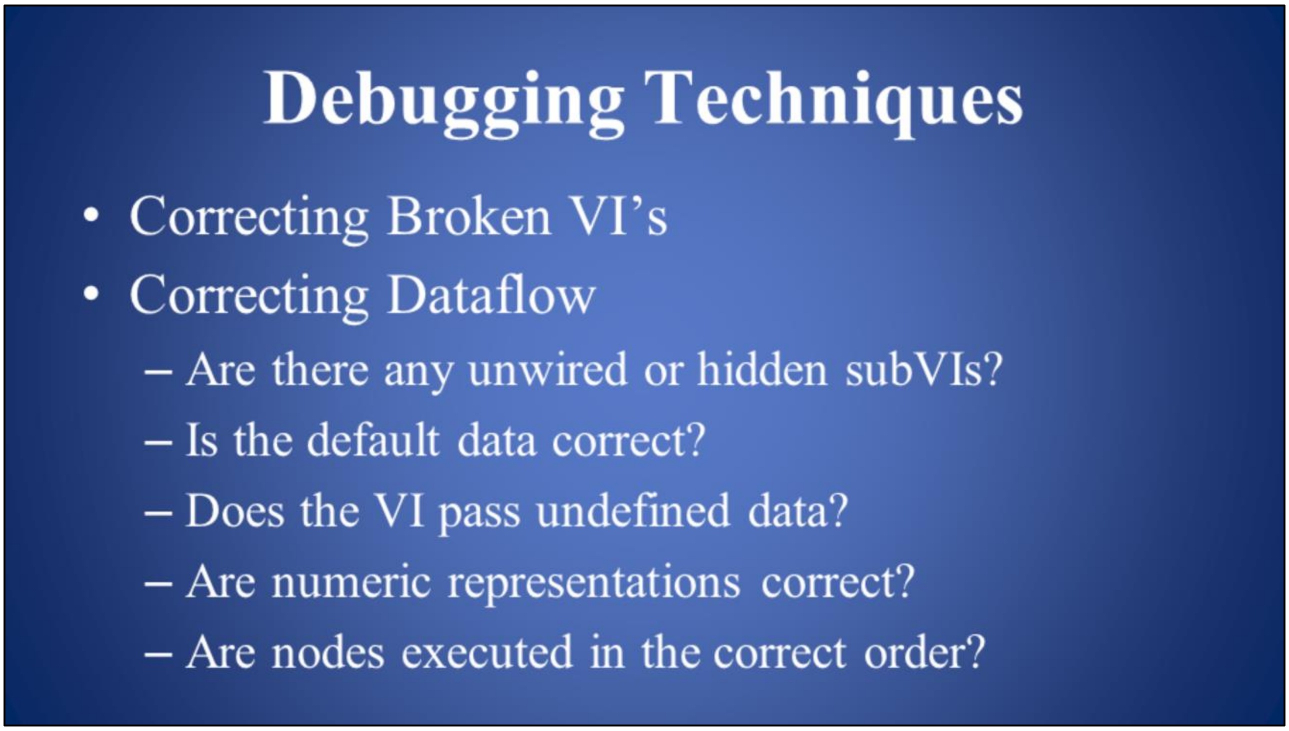
Untitled1.vi



## Debugging Techniques

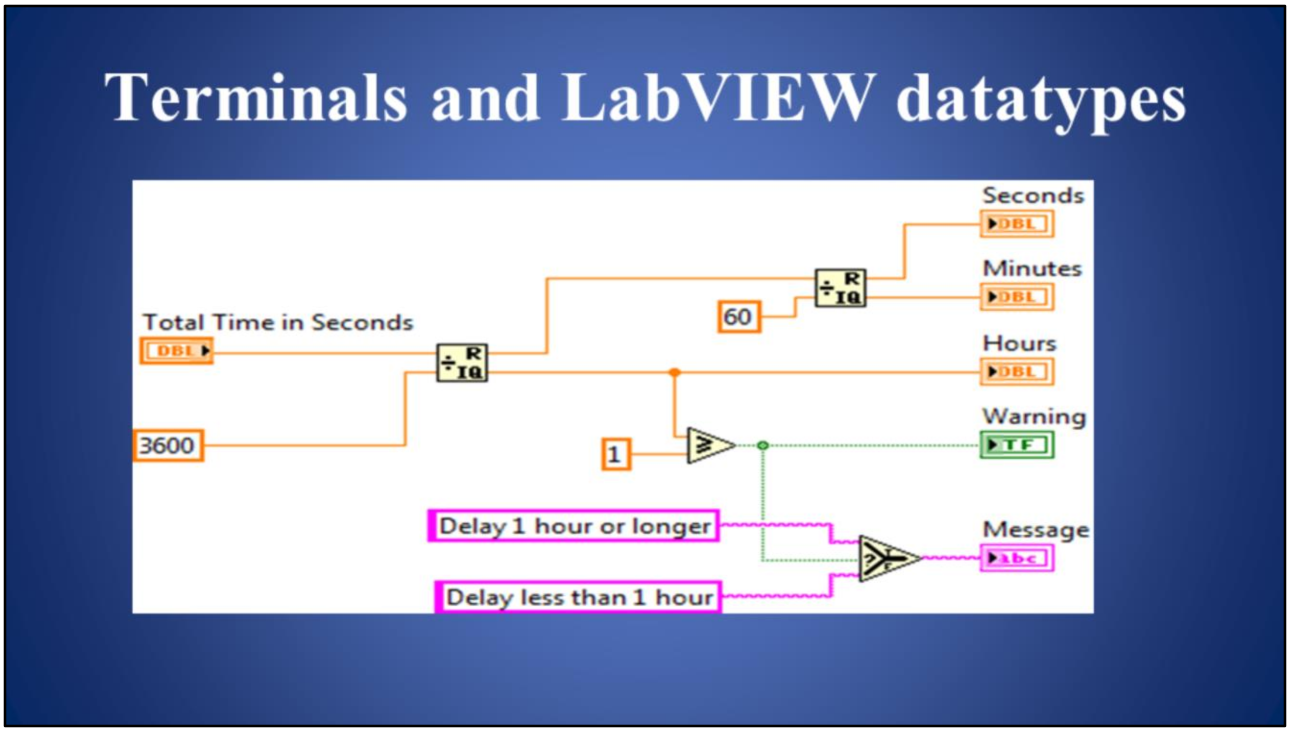
- Correcting Broken VI's
- Correcting Dataflow
  - Execution Highlighting
  - Single-Stepping & Breakpoints
  - Probes

untitled1



## Debugging Techniques

- Correcting Broken VI’s
- Correcting Dataflow
  - Are there any unwired or hidden subVIs?
  - Is the default data correct?
  - Does the VI pass undefined data?
  - Are numeric representations correct?
  - Are nodes executed in the correct order?


## Terminals and LabVIEW datatypes

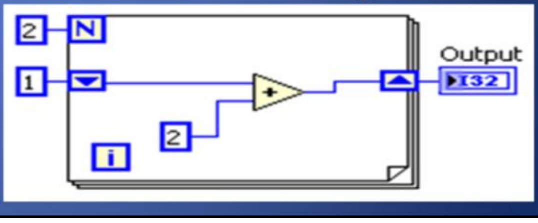
Terminal colors, text, arrow direction, and border thickness all provide visual information about the terminal.

For example, Orange represents floating point numbers.   DBL indicates a double-precision floating point number.

Terminals with thick borders with arrows on the right are control terminals.
Terminals with thin borders with arrows on the left are indicator terminals.

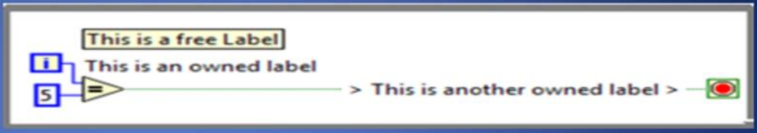
## Data Feedback in Loops

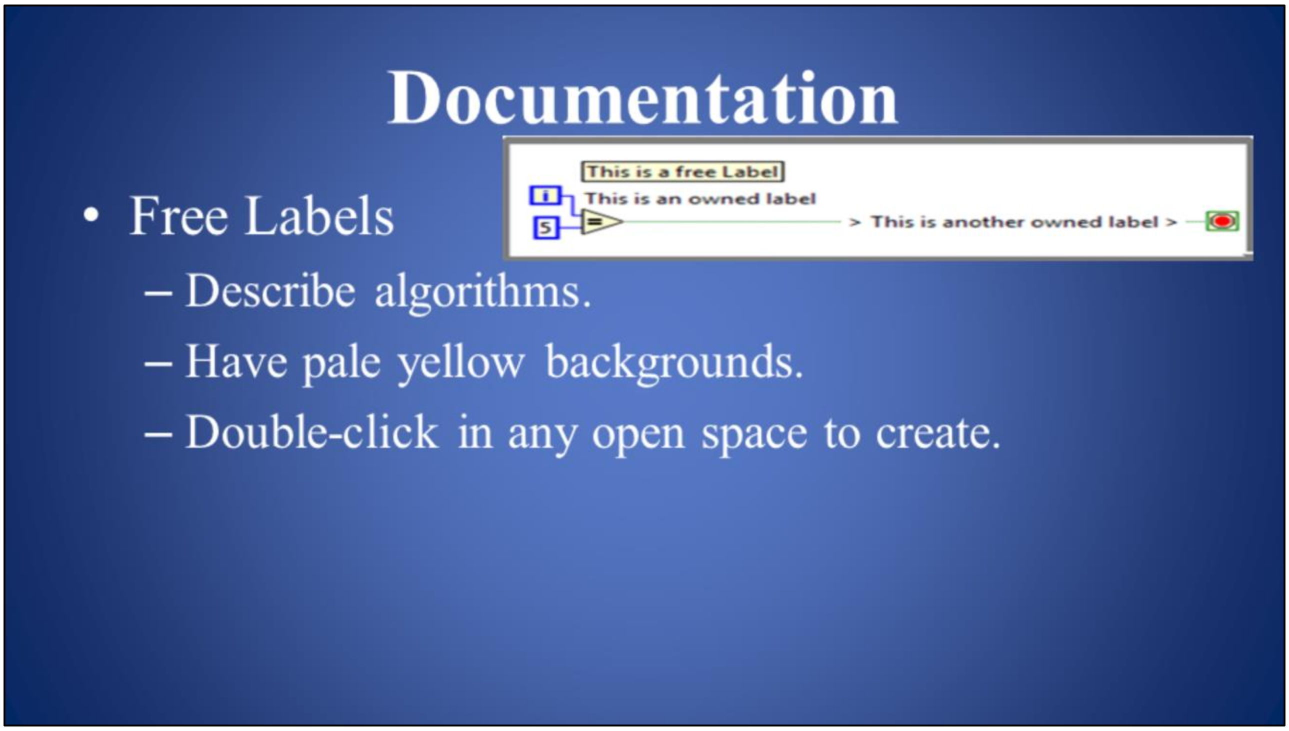
- Shift Registers
  - When programming with loops, you often need to know the values of data from previous iterations of the loop.
  - Shift registers transfer values from one loop iteration to the next.



## Documentation

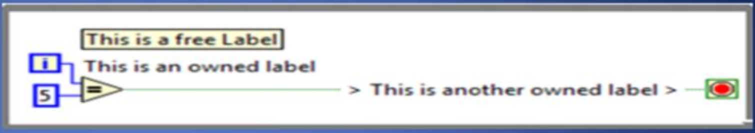
- Free Labels

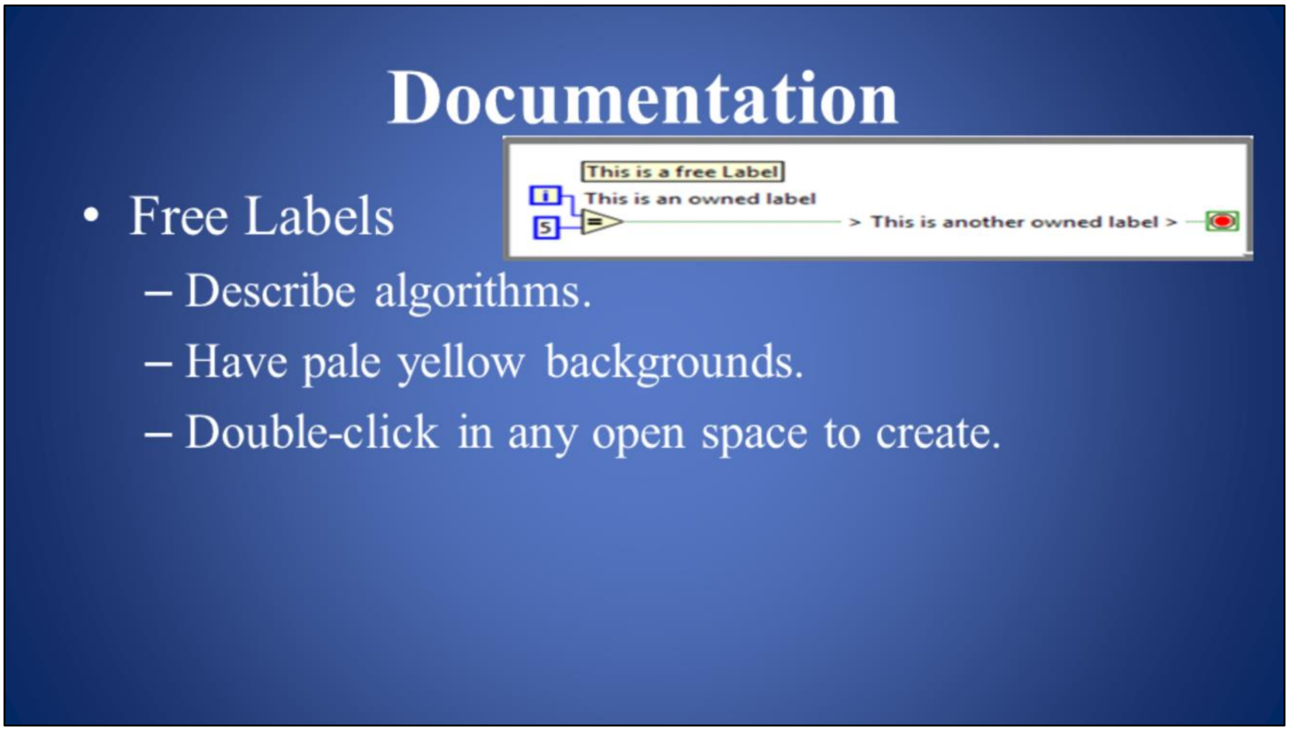




## Documentation

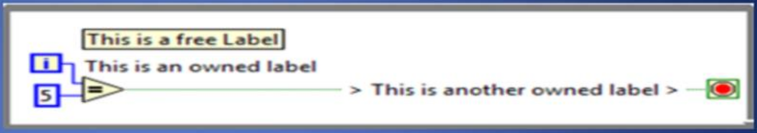
- Free Labels
  - Describe algorithms.
  - Have pale yellow backgrounds.
  - Double-click in any open space to create.



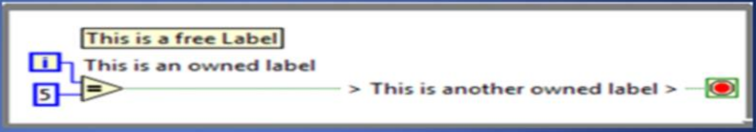


## Documentation

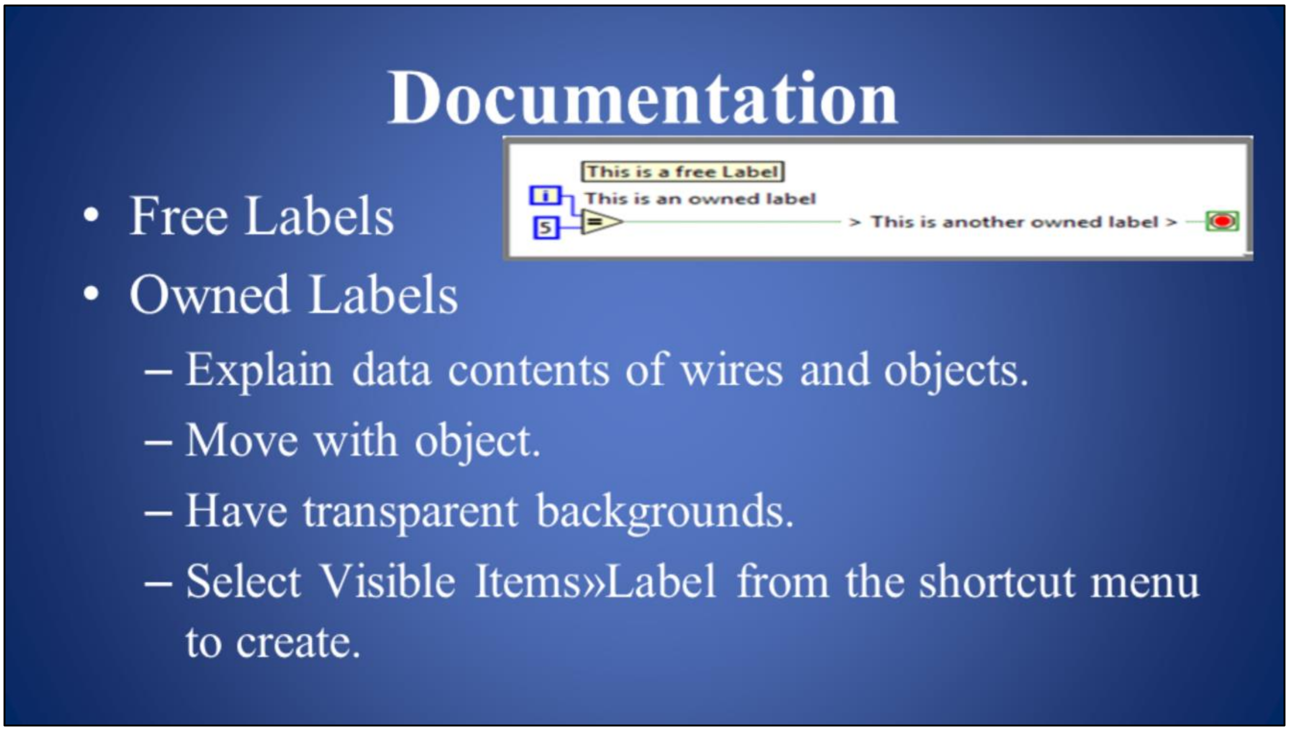
- Free Labels
  - Describe algorithms.
  - Have pale yellow backgrounds.
  - Double-click in any open space to create.



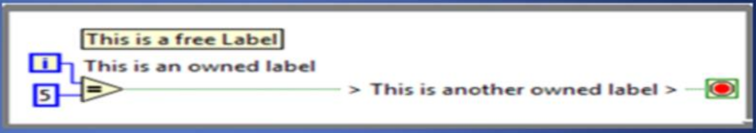
## Documentation



- Free Labels
- Owned Labels
  - Explain data contents of wires and objects.
  - Move with object.
  - Have transparent backgrounds.
  - Select Visible Items»Label from the shortcut menu to create.



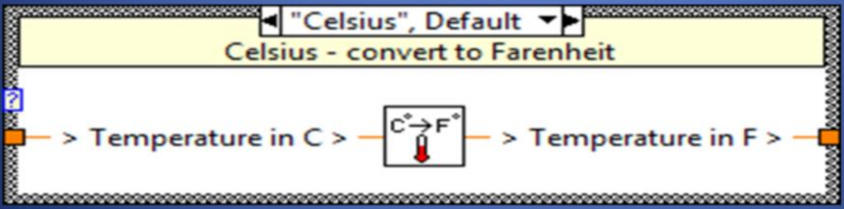
## Documentation



- Free Labels
- Owned Labels
  - Explain data contents of wires and objects.
  - Move with object.
  - Have transparent backgrounds.
  - Select Visible Items»Label from the shortcut menu to create.

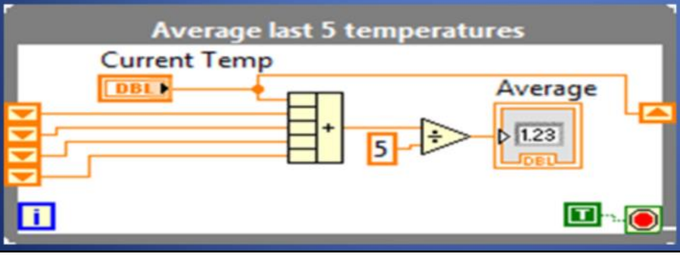
# Documentation

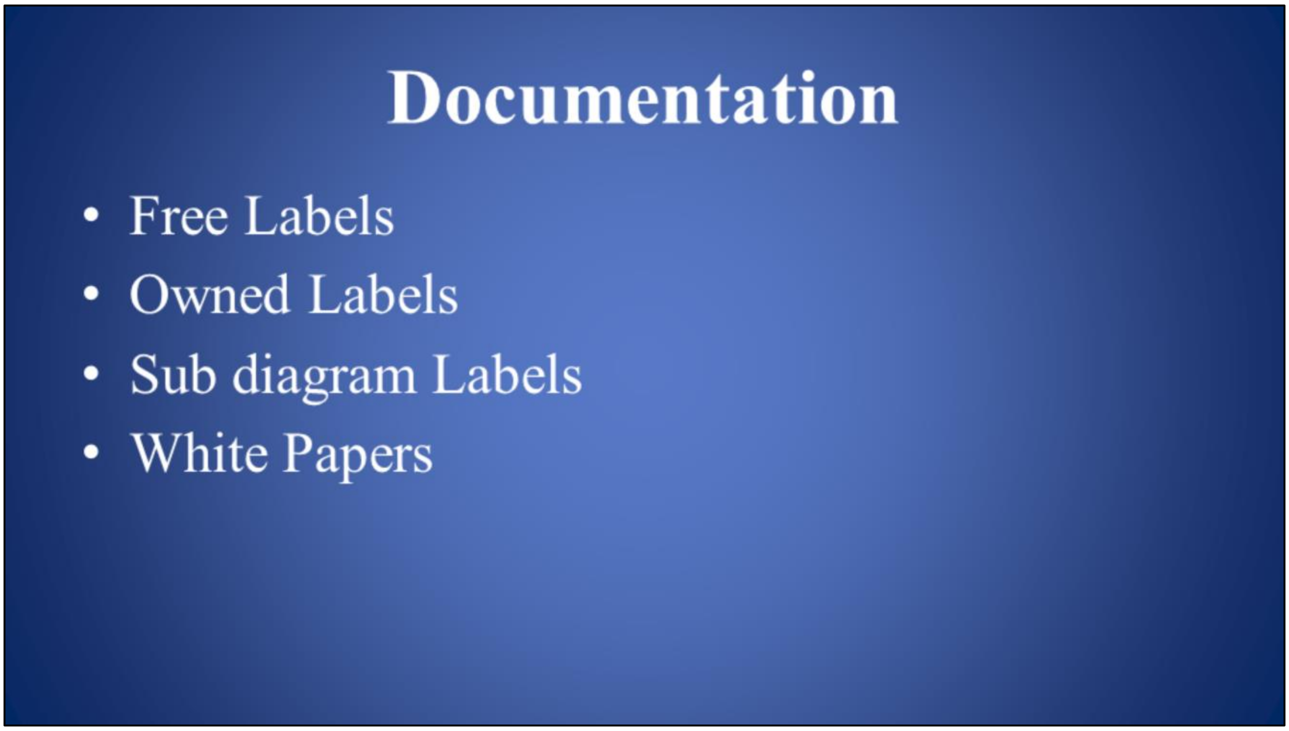
- Free Labels
- Owned Labels
- Sub diagram Labels



## Documentation

- Free Labels
- Owned Labels
- Sub diagram Labels
  - Case Structures

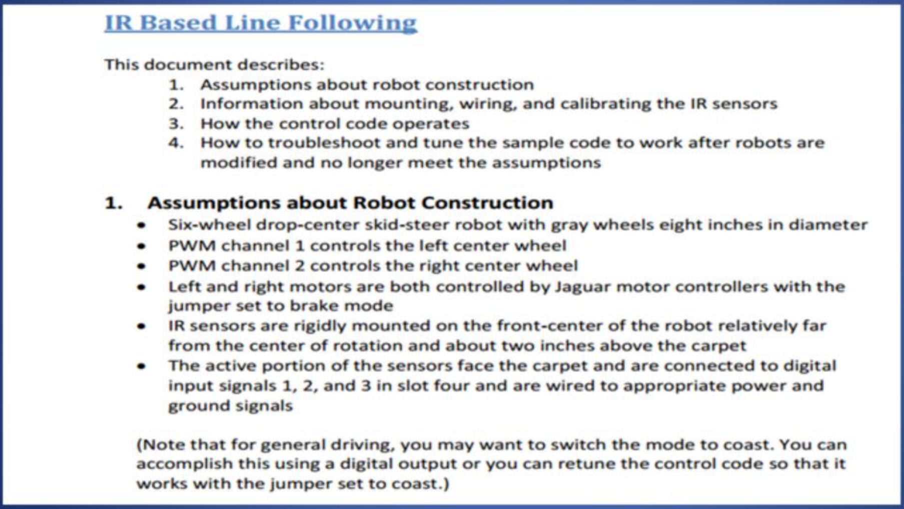




## Documentation

- Free Labels
- Owned Labels
- Sub diagram Labels
- White Papers

## Documentation



IR Based Line Following

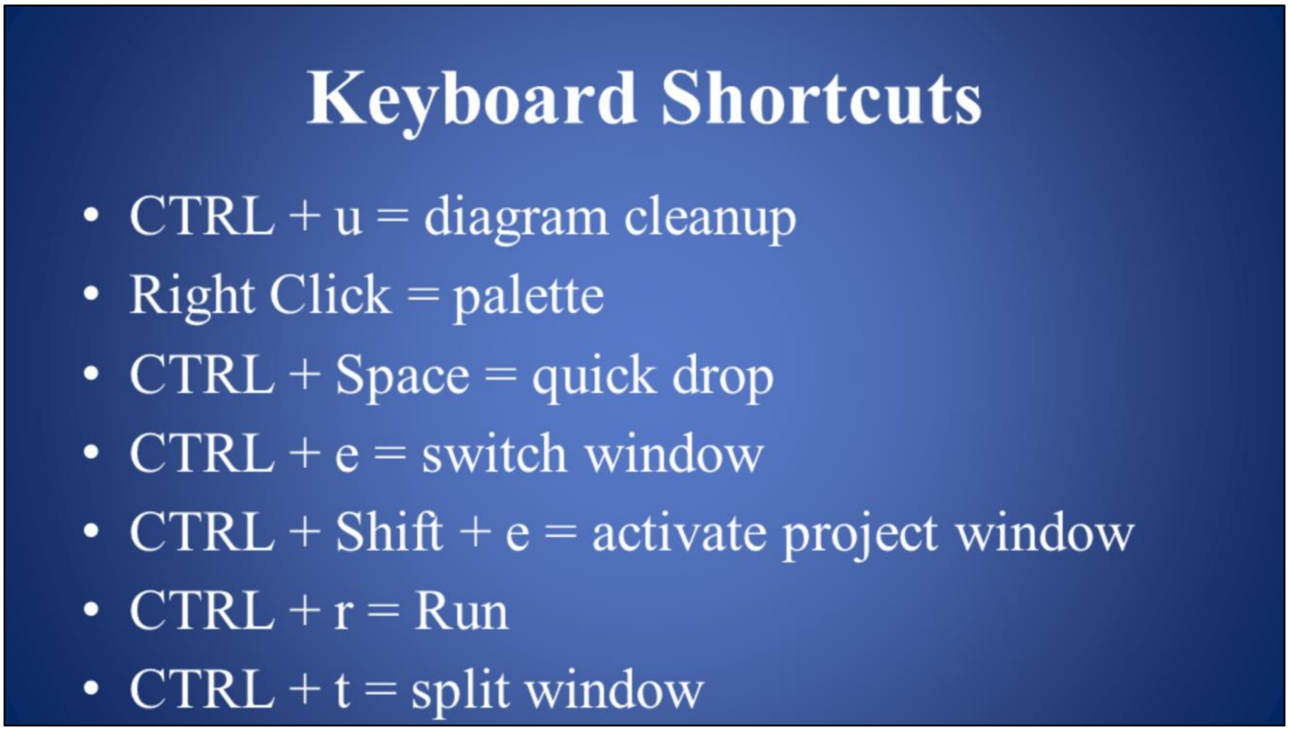
This document describes:

1. Assumptions about robot construction
2. Information about mounting, wiring, and calibrating the IR sensors
3. How the control code operates
4. How to troubleshoot and tune the sample code to work after robots are modified and no longer meet the assumptions

**1. Assumptions about Robot Construction**

- Six-wheel drop-center skid-steer robot with gray wheels eight inches in diameter
- PWM channel 1 controls the left center wheel
- PWM channel 2 controls the right center wheel
- Left and right motors are both controlled by Jaguar motor controllers with the jumper set to brake mode
- IR sensors are rigidly mounted on the front-center of the robot relatively far from the center of rotation and about two inches above the carpet
- The active portion of the sensors face the carpet and are connected to digital input signals 1, 2, and 3 in slot four and are wired to appropriate power and ground signals

(Note that for general driving, you may want to switch the mode to coast. You can accomplish this using a digital output or you can retune the control code so that it works with the jumper set to coast.)



## Keyboard Shortcuts

- CTRL + u = diagram cleanup
- Right Click = palette
- CTRL + Space = quick drop
- CTRL + e = switch window
- CTRL + Shift + e = activate project window
- CTRL + r = Run
- CTRL + t = split window