2024 *FIRST*® Robotics Competition

KitBot LabVIEW Software Guide

1 Contents

2 Document Overview

This document will take you through how to get your 2024 KitBot up and running using the provided LabVIEW example code. To avoid content duplication this document frequently links to WPILib or other documentation for accomplishing specific steps along the way. In addition to getting you up and running with the provided code, this document will walk through the structure of that code so you can understand how it operates. Finally, we'll walk through some of the most likely changes you may wish to make to the code and provide concrete examples of how to make those modifications.

To get started with the example code, or to make some of the modifications described, minimal understanding of LabVIEW is required. The code and modification examples provided will likely provide enough of a pattern to get you going. To understand the walkthrough, or to make modifications not described in this document, a more thorough understanding of LabVIEW is likely required. The NI LabVIEW Tutorial and the NI LabVIEW for FRC pages have resources to help you get started.

This document, and the provided example code, assumes the use of the SPARK MAX controllers provided in the rookie Kickoff Kit.

3 Getting Started with your KitBot code

3.1 Wiring your robot

Use the WPILib Zero-to-Robot wiring document to help you get your robot wired up. Some notes specific to the 2024 KitBot:

- The 2024 KitBot does not utilize pneumatics. You can skip instructions regarding the Pneumatic Hub/Pneumatics Control Module unless you are adding pneumatics to the design.
- In order to use the same IDs for PWM and CAN operation, the 2024 KitBot code does not utilize PWM port 0. Either wire the PWM ports according to the IDs in Begin.vi (Left = 1,2, Right = 3,4) or modify the constants to reflect your wiring.
- The 2024 KitBot contains two additional motors not included in the basic wiring document. Wire these in the same manner as the drivetrain motors. If using PWM, connect the Feeder motor (closer to the center of the robot) to PWM port 5 and the Launcher motor (the motor closer to the outside of the robot) to PWM port 6.

3.2 Configuring hardware and development environment

Before you are able to load code and test out your robot, you will need to configure your hardware (roboRIO, radio, etc.) and get your development environment set up. Follow the WPILib Zero-to-Robot guide steps 2 through 4 to get everything set up and ensure you can deploy a basic robot project.

If using PWM, make sure all 6 SPARK MAXs are in “Brushed” mode. When powered the LED should blink yellow or blue, not magenta or cyan. To change the mode, you can either hold the Mode button down for 3 seconds or use the USB connection and REV Hardware client. You may also wish to check the Idle modes, brake or coast. The Feeder and Launcher motors are recommended to be set to coast mode (blinking yellow). To change the Idle mode, press the Mode button briefly (less than 3 seconds) or use the USB connection and REV Hardware Client. There is no specific recommendation for the drivetrain motors, but you likely want all 4 drivetrain motors to match, you may wish to try driving around with each setting to decide what you prefer.

3.3 Opening the 2024 KitBot Example

The 2024 KitBot example code is provided in individual zip files for each language on the KitBot webpage. To open the LabVIEW code:

1. Download and unzip the LabVIEW example code. Make sure to unzip or copy to a permanent location, not in a temporary folder.
2. The easiest way to open up the code is to leverage the default file associations LabVIEW sets up. Open up the unzipped folder and locate the “2024 Kitbot” file and double click it. This file is the Project file and double clicking it should launch LabVIEW 2023 with the project opened.

3. If you wish to open the project from within LabVIEW instead (e.g. if you have multiple LabVIEW versions installed on your machine), launch LabVIEW 2023 and select File->Open Project. Browse to the folder where you unzipped the code and select the 2024 KitBot file, then click OK.

3.4 Changing to CAN control

If you have wired your SPARK MAX motor controllers using CAN, you will need to do some further configuration and code modification before proceeding. If you are using PWM, skip down to Section 3.5 to deploy and test the code.

3.4.1 Configuring the SPARK MAXs

Before using the SPARK MAXs with CAN control, they each need to be assigned a unique ID. Because your SPARKs all start with the same ID you may wish to unplug the CAN bus from each device while you update and assign an ID.

1. Install the REV Hardware Client
2. With the robot powered off, connect a USB cable between the computer and the SPARK MAX USB port. Leaving the robot powered off ensures only the single SPARK MAX is powered and avoids changing the IDs on unintended devices.
3. Update the firmware on the SPARK MAX
4. Set the CAN ID and Motor Type (you can skip the current limit) and save the settings
    a. CAN IDs for each device can be found in Begin.vi. You can either set the devices to match these IDs or set the IDs as desired (some teams set the CAN ID = the channel number the device is attached to on the PD) and then update these constants.
    b. Note: If you wish to “Spin the motor” as described on that webpage, make sure the robot is in a safe state to do so (wheels not touching the ground or table).
5. Repeat for all 6 devices on the robot.
6. While not required, if using the REV PDH you may wish to check that it has the latest firmware at this time as well. Do not change the ID of the PDH off of the default, each device type has a separate ID space and your PDH will not conflict with your SPARK MAX even if set to the same ID.

Now that all your devices are configured, you can do a preliminary check that your CAN bus is wired properly using the REV Hardware client. While plugged into any REV device on your CAN bus with a USB cable, power on the robot and you should see all the other devices listed in the left pane of the REV Hardware Client, under the CAN Bus heading. If you don’t see all of the devices, you likely have one or more issues with your CAN bus wiring:

1. Verify that your CAN bus starts with the roboRIO and ends with a 120 ohm resistor, or the built in terminator of a Power Distribution Hub or Power Distribution Panel (with the termination set to On using the appropriate jumper or switch.
2. Check that your CAN bus connections all match yellow-yellow and green-green.
3. Check that all CAN wire connections are secure to each other and that the connectors are securely installed in each SPARK Max

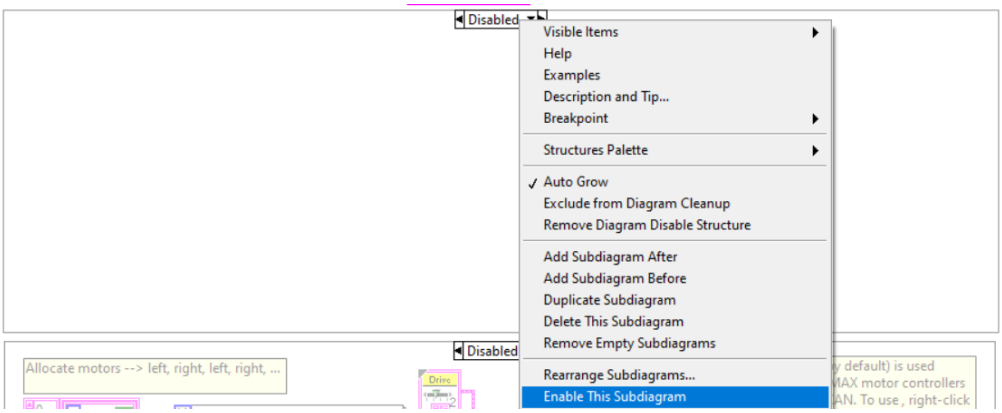
4. If you're still having trouble, moving the USB connection around to different devices and seeing what each device can "see" on the bus can help pinpoint the location of an issue.

3.4.2 Installing REVLib

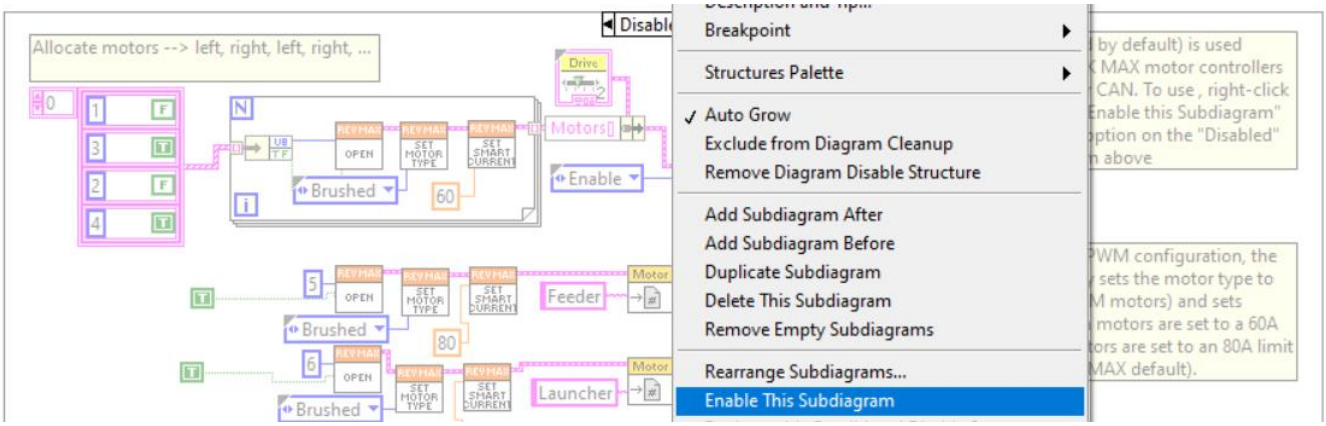
The software library for the SPARK MAX in CAN mode is provided by the vendor (REV Robotics). You will need to install this library, otherwise the CAN VIs for the SPARK MAX will show up as '?' icons. To do this, follow the instructions from the REVLib documentation page.

3.4.3 Updating the Code

The code is mostly in place in the project to switch from PWM to CAN control, you will just need to make a change in Begin.VI to switch over.



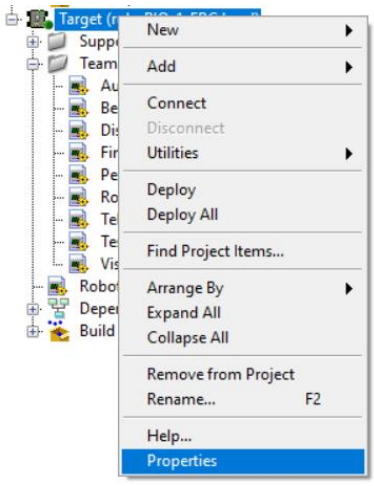
The top block of code for PWM control is Enabled in the project by default. Click the arrow next to "Enabled" to switch to the empty Disabled case, then right-click and select "Enable This Subdiagram". This will disable the PWM code and enable the blank diagram.



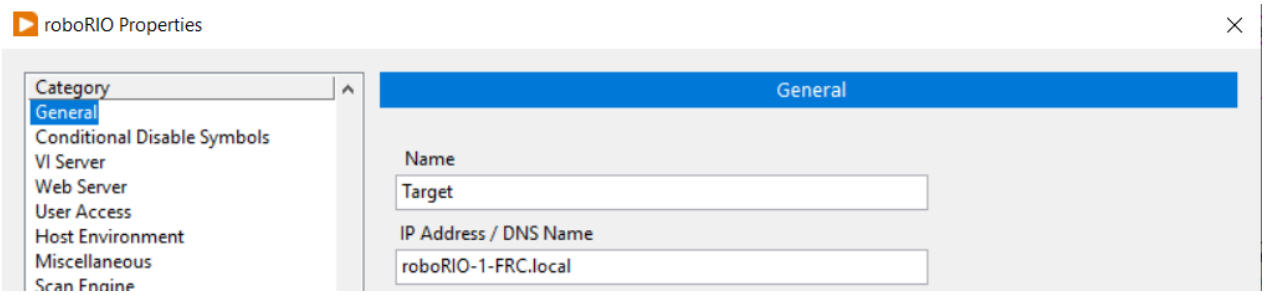
Next, enable the CAN diagram by right-clicking and selecting "Enable This Subdiagram".

3.5 Deploying and testing the KitBot Example

To deploy the example to your robot, you will need to set the Team Number on the project.



In the Project Explorer, right click on the line that says Target(...) and select Properties.



In the "IP Address / DNS Name" box, type **roborio-####-FRC.local** where #### is your team number (example for Team 1 shown). If you're having difficulty deploying, you can also try setting this to **172.22.11.2** for USB connection or **10.TE.AM.2** for ethernet or wireless connection to a roboRIO connected to a programmed radio where TE.AM is your 4 digit team number (e.g. 10.0.1.2 for Team 1, 10.3.12.2 for Team 312, and 10.75.43 for Team 7543).

You are now ready to run or deploy the KitBot example (remember, only Set as Startup will persist through a roboRIO reboot!) just like you deployed the test project in Step 4 of the Zero-to-Robot guide.

Warning: Make sure you have space in all directions when operating a robot. Even with known code, the robot may move with unexpected speed or in unexpected directions. Be prepared to Disable (Enter) or E-stop (Spacebar) the robot if necessary. The 2024 KitBot code contains a very simple autonomous routine that will move the robot backwards at ½ speed for 1 second when the robot is enabled in Autonomous mode.

3.6 Configuring Gamepads

The code is set up to use the Xbox controller class. The Logitech F310 gamepads provided in the Kit of Parts will appear like Xbox controllers to the WPILib software if they are configured in the correct mode. To set up the controllers, check that the switch on the back of the controller is set the 'X' setting. Then when using the controller, make sure the LED next to the Mode button is off, if it is on press the

Mode button to toggle it. When the Mode button is on, the controller swaps the function of the left Analog stick and the D-pad.

3.7 What does the code do?

The provided code implements the following robot controls in Teleoperated:

- Driver controller is an Xbox Controller in Slot 0 of the Driver Station
    - Controls the robot drivetrain using Split-stick Arcade Drive
        - Y-axis (vertical) of left stick controls forward-back movement of drivetrain
        - X-axis (horizontal) of right stick controls rotation of drivetrain
- Operator controller is an Xbox Controller in Slot 1 of the Driver Station
    - Left Bumper – Runs both note launcher wheels inward at different speeds while the button is held. This lets the robot intake a Note
    - A button – Runs a short sequence to launch a Note while the button is held
        - Starts front wheel running to get up to speed
        - Waits 1 second
        - Runs back wheel to feed Note into spinning front wheel

4 Overall Code Structure

The provided code utilizes the default LabVIEW Code structure. This consists of the following VIs. These VIs are in the Team Code folder of the Project Explorer, listed alphabetically. The explanations here are arranged in a more chronological sense.

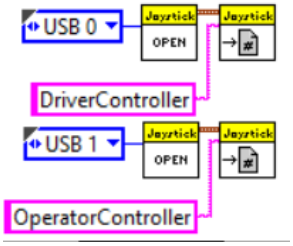
- **Begin.VI** – Used for initializing hardware objects and data. The KitBot code uses this VI for hardware set up.
- **Periodic Tasks.VI** – This VI is called once, but contains loops that run periodically regardless of robot mode. For the KitBot, control of the launcher is handled here. Teams often choose to put advanced mechanism control here so it can be leveraged in both Autonomous and Teleop modes.
- **Disabled.VI** – This code is called periodically (~50hz) while the robot is disabled. You could use this to reset state of things, control lights on the robot, react to autonomous mode selection or any other actions you want the code to do while the robot is disabled (remember, you can't control actuators in Disabled mode!). The KitBot code does not modify the default contents of this VI.
- **Autonomous Independent.VI** – This VI is called once when auto mode begins and is automatically cancelled when auto mode ends. You can either structure your auto mode as a sequence, or add a loop inside if you want to set up a state machine. The KitBot code contains a basic auto to drive backwards at 50% speed for 1 second in addition to the default code in this VI.
- **Robot Global Data.vi** – This VI holds data that you want to access from different parts of the robot program. The KitBot code adds an enumeration for conveying desired Launcher state to the default values.
- **Test.VI** – This VI is called once when the robot is enabled in Test mode and is automatically cancelled when the robot is disabled or the mode changes. You could write code in this VI to test robot functionality, either automatically, or using driver inputs. The KitBot code does not modify the default contents of this VI.
- **Finish.VI** – This VI is called when the Finish button is pressed when running the code from PC. This VI is never called when code is deployed to the robot. Most teams do not utilize this VI. The KitBot code does not modify the default contents of this VI.
- **Vision Processing.VI** – This VI can be used for adding vision processing code. This VI is not run by default, if you choose to use this, you will need to add it to Robot Main.VI yourself, using the other 4 VIs in the bottom section (such as the Timed Tasks icon) as an example. The KitBot code does not modify the default contents of this VI.

5 Code Walkthrough

5.1 Begin.VI

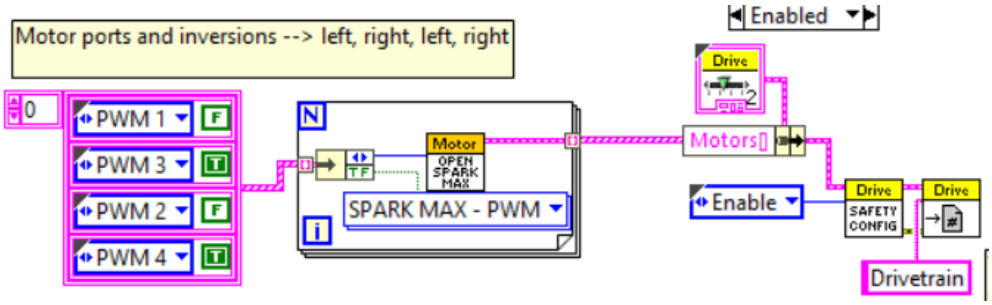
As described in the Section 4, Begin.VI is generally used to open references to all robot hardware and to set configurations. The KitBot code contains a few pieces from the default LabVIEW project template, omitted here, and some KitBot specific code described below

5.1.1 Controller References



This code from the top center of the diagram opens a reference to a controller connected to ports 0 and 1 on the Driver Station and saves those references to RefNums called DriveController and OperatorController respectively.

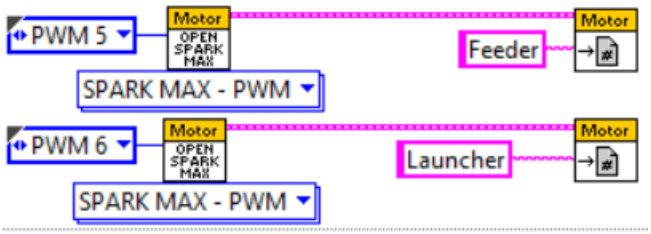
5.1.2 PWM Drivetrain



This section of code in the top of the upper Enable/Disable diagram is for setting up the motor controllers for the drivetrain for PWM control. As noted in the comment, the order of the constants in the array is left, right, left, right. Meaning this code sets up the left controllers on ports 1, and 2 with no inversion and the right controllers on ports 3 and 4 and inverts them. This inversion sets the drivetrain up to work correctly with joystick axis values passed in directly. If, instead, you want both sides of the drivetrain to move forward when passed positive values, invert the right side instead (and negate the joystick axis values in Teleop.VI).

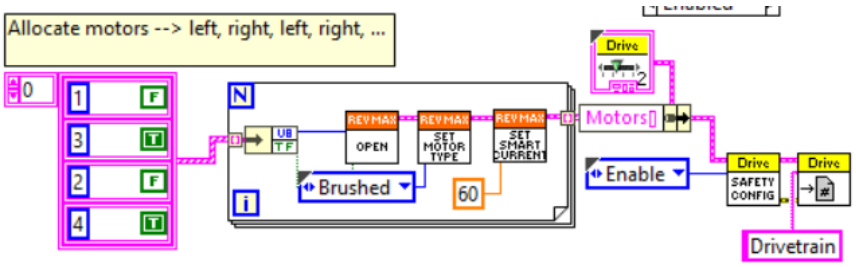
The center section of this code specifies that the motor controllers are SPARK MAX controllers connected over PWM. Then the code enables the Motor Safety feature on the drivetrain, which will disable the motors if you do not provide them an updated command every 100ms. Finally we store the reference in a RefNum called Drivetrain.

5.1.3 PWM Launcher



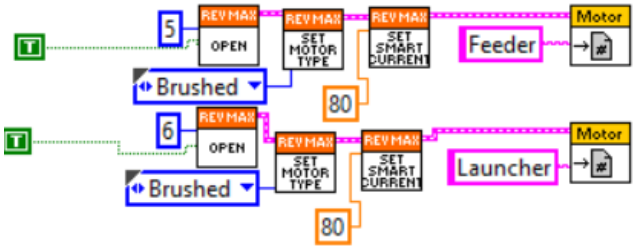
Below the PWM drivetrain code is this code for the PWM Launcher. This code is pretty simple, it opens a reference to the two SPARK MAXs, on ports 5 and 6, and stores the reference in refnums.

5.1.4 CAN Drivetrain



In the next Enable/Disable structure (disabled by default) is the code for setting up the drivetrain if you have connected your motor controllers over CAN. If you have not installed REVLib properly, or not restarted LabVIEW since installing, these icons will show as '?' symbols. This code functions very similar to the PWM drivetrain code described above, but it explicitly sets the motor type to 'Brushed' to match the CIM motors used on the KitBot and sets a 60 Amp current limit on the motors.

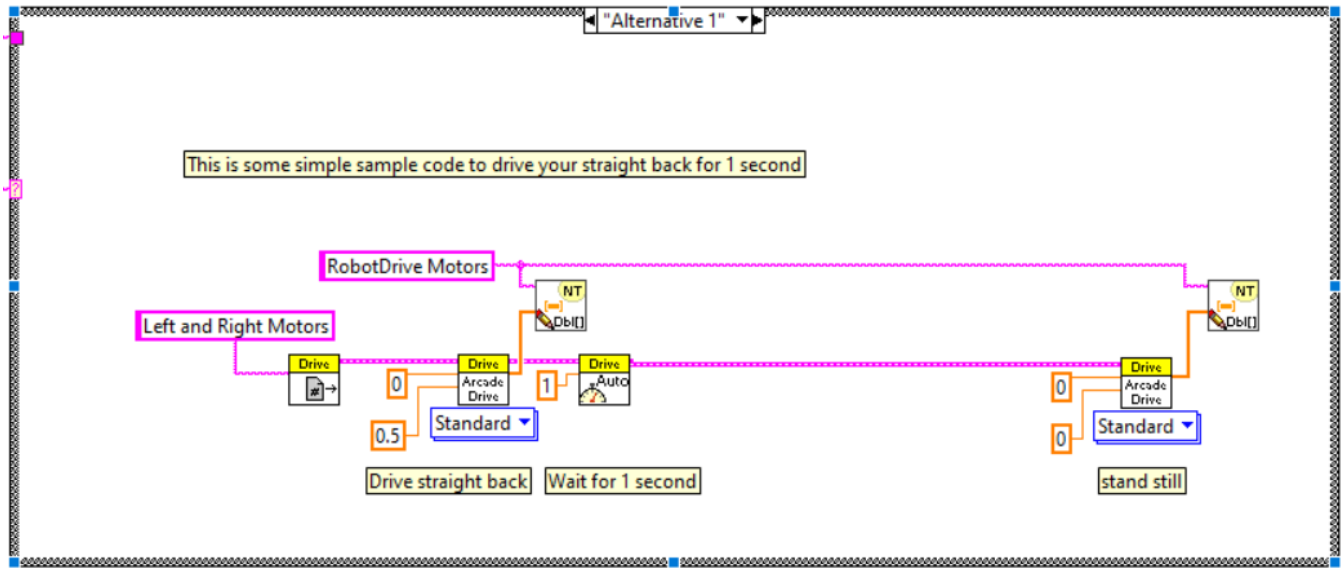
5.1.5 CAN Launcher



The CAN Launcher code opens a reference to the SPARK MAXs, and sets them to be inverted. This was needed on our KitBot to make the wheels spin outward (launching) with a positive command and inward (intaking) with a negative command. If your robot behaves differently, you can either modify these inversion constants or negate the values used in the launcher control. This code also sets the current limit on the motors to the default value of 80 Amps.

5.2 Autonomous Independent.VI

The majority of the code in the Autonomous Independent VI is the default code provided by the LabVIEW project template. Check out the comments and some of the resources on the LabVIEW FRC page in Section 2 for more information.



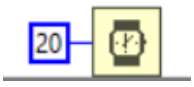
The code in the “Alternative 1” case drive the robot backwards at half speed for 1 second. Utilize the LabVIEW dashboard in order to select this autonomous mode to run. The speed provided is positive in order to drive the robot backwards because of the inversions used in Begin.VI. Because the Y-axis of a joystick is negative as you push it away from you (forward), a positive value here is backward. If you prefer that positive values move the robot forward, change the inversions in Begin and negate the joystick axes in Teleop.

The VI after the speed set is a special VI used in Autonomous that feeds the drivetrain motor safety while delaying for the specified time.

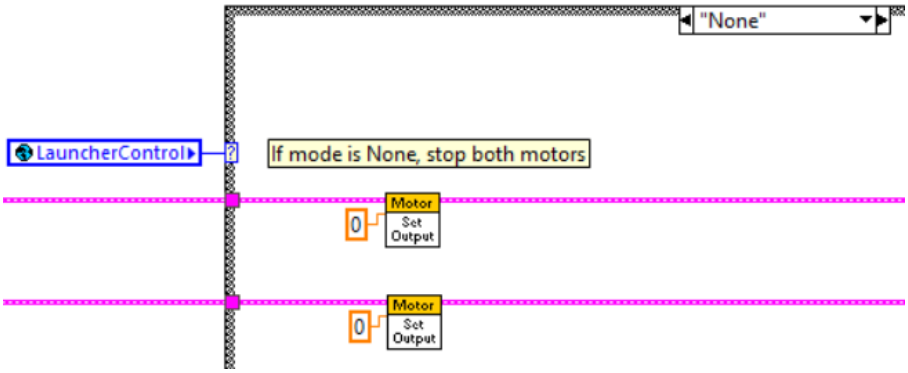
5.3 Periodic Tasks

The Periodic Tasks VI can be used to run robot actions at a fixed rate regardless of robot mode. The KitBot places the Launcher control here so that you can easily extend the code to utilize the launcher controls in autonomous. The provided code only sets the Global Variable used to control the Launcher from within the Teleop code, but you can use that as an example to add Launcher control to Autonomous if desired.

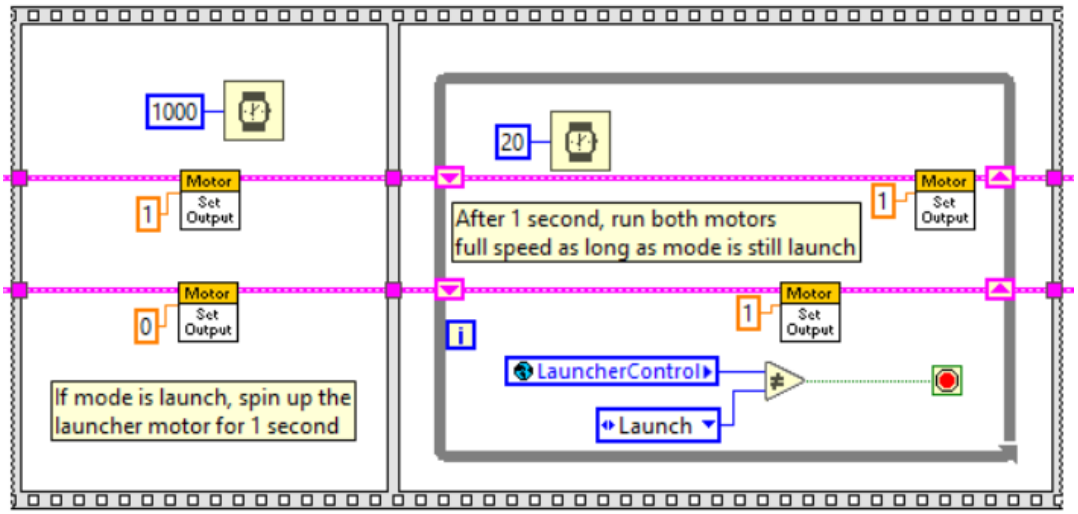
The launcher controls are placed inside a loop structure. The Periodic Tasks VI is only called one time, after Begin.VI finishes, so if we want our code to keep running, we need to provide our own loop(s).



At the bottom of the loop is this 20ms delay. LabVIEW code which is not connected by wires indicating data dependency executes in parallel, so this delay means that our loop will take at least 20ms to run, but may take longer if other loop elements run longer than this. For the KitBot code, the loop will run at the 20ms rate if the LauncherControl is None or Intake, but the outer loop will not be looping at all if the mode is Launch.



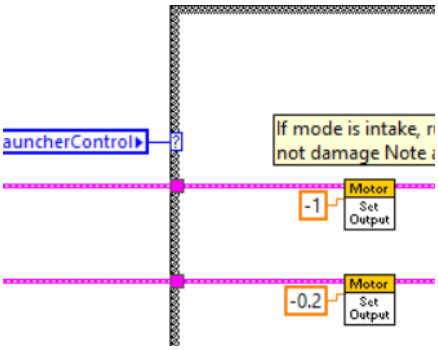
The Launcher code uses a case structure, fed by the LauncherControl global variable, in order to execute different behaviors based on the desired launcher mode. If the desired mode is set to None, the code sets both wheels to 0 to stop them.



If the mode is set to Launch (click the arrows on the top of the case structure to cycle through the diagrams), the code runs this sequence structure. The first frame of this structure sets the Launcher wheel to max speed and keeps the Feeder wheel set to 0 while delaying for 1 second. This gives the Launcher wheel time to reach full speed before the feeder pushes the ring into it. Because this frame uses a delay, the wheel will run for at least 1 second even if you just tap the button. Generally this isn't a big deal, so we didn't add any complexity here to prevent this.

The next frame of the sequence contains a loop that runs every 20ms. This loop sets both motors to full speed to launch the ring. The bottom section of the loop checks to make sure the mode is still set

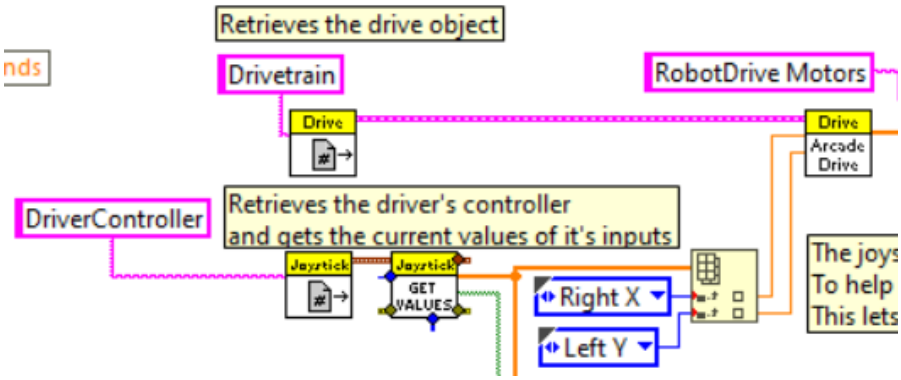
to Launch. As soon as the mode is no longer Launch, the value passed into the stop sign will be True and the loop will exit.



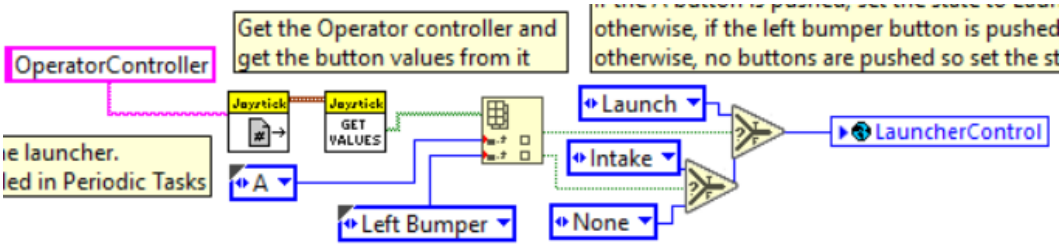
In the Intake portion of the case structure, the Launcher wheel is set to run inwards at max speed and the Feeder wheel is set to run inwards at a low speed. This pulls the Note in fully, but avoids rapidly grinding the wheel against the Note once it is fully inside the robot.

5.4 Teleop.VI

Some aspects of the Teleop code are default parts of the LabVIEW template code. Those items are not described here.



This section of code accesses the joystick values from the DriverController, selects the value of specific axes from the axis array, and uses those values to control the drivetrain in Arcade Drive. Arcade Drive controls the robot rotation with one value (here the Right X axis value) and the robot speed with a separate value (here the Left Y axis value).



This section of code accesses the button values from the OperatorController, selects specific button values from the array and uses a pair of Select VIs to map those button states to desired launcher states. The Select VI furthest to the right takes precedence. Because it is the last Select executed, if the A button is pressed, the mode will be set to Launch, regardless of the state of the Left Bumper. The other Select VI is used to set the mode to Intake if the Left Bumper is pressed and the A button isn't. This desired mode is stored in the global variable and accessed from the Periodic Tasks VI, where the Launcher control actually occurs.

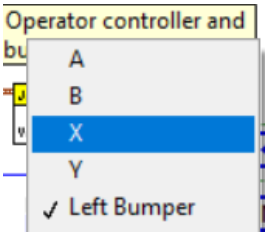
6 Making Changes

This section details some common possible changes you may want to make to the KitBot code and provides some references for how to approach making those changes.

6.1 Changing buttons for actions

One of the easiest changes to make is to switch what buttons control the Launcher. To modify this, simply select a different button option on either of the enumerations by clicking on the value and selecting the new choice in the dropdown menu.

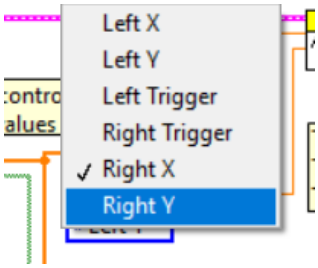
For example, to change the intake command from the left bumper to the **X** button



6.2 Changing Drive Axis Behavior

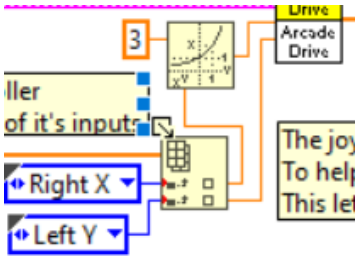
Another easy change to make is to modify which axes of the controller are used as which part of the robot driving and how. To change the axis, just click on the enumeration and select a new axis from the dropdown.

Example: changing the forward-back driving to the right stick Y-axis and leaving the rotation on the right X-axis



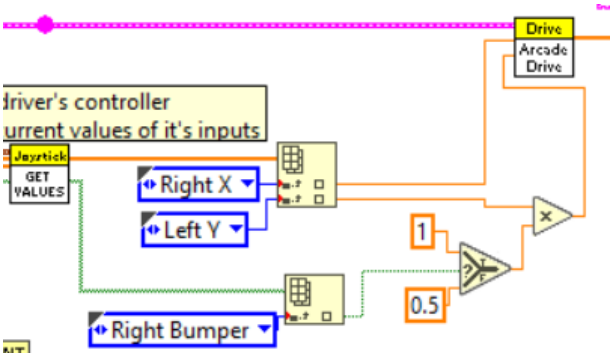
You can also modify the axis values. One common modification is to cube the values. This preserves the sign of the value (positive stays postive, negative stays negative) and the maximum value (doesn't reduce the max speed of the robot) while providing less sensitivity at low inputs, potentially allowing for more precise control at low speeds. To make this type of modification, you can apply the modification to the axis between where it's unpacked from the array and where it enters the Arcade Drive VI.

Example: changing only the rotation axis to be cubed (accomplished here by using the 'x^y' VI from Mathematics-Exponential palette):



Another common modification is to scale the values down by default, but allow for the maximum value if a button is pressed (turbo mode).

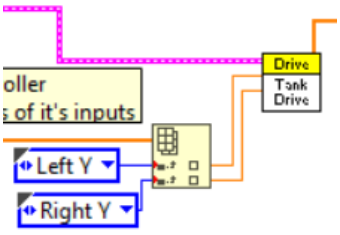
Example: Scale the forward-back driving by 50% unless the right bumper is pressed



6.3 Changing Drive Type

The last likely change we will cover is changing from Arcade Drive to Tank Drive. Unlike Arcade drive which maps one axis to rotation and one to forward/back, Tank drive maps one axis (generally the Y-axis) to each side of a differential drivetrain. To make this change, right click on the ArcadeDrive VI, select Replace and navigate to the WPI Robotics Library -> Robot Drive palette to select the Tank Drive VI. Then modify the joystick axes as appropriate (generally the Y-axis of each stick if using a single controller and not a pair of joystick).

Example:



6.4 Developing Autonomous Routines

The provided code contains a very basic autonomous mode that drives backwards at ½ power for 1 second. You can modify this case of the structure to change or enhance the routine and additional autonomous modes can be developed by adding additional cases to the structure.