Programming Guide (ROBOTC®)

Introduction

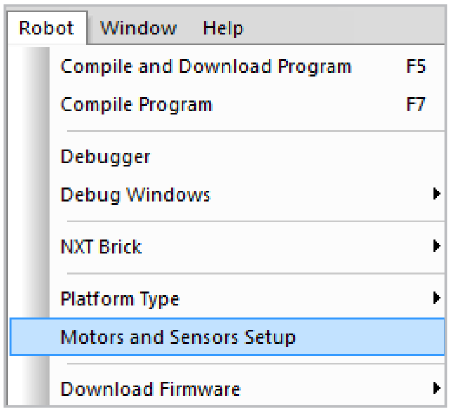
In this guide, a TETRIX® with LEGO® MINDSTORMS® robot with an arm and gripper extension will be programmed to be controlled by a joystick using the Remote Control Editor. The joystick will be configured so that it can make the robot move and pick up objects. Then, the remote control configuration will be generated into code and modified. This guide is for use with the ROBOTC® programming language.

Review:

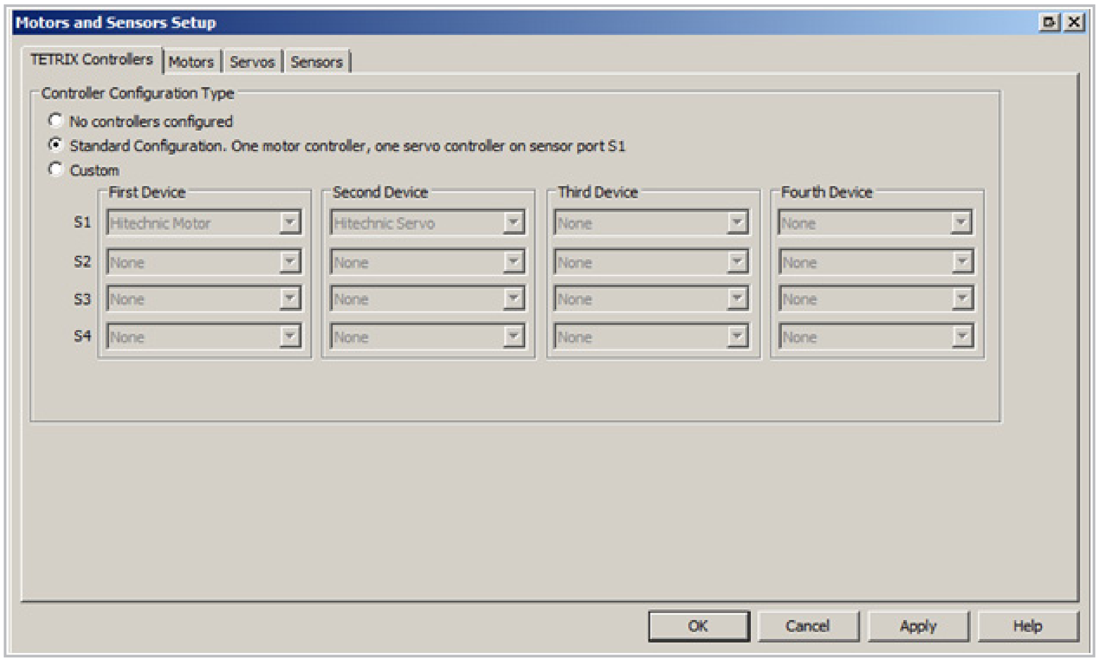
- To review the basics of FTC® programming in ROBOTC, review the Programming Guides and Tutorial Videos from the FTC Programming extension.
- These materials contain detailed information about how to use the FCS to run a FTC game and FTC templates to create FCS compatible programs.

Configure the Motor and TETRIX Controllers:

1. Navigate to the **Robot** menu and select **Motors and Sensors Setup**.



2. Select the **TETRIX Controllers** tab and then select the **Standard Configuration**. **One motor controller, one servo controller on sensor port S1** option.

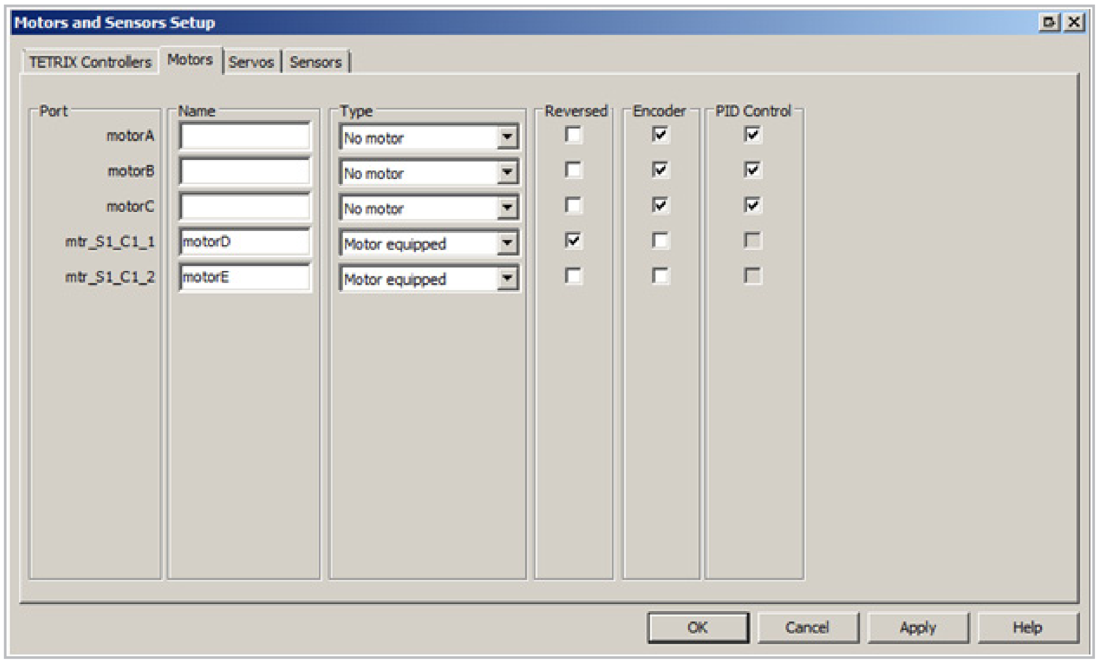


Controller Configuration Type
No controllers configured
[✓ Standard Configuration. One motor controller, one servo controller on sensor port S1]
Custom

	First Device	Second Device	Third Device	Fourth Device
S1	Hitechnic Motor	Hitechnic Servo	None	None
S2	None	None	None	None
S3	None	None	None	None
S4	None	None	None	None

Programming Guide (ROBOTC®)

3. Click the **Motors** tab. Under Type for motorA, motorB, and motorC, select **No Motor** from the pull-down menu. Check the **Reversed** option for MotorD.

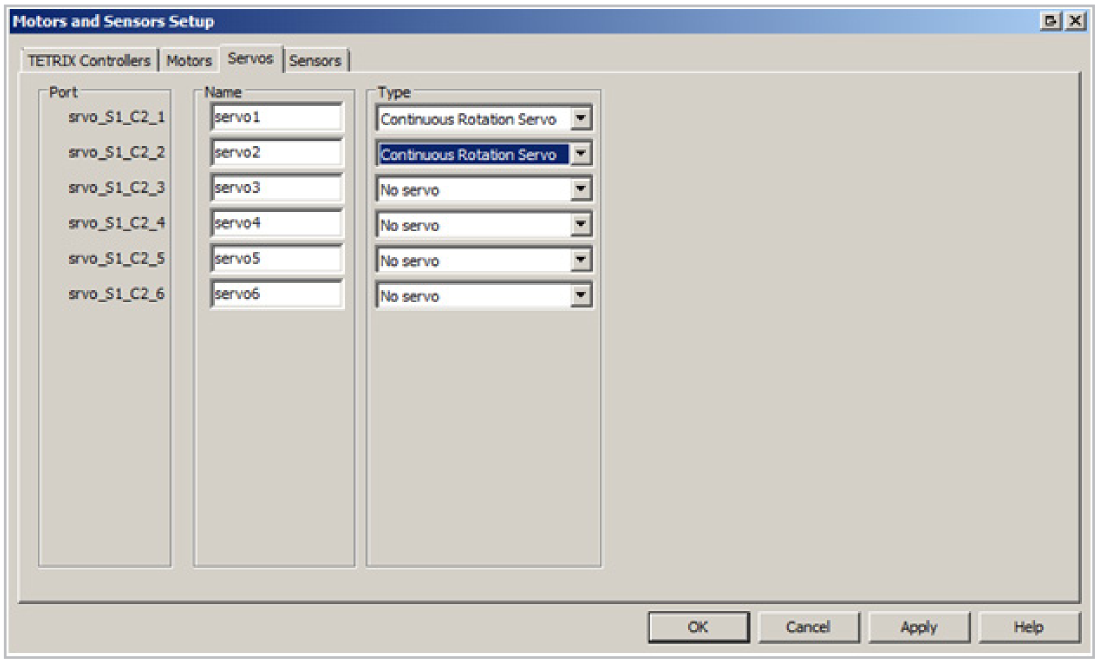


Motors and Sensors Setup

TETRIX Controllers | Motors | Servos | Sensors

Port	Name	Type	Reversed	Encoder	PID Control
motorA		No motor	☐	✓	✓
motorB		No motor	☐	✓	✓
motorC		No motor	☐	✓	✓
mtr_S1_C1_1	motorD	Motor equipped	✓	☐	☐
mtr_S1_C1_2	motorE	Motor equipped	☐	☐	☐

4. Since this is an arm and gripper bot, the servos will also have to be configured. By clicking the down arrow in the Type column, select **Continuous Rotation Servo** for both Servo 1 and Servo 2.



Motors and Sensors Setup

TETRIX Controllers | Motors | Servos | Sensors

Port	Name	Type
srvo_S1_C2_1	servo1	Continuous Rotation Servo
srvo_S1_C2_2	servo2	Continuous Rotation Servo
srvo_S1_C2_3	servo3	No servo
srvo_S1_C2_4	servo4	No servo
srvo_S1_C2_5	servo5	No servo
srvo_S1_C2_6	servo6	No servo

Programming Guide (ROBOTC®)

Getting Started:

1. To start the program, open the TeleOp template from the *FIRST*® Tech Challenge folder. Save the file with a new name in an appropriate directory.
2. Navigate to the **initializeRobot()** function and place the commands that will move the arm and gripper to the initial position, where the Arm will be raised and the Gripper will be open. Set the arm servo, servo2, to an initial position of **127** and the gripper servo, servo1, to an initial position of **255**.

```
void initializeRobot()
{
  // Place code here to initialize servos to starting positions.
  // Sensors are automatically configured and setup by ROBOTC®. They may
  // need a brief time to stabilize.

  servo[servo1] = 255;      // Gripper opened
  servo[servo2] = 127;      // Arm raised

  return;
}
```

Declare the Variables:

3. Scroll down to the **main()** function. Inside this function, below the call to the **initializeRobot()** function, declare two integer variables, **mThreshold** and **aThreshold**. These will function as threshold values for the movement of the analog sticks so that any small, unintentional nudges will not cause the robot or any of the servos to move. Set **mThreshold**, the threshold for the drive motors, to a value of **15** and **aThreshold**, the threshold for the arm servo motor, to a value of **30**.

```
int mThreshold = 15;        // To avoid unnecessary movement
int aThreshold = 30;
```

Note: Two different thresholds have been declared because one of the analog sticks will control the motors, while the other will control the servo attached to the arm of the robot.

4. Declare two more integer variables, **xVal** and **yVal**. These variables will store the x and y values of the left analog stick, which controls the drive motors. It is not necessary to set initial values for these, as they will be set later in the program before they are used.

```
int xVal, yVal;             //stores left analog stick values
```

Note: Variables of the same type can be declared in one line of code, separated by commas. However, code will looks neater and is easier to debug later if it has been organized effectively. Any variables being initialized when they are declared should be put on a separate line. The two variables in Step 4 should be grouped together on one line because they are not being initialized at this point and because they are related to one another in how they will be used.

Programming Guide (ROBOTC®)

5. Declare two variables with a type of **float**, **scaleFactor** and **aPosition**. A float, or floating point number, stores seven decimal digits in total. Set the **scaleFactor** variable to a value of **40.0/127**, which will store the number 0.3149606 into scaleFactor. The scale factor will be used to set a maximum power for the drive motors, as well as map the range of the analog stick to the maximum power that has been set. In this case, the maximum power set is 40.0. Set the **aPosition** variable to **ServoValue[servo2]**. This will retrieve the arm servo's position value and store it in the aPosition variable.

```
float scaleFactor = 40.0 / 127; // Sets max. average motor power and maps range
                                // of analog stick to this power range
float aPosition = ServoValue[servo2];    // Stores the current position of the arm servo
```

6. Once the variables have been defined, scroll down to the infinite loop found below the **waitForStart()** function call. The template contains a comment that says "**Add your robot specific tele-op code here.**" The rest of the code should be placed here. The first command to be placed here will be the **getJoystickSettings(joystick)** command, which will retrieve the data from the joystick every single time that the loop is run. Action will be taken according to the input received from the joystick through this command.

```
while (true)
{
      getJoystickSettings(joystick); // Retrieves data from the joystick
```

7. Set **xVal** to the x position value and **yVal** to the y position value retrieved from the left analog stick.

```
xVal = joystick.joy1__x1;
yVal = joystick.joy1__y1;
```

8. Use conditional statements, also referred to as if statements, to check if the values of the left analog stick are within the dead zone defined by the **mThreshold** variable. If either is within this dead zone, set the corresponding **xVal** or **yVal** variable to 0, so that the motor power will not be affected.

```
if (abs(xVal) < mThreshold)
   {
     xVal = 0;
   }
if (abs(yVal) < mThreshold)
   {
     yVal = 0;
   }
```

Note: Because the analog stick reports values from -128 to 127, the absolute value of the position is taken to make the comparison easier. Otherwise, large negative values would be filtered out in the dead zone as well.

9. Set the drive motor power using the **yVal**, **xVal** and **scaleFactor** variables.

```
motor[motorD] = ((yVal) + (xVal)) * scaleFactor;
motor[motorE] = ((yVal) - (xVal)) * scaleFactor;
```

Programming Guide (ROBOTC®)

10. The right analog stick will control the movement of the servo connected to the arm and will use a set of **if** and **else if** statements. The value of the analog stick needs to be compared to the value of **aThreshold** to determine if the analog stick is within the dead zone. If the value of the analog stick is positive and within the dead zone, **aPosition** should be decreased by a value of 0.15, to a minimum value of 0. If the value of the analog stick is negative and within the dead zone, **aPosition** should be increased by a value of 0.15, to a maximum value of 255.

```
if((joystick.joy1__y2 > aThreshold) && (aPosition > 0))
{
     aPosition -= 0.15;
}
else if((joystick.joy1__y2 < (aThreshold * -1)) && (aPosition < 255))
{
     aPosition += 0.15;
}
```

11. Set the arm servo's position as defined by the **aPosition** variable.

```
servo[servo2] = aPosition;     //Moves the arm servo
```

12. Use another set of **if** and **else if** statements to respond to the pressing of Buttons 5 and 6. The gripper should open if Button 5 is pressed, and close if Button 6 is pressed.

```
if(joy1Btn(5))          // Open Gripper
{
     servo[servo1] = 255;
}
else if(joy1Btn(6))      // Close Gripper
{
     servo[servo1] = 0;
}
```

Programming Guide (ROBOTC®)

Completed Code:

```
#pragma config(Hubs,  S1, HTMotor,  HTServo,  none,     none)
#pragma config(Motor,  mtr_S1_C1_1,     motorD,      tmotorNormal, openLoop, reversed)
#pragma config(Motor,  mtr_S1_C1_2,     motorE,      tmotorNormal, openLoop)
#pragma config(Servo,  srvo_S1_C2_1,    ,                       tServoStandard)
#pragma config(Servo,  srvo_S1_C2_2,    ,                       tServoStandard)
//*!!Code automatically generated by 'ROBOTC' configuration wizard                !!*//

/////////////////////////////////////////////////////////////////////////////////////////////////////
//
//                              Tele-Operation Mode Code Template
//
// This file contains a template for simplified creation of an tele-op program for an FTC® competition.
//
// You need to customize two functions with code unique to your specific robot.
//
/////////////////////////////////////////////////////////////////////////////////////////////////////

#include "JoystickDriver.c"  //Include file to "handle" the Bluetooth messages.


/////////////////////////////////////////////////////////////////////////////////////////////////////
//
//                                    initializeRobot
//
// Prior to the start of TeleOp mode, you may want to perform some initialization on your robot and the
// variables within your program.
//
// In most cases, you may not have to add any code to this function and it will remain "empty".
//
/////////////////////////////////////////////////////////////////////////////////////////////////////

void initializeRobot()
{
   servo[servo1] = 255;  // Gripper opened
   servo[servo2] = 127;  // Arm raised

   return;
}
```

Programming Guide (ROBOTC®)

Completed Code (continued):

```
////////////////////////////////////////////////////////////////////////////////////////////////////////
//
//                                      Main Task
//
// The following is the main code for the tele-op robot operation. Customize as appropriate for
// your specific robot.
//
// Game controller / joystick information is sent periodically (about every 50 milliseconds) from
// the FMS (Field Management System) to the robot. Most TeleOp programs will follow the following
// logic:
//   1. Loop forever repeating the following actions:
//   2. Get the latest game controller / joystick settings that have been received from the PC.
//   3. Perform appropriate actions based on the joystick + buttons settings. This is usually a
//      simple action:
//      *  Joystick values are usually directly translated into power levels for a motor or
//         position of a servo.
//      *  Buttons are usually used to start/stop a motor or cause a servo to move to a specific
//         position.
//   4. Repeat the loop.
//
// Your program needs to continuously loop because you need to continuously respond to changes in
// the game controller settings.
//
// At the end of the TeleOp period, the FMS will automatically abort (stop) execution of the program.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////

task main()
{
   initializeRobot();

   int mThreshold = 15;        // Sets dead zones to avoid unnecessary movement
   int aThreshold = 30;
   int xVal, yVal;             // Stores left analog stick values
   float scaleFactor = 40.0 / 127;              // Sets max. power and maps range of analog stick to max. power
   float aPosition = ServoValue[servo2];        // Stores the current position of the arm servo

   waitForStart();             // Wait for start of TeleOp phase
```

Programming Guide (ROBOTC®)

Completed Code (continued):

```
    while (true)
    {
     getJoystickSettings(joystick);        // Retrieves data from the joystick

      xVal = joystick.joy1__x1;
      yVal = joystick.joy1__y1;

     // Checks if analog stick values are within the dead zone

      if (abs(xVal) < mThreshold)
      {
      xVal = 0;
      }
     if (abs(yVal) < mThreshold)
      {
        yVal = 0;
      }

      motor[motorD] = ((yVal) + (xVal)) * scaleFactor;        // Sets left motor power
      motor[motorE] = ((yVal) - (xVal)) * scaleFactor;        // Sets right motor power

      //Controls arm servo position

      if((joystick.joy1__y2 > aThreshold) && (aPosition > 0))
      {
        aPosition -= 0.15;
      }
      else if((joystick.joy1__y2 < (aThreshold * -1)) && (aPosition < 255))
      {
        aPosition += 0.15;
      }
      servo[servo2] = aPosition;            // Moves the arm servo
      if(joy1Btn(5))               // Open Gripper
      {
        servo[servo1] = 255;
      }
      else if(joy1Btn(6))          // Close Gripper
      {
        servo[servo1] = 0;
      }
    }
  }
```