Using the Mobile Studio IOBoard and LabVIEW 8.5 for Motor Control

Michael Kleinigger
November 2, 2008 (Updated 12/4/2008)

From portable hard drives to hybrid cars, electric motors can be found in a wide range of devices. In Introduction to Engineering Design (IED), projects often utilize motors to perform a number of different tasks. While there are a wide variety of different motors available to students, brushed DC motors are often the first choice for their simplicity and low cost.

When students need to control the speed and direction of a motor, the IED H-Bridge module provides a quick solution. But what if the motor needs to be controlled by software? This is where the Mobile Studio IOBoard can help. The IOBoard is a powerful device for data acquisition and signal generation (See http://www.mobilestudioproject.com/ for details and to download the supporting software). In this tutorial we will describe a procedure for controlling motors using the IED H-Bridge, an IOBoard, and LabVIEW, a graphical programming language.

Before getting started, you will need the following hardware and software:

- **The Red2 IOBoard**
- **The IED H-Bridge module** (or equivalent motor speed control, see http://mfg.eng.rpi.edu/IED/The%20IED%20H%20Bridge%20050928.pdf for specifications)
- **Mobile Studio Desktop** (download from http://mobilestudio.rpi.edu/Downloads.aspx)
- **The LabVIEW 8.5 IOBoard interfaces** (download from http://mobilestudio.rpi.edu/Downloads.aspx - see the list at the bottom of the page. Note that LabVIEW 8.0 and 8.2 can also be used if version 8.5 is not available.)
- **National Instruments LabVIEW 8.5** (available to RPI students as part of the default laptop image or via download at http://www.rpi.edu/dept/arc/web/licenses/labview_license.html)

Alternatives

Note that there are several alternative methods of motor control possible with the IOBoard:

- **Unidirectional motor on/off control** – when speed control is not important, a simple relay or transistor may be used to provide simple on/off control. The IOBoard has 16 digital outputs which can be utilized for relay or transistor control (through a buffer). This can be accomplished by using "**DigitalIO_Single.vi**" (which must be first connected to the "**OpenBoard.vi**" block) within LabVIEW.

- **Unidirectional speed control** – using a high-power MOSFET and pulse-width modulation, the speed of a motor rotating in one direction can be controlled easily. See http://www.embedded.com/story/OEG20010821S0096 and http://homepages.which.net/~paul.hills/SpeedControl/SpeedControllersBody.html for details on PWM and speed control circuits. The Red2 IOBoard's PWM generators can be controlled using "**PWMConfig.vi**" (which must be first connected to the "**OpenBoard.vi**" block) within LabVIEW.

- **Bidirectional speed control** – using controllers other than the IED H-bridge. For smaller motors, an integrated circuit (e.g. http://www.st.com/stonline/products/literature/ds/1330/l293d.pdf) can be utilized. This chip is controlled by changing digital inputs. The user applies a constant digital input to determine direction, and a PWM input to determine speed. See the example file, "**EXAMPLE Motor P-Control.vi**" for details on how to interface with such a device.

The IED H-Bridge

The IED H-Bridge is controlled by a single analog input voltage applied to the A_{in} terminal. If the voltage is set to 2.5V, the motor will be stopped. If the voltage is set to 5.0V, the motor will rotate at full speed in one direction. If the voltage is set to 0V, the motor will rotate at full speed in the opposite direction. At intermediate voltages the motor speed will be set proportional to input voltage. **IMPORTANT**: Never apply a negative voltage to the A_{in} terminal.

The Red2 IOBoard includes two analog outputs which can be utilized to control the IED H-Bridge (HB). However, these outputs range from -3.5V to 3.5V. Since the HB operates in the 0-5V range, by default a motor can be commanded to 100% power in one direction (0V), but only 40% power (3.5V) in the opposite direction. This can be overcome by implementing an amplifier circuit between the IOBoard and the HB (See http://ecow.engr.wisc.edu/cgi-bin/get/ece/340/schowalter/opampckts.pdf for circuit details - look for the "Non-Inverting Amplifier" and set (R1 + R2)/(R1) = 1.4).

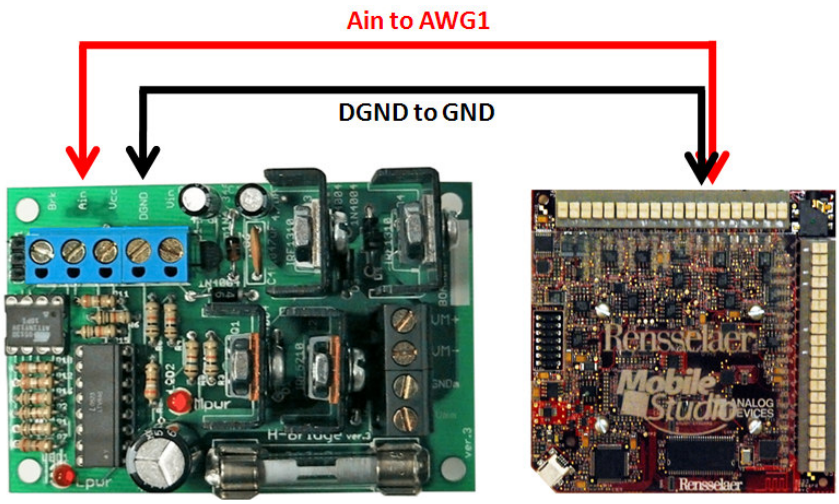

Figure 1 - IED HB and IOBoard Connections

Note: The IED H-bridge also contains a digital terminal labeled “Brk” which will put the motor in an electronic braking mode. Essentially, passing a logic high value to this terminal will cause the board to disable the motor drive and short the two motor terminals together. This causes the motor’s back-EMF to slow and then stop the motor. However, the braking force will vary from motor to motor. This terminal can be driven using the IOBoard’s digital I/O pins and the LabVIEW “DigitalIO” blocks.

Creating a Speed Control Program with LabVIEW

1. To begin, open LabVIEW 8.5. Once the “Getting Started” screen has appeared (Figure 2), under the “New” menu select “Blank VI”. You are now presented with a blank front panel and behind it, an empty block diagram (Figure 3). The front panel is the user interface for your LabVIEW program, and the block diagram is where the program is built, using various types of program blocks.

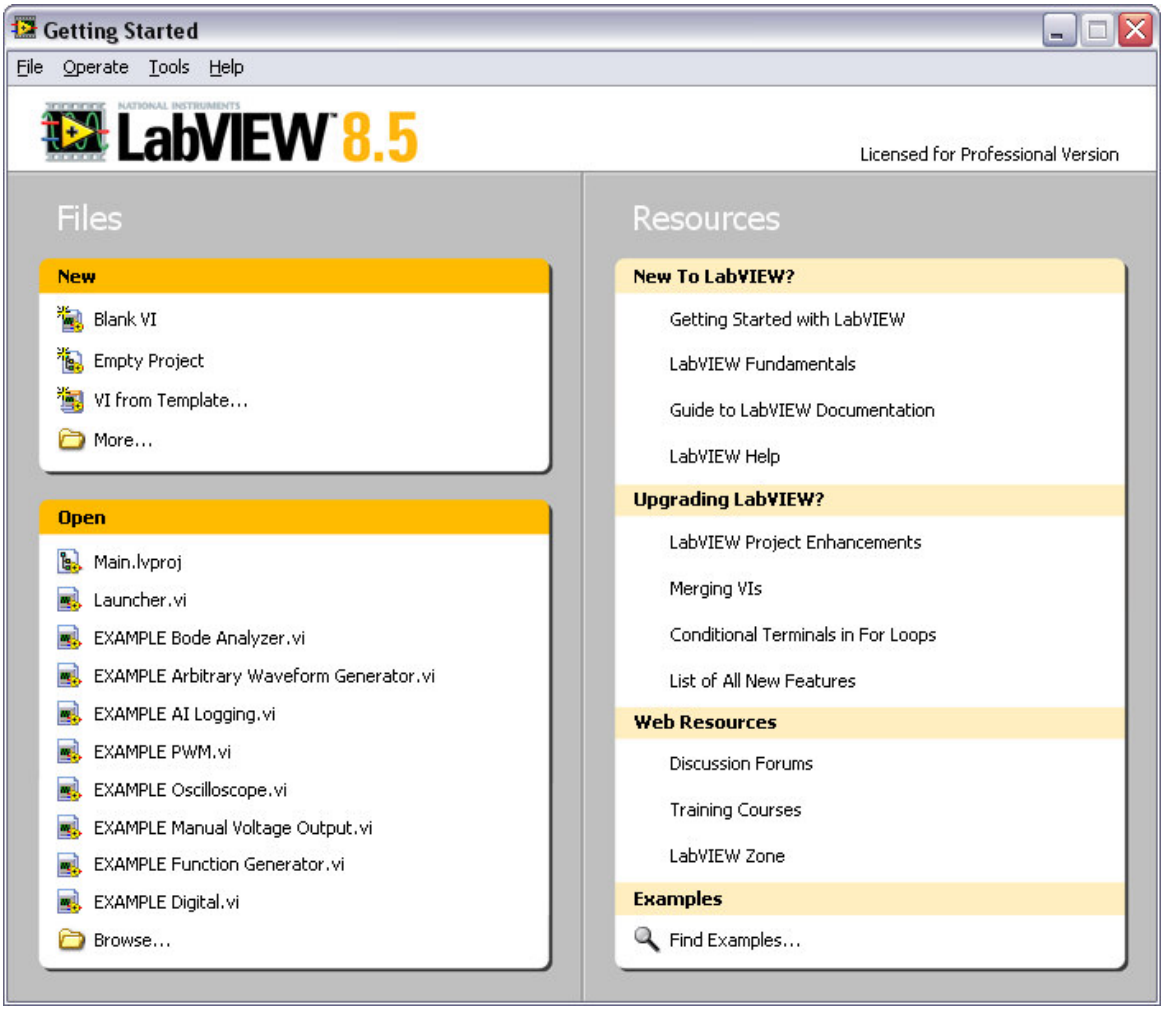

Figure 2 - LabVIEW "Getting Started" Screen

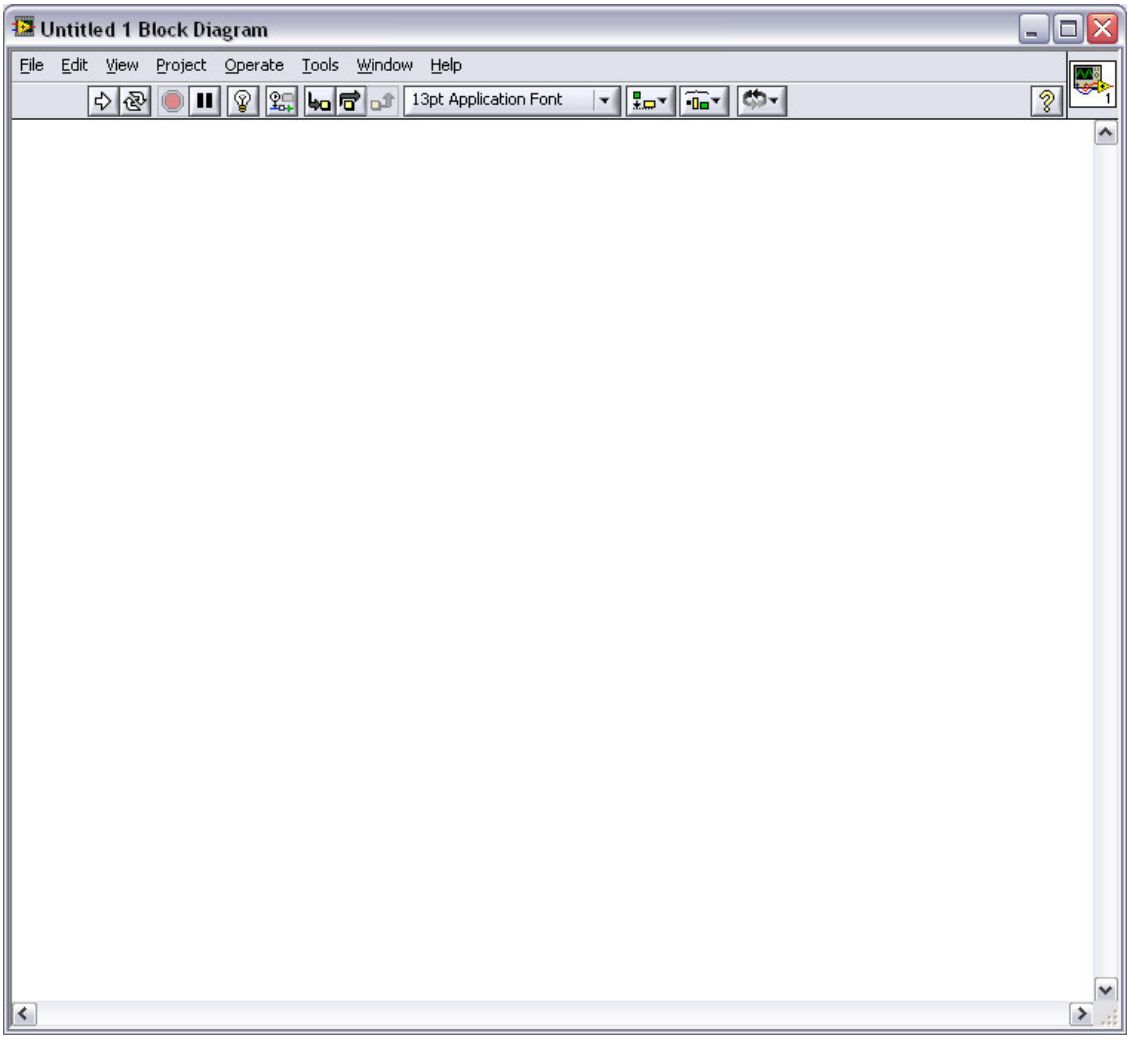

Figure 3 – Blank LabVIEW Block Diagram

2. Select the block diagram window (located behind the grey front panel). To do this you may either click the window's title bar, or simply press Ctrl+E. This shortcut can also be used to switch back to the front panel when needed.

3. The first step for any IOBoard program is to add the block which will initialize our communications with the board. To do this, open the file folder where you've saved the LabVIEW IOBoard interface VIs. Resize this window and the LabVIEW block diagram so they are both visible on your screen. Locate the file named "**OpenBoard.vi**", then click and drag the filename onto the block diagram. When you release the left mouse button, you will see that the "Open Board" block has been added to your program.

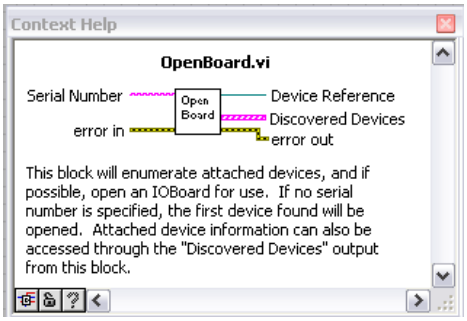

Figure 4 - Context Help

If it is not already visible, open the context help window by pressing Ctrl+H (Figure 4). This window displays valuable information for whatever block you move your cursor over. Notice the colored, labeled lines connecting to the block. These are called wires, and are used to pass data between blocks within LabVIEW. However, not all inputs and outputs from certain blocks need to be wired (LabVIEW will complain if you try and run a program without all necessary wiring). The "Open Board" block does not require any inputs, but can be optionally wired to a serial number so that if you're using more than one IOBoard, you can choose between them.

NOTE: If using the LabVIEW 8.0 interface VIs, you will need to make the following adjustment before the VIs recognize your attached IOBoard. Open **OpenBoard.vi** by double-clicking the block, then hit Ctrl+E to switch to the block diagram. Find the leftmost block labeled "IOBoard" (Figure 5) next to the "error in" control. Double-click this block, then select "IOBoard" from the objects list and click OK. Run the VI to make sure your device is found, then save and close.

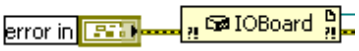

Figure 5 - Fixing OpenBoard.vi

4. For this program, we are only concerned with a single output from the "Open Board" block: "Device Reference". This wire contains data for communicating with the IOBoard and is used by all other IOBoard function blocks. Before closing the folder with your IOBoard function VIs, drag "**AnalogOutput.vi**" onto your block diagram. Connect the "Device Reference" output from the "Open Board" block to the "Device Reference Input" of the "Analog Output" block by clicking once on the terminal of each block (**Error! Reference source not found.**).

Figure 6 - Analog Output Blocks

5. Next, we will create a slider to control the output voltage of channel 1 (AWG1). Switch to the front panel by pressing Ctrl+E. Right-click on any blank space on the front panel to bring up the controls palette (Figure 7). Select "Horizontal Pointer Slide" from the "Numeric" group (under the "Modern" group, which is opened by default). Click once on the front panel to place this control. Rename the control to "Voltage." Switch back to the block diagram by pressing Ctrl+E again and you will see that a new icon has appeared which represents the value of the control.

   Since the Analog Output block accepts array input only, we need to convert the single value from our slider into an array. To do this, right-click on any blank part of the block diagram to bring up the functions palette (Figure 8). Select "Build Array" from the "Array" group (under the "Programming" group, which is opened by default).

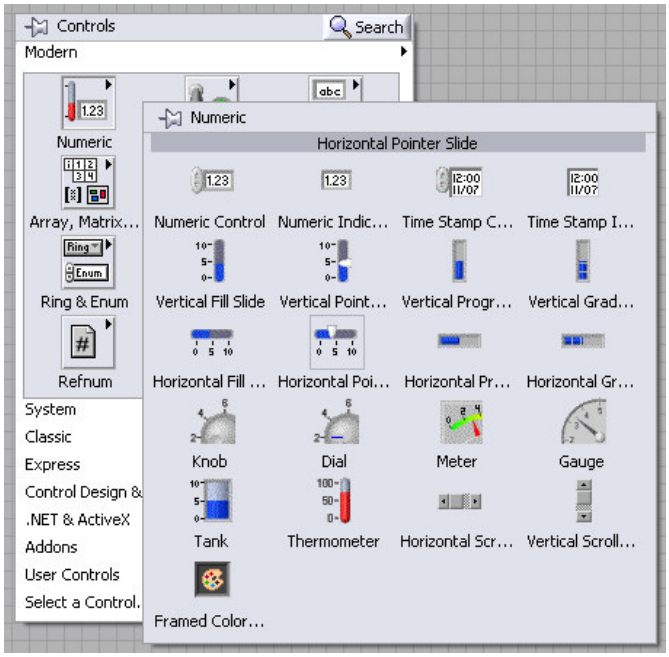

Figure 7 - Select Horizontal Pointer Slide

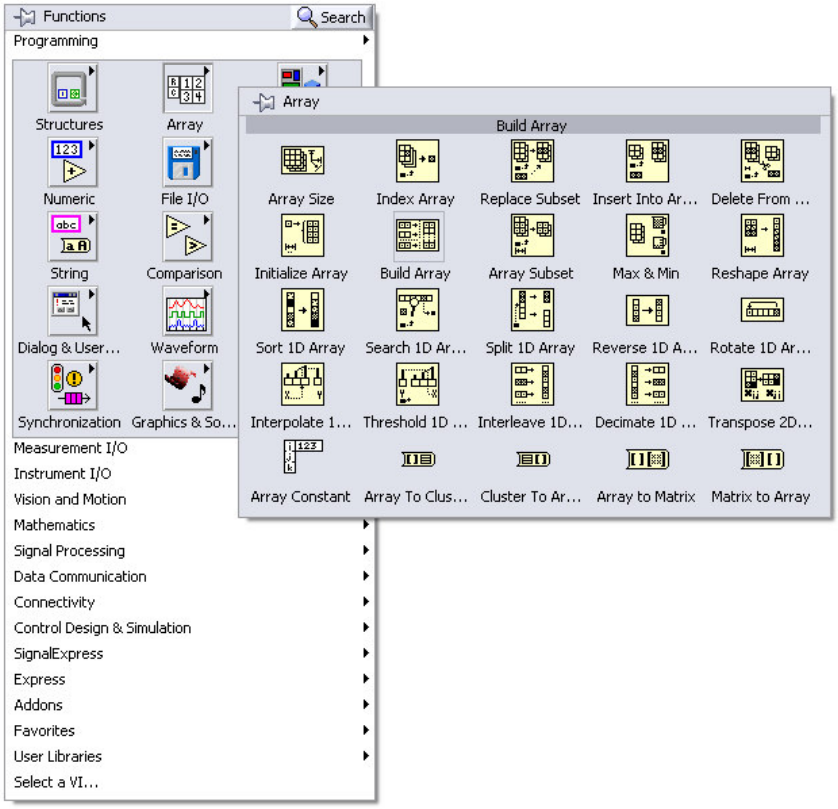

Figure 8 - Select Build Array

Place the “Build Array” block on the block diagram, then connect the output from the “Voltage” control to the input of “Build Array.” Connect the “Build Array” output to the “Channel 1 Data” input on the Analog Output block. Also right-click the “Channel 1 Enabled” terminal of the Analog Output block and select Create ⟶ Constant. A Boolean value will be created. Click this value to change it to True (Figure 9).

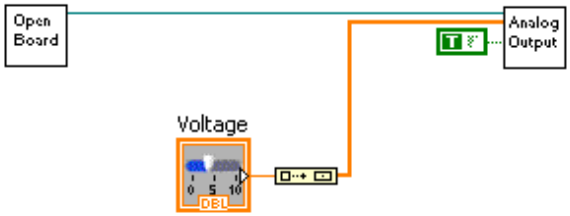

Figure 9 - Voltage Control Blocks

That’s all there is to creating a single analog output. However, if you run this VI (with Ctrl+R or by clicking the play button), the IOBoard voltage will only update once before the program terminates. We need to wrap our blocks in a while loop that executes until the VI is stopped. The while loop is found under the block diagram functions palette (right-click to make it appear) under “Programming” (open by default) ⟶ “Structures” ⟶ “While Loop”. Click and drag the loop around all blocks except the Open Board block (since we only need to open the board once per session). Next, right-click on the stop icon and select “Create” ⟶ “Constant”. This ensures that the loop will continue to execute until the VI is manually stopped (Figure 10). Press Ctrl+R or click the play button to run the VI. Move the slider on the front panel to change the voltage of the IOBoard’s AWG1 output which will change the speed of your motor. That’s all there is to creating a manual speed control program!

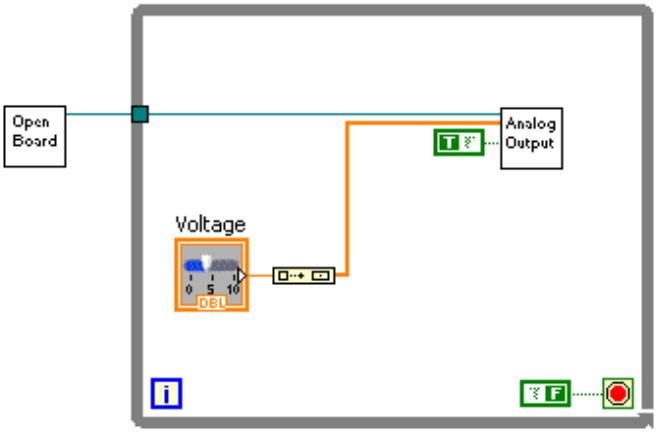

Figure 10 - While Loop Added

Closing the Loop: Motor Position Control with LabVIEW

Quite often motors are used in a closed-loop control configuration for position control. For example, imagine we want to turn a motor to an arbitrary angle then stop. To do this requires a second device to sense the position of our motor. Once the position is known we can write a simple program to send the appropriate control commands to our motor.

Sensors

There are a number of sensors available for this task. Perhaps the simplest device is a potentiometer, or variable resistor. Turning the input knob of a potentiometer sweeps a center terminal from one side of a resistor to another. Thus, a potentiometer can be used as a variable voltage divider. By connecting the first terminal to 3.3V and the third to ground, the middle wiper terminal will produce an output voltage which varies between 0 and 3.3V proportional to angular position. By connecting the input knob of a potentiometer to the shaft of a motor, we can monitor the motor shaft's position. Note that most potentiometers can only be turned a certain amount before hitting a mechanical stop. If selecting a potentiometer, be sure to purchase one designed for continuous rotation.

Another similar device which can produce an analog voltage proportional to angular position is the absolute encoder. These devices offer much higher precision. For example, the US Digital MA3 (http://www.usdigital.com/products/encoders/absolute/rotary/shaft/ma3/) offers 10-bit resolution. However, these devices are typically much more expensive than a simple potentiometer.

Incremental optical encoders are also common solutions for motor position control. These devices typically have two digital outputs that pulse between logic high and low as the encoder shaft is rotated. For a more detailed explanation of optical encoders, see http://en.wikipedia.org/wiki/Rotary_encoder for more details on absolute and incremental rotary encoders. Unfortunately, the Red2 IOBoard does not currently support the high speed counter inputs necessary for such a device. The Red2 hardware itself does support counter inputs, so future firmware updates may enable this feature.

Software

For the purpose of this tutorial, we will assume that a potentiometer (or similar device) is being used which returns an analog voltage (in the -3.3 to 3.3V range) based on the position of our motor shaft. Connect the sensor input to the IOBoard's A1+ input and the sensor's ground to the IOBoard's GND input. Once that connection is made, we need to add a couple blocks to read in our analog voltage.

Add the "**AnalogInputConfig.vi**" and "**AnalogInputRead.vi**" blocks to your block diagram by dragging them from the folder where your interface VIs are saved onto your block diagram. Place the Analog Config Start block outside the while loop, and connect its "Device Reference" input and output terminals between the Open Board block and the while loop. Then connect the Analog Input Read block's "Device Reference" terminals between the while loop input and the Analog Output block as shown in Figure 11.

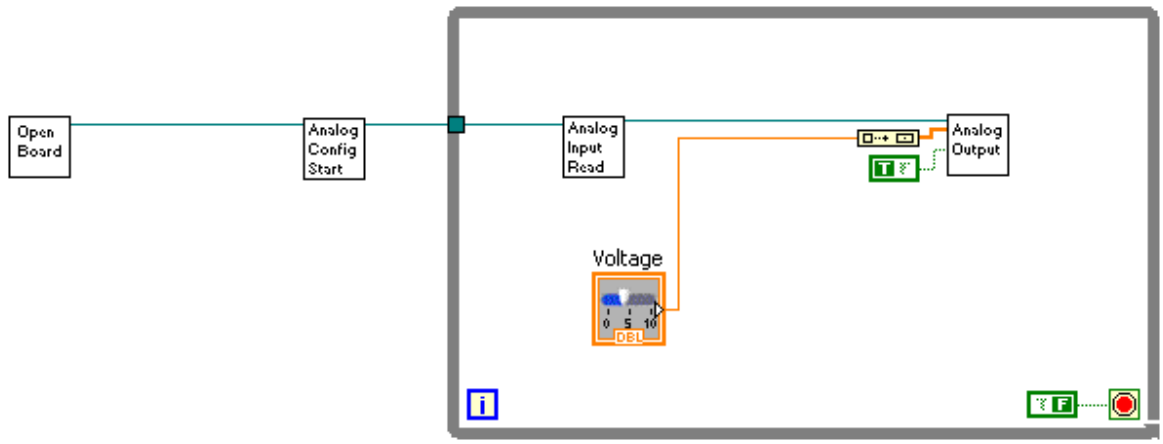

Figure 11 - Adding the Analog Input Blocks

Next, right-click on the "Channel Configuration" input of the Analog Config Start block and select "Create" ⟶ "Constant." This will create a cluster input containing the channel settings. The default values should be correct. Also right-click on the Analog Input Read block's "Channel 1 Enable (T)" terminal and create a true constant, then right-click on the "Channel 2 Enable (T)" terminal and create a false constant (Figure 12). This will speed up execution by only returning data from channel 1.

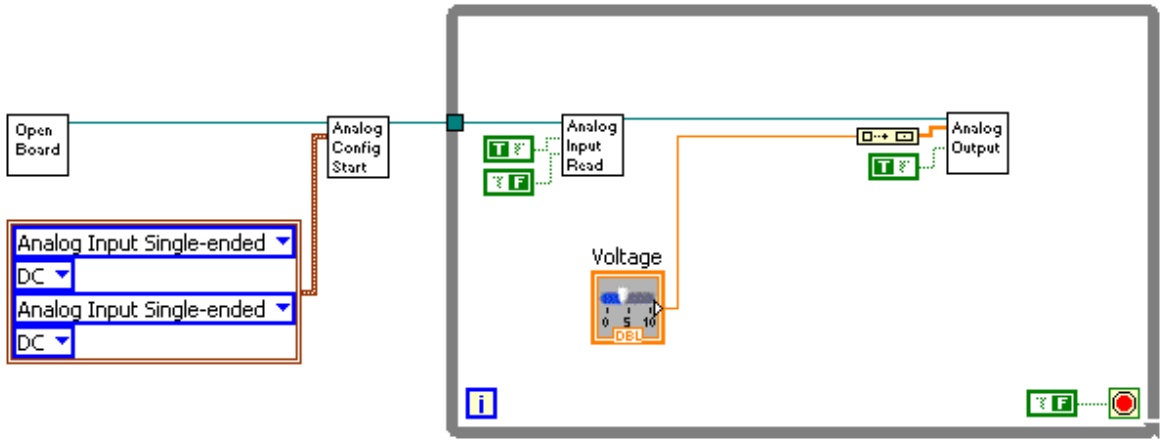

Figure 12 - Analog Input Configuration Constants

Next, we will add the blocks to create a proportional control system. To do this, first delete the wire coming out of the Voltage control, then double-click on the word "Voltage" and rename the control to "Command." Next, create a copy of this control by holding the Ctrl key while dragging. Alternatively, you can select the "Command" block, press Ctrl+C, then Ctrl+V to copy and paste it in a convenient location within the while loop. Rename this block to "Proportional Gain." Your block diagram should now look similar to that shown in Figure 13.

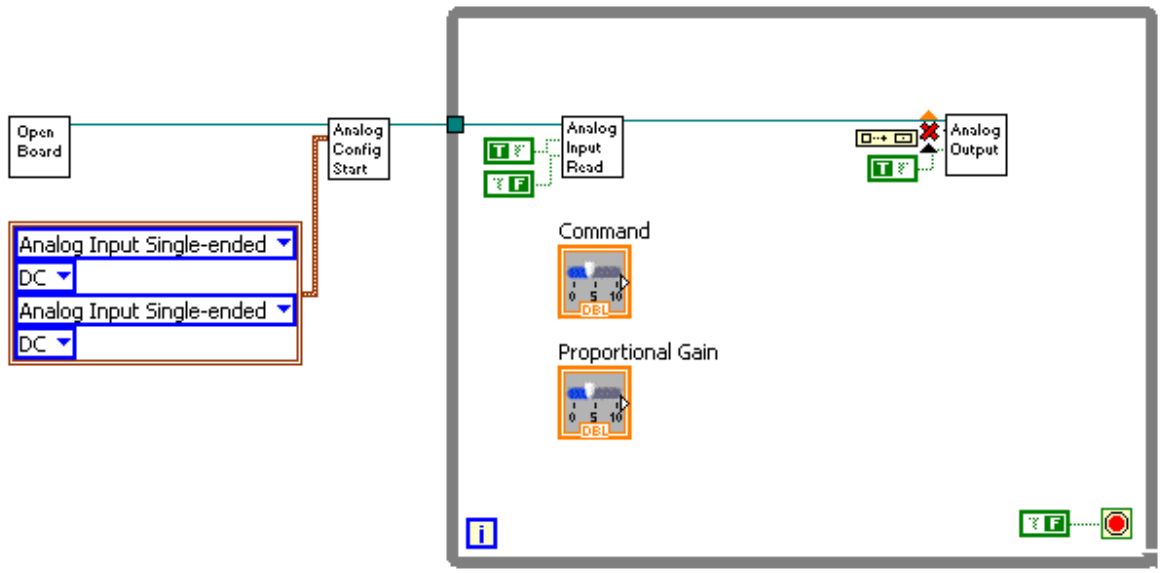

Figure 13 - Adding Controls

Next, right-click in any blank space on the block diagram to bring up the functions palette. Add the following blocks to your block diagram and arrange and connect them as shown in Figure 14. Don't forget to add constants to the "In Range and Coerce" block:

- Index Array (Found under "Programming" ⟶ "Array" ⟶ "Index Array")
- Subtract (Found under "Programming" ⟶ "Numeric" ⟶ "Subtract")
- Multiply (Found under "Programming" ⟶ "Numeric" ⟶ "Multiply")
- In Range and Coerce (Found under "Programming" ⟶ "Comparison" ⟶ "In Range and Coerce")

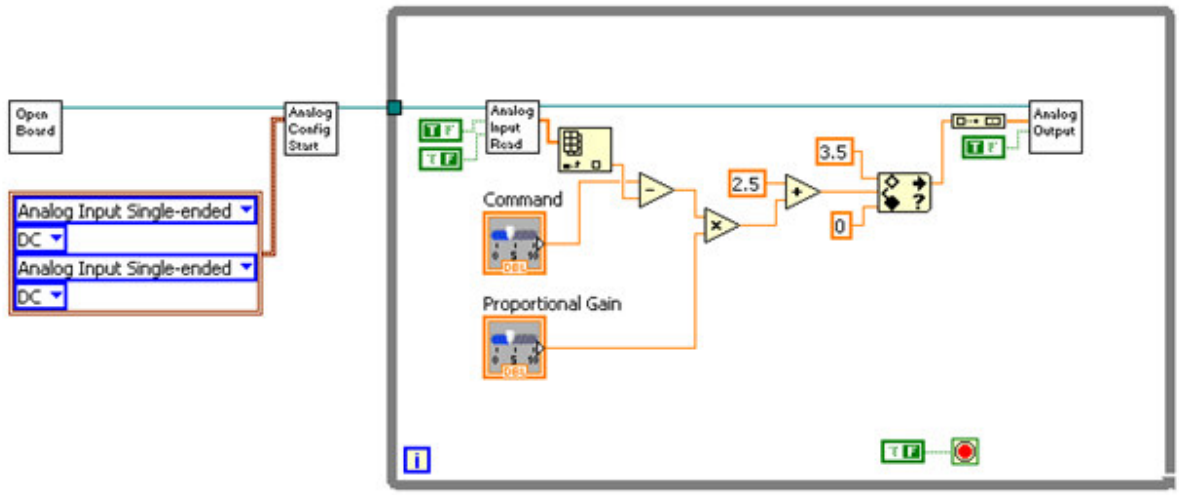

Figure 14 - Completed Proportional Control

As you may have guessed, the output voltage will be controlled according to the following formula:

Output = (Command Voltage – Input Voltage)*(Proportional Gain) + 2.5V

However, the "In Range and Coerce" block limits the output voltage to $0 < x < 3.5V$. This prevents our program from sending any invalid values to the board. Essentially, this controller reads in a voltage from our sensor. Then it computes the difference between our desired sensor voltage (representing some desired angle) and the actual sensor voltage (representing some actual angle). This error value is multiplied by a gain and represents a command to rotate the motor in a certain direction at a speed proportional to the error. In the case of the IED H-bridge, this command voltage should be centered around 2.5V (since this is the voltage which commands the motor to stop turning), so we add 2.5V to our command value before sending it through the "In Range and Coerce" block and out to the board.

Thus, we have created a basic proportional control. However, this system will only work well for certain motors at certain proportional gains. If the proportional gain is set too high, the motor will likely overshoot the desired angle and then oscillate around it without ever stopping. If the proportional gain is too low, the motor will approach but never reach the desired angle. One way to deal with this problem is to create a proportional-derivative (PD) or full-blown proportional-integral-derivative (PID) controller. It's actually quite simple to create, but tuning the gains of such a controller may require significant time and patience. Here are the blocks you'll need to arrange as shown in Figure 15:

- Create two more slider controls called "Derivative Gain" and "Integral Gain"
- Add two more multiply blocks (Found under "Programming" → "Numeric" → "Multiply")
- Compound Arithmetic (Found under "Programming" → "Numeric" → "Compound Arithmetic")
- Derivative x(t) (Found under "Signal Processing" → "Point by Point" → "Integ and Diff")

- Integral x(t) (Found under “Signal Processing” ⟶ “Point by Point” ⟶ “Integ and Diff”)

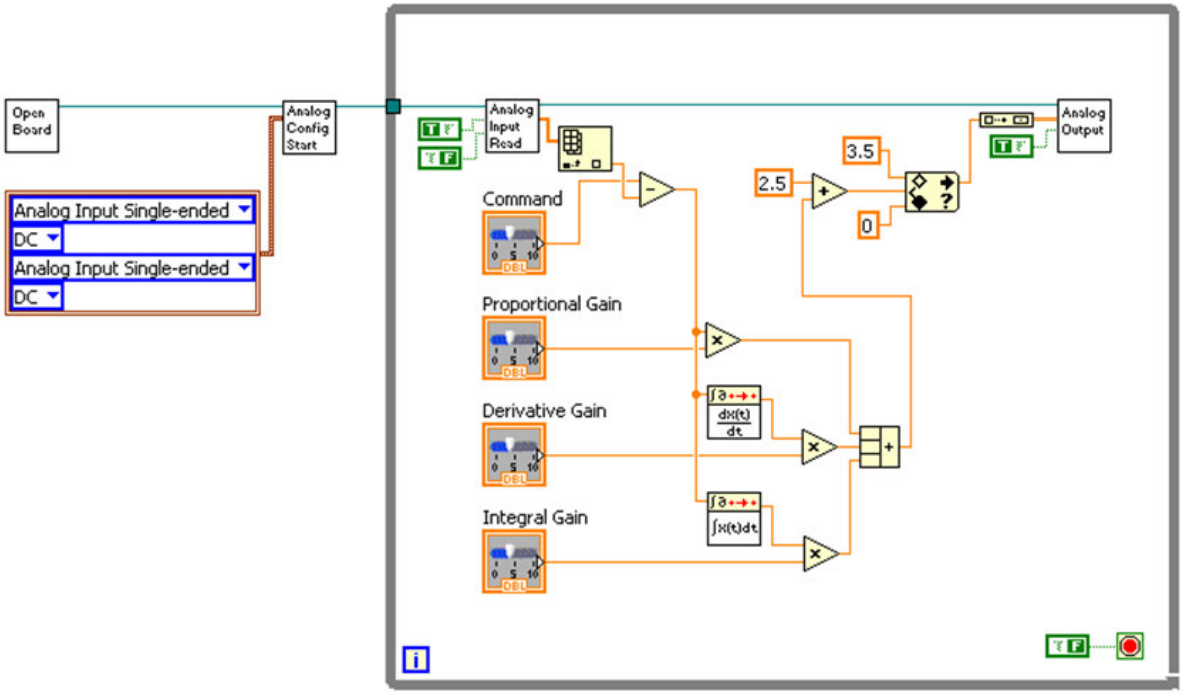

Figure 15 - Full PID Control

In this configuration, the output voltage is also influenced by the rate of change of the error signal as well as the integral of the error signal. The goal of the derivative control is to smooth the speed changes of the motor. It always operates to oppose the motion of the motor. Thus, if the proportional control is driving the motor quickly towards the desired angle, the derivative control acts to slow the motor to a comfortable stop without overshoot. The integral control is added to correct error over time. For instance, if the motor comes close to a desired angle but stops before reaching it, that positional error will be integrated over time and will eventually become large enough to force the motor to move towards the desired angle.

The full details of PID control will not be discussed here, but the following article may be of use: http://en.wikipedia.org/wiki/PID_controller. You may also wish to consult the examples included with the IOBoard interface VIs. Particularly, “**EXAMPLE Motor P-Control.vi**” and “**EXAMPLE Motor PID-Control (REQUIRES SIMULATION MODULE).vi**” (the simulation module can be obtained through the RPI VCC by any RPI student.)