AURA PRESENTS

ROBOTICS WORKSHOP

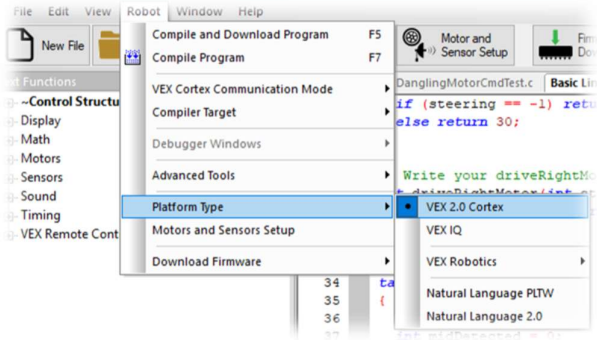
PARTS 2-3 OF 3 — LINE FOLLOWING CHALLENGE — ROOM: 405-328

1 INSTALLING ROBOTC

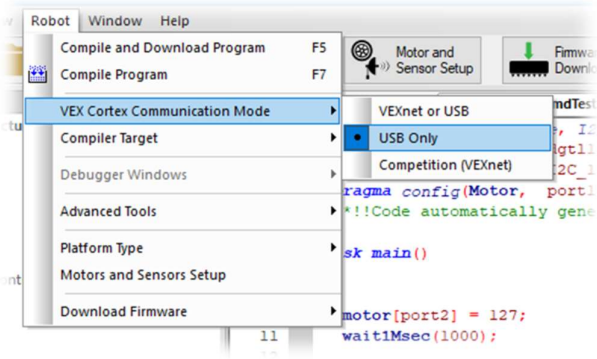
Visit http://www.robotc.net/download/vexrobotics/ to download and install ROBOTC on your computer. Once installed, run ROBOTC.

IMPORTANT SETTINGS

Make sure that the Platform Type is selected properly. Go and select Robot > Platform Type > VEX 2.0 Cortex.



Also make sure that the VEX Cortex Communication Mode is set properly. Go and select Robot > VEX Cortex Communication Mode > USB Only.



RobotC Official Reference Documentation

Throughout this handout we will give you links to the official documentation for ROBOTC, which gives a lot more information than this flimsy document. If in doubt, check it out! The home page for this official documentation is here:

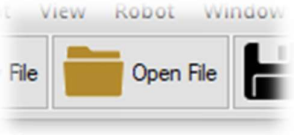
http://help.robotc.net/WebHelpVEX/index.htm

2 CREATING A NEW PROGRAM

Throughout this part of the workshop, we will be using a preconfigured program template rather than starting from scratch. Download the template here:

http://aura.org.nz/workshop2019
(Select the file *WorkshopTemplate.c*)

Open this template using the big "Open File" button.



Whenever you want to work on a new program for this workshop, just Save-As the file under a new name (File > Save As).

3 VEX CORTEX

The *VEX Cortex* is the brain of the robot you will be programming in these workshops. The motors and sensors you will be using are wired up to the ports in the Cortex. You can download your program onto the Cortex to control them.

4 MOTORS & SENSORS SETUP

Before you can write a program, you need to let ROBOTC know about the motors and sensors you are

using and what ports you are plugging them into on your Cortex. For the workshop, you won't need to do this as we're using the template code that has everything preconfigured for the robot you'll be using.

To setup the motors and sensors, use the toolbar to go to the Motors and Sensors Setup window (the big *"Motors and Sensors Setup"* button > Robot > Motors and Sensors Setup.

In this window, you can configure your motors and sensors in the ***Motors*** and ***Sensors*** tabs respectively. Enter appropriate names for the motors and sensors in the boxes that correspond to the physical ports they are plugged into.

RobotC Reference – Motors & Sensors Setup

http://help.robotc.net/WebHelpVEX/index.htm#Resources/topics/Getting_Started/Motors_and_Sensors_Setup.htm

The motor and sensor names used in the template are:

- `leftDrive`
- `rightDrive`
- `leftLineTracker`
- `midLineTracker`
- `rightLineTracker`

5 DRIVING & TURNING

To make the wheels on your robot turn, you need to set the "speed" of the motor connected to the wheel. The "speed" value ranges from `-127` to `+127`, where `-127` is full speed in reverse, `+127` is full speed forwards and `0` is no power. The speed values for all motors are stored in a special array called `motor`.

In the previous Motors & Sensors Setup section, we gave names to each motor. We would use the same name here when we're setting the speed of the motors. For instance, if we've named a motor `leftDrive`, the following would turn that motor on at full speed forwards:

```
motor[leftDrive] = 127;
```

RobotC Reference – Motors

http://help.robotc.net/WebHelpVEX/index.htm#Resources/topics/VEX_Cortex/ROBOTC/Motor_and_Servo/motor.htm

Once the motor speed is set, it will continue to be powered on until you explicitly tell it to stop or until the program terminates. To leave it on for a fixed amount of time, use the following function to pause the program for the specified amount of time:

```
sleep(<number of milliseconds>);
```

RobotC Reference – Sleeping

http://help.robotc.net/WebHelpVEX/index.htm#Resources/topics/VEX_Cortex/ROBOTC/Timing/sleep.htm

EXAMPLE – DRIVE STRAIGHT FOR 1 SECOND THEN STOP

```
// Drive Straight
motor[leftDrive] = 127;
motor[rightDrive] = 127;

// Leave the motors on for 1 second
sleep(1000);

// Sets the motor speed to 0
motor[leftDrive] = 0;
motor[rightDrive] = 0;
```

EXAMPLE – TURNING

In the previous example, both wheels ran forwards to make the robot drive straight. Similarly, you can make one wheel go forwards and one wheel go backwards to make the robot turn on the spot.

```
// Turn Right
motor[leftDrive] = 127;
motor[rightDrive] = -127;

// Leave the motors on for 1.5 seconds
sleep(1500);

// Turn Left
motor[leftDrive] = -127;
motor[rightDrive] = 127;

// Leave the motors on for 1.5 seconds
sleep(1500);

// Sets the motor speed to 0
motor[leftDrive] = 0;
motor[rightDrive] = 0;
```

Real motors are not perfect

Due to friction, wear and tear inside the motors, each motor may run slightly slower or faster than other motors, and the robot usually won't drive perfectly straight with our simple code above.

6 TASK – CONTROLLING MOTORS

First, open the template program and save it under a different name.

Look at the template code. You will see some `#pragma` statements, then a `task auton()`, and finally a `task main()`. For the rest of the workshop, you should only write code into the `auton` task and write functions between the `auton` task and the `#pragma` statements.

Next, type the following code inside the `task auton()`.

```
// Drive Straight
motor[leftDrive] = 127;
motor[rightDrive] = 127;

// Leave the motors on for 1 second
sleep(1000);

// Turn Right
motor[leftDrive] = 127;
motor[rightDrive] = -127;

// Leave the motors on for 1.5 seconds
sleep(1500);

// Turn Left
motor[leftDrive] = -127;
motor[rightDrive] = 127;

// Leave the motors on for 1.5 seconds
sleep(1500);

// Sets the motor speed to 0
motor[leftDrive] = 0;
motor[rightDrive] = 0;
```

To try this program out, read the next two subsections.

COMPILE, DOWNLOAD, & RUN ON A PHYSICAL ROBOT

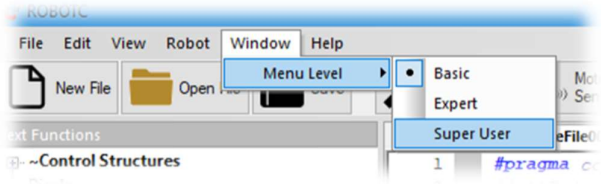
To send your code to the robot, you need to ***Compile*** it and then ***Download*** it to the robot. To do this, you can hit F5 on your keyboard or press the big ***"Download to Robot"*** button. Make sure that you've connected your Laptop to the VEX Controller before you download your program.

It is also a good idea to regularly compile your code as you go to check for errors. You can do this by hitting F7 on your keyboard or by clicking the big ***"Compile Program"*** button.

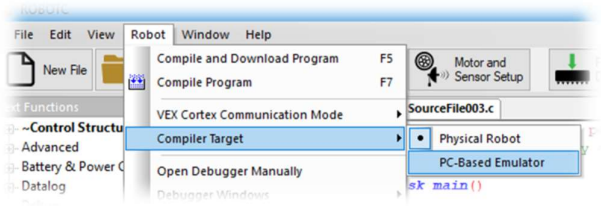
TESTING WITHOUT A ROBOT – ROBOTC PC-EMULATOR

Waiting in the queue for the robot probably takes ages. Fortunately, there's a way to test your program without needing a physical robot.

First, select Window > Menu Level > Super User.



Next, select Robot > Compiler Target > PC-Based Emulator.

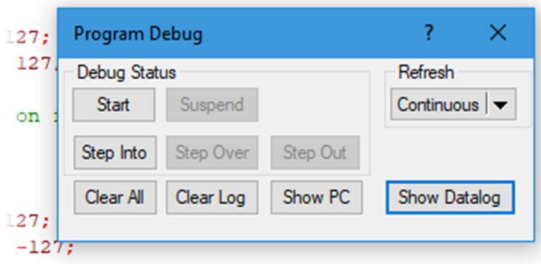


Now, when you click the "Download to Robot" button, it will no longer download the program to a physical robot. Instead, it is downloaded to a system inside ROBOTC that simulates a robot's VEX Cortex.

Common Mistake – Wrong Compiler Target

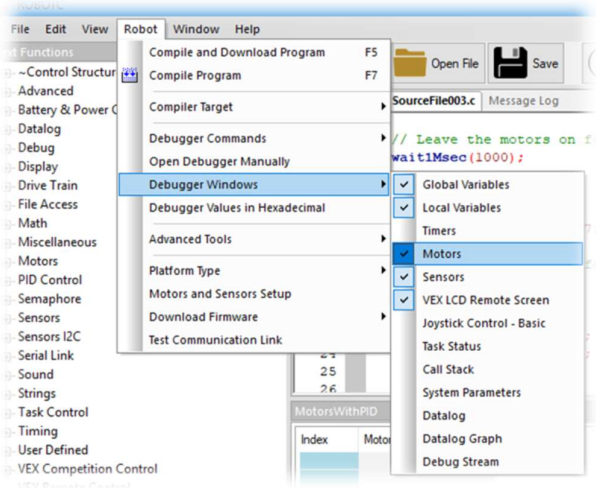
When it's time to test your code on an actual, physical robot, don't forget to switch the compiler target back to "Physical Robot", or else the robot will be running the previous person's code and you will be spectacularly confused. *I learnt the hard way :'(*

Downloading your code to the emulator will open a new dialog box:

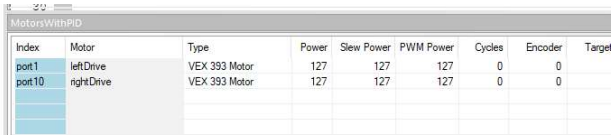


When you see this "Program Debug" dialog box, you are in ***Debugging Mode***. To exit out of debugging mode, simply close this dialog box, or click the giant "Exit Debugger" button in the toolbar.

At first, the emulator starts up as paused. Before we start the emulation, let's open up a debugger window that shows the motor status in real time. Select Robot > Debugger Windows > Motors.



This opens a panel at the bottom. At first, it will be empty, but it will show what motors are available when you click "start" in the "Program Debug" dialog box. So, without further ado, start the emulation and see the values change over time!

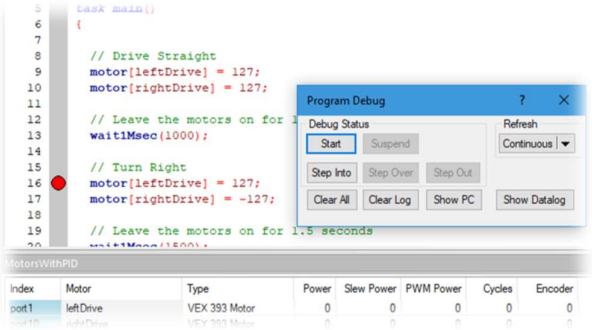


DEBUGGING TECHNIQUES – BREAKPOINTS & STEPPING

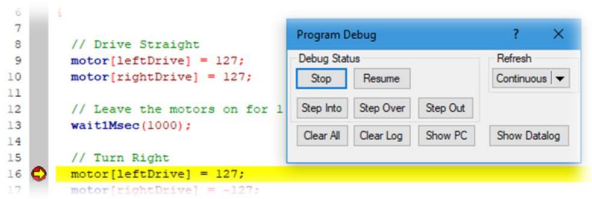
While the compiler can detect most mistakes in your program, in a lot of times the program is perfectly valid and compiles successfully, but it doesn't do what you want it to do. Here is when the debugger is very useful. You can use the debugger both when you're using the emulator and when you're connected to a physical robot.

First, enter Debugging Mode either by downloading the code to the robot, or by selecting Robot > Open Debugger Manually. While you're at it, also select Robot > Debugger Windows > Local Variables.

Next, see the grey bar between the line numbers and the code? Click on this grey bar right after the `// Turn Right` comment. You should then see a red dot. This puts a ***breakpoint*** on that line and tells the debugger to pause the program when we reach that line. (To remove the breakpoint, click on the red dot again).



Now, start the program by pressing "Start". After one second, you'll get notified that you've hit a breakpoint. The yellow line shows which line the program is about to execute.



Next, click "Step Into" to advance the program by one statement. Notice that the yellow line has shifted. Now look at the MotorsWithPID panel, click "Step Into" again, and see how the motor power for the `rightDrive` motor has changed from 127 to -127.

Finally, click "Resume" to let the program finish on its own.

When your program compiles but doesn't work as intended, use this tool to step through your program and check that each line is doing what you expect it to do. If you're using variables, the Local Variables panel will show what value each variable has while you step through it, so you can also use that to verify your program's correctness.

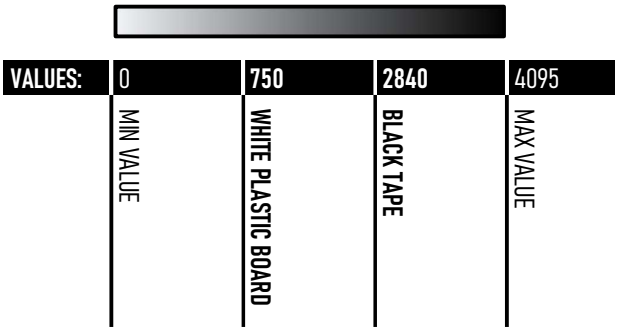
RobotC Reference – Program Debug

http://help.robotc.net/WebHelpVEX/index.htm#Resources/topics/ROBOTC_Debugger/Debug_Windows/Program_Debug.htm

7 THE LINE TRACKER SENSOR

The line tracker sensor can be used by your robot to detect lines and follow it.
In ROBOTC, the Line Tracker Sensor can give you a number between 0 and 4095, depending on the surface is detecting. Light surfaces will return a low value while darker surfaces will return a higher value.

For your workshop, you will be following a line made of black tape on top of a white plastic corflute board. The black taped line gives a sensor value of `2840`, and the white plastic board gives a sensor value of `750`:



The value that the Line Tracker senses can be found using the `SensorValue(NameOfSensor)` function, where the variable `NameOfSensor` is replaced with the name of the line tracker in the Motors and Sensors setup. In our case, we would be using the following function calls:

```
SensorValue(leftLineTracker);
SensorValue(midLineTracker);
SensorValue(rightLineTracker);
```

To make a simple Line Following Robot, you will need to determine a threshold value that will help the robot decide if it's seeing a black line or not.

You can use the Debugger and ROBOTC Emulator to calibrate your Line Tracker Sensor and find out what values it reads when it senses a light and dark surface.

Note

For the purpose of these workshops, you won't need to calibrate the sensors and measure the sensor values yourselves, as the sensor values are given above.

Here's an example code showing you how you can read the current sensor value from the line trackers and react to it:

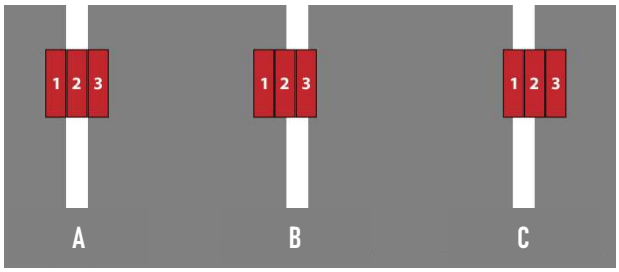
```
// Get the current detected value from the
// Middle Line Tracker
midSensor = SensorValue(midLineTracker);

if (midSensor < 1000){
  // Drive straight
  motor[leftDrive] = 127;
  motor[rightDrive] = 127;
}
```

RobotC Reference – SensorValue

http://help.robotc.net/WebHelpVEX/index.htm#Resources/topics/VEX_Cortex/ROBOTC/Sensor/sensorValue.htm

8 TASK: SIMPLE LINE FOLLOWING



You now know how to use the Line Tracker Sensor. In the previous example, you learnt how to make use of one sensor to determine whether a robot was on the line or not. Using more sensors means you can get more data and therefore make your robot follow the line better.

The robot you will be using comes with 3 sensors. The diagram above illustrates how you could use these sensors to determine your robot's position on the line it should be following.

If the middle sensor detects the line (**A**), then it means your robot is on the line and therefore your robot should drive straight.

If the right sensor detects the line (**B**), then it means that your robot has turned too far to the left and therefore needs to turn to the right to correct its position.

If the left sensor detects the line (**C**), then it means that your robot has turned too far to the right and therefore needs to turn to the left to correct its position.

Does this sound familiar? This is very similar to the problem you solved in **Section 7 of the C portion of this workshop.**

Using the information that you have now learnt about ROBOTC, try to adapt your code from Section 7 so that you can create a ROBOTC Line Following Program.

9 FINER LINE POSITION

So now you know whether your robot is on the left-side of the line, the right-side of the line, or square in the middle. Great! Can we do better though? In the previous task, if the robot wasn't exactly on the line, you made the robot ***turn at a certain speed.*** What if the speed at which the robot corrected itself depended on how far the sensor was from the line? For example, exactly how far away from the line are we?

One way of obtaining greater precision when measuring the line's position is to add more sensors. With five sensors, we can detect the line in five different locations. With twenty sensors, it's even better. With more than that, eventually we've made ourselves a 1D camera and we'll need to start teaching you an entire course on computer vision as well.

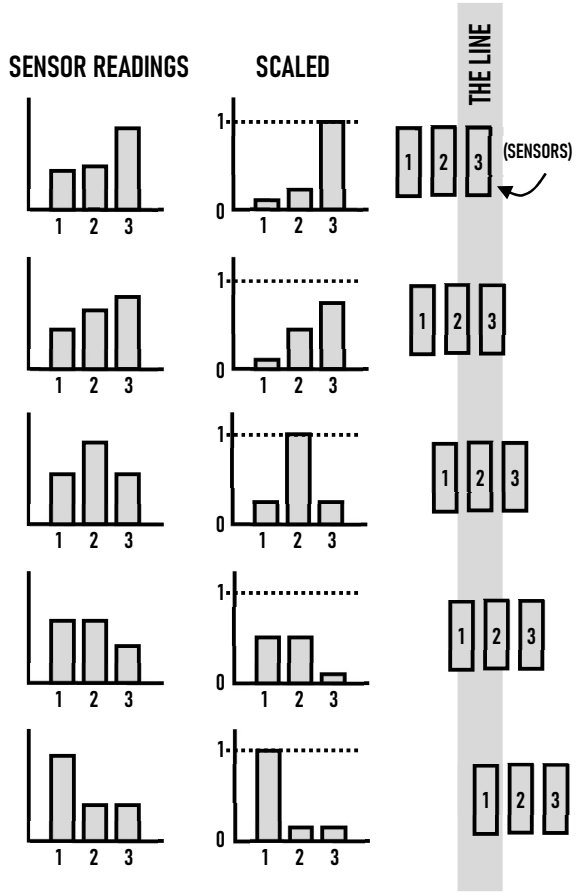
However, sensors take up space and money, and you're limited to the three sensors for this workshop anyway. Instead of adding more sensors, we will show you an alternative method of obtaining greater precision when measuring the line's position. We will rely on the fact that the sensors are analogue sensors.

Refer back to the diagram at the top of Section 7. Here's how the sensor reacts to each situation.

SENSOR	EXAMPLE A SENSOR SEES/ SENSOR VALUE	EXAMPLE B SENSOR SEES/ SENSOR VALUE	EXAMPLE C SENSOR SEES/ SENSOR VALUE
1	Light/Low	Light/Low	Dark & Light/ Medium
2	Dark/High	Dark & Light/ Medium	Dark & Light/ Medium
3	Light/Low	Dark & Light/ Medium	Light/Low
ACTION TO TAKE	No change to motor power	More power to left side	More power to right side

As you can see, the reading from the Line Tracker Sensors changes as the amount of white/black changes. The closer the black line is to a sensor, the higher the sensor value.

What happens when we try visualising these sensor readings onto bar graphs? To make things easier to see, lets scale the sensor values so that changes are emphasised.



Notice how the actual line's position is hinted by where the inverted bars' average horizontal location is. What's that term called in statistics again? It's the ***mean***. Let's see how we can calculate this with the following example sensor values:

SENSOR 1	SENSOR 2	SENSOR 3
`952`	`1010`	`2740`

First, let's clean up the sensor data so it makes most sense to us. If we knew that the sensor value for black tape is `2840`, and the sensor value for the white ground is `750`, then we can subtract the sensor values by `750` and divide each result by `(2840 - 750)`. Doing this gives us nice values between `0` and `1` where `1` represents being on black tape and `0` represents being on the ground.

SENSOR 1	SENSOR 2	SENSOR 3
`0.0967`	`0.1244`	`0.9522`

Next, we can think of these values as giving us a relative *measure of confidence* that the white line is underneath it. Say, to illustrate, that the sensors are positioned at -1cm, 0cm, and 1cm from the centre of the robot. We can then calculate the mean position weighted by the adjusted sensor values:

$$\frac{-1 \times 0.0967 + 0 \times 0.1244 + 1 \times 0.9522}{0.0967 + 0.1244 + 0.9522} = 0.7291$$

We then get a pretty good estimate that the line is at position 0.7291cm from the robot's centre.

For Astute Students

If you carefully look at the mathematics behind the concepts above, you might realise that estimating the line's position can be amazingly simplified to:

```
someConstant * (leftValue - rightValue)
```

It's a bit anticlimactic, isn't it? I could've just given you this line of code from the beginning. But, what's the fun in that.

10 TASK – BETTER LINE FOLLOWER

Create a line following program that estimates the actual position of the taped line and uses this estimate to determine how much to turn. The further away from the line, the greater the amount the robot should turn. The closer the robot is to the line, the less it should turn.

Hint: For each millimetre the line is further away from the centre of the robot, how much more power should be given to the motors for turning the robot? You'll need to test and find this out yourself.

11 BONUS TASK: PID CONTROL

In the previous task, we have essentially programmed something called a "Proportional Controller". While it has the potential to out-perform the basic line following algorithm from the earlier task, you may find that it may not work consistently well throughout the course.

The ultimate general-purpose algorithm we will show you is called the PID Controller. It requires more code and more tuning, but it should make your line following perform much better.

We have material for PID Control for which this column is too narrow to contain. Instead, we recommend you to the following book by our one and only George Gillard (2017-2018, as the chairperson of AURA that is).

Recommended reading:

- George Gillard. *An Introduction to PID Controllers*. Second Edition. http://georgegillard.com/documents

The End.