# FTC-Iowa 3-day Coaches Workshop June 13<sup>th</sup> to 15<sup>th</sup>, 2010 – University of Iowa

Supplemental Guide for ROBOTC Programming
Using an ORTOP Tetrix Try-It demo robot

# **Contents:**

ABOUT THIS GUIDE	3
STARTING UP ROBOTC FOR THE FIRST TIME	4
STARTING A NEW FILE	7
DRIVE CONTROL	10
Bare Minimum	10
Testing	11
Basic	15
Better	17
MANIPULATOR CONTROL	19
Using Servo Debug Window	19
Arm Control	21
Gripper Control	22
Enhancing the Servo Control Code	23
SOME TIPS & ADVICE	29
REFERENCES/RESOURCES	31
APPENDIX / FULL CODE EXAMPLES	33
Getting Started	33
Drive Control	33
Manipulator Control	35

## **About this Guide**

This guide is a supplementary aide for those attending the 3-day FTC coaches training camp at the University of Iowa. It does not contain all the material needed to jump into ROBOTC, but rather it contains step-by-step references for the example code used in the workshop. It is the intent that this guide be used in conjunction with a training class or other materials in order to explain the reasoning and fundamentals behind many of the examples.

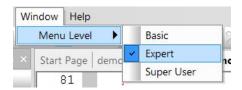
The code examples are specific to the demo robot design exceptionally well-documented in ORTOP's Tetrix Try-It instructions which can be found here courtesy of Oregon Robotics Tournament and Outreach Program:

http://www.ortop.org/ftc/res.htm#TT

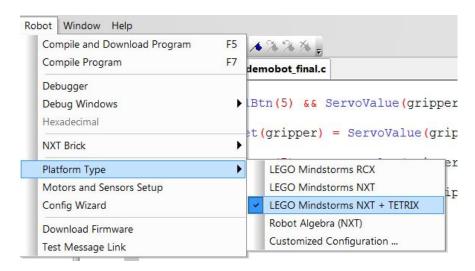
## Starting up ROBOTC for the first time

The first time you fire up the ROBOTC IDE, there are a few quick things you will want to do before you begin programming a FTC robot.

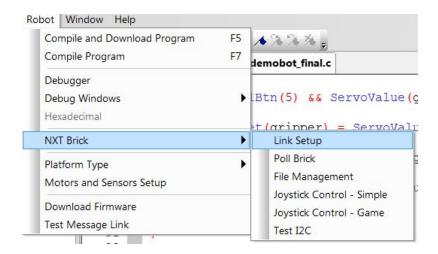
- 1. Set menu level from basic to "Expert".
  - a. Window, Menu Level, Expert



- 2. Set platform type to "LEGO Mindstorms NXT + TETRIX".
  - a. Robot, Platform Type, LEGO Mindstorms NXT + TETRIX



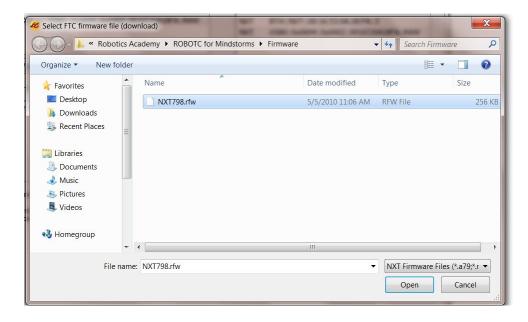
- 3. Download latest firmware to NXT brick.
  - \* Note that this only needs to be done the first time you setup a new NXT brick with ROBOTC and when you upgrade your version of ROBOTC.
  - \*\* Also note that the battery level must be high enough on the NXT before it will allow you to download new firmware.
    - a. Make sure NXT is connected via USB and turned on.
    - b. Open Link Setup dialog
      - i. Robot, NXT Brick, Link Setup



c. Select NXT in top left corner and press "F/W Download" button.



- d. Select .rfw file in default firmware directory in popup and press "Open".
  - i. For ROBOTC v2.03, the firmware file should be: "NXT798.rfw".

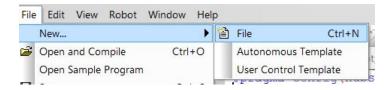


e. After a few moments, the Link Setup dialog box will display some verbose information in the Message Log and your NXT should beep several times and restart. When complete, you should see a series of messages similar to below in the Message Log:

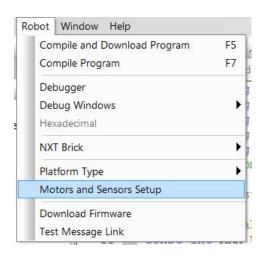
Message Log
Waiting for brick to enter 'boot' mode. 12 seconds elapsed
Downloading firmware file 'L:\Program Files (x86)\Robotics Academy\ROBOTC for Mindstorms\Firmware\NXT798.rfw'
Opening connection to NXT brick
Starting firmware download....
Firmware download completed. Waiting for NXT restart.
Opening connection to NXT brick
Waiting for NXT brick initialization to complete
Firmware download completed

## Starting a new file

- 1. Open a new file.
  - a. File, New..., File



- 2. Configure motors.
  - a. Robot, Motors and Sensors Setup

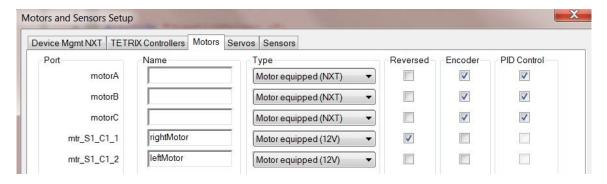


- b. Select the TETRIX Controllers tab.
- c. Select the Standard Configuration in the bullet list. This will default your motor controller to be the first device on sensor port 1 and your servo controller as the second device.



- d. Select the Motors tab.
- e. Give the drive motors more meaningful names.
  - i. Rename motorD to "rightMotor"

- ii. Rename motorE to "leftMotor"
- f. Select the Reversed checkbox next to the right motor (mtr\_S1\_C1\_1).



- g. Click "Apply" and then "OK".
- h. You should now see a set of pragma statements auto-generated at the top of your file. To edit these, use the Motors and Sensors Setup by accessing it the previously described method or by double clicking on the pragma statements.

```
#pragma config(Hubs, S1, HTMotor, HTServo, none, none)
#pragma config(Motor, mtr_S1_C1_1, rightMotor, tmotorNormal, openLoop, reversed)
#pragma config(Motor, mtr_S1_C1_2, leftMotor, tmotorNormal, openLoop)
#pragma config(Motor, mtr_S1_C1_1, rightMotor, tmotorNormal, openLoop, reversed)
#pragma config(Motor, mtr_S1_C1_2, leftMotor, tmotorNormal, openLoop)
#pragma config(Motor, mtr_S1_C1_2, leftMo
```

3. Create a main task.

Every ROBOTC program needs a main task. It is the code which is initially executed by the runtime and is the starting point for any program.

a. Below the pragmas, define a main task using the following syntax:

```
task main()
{
```

```
6  task main()
7  {
8  }
9  }
```

b. Add an infinite loop inside the main task to keep your program running until you want to manually stop it.

```
while(true)
{
}
```

- 4. Import JoystickDriver.c
  - a. Add the following import statement below the pragmas, but before your main task:

```
#include "JoystickDriver.c";
```

```
6 #include "JoystickDriver.c";
7
```

- 5. Compile your code and save.
  - a. To compile, select Robot, Compile Program, or simply press F7.
    - i. You should not receive any errors, but may see a compiler warning about the infinite loop which you can safely ignore at this time.
  - b. Save your file by selecting File, Save, or click the save icon, or simply press Ctrl + s.
    - i. Make sure to give your file a meaningful name.

#### **Drive Control**

For this tutorial, we will use a simple tank drive approach, also known as a skid-steer. We will use the two analog sticks on a single joystick to control the left and right drive motors respectively.

In ROBOTC, motors are typically controlled by setting the desired power output to the global motor[] array.

```
motor[motorName] = power;
```

where motorName is the name of the motor you want to set and power is an integer value between -100 and +100. A power setting of zero will stop the motor.

#### Bare Minimum

To do this, we will directly set the output power to the motors from -100 to +100 based on the input value from the joysticks.

- 1. Get current joystick values.
  - a. Each time the code loops through, we want to read the most recent and current state of the joystick. To do this, we need to add the following line of code inside the infinite loop inside the main task:

```
getJoystickSettings(joystick);
```

```
13 | getJoystickSettings(joystick);
15 | getJoystickSettings(joystick);
```

- 2. Get left analog stick values and pass them to the left drive motor.
  - a. Here we are setting the output to the left motor equal to the value of the left analog stick on the joystick. Add the following line of code below the getJoystickSettings() function call.

```
motor[leftMotor] = joystick.joy1 y1;
```

b. Repeat the same syntax for the right motor:
 motor[rightMotor] = joystick.joy1 y2;

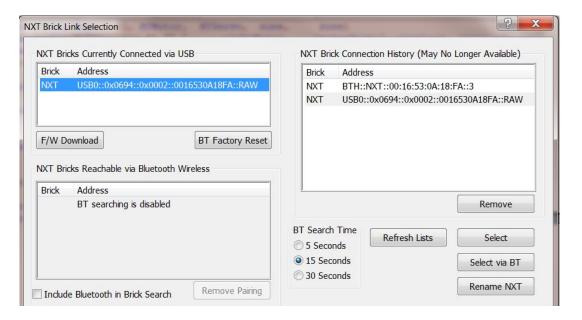
```
8
      task main()
9
10
11
        while (true)
12
13
          getJoystickSettings(joystick);
14
15
16
         motor[leftMotor] = joystick.joy1_y1;
17
          motor[rightMotor] = joystick.joy1 y2;
18
19
```

This is all you need to drive the robot around a little. At this time you should download and test driving your robot.

#### **Testing**

To test and run your code, we will have to first compile and download the code to the NXT and then use various debug windows to control and test the robot.

- 1. Establish connection with NXT (if not already connected).
  - a. Make sure NXT is turned on and is connected via USB or you have your Bluetooth dongle connected.
  - b. Open Link Setup dialog
    - i. Robot, NXT Brick, Link Setup
  - c. If using USB, select the NXT brick from the top left and click "Select" and then "Close".

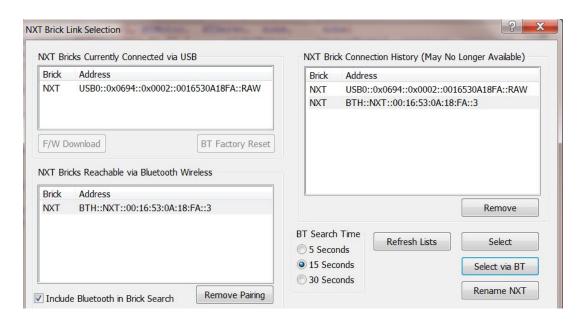


- d. If you want to use Bluetooth, you will need to either
  - i. Enable Bluetooth searching by checking the "Include Bluetooth in Brick Search" checkbox in the lower left of the window and click "Refresh Lists".
    - 1. You should now be able to select the NXT brick from the middle left Bluetooth list. Select the NXT brick, click "Select via BT", then click "Close".

Or:

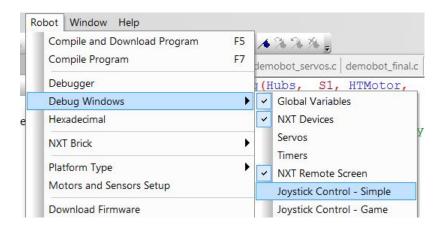
- ii. First connect via USB, select the NXT brick from the top left and click "Select via BT" instead of the normal "Select" button. Then click "Close".
  - \* Note that this is the preferred method of connecting via Bluetooth as the search function can take too long and becomes unreliable if there are many other NXT bricks around (such as at FTC events).

iii. If you are connecting via Bluetooth to a NXT for the first time, you may need to press the orange button on the NXT when prompted to accept the new device pairing.



\*\*Note that once you have connected to the NXT at least once using your preferred method (USB or Bluetooth), you should not need to manually establish the connection for subsequent downloads. Simply pressing the F5 shortcut will attempt to connect using the most recent method/settings.

- 2. Compile and Download
  - a. Robot, Compile and Download Program
  - b. It is easiest to use the F5 shortcut.
- 3. Open up debug windows
  - a. Open the Simple Joystick debug window
    - i. Robot, Debug Windows, Joystick Control Simple
    - ii. Note that you will want to use the Game Joystick debug window when programming for a FTC competition using the template. It has some additional controls for simulating match start/stop and autonomous/tele-op flags that are used by the FMS/FCS.



- b. Open the NXT Remote Screen debug window.
  - i. Robot, Debug Windows, NXT Remote Screen
- c. Open the NXT Devices debug window
  - i. Robot, Debug Windows, NXT Devices

#### 4. Start the program.

- a. If you have not already done so, make sure the power switch for the 12V battery has been turned on.
- b. The Program Debug window should have automatically opened when you compiled and downloaded.
- c. Click the "Start" button.

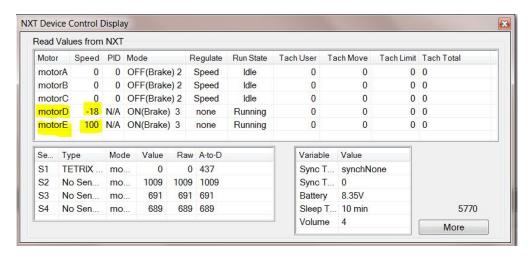


#### 5. Drive!

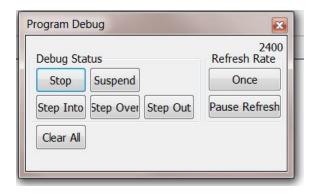
a. At this point you should be able to see anything you do to the joystick reflected in near real-time in the joystick debug window. The joystick debug window will show you the raw values for the analog sticks and buttons pressed which is very useful for debugging. These are the same values that you will be reading in the code.



b. When you move the analog sticks up and down, you should see the y values from them passed to the two dc motors. Inside the NXT Devices debug window, you should see the Speed for motors D and E (named rightMotor and leftMotor in our code) change in value with the movement of the analog sticks on the joystick.



- c. When you're done testing, click the "Stop" button on the Program Debug window which is where the "Start" button was previously. You can then hide all the debug windows by only closing the Program Debug window using the little red X at the top right-hand corner.
  - Note than the next time you press F5 to compile and download, ROBOTC will open all the debug windows you previously had open.



\*\* You can test your code at any point at which is successfully compiles. It is recommended to do this often as programming is a pragmatic and iterative process. The remaining sections of this guide will not explicitly indicate to test, but you should compile and download at the end of each section which adds functionality to your program.

#### Improving the Drive Code

Although the code we have written so far will drive the robot around, it has a few issues.

While the Tetrix motors accept an output power range of -100 to +100, the analog sticks on the joystick return values between -128 to +127. This means if the analog stick is in the range under -100 or over +100, the value passed to the motor is simple transformed to -100 or +100. This is less than ideal because it reduces the usable range of the analog stick by about 20%.

There is another issue related to deadzones within the physical joysticks themselves. Due to inaccuracies and tolerances within the manufacturing process, the analog sticks on most FTC joysticks will rarely return to a perfect zero when released. This has a tendency to cause a robot to slowing spin in a circle when no one is touching the analog sticks. A simple code solution is to account for a deadzone at low values for the analog sticks which can then be treated as zero.

Better approaches to scaling the joystick values to the motor output are described below.

#### Basic

Here we will apply a simple proportional scale to get to full range of the analog sticks. The basic formula we want to use is:

$$motorOutput = \left(\frac{joystickValue}{max\ joystickValue}\right) \times max\ motorValue$$

1. Below the import for JoystickDriver.c, but before your main task, create a new function to accept the joystick value and return a scaled value for the motor.

```
int scaleForMotor(int joyValue)
{
      }

int scaleForMotor(int joyValue)
{
      int scaleForMotor(int joyValue)
}
```

- 2. The new function will need to apply a basic proportional scale knowing the maximum ranges of the inputs and outputs.
  - a. Define constants for the max ranges inside the function block as such: const int MAX\_MOTOR\_VAL = 100;

```
const float MAX_JOY_VAL = 127.0;
```

- i. Note that the second constant was defined as a float instead of an integer. This is because we will use it as a divisor where we want a floating point result.
- b. Calculate the ratio of the value of the analog stick to its max range and multiply that by the max range of the motor output:

```
float ratio = joyValue / MAX_JOY_VAL;
int scaledVal = ratio * MAX_MOTOR_VAL;
```

c. Finally, return the scaled value.

```
return scaledVal;
```

```
8 💢 int scaleForMotor(int joyValue)
 9
10
        const int MAX MOTOR VAL = 100;
11
        const float MAX JOY VAL = 127.0;
13
14
       //calculate scaled value
15
        float ratio = joyValue / MAX JOY VAL;
        int scaledVal = ratio * MAX_MOTOR_VAL;
16
17
18
19
```

3. Now that we have a function to scale the joystick value to the desired ratio to pass to the motors, we need to change the code which is passing the joystick value directly to the motors to use the new function.

```
motor[leftMotor] = scaleForMotor(joystick.joy1_y1);
motor[rightMotor] = scaleForMotor(joystick.joy1 y2);
```

```
29     motor[leftMotor] = scaleForMotor(joystick.joy1_y1);
30     motor[rightMotor] = scaleForMotor(joystick.joy1_y2);
```

6. There is still the issue of the deadzone for the joystick. We can add a few lines of code to our new scaling function to take care of this.

f. First, define another constant for the deadzone range. Note that the actual value here is very specific to each joystick, but we will use 5 for this example.

```
const int DEADZONE = 5;
```

g. Now we need to check if the joystick value is less than the deadzone range and return zero if it is. This should be done before calculating the scaled value inside the method. We can use the built in absolute value function in ROBOTC to simply this:

```
if(abs(joyValue) < DEADZONE) {
    return 0;
}</pre>
```

```
8
     int scaleForMotor(int joyValue)
10
11
       const int DEADZONE = 5;
      const int MAX MOTOR VAL = 100;
12
      const float MAX JOY VAL = 127.0;
13
14
15
       //check for deadzone
16
      if(abs(joyValue) < DEADZONE) {</pre>
17
         return 0;
18
19
20
       //calculate scaled value
21
       float ratio = joyValue / MAX_JOY_VAL;
       int scaledVal = ratio * MAX MOTOR VAL;
23
24
       return scaledVal;
```

#### Better

Here we will apply a logarithmic scale to get fine grain control at lower speeds for a greater range of the analog sticks and quickly scale the power up at the end of the range. The basic formula for this will be:

$$motorOutput = \left(\frac{joystickValue^2}{\max joystickValue^2}\right) \times \max motorValue$$

1. We can modify our proportional scaling function by squaring the values used to calculate the ratio. The simplest way to accomplish this in ROBOTC is to just multiply them with themselves:

```
float ratio = ((joyValue * joyValue) / (MAX_JOY_VAL *
MAX_JOY_VAL));
```

2. There is a problem if we stopped here though. Because we are squaring the joystick value, we are losing sign if the value is negative. Since we still want to be able to drive in reverse, we need to calculate the direction and feed it back into the result.

a. An easy method of capturing the sign of the variable is to divide it by the absolute value of itself which should give you -1 or +1:

```
int direction = joyValue / abs(joyValue);
```

b. To feed the direction back into the result of the scaled calculation, simply multiply the result with the -1 or +1 we calculated above:

```
int scaledVal = (ratio * MAX_MOTOR_VAL) *
direction;
```

```
8
      int scaleForMotor(int joyValue)
9
10
11
        const int DEADZONE = 5;
        const int MAX MOTOR VAL = 100;
12
         const float MAX_JOY_VAL = 127.0;
13
14
        //check for deadzone
15
16
        if(abs(joyValue) < DEADZONE) {</pre>
17
          return 0;
18
19
20
        //calculate scaled value
         int direction = joyValue / abs(joyValue); // 1 or -1
21
        float ratio = ((joyValue * joyValue) / (MAX_JOY_VAL * MAX_JOY_VAL));
int scaledVal = (ratio * MAX_MOTOR_VAL) * direction;
23
24
25
        return scaledVal;
26
```

## **Manipulator Control**

The Tetrix Try-It demo robot has two servos, one which raises and lowers the arm and one which opens and closes the gripper. The HiTechnic servos included with the FTC kit of parts have just over 180° of movement and operate with position values between 0 and 255.

It is therefore important to know what position values we want to set to have the arm up or the gripper open. Although we could code a guess, test, adjust, and repeat, the ROBOTC IDE offers a much easier solution.

In ROBOTC, servos are controlled by setting the desired position to the setServoTarget() function.

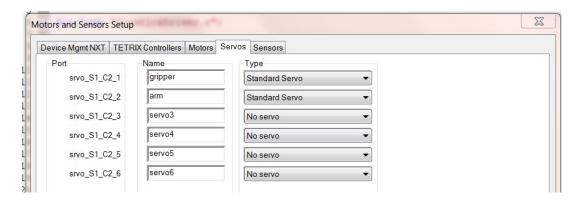
```
setServoTarget(servoName) = position;
```

where servoName is the name of the servo you want to set and postion is an integer value between 0 and +255.

## Using Servo Debug Window

We will use the Servo Debug dialog to find the ideal values for the positions for each servo. But first, we need to configure the servos in the Motor and Sensor Setup.

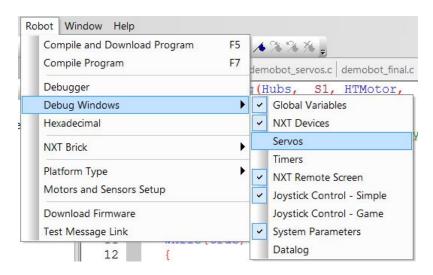
- 1. Configure the servos
  - a. Open the Motor and Sensor Setup
    - i. Robot, Motor and Sensor Setup or simply double-click the existing pragmas at the top of your program.
  - b. Click the Servo tab
  - c. Name the first servo (srvo\_S1\_C2\_1) to "gripper" and set the Type to "Standard Servo".
  - d. Name the second servo (srvo\_S1\_C2\_2) to "arm" and set the Type to "Standard Servo"



- e. Click "Apply" and then "OK".
- f. You should now see two additional pragma lines auto-generated for you.

```
#pragma config(Hubs, S1, HTMotor, HTServo, none, none)
#pragma config(Motor, mtr_S1_C1_1, rightMotor, tmotorNormal, openLoop, reversed)
#pragma config(Motor, mtr_S1_C1_2, leftMotor, tmotorNormal, openLoop)
#pragma config(Servo, srvo_S1_C2_1, gripper, tServoNormal)
#pragma config(Servo, srvo_S1_C2_2, arm, tServoNormal)
```

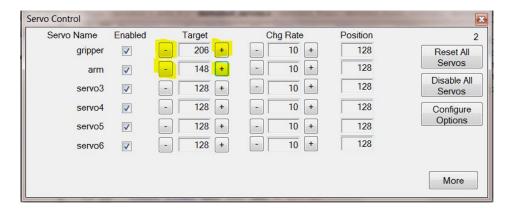
- 2. Run existing code to test.
  - a. Make sure NXT and robot 12V switch are turned on.
  - b. Use the F5 shortcut to compile and download
  - c. Click "Start" button on the Program Debug window.
- 3. Open Servos debug window
  - a. Robot, Debug Windows, Servos



- 4. Test servo positions.
  - a. By default, the servos will be disabled and set to 128 (center).



b. Click the "-" and "+" buttons next to the Target box for the gripper and arm servos to see what physical position the values actually correspond to.



- c. Raise and lower the values for the gripper servo until you find the desired values for open and closed positions.
  - i. Note that you do not want target values that force the servos beyond their movable range. For example, if the gripper servo shuts the gripper at 200, you may want a target of 210 to keep the gripper a little open rather than 190 which squeezes it extra tight and can wear down the servo's internal gears.
  - ii. Write down the open and close position values. We will use these in the code later on.
- d. Repeat the process for the arm servo to find the values for the arm being raised up and lowered down.
  - i. Write down the up and down position values as well.
- 5. Stop the program and close the debug windows via the Program Debug window.
- 6. Now that we know the desired servo targets, we can define these values as constants in our code.
  - a. Below the import for the JoystickDriver.c and before the scaleForMotor() function, define four constants, one for each servo target using the values written down from step 4 above.

```
const int ARM_UP_POSITION = 128;
const int ARM_DOWN_POSITION = 210;
const int GRIPPER_OPEN_POSITION = 255;
const int GRIPPER CLOSE POSITION = 200;
```

```
//global variables for servo targets
const int ARM_UP_POSITION = 128;
const int ARM_DOWN_POSITION = 210;
const int GRIPPER_OPEN_POSITION = 255;
const int GRIPPER_CLOSE_POSITION = 200;
```

#### Arm Control

Now that we have constants defined for our servo targets, we just need to code up our joystick buttons to pass them to the servo controller.

Before writing any control code, you should have a control scheme in mind. For this example, we will extend/lower the arm using button 4 on the joystick and raise it using

button 2. If you do not like this configuration, feel free to use whichever buttons you like instead.

- 1. Code arm down block.
  - a. Inside of the main task, inside of the infinite loop, and below the drive motor code, create an if condition which evaluates to true if button 4 is pressed.

```
if(joy1Btn(4))
{
```

b. Inside of the if block, set the target for the arm servo to the down position using the constant defined earlier.

```
servoTarget(arm) = ARM DOWN POSITION;
```

```
//arm control
if(joy1Btn(4)) //down

{
servoTarget(arm) = ARM_DOWN_POSITION;
}
```

- 2. Code the arm up block
  - a. Below the if condition for button 4, add an 'else if' block for button 2.

```
else if(joy1Btn(2))
{
```

b. Inside of the 'else if' block, set the target for the arm servo to the up position using the constant defined earlier.

```
servoTarget(arm) = ARM UP POSITION;
```

## **Gripper Control**

For the gripper control, we will use button 1 on the joystick to close the gripper and button 3 to open it.

- 3. Code gripper close block.
  - a. Inside of the main task, inside of the infinite loop, and below the arm control, create an if condition which evaluates to true if button 1 is pressed.

```
if(joy1Btn(1))
{
}
```

b. Inside of the if block, set the target for the gripper servo to the closed position using the constant defined earlier.

```
servoTarget(gripper) = GRIPPER CLOSE POSITION;
```

```
//gripper control
if(joylBtn(1)) //close
{
    servoTarget(gripper) = GRIPPER_CLOSE_POSITION;
}
```

4. Code the gripper open block

```
a. Below the if condition for button 1, add an 'else if' block for button 3.
  else if(joy1Btn(3))
{
}
```

b. Inside of the 'else if' block, set the target for the gripper servo to the open position using the constant defined earlier.

```
servoTarget(gripper) = GRIPPER OPEN POSITION;
```

# **Enhancing the Servo Control Code**

At this point, you should be able to control your arm and gripper using the four primary buttons on the joystick. The main task should look something like this:

```
36 task main()
37
38
39
        while(true)
40
41
42
          getJoystickSettings(joystick);
43
44
         motor[leftMotor] = scaleForMotor(joystick.joy1 y1);
         motor[rightMotor] = scaleForMotor(joystick.joy1 y2);
45
46
47
          //arm control
          if(joy1Btn(4)) //down
48
49
50
            servoTarget(arm) = ARM_DOWN_POSITION;
51
52
          else if(joy1Btn(2)) //up
53
54
            servoTarget(arm) = ARM UP POSITION;
55
56
57
          //gripper control
58
          if(joy1Btn(1)) //close
59
            servoTarget(gripper) = GRIPPER_CLOSE_POSITION;
60
61
62
          else if(joy1Btn(3)) //open
63
64
            servoTarget(gripper) = GRIPPER OPEN POSITION;
65
66
67
68
```

The main task is starting to get filled up with low level control details. Let's move the servo control code to a new function.

1. Move servo code to a new servo control function

a. Create a new function named "armAndGripperControl" below the scaleForMotor() function and the main task .

```
void armAndGripperControl()
{
...
```

b. Cut and paste servo code from main task to the new function.

```
37
       \ensuremath{^{\star}} Function to manually operate servos for arm and gripper.
38
39
      void armAndGripperControl()
40
41
          //arm control
42
          if(joy1Btn(4)) //down
43
            servoTarget(arm) = ARM DOWN POSITION;
44
45
46
          else if(joy1Btn(2)) //up
47
48
            servoTarget(arm) = ARM UP POSITION;
49
50
51
          //gripper control
52
          if(joy1Btn(1)) //close
53
54
            servoTarget(gripper) = GRIPPER CLOSE POSITION;
55
56
          else if(joy1Btn(3)) //open
57
58
            servoTarget(gripper) = GRIPPER OPEN POSITION;
59
60
```

c. Add a call to the new function from the main task where we cut the control code from:

```
armAndGripperControl();
```

```
task main()
63
64
65
        while (true)
66
67
68
          getJoystickSettings(joystick);
69
70
          motor[leftMotor] = scaleForMotor(joystick.joy1 y1);
          motor[rightMotor] = scaleForMotor(joystick.joy1 y2);
71
72
73
         armAndGripperControl();
74
75
76
77
```

Now that we have isolated the servo control code, we can add to it without crowding the main task up with more low level code. Here we want to allow the driver to operate the arm and gripper with a more fine-grained incremental control using the shoulder buttons on the joystick.

- 2. Add code to increment arm servo down
  - a. Add 'else if' condition below 'else if' condition for button 2 to check for button 8.

```
else if(joy1Btn(8)
{
}
```

b. Add max constraint to condition prevent the arm servo from over extending the arm down.

```
else if(joy1Btn(8) && ServoValue(arm) <
ARM DOWN POSITION)</pre>
```

c. Add code to increment the servo target inside of condition block. servoTarget(arm) = ServoValue(arm) + 1;

```
50     else if(joy1Btn(8) && ServoValue(arm) < ARM_DOWN_POSITION) //manual down
51     {
52         servoTarget(arm) = ServoValue(arm) + 1;
53     }</pre>
```

- 3. Add code to decrement arm servo up
  - a. Add 'else if' condition below 'else if' condition for button 8 to check for button 6.

```
else if(joy1Btn(6)
{
```

b. Add min constraint to condition to prevent the arm servo from over extending the arm up.

```
else if(joy1Btn(6) && ServoValue(arm) >
ARM UP POSITION)
```

c. Add code to decrement the servo target inside of condition block.
 servoTarget (arm) = ServoValue (arm) - 1;

- 4. Add code to decrement gripper servo shut
  - a. Add 'else if' condition below 'else if' condition for button 3 to check for button 5.

```
else if(joy1Btn(5)
{
```

 Add min constraint to condition to prevent the gripper servo from over extending the gripper closed.

```
else if(joy1Btn(5) && ServoValue(gripper) >
GRIPPER CLOSE POSITION)
```

c. Add code to decrement the servo target inside of condition block.
 servoTarget (gripper) = ServoValue (gripper) - 1;

```
68     else if(joy1Btn(5) && ServoValue(gripper) > GRIPPER_CLOSE_POSITION) //manual close
69     {
        servoTarget(gripper) = ServoValue(gripper) - 1;
71     }
```

5. Add code to increment gripper servo open

a. Add 'else if' condition below 'else if' condition for button 5 to check for button 7.

```
else if(joy1Btn(7)
{
```

b. Add max constraint to condition to prevent the gripper servo from over extending the gripper open.

```
else if(joy1Btn(7) && ServoValue(gripper) <
GRIPPER OPEN POSITION)</pre>
```

c. Add code to increment the servo target inside of condition block. servoTarget (gripper) = ServoValue (gripper) + 1;

```
72    else if(joy1Btn(7) && ServoValue(gripper) < GRIPPER_OPEN_POSITION) //manual open
73    {
74        servoTarget(gripper) = ServoValue(gripper) + 1;
75    }</pre>
```

The arms and gripper should not be able to be operated using the quick targets with buttons 1-4 or manually/incrementally using the shoulder buttons 5-8. The completed servo control function should look something like this:

```
39
      void armAndGripperControl()
40
41
        //arm control
42
        if(joy1Btn(4)) //down
43
          servoTarget(arm) = ARM DOWN POSITION;
44
45
46
        else if(joy1Btn(2)) //up
47
48
          servoTarget(arm) = ARM UP POSITION;
49
50
        else if(joy1Btn(8) && ServoValue(arm) < ARM DOWN POSITION) //manual down
51
52
          servoTarget(arm) = ServoValue(arm) + 1;
53
54
        else if(joy1Btn(6) && ServoValue(arm) > ARM UP POSITION) //manual up
55
56
          servoTarget(arm) = ServoValue(arm) - 1;
57
58
59
        //gripper control
60
        if(joy1Btn(1)) //close
61
          servoTarget(gripper) = GRIPPER CLOSE POSITION;
63
64
        else if(joy1Btn(3)) //open
65
          servoTarget(gripper) = GRIPPER OPEN POSITION;
66
67
        else if(joy1Btn(5) && ServoValue(gripper) > GRIPPER CLOSE POSITION) //manual close
68
69
70
          servoTarget(gripper) = ServoValue(gripper) - 1;
71
72
        else if(joy1Btn(7) && ServoValue(gripper) < GRIPPER OPEN POSITION) //manual open
73
74
          servoTarget(gripper) = ServoValue(gripper) + 1;
75
76
77
```

There is one last enhancement to be concerned with for this example. When the program first starts, there is currently no default position being set properly for the servos. To make sure the robot starts with the arm up and the gripper closed, we want to add some initialization code.

- 6. Initialize servos to default positions.
  - a. Inside of the main task, before the infinite loop, set the arm to the up position:

```
servoTarget(arm) = ARM UP POSITION;
```

b. Below the call to default the arm up, set the gripper to the closed position servoTarget(gripper) = GRIPPER CLOSE POSITION;

```
79
       task main()
80
81
82
         //set default servo positions
83
        servoTarget(arm) = ARM_UP_POSITION;
        servoTarget(gripper) = GRIPPER CLOSE POSITION;
84
85
86
         while (true)
87
88
89
            getJoystickSettings(joystick);
90
           motor[leftMotor] = scaleForMotor(joystick.joy1_y1);
motor[rightMotor] = scaleForMotor(joystick.joy1_y2);
91
92
            armAndGripperControl();
94
95
96
97
98
```

\*Note that there is an initialization function pre-defined in the official FTC competition template. When using the template, you should place any initialization code within that function.

## Some Tips & Advice

#### Compile, compile, compile

The ROBOTC IDE will not compile in the background, you need to press F7 to compile and update the code structure in the IDE memory. It only takes one key press to compile, so feel free to wail on F7 as you code. The worst that can happen is you find syntax errors and other compile time bugs sooner than later.

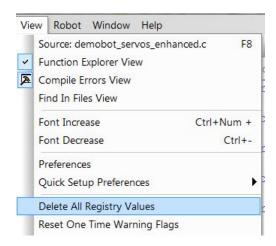
#### Be wary of Bluetooth

A lot of FTC coaches and students have lost a bit of sleep, patience, and even sanity troubleshooting why Bluetooth suddenly stopped working for them. This is a known issue beyond ROBOTC and includes the underlying phantom.dll, Windows, and the protocol itself. I could easily dedicate an entire guide to troubleshooting Bluetooth connectivity issues, but with FIRST's adoption of a new WiFi based wireless communication system for the upcoming 2010-2011 season, I am hopeful that those dark days are behind us.

#### If ROBOTC gets too messed up, clear the registry values

On occasion, a team has lost the menu bar or some other key component of ROBOTC. In these cases, the only proven fix has been to clear out all registry entries for ROBOTC. !Do not use regedit unless you are experienced and already comfortable with its use and risks!

There is a built in option for this in the IDE if you need to reset things. You can find it under View, Delete All Registry Values.



<sup>\*</sup>Note that this menu option won't save you if you've lost the menu. In that case, find someone familiar with regedit to help you out.

## Use good programming processes

Applying some simple best practices upfront will make coding in ROBOTC much easier and a lot less work. Try to apply some simple things such as:

- Develop your code iteratively. Divide and conquer complex problems by attacking them individually and coding solutions through testing in a constant cycle. Don't try to program an entire large autonomous routine in one go; get the first core part to work, then the second, and so forth. Constantly compiling and testing is part of this.
- Use some style. A much wiser programmer than I once stated that any fool can write code a computer can understand, but a good programmer writes code other programmers can understand. This is a fundamental truth I've come to realize and it will serve you well if you ever want to get help with your code or pass the programming torch to another student. Use good variable names that instantly indicate what they are for. Keep functions short and readable. Use comments. Use whitespace and indentations to keep your code clean and readable. There are too many best practices for good coding style to list here, but I highly recommend looking some up and applying them.
- Save regularly and version your files. Far too often a student will lose all the code they've been working on for a season in the blink of an eye because they didn't properly back up their code. It is generally not enough to simply back up a copy of your code; you generally want to version it so that you can go back and reference why you changed something or undo/redo something. Also remember that compiling and downloading doesn't save, so you must do this consciously to avoid losing changes if the ROBOTC IDE crashes when programming and testing. Lastly, give your files meaningful names.

## Keep learning

ROBOTC is not exactly ANSI-C, but it is close enough for you to apply a lot of what you learn about general C programming to ROBOTC. ROBOTC does not support pointers or recursion. It does however support more rare/useful constructs such as boolean primitives and union typedefs. There is lot of power in ROBOTC that requires just a little effort on the programmers part to learn and understand some basic fundamentals. Reading a good introductory book on C programming will take you far. Reading a good introductory book on computer science basics (data structures, algorithms, etc.) will take you even further.

I also recommend reading through or at least familiarizing with as much of the reference materials linked below.

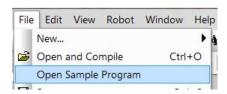
### References/Resources

The following is a list of resources for learning more or helping you troubleshoot issues.

#### IDE help files & Sample Programs

The help files contain a lot of good information and usage examples. The included sample files do a good job illustrating the use of various functions and features in ROBOTC.

You can access them through the IDE by going to File, Open Sample Program.



#### ROBOTC.net

The main website for ROBOTC is one of the best resources for getting help or interacting with other members of the ROBOTC community. You can email their technical support if you have a technical issue with the software/IDE itself or if you want to report a bug. But often more useful are the forums. There is a sub-forum dedicated to FTC and most posts there are responded to the same day. I personally make an effort to visit the forum daily during the build season as do a number of other mentors. It may be the best place to get help and feedback from a largest number of smart folks familiar with ROBOTC.

Main website: <a href="http://www.robotc.net/">http://www.robotc.net/</a>

Forums: <a href="http://www.robotc.net/forums/">http://www.robotc.net/forums/</a>
Blog: <a href="http://www.robotc.net/blog/">http://www.robotc.net/blog/</a>

## Xander's 3<sup>rd</sup> Party Drivers

Xander Soldaat is a regular contributor to the NXT community who maintains a set of excellent third party drivers for NXT compatible sensors. These drivers offer enhanced functionality for using just about any sensor with ROBOTC and I highly recommend them; particularly so if you are using the IR Seeker or Sensor Multiplexer.

3<sup>rd</sup> Party Drivers: <a href="http://rdpartyrobotcdr.sourceforge.net/">http://rdpartyrobotcdr.sourceforge.net/</a>

Blog: <a href="http://mightor.wordpress.com/">http://mightor.wordpress.com/</a>

## **Robotics Academy Curriculum**

Robotics Academy, the non-profit affiliated with Carnegie Melon University who created ROBOTC, offer a number of training resources. Note that some of these are not free.

Main Website: http://www.robotc.net/education/curriculum/nxt/

**Preview Website:** 

http://www.education.rec.ri.cmu.edu/previews/robot c products/teaching rc tetri

x preview/index.html

Classes: <a href="http://www.robotc.net/education/training/nxt/">http://www.robotc.net/education/training/nxt/</a>
Webinars: <a href="http://www.robotc.net/education/webinars/">http://www.robotc.net/education/webinars/</a>

Older videos & tutorials:

http://www.education.rec.ri.cmu.edu/content/events/ftc/robotc/index.htm

#### **Chief Delphi**

For years, the forums on Chief Delphi have been the most active and engaging source for unofficial FIRST related information. It is an extremely dynamic online community of FIRST students, mentors, and volunteers. As such, it is another great resource for help with not only ROBOTC, but FTC and FIRST in general.

Main website: <a href="http://www.chiefdelphi.com/forums/index.php">http://www.chiefdelphi.com/forums/index.php</a>

FTC sub-forum: http://www.chiefdelphi.com/forums/forumdisplay.php?f=146

#### Course Notes – Albert Schueller, Whitman College

One of the best ROBOTC tutorials can be found in the class notes from Albert Schuellers's course on programming with robots at Whitman College. This is a good read through which covers many of the basics/fundamentals that this guide does not.

PDF of tutorial: <a href="http://carrot.whitman.edu/Robots/notes.pdf">http://carrot.whitman.edu/Robots/notes.pdf</a>

## **Local Planning Committees and Other FTC Coaches**

It is important to know that you're not alone when you mentor a FIRST team. Network with your fellow coaches and regional volunteers. Everyone involved generally volunteers their person time and effort with the mission of FIRST in mind; to inspire students to take up science and technology. Helping each other makes that goal that much easier and I hope you'll find as much support as I did when I was a rookie mentor.

# **Appendix / Full Code Examples**

Below are complete code examples for each of the sections detailed above.

## **Getting Started**

```
#pragma config(Hubs, S1, HTMotor, HTServo, none, none)
#pragma config(Motor, mtr_S1_C1_1, rightMotor, tmotorNormal, openLoop, reversed)
#pragma config(Motor, mtr_S1_C1_2, leftMotor, tmotorNormal, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard !!*//
#include "JoystickDriver.c";

/**
    * Main task.
    */
task main()
{
    while(true)
    {
        }
    }
}
```

#### **Drive Control**

```
#pragma config(Hubs, S1, HTMotor, HTServo, none, none)
#pragma config(Motor, mtr_S1_C1_1, rightMotor, tmotorNormal, openLoop, reversed)
#pragma config(Motor, mtr_S1_C1_2, leftMotor, tmotorNormal, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard !!*//
#include "JoystickDriver.c";
```

```
/**
 * Helper function to scale analog joystick values for tetrix dc motors.
int scaleForMotor(int joyValue)
  const int DEADZONE = 5;
      const int MAX MOTOR VAL = 100;
      const float MAX JOY VAL = 127.0;
      //check for deadzone
      if(abs(joyValue) < DEADZONE) {</pre>
            return 0;
      //calculate scaled value
      int direction = joyValue / abs(joyValue); // 1 or -1
      float ratio = ((joyValue * joyValue) / (MAX JOY VAL * MAX JOY VAL));
      int scaledVal = (ratio * MAX MOTOR VAL) * direction;
      return scaledVal;
/**
* Main task.
* /
task main()
  while(true)
   getJoystickSettings(joystick);
   motor[leftMotor] = scaleForMotor(joystick.joy1 y1);
   motor[rightMotor] = scaleForMotor(joystick.joy1 y2);
```

}

## **Manipulator Control**

```
#pragma config(Hubs, S1, HTMotor, HTServo, none,
                                      rightMotor,
#pragma config(Motor, mtr S1 C1 1,
                                                      tmotorNormal, openLoop, reversed)
#pragma config(Motor, mtr S1 C1 2, leftMotor,
                                                       tmotorNormal, openLoop)
#pragma config(Servo, srvo S1 C2 1, gripper,
                                                             tServoNormal)
#pragma config(Servo, srvo S1 C2 2,
                                                             tServoNormal)
                                       arm,
//*!!Code automatically generated by 'ROBOTC' configuration wizard
                                                                                !!*//
#include "JoystickDriver.c";
//global variables for servo targets
const int ARM UP POSITION = 128;
const int ARM DOWN POSITION = 210;
const int GRIPPER OPEN POSITION = 255;
const int GRIPPER CLOSE POSITION = 200;
/**
 * Helper function to scale analog joystick values for tetrix dc motors.
int scaleForMotor(int joyValue)
      const int DEADZONE = 5;
      const int MAX MOTOR VAL = 100;
      const float MAX JOY VAL = 127.0;
      //check for deadzone
      if(abs(joyValue) < DEADZONE) {</pre>
           return 0;
     //calculate scaled value
      int direction = joyValue / abs(joyValue); // 1 or -1
```

```
float ratio = ((joyValue * joyValue) / (MAX JOY VAL * MAX JOY VAL));
     int scaledVal = (ratio * MAX MOTOR VAL) * direction;
     return scaledVal;
 * Function to manually operate servos for arm and gripper.
void armAndGripperControl()
     //arm control
      if(joy1Btn(4)) //down
            servoTarget(arm) = ARM DOWN POSITION;
      else if(joy1Btn(2)) //up
            servoTarget(arm) = ARM UP POSITION;
     else if(joy1Btn(8) && ServoValue(arm) < ARM DOWN POSITION) //manual down
            servoTarget(arm) = ServoValue(arm) + 1;
      else if(joy1Btn(6) && ServoValue(arm) > ARM UP POSITION) //manual up
            servoTarget(arm) = ServoValue(arm) - 1;
      //gripper control
      if(joy1Btn(1)) //close
            servoTarget(gripper) = GRIPPER CLOSE POSITION;
      else if(joy1Btn(3)) //open
            servoTarget(gripper) = GRIPPER OPEN POSITION;
```

```
else if(joy1Btn(5) && ServoValue(gripper) > GRIPPER CLOSE POSITION)
                                                                               //manual close
           servoTarget(gripper) = ServoValue(gripper) - 1;
     else if(joy1Btn(7) && ServoValue(gripper) < GRIPPER OPEN POSITION) //manual open
           servoTarget(gripper) = ServoValue(gripper) + 1;
/**
* Main task.
task main()
 //set default servo positions
 servoTarget(arm) = ARM UP POSITION;
 servoTarget(gripper) = GRIPPER CLOSE POSITION;
 while(true)
   getJoystickSettings(joystick);
   motor[leftMotor] = scaleForMotor(joystick.joy1 y1);
   motor[rightMotor] = scaleForMotor(joystick.joy1 y2);
   armAndGripperControl();
```