ROBOTC Tutorial Documentation

Release

Ian Moore

Mar 16, 2018

Contents:

1 Setting up ROBOTC **1**
- 1.1 Switching the compiler to VEX 2.0 Cortex from VEX IQ . . . 1

2 Updating the Cortex and Controller **3**
- 2.1 Updating the Cortex . . . 3
- 2.2 Updating the Controller . . . 3
- 2.3 Troubleshooting . . . 4

3 Creating your first program **5**
- 3.1 Opening the sample program . . . 5
- 3.2 Getting familiar . . . 5
- 3.3 Adding your own controls . . . 6

4 Stepping it up a notch **9**
- 4.1 Single-joystick control . . . 9
- 4.2 Thresholds . . . 9

5 Indices and tables **11**

CHAPTER 1

Setting up ROBOTC

Often times, ROBOTC will not already be configured and ready to use for the VEX 2.0 Cortex. The process to set it up though is quite easy.

1.1 Switching the compiler to VEX 2.0 Cortex from VEX IQ

1. Click on **Robot** in the top menu bar
2. Click on **Platform Type**
3. Click on **VEX 2.0 Cortex**, you should notice the Text Functions menu on the side update to reflect this change.

CHAPTER 2

Updating the Cortex and Controller

It is important to stay up to date of firmware updates for both the VEX 2.0 Cortex and Controller. Firmware updates often fix bugs and can sometimes add new features that are only available in newer versions of ROBOTC. Updating these devices is easy.

2.1 Updating the Cortex

1. Plug one end of the orange cable into the USB port on the Cortex and the other end into the computer.

Note: If this is your first time plugging in the Cortex, an icon may appear in the system tray or a window that says it is installing the necessary drivers for the Cortex. Simply wait for it to finish before continuing.

2. Click on **Robot** in the menu bar, then **Download Firmware**, and then **Automatically Update VEX Cortex**.

Warning: Occasionally the Cortex, will get stuck while flashing the update, if this happens try updating it manually. First update the **Master CPU Firmware** by going to the **Download Firmware** drop down and then **Manually Update Firmware** and then **Master CPU Firmware** and select **Standard File**. Next update the **ROBOTC Firmware** by going to the same **Manually Update Firmware** drop down and then **ROBOTC Firmware** and then selecting **Standard File**.

2.2 Updating the Controller

1. Plug one end of the orange cable into the USB port on the Controller and the other end into the computer.
2. Click on **Robot** in the menu bar, then **Download Firmware**, and then **Automatically Update VEXnet Joystick**.

2.3 Troubleshooting

Sometimes the updater will fail to find the Cortex or Controller, when this happens, try the following.

1. Unplug the device from the orange cable and the plug it back in.
2. If that doesn't solve the issue, try restarting the device (which requires you to unplug it first).
3. If that still doesn't work, try restarting the computer.

CHAPTER 3

Creating your first program

ROBOTC provides many sample programs that make for a great way to get started. We will be starting by using **Dual Joystick Control.c**

3.1 Opening the sample program

1. Click on **File** in the menu bar.
2. Click on **Open Sample Program**
3. Make sure you are in the folder for VEX2 and then open the folder **Remote Control**
4. Select **Dual Joystick Control.c** and open it up
5. Go to **File**, **Save as** and save it in a safe location with a relevant name

3.2 Getting familiar

By now you should be looking at the sample program for single joystick control.

At the top you will notice the motor configuration, it should look like this:

```
1 #pragma config(Motor,  port2,           rightMotor,    tmotorNormal, openLoop,␣
  ↪reversed)
2 #pragma config(Motor,  port3,           leftMotor,     tmotorNormal, openLoop)
3 //*!!Code automatically generated by 'ROBOTC' configuration wizard
```

Here we have two motors configured, the right motor (named `rightMotor`) in port 2, and the left motor (named `leftMotor`) in port 3. If we click on **Motor and Sensor Setup** in the top bar and then go to the **Motors** tab, we can see the listing for all the motors and their ports. This setup window is the best way to configure your motors. Make sure to always give them a recognizable name, set the motor type to the correct one listed on the back of the motor, and enable or disable Reversed depending on your needs.

If we look further down in the file, we see the code for controlling the motors with the joysticks:

```
23 task main ()
24 {
25
26   while(1 == 1)
27   {
28     motor[leftMotor]  = vexRT[Ch3];   // Left Joystick Y value
29     motor[rightMotor] = vexRT[Ch2];   // Right Joystick Y value
30   }
31
32 }
```

Here we can see one of the most basic programs you can write in ROBOTC. The program begins with `task main ()` which is the way of telling the computer, this is where our code begins.

Next we see `while(1 == 1)`, this is an infinite loop that will run whatever code is inside the curly braces as long as 1 is equal to 1, or forever.

Going down further we see `motor[leftMotor]` this is the way we access one of our named motors. Then we see it is equal to `vexRT[Ch3]`, which if you look on your controller, corresponds with the label for vertical axis of the left joystick. By saying `motor[leftMotor] = vexRT[Ch3]`, we are telling the robot to set the motor power for `leftMotor` to the value of the `Ch3` joystick, which is 0 when its in the middle, -127 when it is all the way down, and 127 when it is all the way up. We see the same idea with `rightMotor`. Taking this basic idea, you can change which channels motors use and add more motors on the same or different channels.

3.3 Adding your own controls

While it is nice to be able to control motors by different joystick channels, the controller has tons more buttons that can also be utilized. Here are some examples on how to add more functionality to your bot.

3.3.1 Adding an arm

In this example, we will be using a motor named `arm` that is controlled by `Btn5U` and `Btn5D` on the controller.

```
if(vexRT[Btn5U])
{
    motor[arm] = 127;
}
else if(vexRT[Btn5D])
{
    motor[arm] = -127;
}
else
{
    motor[arm] = 0;
}
```

This here is a basic, if else-if else. The program will check the first condition (`vexRT[Btn5U]`) to see if it is true, and if it is it will run the code inside the first set of curly braces. If the first condition is not true, it will then look at the next one (`vexRT[Btn5D]`), if that is true it will run the code inside the second set of curly braces. If none of those are true, it will run the code in the last set of curly braces.

You may be thinking why do "I need `else if`"? The reason we use `else if` is because we want to make this one whole statement where if the first one is not true it will then check the second one and if that is not true, finally default

on the last one. A common mistake is the make two separate if else statements like this:

```
//Do not do it like this

if(vexRT[Btn5U])
{
    motor[arm] = 127;
}
else
{
    motor[arm] = 0;
}

if(vexRT[Btn5D])
{
    motor[arm] = -127;
}
else
{
    motor[arm] = 0;
}
```

The problem with this is that if you press `Btn5U` it will trigger the arm to go at 127 power as expected, however when the program reaches the next if statement for `Btn5D`, it will be false and it will set the arm to 0. This will cause the bot to spasm between to contradicting statements. By simply combining it into an if else-if else we avoid this by making sure the arm will stop only if both are buttons are not pressed.

CHAPTER 4

Stepping it up a notch

Now that you have a basic program, why not work on adding a few more advanced things.

4.1 Single-joystick control

Sometimes it is preferable to control the bot using a single joystick.

Earlier we had:

```
motor[leftMotor]  = vexRT[Ch3];   // Left Joystick Y value
motor[rightMotor] = vexRT[Ch2];   // Right Joystick Y value
```

This allowed us to use the left and right joysticks to control their corresponding wheels. However, let's say we only wanted to use the left joystick, more like a video game controller is used to move the player.

Here, we will use some math the calculate the speed at which each wheel should move:

```
motor[leftMotor]  = vexRT[Ch3] + vexRT[Ch4];  // Left Joystick Y value + Left␣
↪Joystick X Value
motor[rightMotor] = vexRT[Ch3] - vexRT[Ch4];  // Left Joystick Y value - Left␣
↪Joystick X Value
```

As you can see, we are adding `Ch4` (x-axis) to `Ch3` (y-axis) to move the left motor, and we are subtracting `Ch4` from `Ch3` to move the right motor.

This works because as we push the joystick further right (a positive x-value) we increase the power of the left motor while decreasing the power of the right motor, turning the bot rightwards, and vice versa.

4.2 Thresholds

The joysticks are not perfect, and sometimes they will report value (albiet small) even when untouched. To prevent this, we can implement a threshold to make sure the wheels (or anything else that uses joystick values) only move

when the joystick is pushed past a certain value.

First we will create a function called `threshold` at the bottom of our code (below `task main`):

```
int threshold(int value, int threshold)
{
        return abs(value) > abs(threshold) ? value : 0;
}
```

You may be wondering what the `? :` is, that is a ternary operator, it works essentially like an inline if-else statement. They can help us to write out code in a shorter amount of lines, however they should only really be used for basic things. You should never but a ternary operator in a ternary operator.

Next we need to declare our function at the top of our code, since it is at the bottom, any code in `task main` that references it will throw an error since to compiler does not know the function exists until it reaches the bottom of the code. A declaration is our way of telling the compiler "don't worry, I'll define it later".

So, at the top of your code (right above `task main`), simply add:

```
int threshold(int value, int threshold);
```

Now we can use our new function in our code! It takes two inputs, the value (an integer), the threshold (an integer) and it outputs an integer (the value if it is greater than the threshold or 0 if not).

An example of how we would use it is:

CHAPTER 5

Indices and tables

- genindex
- modindex
- search