



Configuring motors and sensors

Objectives

In this lab, you will write some simple programs in RobotC to implement a few requested behaviours for a simple controller for electro-mechanical systems.

Overview

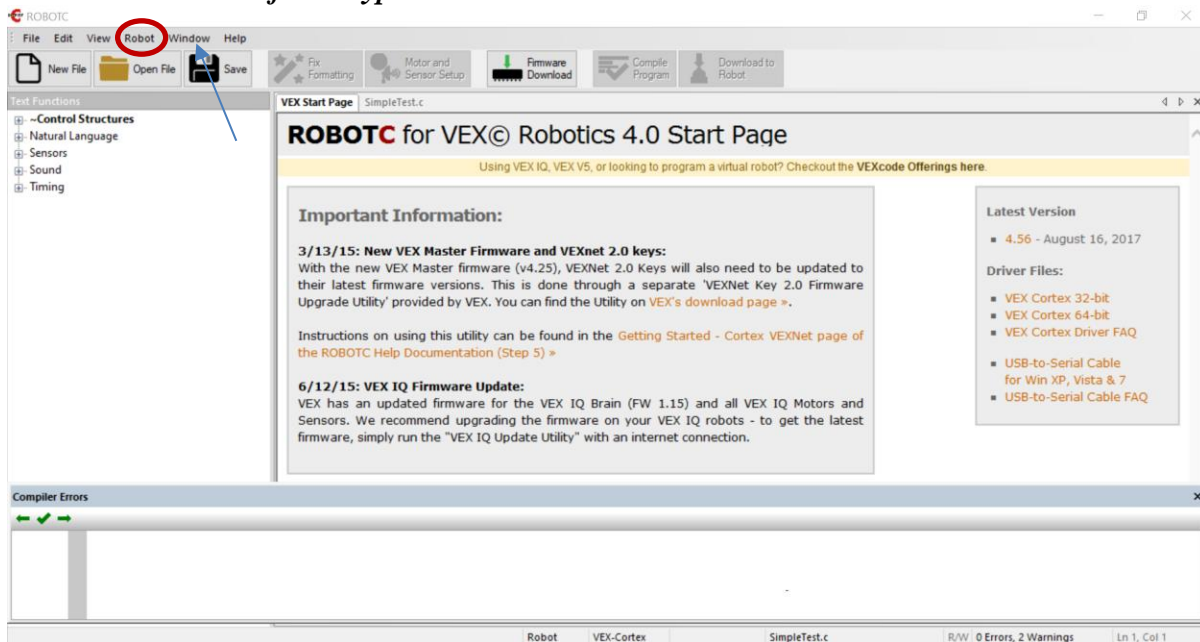
In this lab, you will work in teams to design and implement controllers for simple electro-mechanical systems. The use of VEX peripherals, such as push button controls, encoders, and motors, can be controlled by RobotC code.

Note: Since RobotC comes with a fairly good debugger, and you have your VEX kits already, feel free to test your code outside of lab time. You can refer to the videos given here:

https://www.robotc.net/teaching_rc_cortex_v2/lesson/index_reference.html

Getting Started

- 1) Connect your VEX microcontroller to a lab computer using the USB cable
- 2) Connect the battery to the VEX microcontroller and turn-on the power button
- 3) Open *RobotC for VEX* in the computer
- 4) Select **Robot** → **Platform Type** → **VEX 2.0 Cortex**






- 5) Select **Robot** → **VEX Cortex Communication Mode** → **VEXnet or USB**
- 6) Select **Robot** → **Download Firmware** → **Automatically Update VEX Cortex**
- 7) Select **Window** → **Menu level** → **Super User**



Exercise 1 – Motor Speed Control

Parts list:

Component	Quantity	Image
VEX 393 motor (<i>GreenMotor</i>)	1	
VEX 393 motor with integrated encoder module (<i>BlackMotor</i>)	1	
Motor controller	1	

Wiring instructions:

1. Observe your microcontroller and identify the MOTOR ports. A more formal approach would be to examine the engineering drawing of the microcontroller and to identify the MOTOR ports. The engineering drawing of the microcontroller is given in Figure 1.
2. Connect the VEX 393 motor (*GreenMotor*) to port 1 of the MOTOR ports.
3. Connect the VEX 393 motor with the integrated encoder module (*BlackMotor*) to port 2 of the MOTOR ports.

Note: MOTOR ports 2 to 9 have three pin slots whereas the motors have only two pin terminals. One needs to use a motor controller to connect a motor to any MOTOR port with three pin slots.



Once the wiring is completed, please wait for your TA to check it. Once the wiring is verified by your TA, you can proceed.

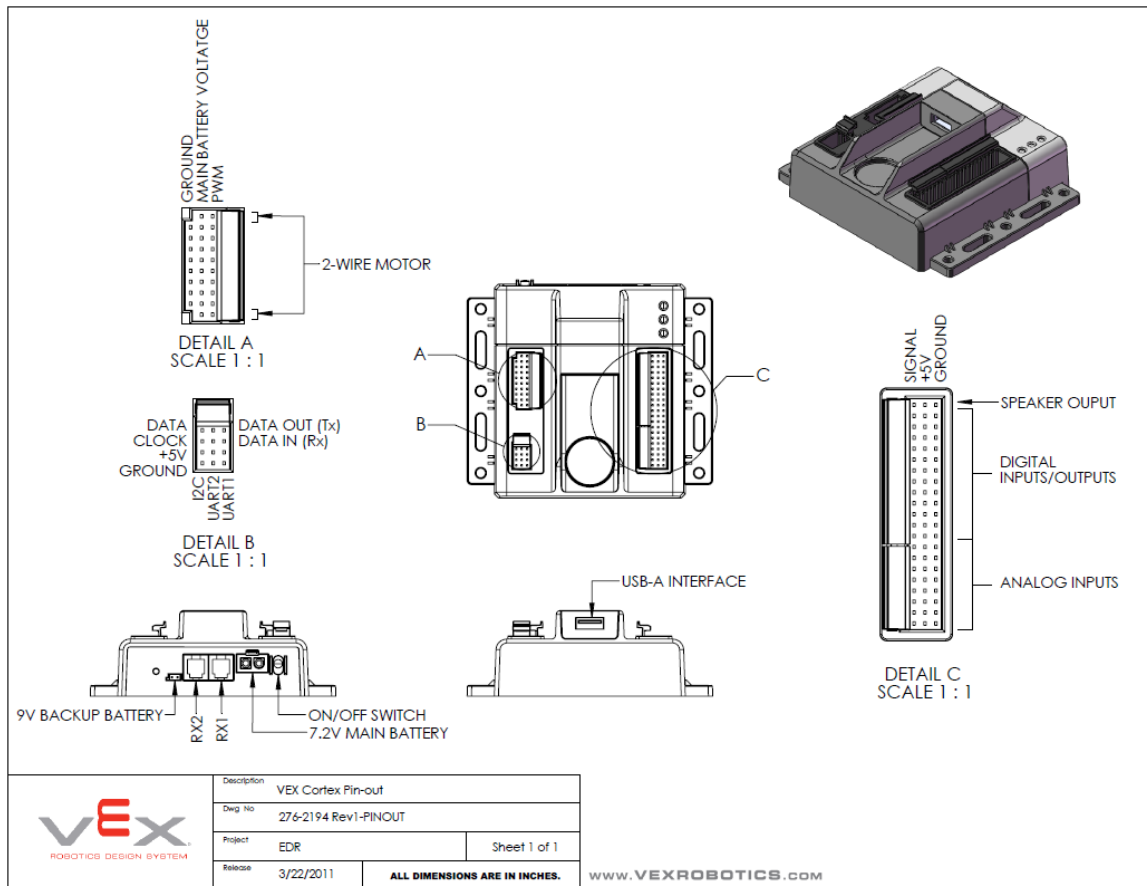
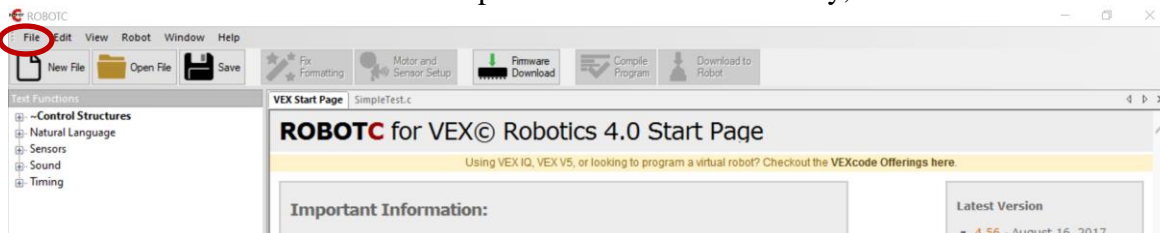


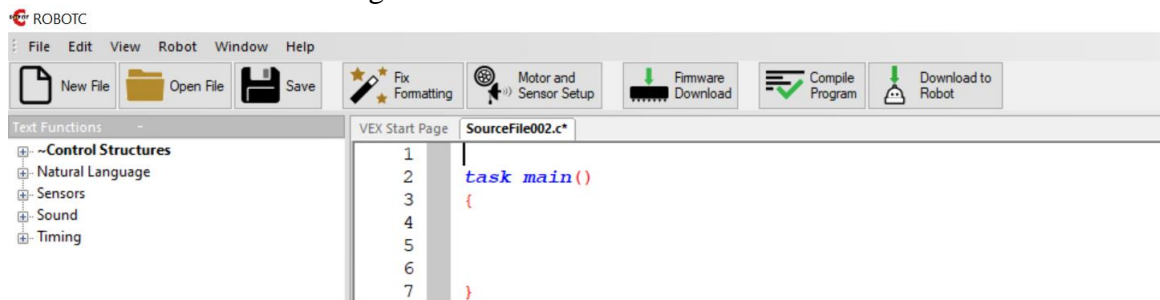
Figure 1. Engineering drawing of the VEX Cortex microcontroller. *Section A* corresponds to the motor ports.

Configuring the motors using software:

1. Select **File** → **New** → **New File** to open a new file. Alternatively, click on the **New File** icon.

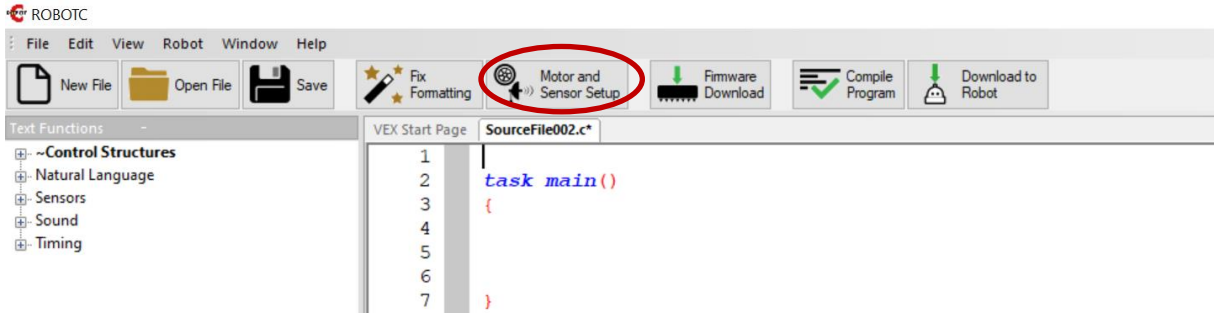


You should see the following.

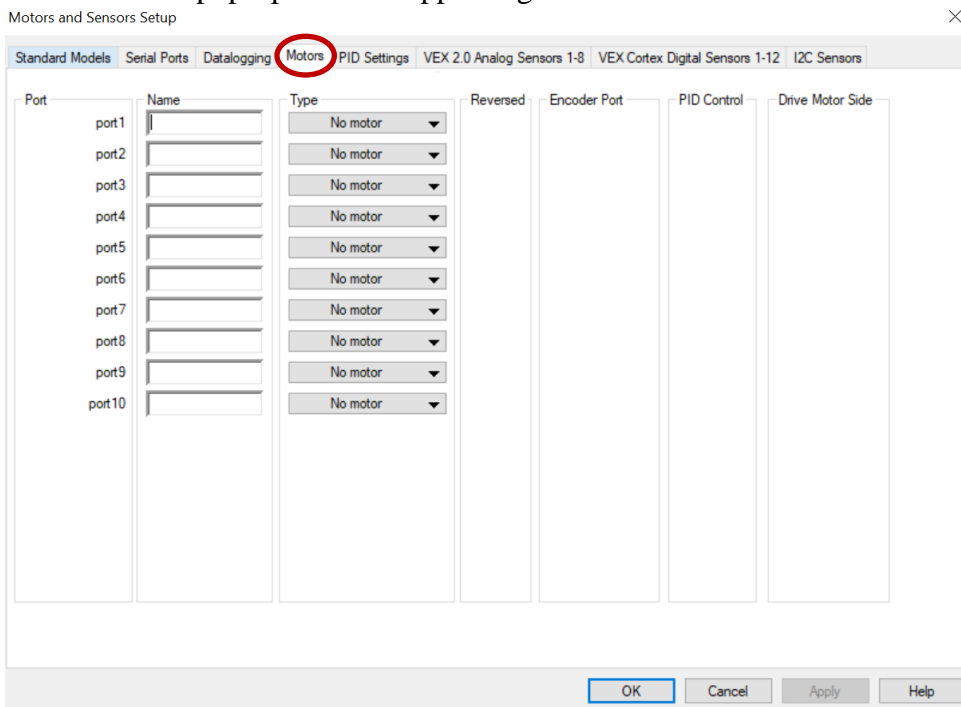




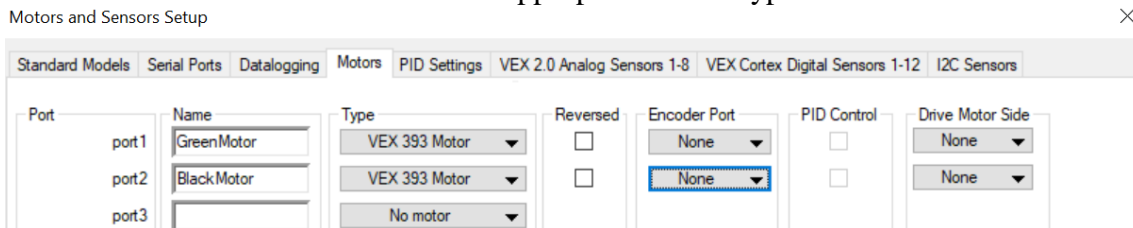
2. Save the file as *MotSpdCtrl1.c* (Feel free to choose a different name).
3. Click on **Motor and Sensor Setup** icon.



You will see a pop-up window appearing as shown below. Select **Motors** tab.



4. Create labels for the motors and select appropriate motor type.



Note: Port numbers in this pop-up window must correspond to the wiring. If a motor is wired to port 3, then the label must be created for port3 in this window.



- Click *Apply* and *OK*. The pop-up window will disappear. You should now see that a few lines of code are inserted in your source file. The labels you created will appear in this auto-generated code.

```

1 #pragma config(Motor, port1, GreenMotor, tmotorVex393_HBridge, openLoop)
2 #pragma config(Motor, port2, BlackMotor, tmotorVex393_MC29, openLoop)
3 /** Code automatically generated by 'ROBOTC' configuration wizard */
4
5 task main()
6 {
7
8     Space for your code
9
10 }

```

Programming to control motor speed:

Once motors are successfully configured, you can program them to control their speed.

- Your code must be written within the *main* function.
- Motor speed can be controlled from the programming environment by assigning different power levels to it. In order to assign a power level to a motor, the following syntax must be used.

motor[label_of_your_motor] = value of power level;

The *value of power level* can be any integer from -127 to +127. Setting the power level to 127 will apply full power to the motor and will make it spin at its maximum speed. Setting the power level to 0 will apply zero power to the motor and will turn it off. The sign associated with the power level indicates direction of rotation. Setting positive values will result in one direction of rotation while setting negative values will result in opposite direction of rotation.

Note: Two motors assigned with the same power level might spin at different speeds due to inherent variations in them.

- In order to insert a comment about your program, you must use two forward slashes. A comment will appear green in colour.

// assign full power to the motor connected to port 1

- Typically, you would want a particular section of code to be executed for a period of time. In order to achieve this goal, *wait* command is used. Here is the syntax for a *wait* command.

wait1Msec(wait_time);

// this code will cause the robot to wait a specified number of milliseconds before executing the next instruction in a program.

- To get a better understanding about the fundamentals of programming in RobotC, please refer to the document titled *Fundamentals_ROBOTC.pdf*. It is uploaded on Brightspace (*Contents* → *Design labs* → *Configuring motors & sensors lab* – Week of Jan 30).



By working together as a team, develop a piece of code to make the two motors connected to the microcontroller to spin in the same direction for five seconds. Feel free to choose any power level.

Testing the code:

- Before compiling your code, you **MUST** save it, or changes will not take effect
- To compile your code, go to **Robot** → **Compile Program**. If your code fails to compile, fix the reported errors and try again.



3. To compile your code and run it on the VEX controller, go to **Robot** → **Compile and Download Program**.
If your code fails to compile, fix the reported errors and try again.
4. When your code compiles and downloads, a *Program Debug* window will be opened.
5. To run your code, hit *Start* button on the *Program Debug* window.





Test your code to verify if the two motors connected to the microcontroller are spinning in the same direction for five seconds. Demonstrate your functioning code to the TA.

Questions to ponder:

1. What changes will you make to your code if the motors have to spin in opposite directions?
2. What changes will you make if the motors were connected to ports 5 and 9?
3. What changes will you make if the motors have to spin for 2 seconds and remain stationary for 2 seconds?

Exercise 2 – Motor ON/OFF Control Using Touch Sensors

Parts list:

Component	Quantity	Image
VEX 393 motor with integrated encoder module (<i>BlackMotor</i>)	1	
Bumper Switch (<i>button1; button2</i>)	2	

Wiring instructions:

1. Connect the VEX 393 motor with the integrated encoder module (*BlackMotor*) to port 1 of the MOTOR ports.
2. Observe your microcontroller and identify the DIGITAL ports. A more formal approach would be to examine the engineering drawing of the microcontroller and to identify the DIGITAL ports. The engineering drawing of the microcontroller is given in Figure 1. DIGITAL ports are in *section C* in the drawing. There are 12 DIGITAL ports.
3. Connect the Bumper Switches to *slot 1* and *slot 2* of the DIGITAL ports block on the VEX controller.

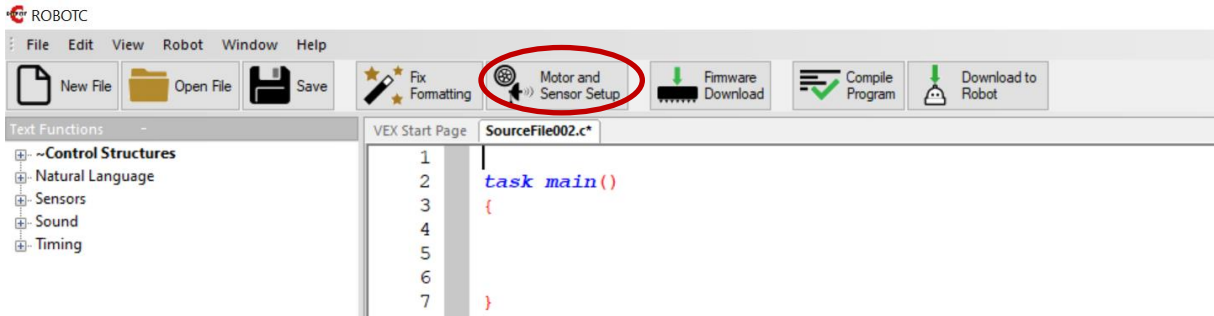


Once the wiring is completed, please wait for your TA to check it. Once the wiring is verified by your TA, you can proceed.



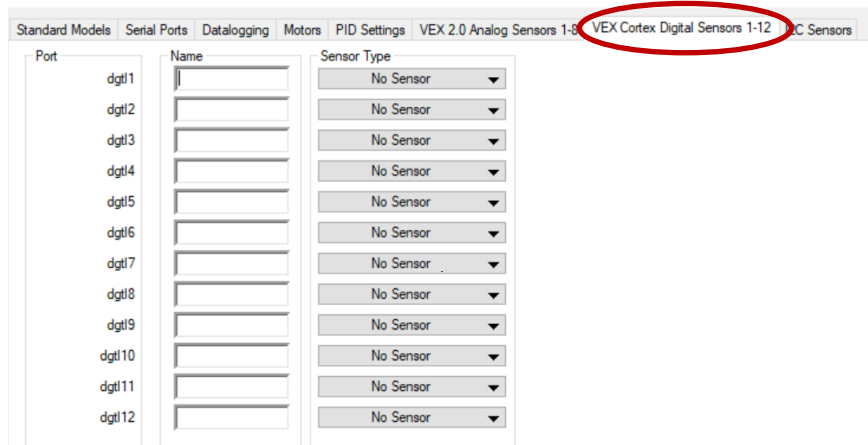
Configuring the touch sensors using software:

1. Create a new file and save it as *MtrSpdCtrl2.c*
2. Click on **Motor and Sensor Setup** icon.



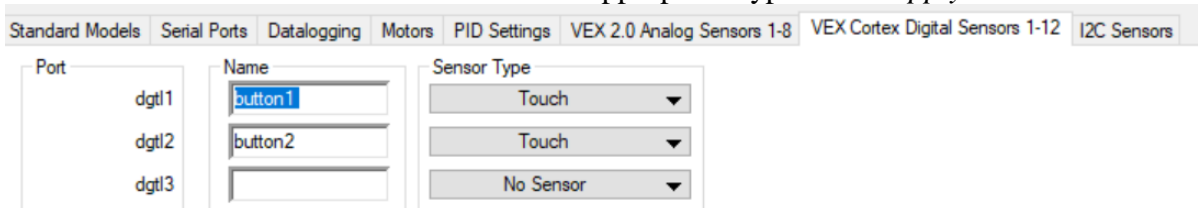
You will see a pop-up window appearing. Select the *VEX Cortex Digital Sensors 1-12* tab.

Motors and Sensors Setup



Note: Touch sensors are digital sensors. That is they can exist only in two possible states. A touch sensor can either be in “pressed state” or in “not pressed state”. Whenever it is pressed, it will return a value of 1 to the microcontroller. Whenever it is not pressed, it will return a value of 0 to the microcontroller.

3. Create labels for the touch sensors and select appropriate type. Click *Apply* and *OK*.



Note: Port numbers in this pop-up window must correspond to the wiring. If a touch sensor is wired to DIGITAL port 3, then the label must be created for dgtl3 in this window.



Programming using touch sensor values:

1. A touch sensor connected to the microcontroller will continuously return a value (0 or 1) depending upon its state. Typically, a decision is taken based on the state of a touch sensor. Therefore, it is essential to read the state of a touch sensor in the programming environment. The syntax to read a sensor value is `SensorValue(button1);` // *button1 is the user-defined label of the touch sensor connected to a port*
2. Once the sensor value is known, a decision can be made using a conditional statement in the programming environment. For example, if the touch sensor value is 1, then turn-on the motor. Figure 2 and Figure 3 depict the pseudocode of the popular conditional statements: *if* and *if-else*.

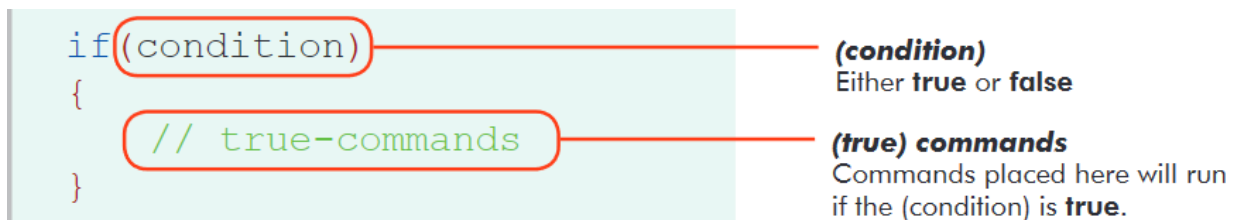


Figure 2. Pseudocode of an *if* statement.

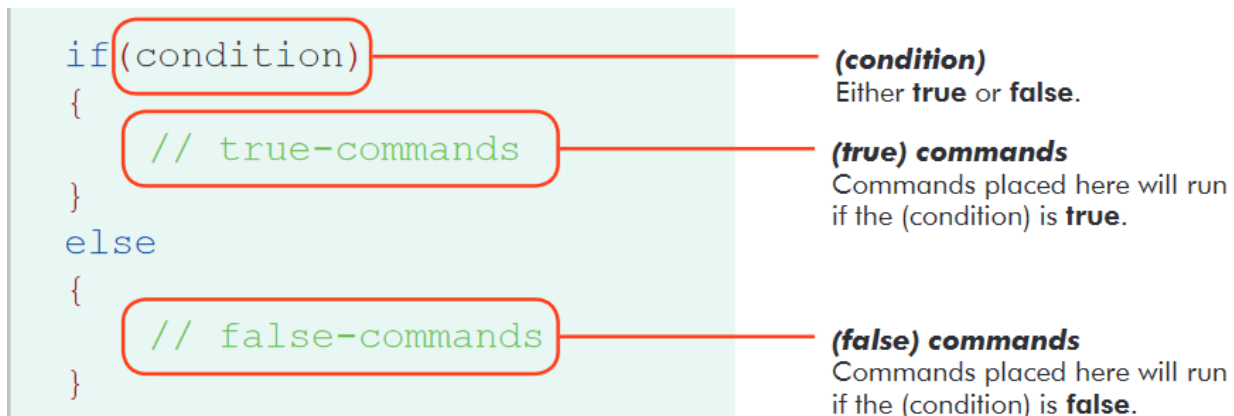


Figure 3. Pseudocode of an *if-else* statement.

3. To view a couple of examples using these conditional statements in ROBOTC environment, please refer to the document titled *ROBOTC_if-else.pdf* uploaded on Brightspace (*Contents* → *Design labs* → *Configuring motors & sensors – Week of Jan 30*).



By working as a team, write a piece of code to turn-on the motor connected to MOTOR port 1 at full power level when *button1* is pressed and turn-off the motor when *button2* is pressed. Remember to comment the code. Test the code by compiling and downloading it onto the microcontroller. Demonstrate your functional code to the TA.

Questions to ponder:

1. What will happen if *button1* is pressed twice?
2. What will happen if both *button1* and *button2* are pressed simultaneously?





Debugging

Many times the piece of code you write might not give you the intended outcome. This is usually because there is a mismatch between your way of thinking and your way of programming (how your code actually gets executed). In order to follow your code execution, it is important to familiarize with *debugging*.

1. Once your program is compiled and downloaded on to the robot, you will see the *Program Debug* window.
2. To run one line of your code, hit the *Step Into* button on the *Program Debug* window. This will allow you to see what a single line of code does. You can advance through the code by one line at a time by hitting this button.
3. You can also check the values of sensors and variables in the debugger.
 - Select **Robot** → **Debugger Windows** and ensure that there are check marks next to ‘Global Variables’, ‘Local Variables’, ‘Motors’ and ‘Sensors’.
 - You should see ‘Global Variables’, ‘Local Variables’, and ‘Motors’ and ‘Sensors’ tags in the bottom pane of the screen.
 - When you click the ‘Sensors’ tag, you can see what the current values from the sensors are. Push both buttons and see how the values change in the debugger window. Examine what is visible under the ‘Motors’ tab when the motor is running and not running.
4. Setting a *breakpoint* is another helpful way to debug your code. A *breakpoint* can be set at any line of your code by clicking once on the grey bar. If a *breakpoint* is set, a red dot should appear next to the line in the grey bar. To remove a *breakpoint*, click on the red dot once.
5. After setting a breakpoint, run the code by hitting the *Start* button in the *Program Debug* window. You will see that the execution stops at the line where the breakpoint is set. When this happens, there should be a yellow arrow on top the red dot on the grey bar. By selecting *Resume* the code executes and stops at the next breakpoint (or in the same one after looping through).
6. To leave the debugger and return to the editor, just close the Program Debug window.

Exercise 3 – Motor Turn-Off Control Using Encoder Value

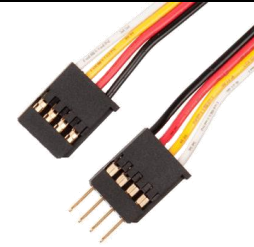
Parts list:

Component	Quantity	Image
VEX 393 motor with integrated encoder module (<i>BlackMotor</i>)	1	
Bumper Switch (<i>button1</i>)	1	



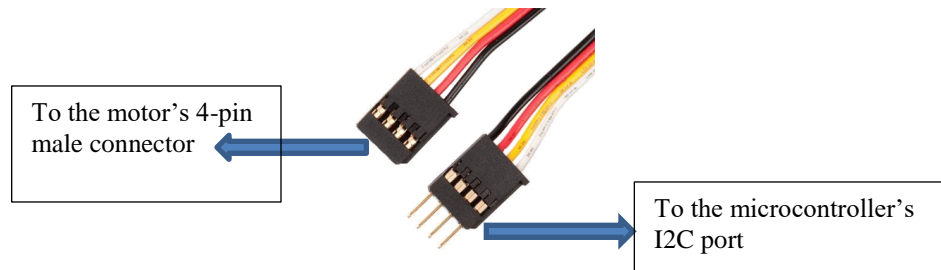
4-wire extension cable for I2C connection
between the motor and the microcontroller

1



Wiring instructions:

1. Connect the VEX 393 motor with the integrated encoder module (*BlackMotor*) to port 1 of the MOTOR ports.
2. Connect the Bumper Switch (*button1*) to *slot 1* of the DIGITAL ports block on the VEX controller.
3. Observe your microcontroller and identify the I2C port. A more formal approach would be to examine the engineering drawing of the microcontroller and to identify the I2C port. The engineering drawing of the microcontroller is given in Figure 1. I2C port is in *section B* in the drawing.
4. Connect the 4-wire extension cable from the output of the integrated encoder module in the motor to the I2C port of the microcontroller. The VEX 393 motor with the integrated encoder module has a 4-pin slot to accept I2C input and a 4-pin male connector to send I2C output. Figure 4 given below shows the appropriate terminals for connection.



Note: Sometimes the VEX controller does not properly initialize an encoder so it is not read correctly. If your motor's encoder is properly connected the light on the back of the motor should give two quick green flashes about every 3 seconds. If it is not addressed correctly, the light will not flash or it will be flashing or solid amber. In this case, power down the VEX controller completely by turning off the power and disconnecting the programming cable followed by reconnecting the programming cable and powering the controller back up to see if this causes the encoder to be properly initialized. If this does not work call a TA over to see if they can help you.



Once the wiring is completed, please wait for your TA to check it. Once the wiring is verified by your TA, you can proceed.



Configuring the encoder using software:

1. Click on **Motor and Sensor Setup** icon. Select the **Motors** tab. Configure the motor and select the **Encoder Port** to I2C_1. Click **Apply** and **OK**.

Motors and Sensors Setup

Port	Name	Type	Reversed	Encoder Port	PID Control	Drive Motor Side
port1	motor1	VEX 393 Motor	<input type="checkbox"/>	I2C_1	<input type="checkbox"/>	None
port2		No motor				
port3		No motor				

Programming using encoder value:

An encoder outputs the angular position of the motor to which it is connected to. The encoder connected to the VEX 393 motor outputs a value of 627 when the motor completes one rotation. The output of the encoder continues to increase as long as the motor spins. In order to read the value of the encoder in the programming environment, the following command is used.

`getMotorEncoder(BlackMotor);` // *BlackMotor is the label of the motor connected to a MOTOR port*

The initial value of the encoder output can be anything. Hence, it is important to reset the value of encoder output to zero before measuring angular position. The command to do that is given below.

`resetMotorEncoder(BlackMotor);` // *BlackMotor is the label of the motor connected to a MOTOR port*



By working as a team, program the microcontroller to turn the motor once (i.e. about 360 degrees) when *button1* is pushed and if the motor is currently stationary. Pressing the button again during the rotation should not cause the rotation to repeat. Choose 50 for motor power level.

Demonstrate the functional code to the TA. Remember to comment your code.

Questions to ponder:

1. What is the value of the encoder output when the motor stops? Why? What is its implication?
2. What changes will you make to your code to make the final encoder value close to 627?

END OF LAB