Control Mapping  Radio Control Setup

In this section, you will learn the Radio Control subsystem components and make sure the system is ready for use.

The VEX Radio Control Transmitter is your primary means of human control over the robot while it is running.

The Radio Transmitter includes a number of menu options and special modes that operate independently of ROBOTC. However, for these lessons, you must use the default configuration of the Radio Transmitter or you may experience strange behavior. This section will show how to put the Radio Transmitter into its default configuration.
Control Mapping  Radio Control Setup (cont.)

1. Reset the Transmitter to its default settings.

1a. Turn the Transmitter ON
Push the power switch on the Transmitter to the ON position.

1b. Enter Menu Mode
Press both the Mode and Select buttons at the same time. Hold until the Transmitter beeps and the screen changes. Release the buttons when you hear the beep.

1c. Select Transmitter Configuration 1
Config 1 is the first menu item to appear by default, so simply press the Select button to select it.

1d. Reset (CL) the channel data
Push and hold the Data Input slider in the up or down position until you hear a double-beep confirming that the configuration data has been Cleared. Once it beeps, take your finger off of the button.
Radio Control Setup (cont.)

**Checkpoint**

The Radio Transmitter includes two **joysticks** on the front and two **pairs of buttons** on the back. Together, these controls manipulate six **channels** of information. Each channel encodes a single **value**, sent from the Transmitter to the Receiver on the robot using short range FM radio waves. Once the radio signal arrives on the robot, the values are decoded and made available to the ROBOTC program.

The six channels of information correspond to the following physical parts of the Radio Transmitter:

*Joysticks have two channels each (one for horizontal position of the stick, and one for vertical).*
*Buttons have one channel per pair of buttons.*

<table>
<thead>
<tr>
<th>Transmitter Part</th>
<th>Radio Channel</th>
<th>ROBOTC Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Joystick" /></td>
<td>((( Channel 1 ))))</td>
<td>vexRT[ch1]</td>
</tr>
<tr>
<td><img src="image" alt="Joystick" /></td>
<td>((( Channel 2 ))))</td>
<td>vexRT[ch2]</td>
</tr>
<tr>
<td><img src="image" alt="Joystick" /></td>
<td>((( Channel 3 ))))</td>
<td>vexRT[ch3]</td>
</tr>
<tr>
<td><img src="image" alt="Joystick" /></td>
<td>((( Channel 4 ))))</td>
<td>vexRT[ch4]</td>
</tr>
<tr>
<td><img src="image" alt="Buttons" /></td>
<td>((( Channel 5 ))))</td>
<td>vexRT[ch5]</td>
</tr>
<tr>
<td><img src="image" alt="Buttons" /></td>
<td>((( Channel 6 ))))</td>
<td>vexRT[ch6]</td>
</tr>
</tbody>
</table>
Control Mapping Radio Control Setup (cont.)

When multiple Radio Transmitters are being used in close proximity (for example, neighboring teams who are testing their robots), the radio signals from one Transmitter can sometimes interfere with the signals from another.

In order to prevent this, Frequency Crystals are used to shift the radio waves into different frequency bands (sometimes simply called frequencies). Transmitters will not interfere with each other as long as they are transmitting on different frequencies.

Since the goal is to have each Transmitter’s signal picked up by only the one correct Receiver, Receivers also use frequency crystals. The crystal in the Receiver ensures that it only receives signals from the Transmitter with the matching crystal.

2. Check your Frequency Module pairs and Receiver connection.

2a. Check Transmitter Frequency
Find the Transmitter (Tx) frequency crystal on the back of the Transmitter. The large number identifies the frequency band for this crystal.
Control Mapping  Radio Control Setup (cont.)

**2b. Check Receiver Frequency**
Find the Receiver (Rx) frequency crystal on the top of your Receiver. Make sure that it matches the frequency that you have on your Transmitter!

**2c. Check Receiver plug**
Make sure that the Receiver Module on the robot is plugged into the Rx1 port on the VEX Micro Controller.

**2d. Check Neighbor Frequencies**
Check the frequency numbers of the other teams in your class to make sure that they are not using the same frequencies.

End of Section
You have successfully chosen your crystals and configured the Transmitter. Your radio control subsystem is now set up and ready to use.
In this lesson, you will learn how values sent from the Transmitter joysticks can be used to control motor powers.

To set motor powers, you use commands of the form:

Example:

```c
motor[motorName] = motorPower;
```

For example:

```c
motor[port2] = 127;
motor[port2] = 0;
motor[port2] = -127;
```

Right motor: Full power forward
Right motor: Stop and brake
Right motor: Full power reverse

Whenever you want to set the motor to a different power, you change the number that you use as the "motorPower". Any number between -127 and +127 works, and each represents a certain amount of forward or reverse power that the motor should output.

Let's think about the value `motorPower` (the 127, 0, or -127 above). In practice, it's like a slot in your program where you need to insert a numeric value to complete the command. Different values in that position yield different results.

So far, we've always filled that number in at the time the program is written. But, now that we want the robot to respond to real-time commands from the transmitter, a pre-typed value won't work. Instead, the motor power needs to be set based on a live value sent from the transmitter.

The joysticks on the front of the transmitter have two values, corresponding to two channels, each. Consider what happens when you move a joystick up and down.
Control Mapping Values and Axes (cont.)

The joystick’s vertical position is measured by a device inside the Radio Transmitter called a *potentiometer*. The signal from the potentiometer is translated into a value from -127 to +127 as shown, with 0 being the neutral center position.

The right stick’s vertical position corresponds to the value on channel 2. The value, between -127 and +127 depending on the stick’s vertical position, is sent from the Transmitter, through the Receiver, and into the ROBOTC program as the value `vexRT[ch2]`.

<table>
<thead>
<tr>
<th>Transmitter Part</th>
<th>ROBOTC name</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Joystick at +127" /></td>
<td><code>vexRT[ch1]</code></td>
</tr>
<tr>
<td><img src="image2.png" alt="Joystick at -127" /></td>
<td><code>vexRT[ch2]</code></td>
</tr>
<tr>
<td><img src="image3.png" alt="Range of joystick movement" /></td>
<td><code>vexRT[ch3]</code></td>
</tr>
<tr>
<td><img src="image4.png" alt="Range of joystick movement" /></td>
<td><code>vexRT[ch4]</code></td>
</tr>
</tbody>
</table>
Control Mapping  Values and Axes  (cont.)

Now, put these two ideas together:
• You set a motor to run at any power by specifying the power value in a motor command.
• The joysticks on the Transmitter are constantly providing values to the robot that correspond to the positions of the sticks in certain directions.

By using the joystick’s position value as the motor’s power, the joysticks can be used to control the robot’s motors.

Example:
```
motor[motorName] = motorPower;
motor[port2] = vexRT[ch2];
```

Setting motors and channels
In this example, vexRT[Ch2] is the ROBOTC keyword that assigns the power level received from Channel 2 of the Radio Transmitter to Motor port 2.

The other axes along which each joystick can move are also assigned channel numbers. Pairing each Transmitter input channel with a motor would allow us to set both motor powers using joystick movements!

Example:
```
motor[port3] = vexRT[ch3];
motor[port2] = vexRT[ch2];
```

<table>
<thead>
<tr>
<th>Transmitter Part</th>
<th>Radio Channel</th>
<th>ROBOTC name</th>
<th>In THIS program, used to control...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>((Ch1)))</td>
<td>vexRT[ch1]</td>
<td>nothing</td>
</tr>
<tr>
<td></td>
<td>((Ch2)))</td>
<td>vexRT[ch2]</td>
<td>motor[port2] (right motor)</td>
</tr>
<tr>
<td></td>
<td>((Ch3)))</td>
<td>vexRT[ch3]</td>
<td>motor[port3] (left motor)</td>
</tr>
<tr>
<td></td>
<td>((Ch4)))</td>
<td>vexRT[ch4]</td>
<td>nothing</td>
</tr>
</tbody>
</table>
Control Mapping Values and Axes (cont.)

1. Start with a basic moving forward program. One is provided in the ROBOTC samples folder. Open the Moving Forward sample program in the Training Samples.

2. Before making changes, save the program with a new name.
Control Mapping Values and Axes (cont.)

2b. **Name the program**
Name the program “RadioControl”.

2c. **Save the program**
Click Save to save the program with the new name.

3. Enable radio control processing on the robot. This is not on by default, because it consumes additional power and processor time.

```c
1  task main()
2  {
3  
4    bIfiAutonomousMode = false;
5    bMotorReflected[port2] = 1;
6
7    motor[port3] = 127;
8    motor[port2] = 127;
9    wait1Msec(2000);
10  }
```

3. **Add this code**
This line of code takes the robot out of autonomous-only mode, allowing the VEX controller to accept signals from the Radio Transmitter.
4. Change the motor power commands to use values from the Transmitter joysticks instead of pre-set values.

```c
4. Modify this code
Replace the motor power values with the values provided by the joysticks. Channel 2 is the right joystick vertical position, and Channel 3 is the left joystick vertical position.

```vex
if AutonomousMode = false;            
MotorReflected[port2] = 1;

motor[port3] = vexRT[Ch3];  
motor[port2] = vexRT[Ch2];
wait1Msec(2000);            
```vex
```

5. Compile and download the program to the robot. (Make sure your robot is on and that the robot is connected to your computer with the programming cable.)

```vex
5. Compile and download program
Choose Robot > Compile and Download Program
```

Caution

Your program is not yet completely functional and will not behave as planned. Test it to see what it does instead.
Control Mapping Values and Axes (cont.)

6. Turn on the Transmitter.

7. Hold the Transmitter joysticks in various different positions, and run the program several times. For this test, you must keep the robot plugged into the programming cable and use the on-screen Start button to start the program.

7a. Hold joysticks in “up” position
Hold both joysticks in the “up” position. This will cause both vexRT[ch2] and vexRT[ch3] to take on values at or around +127.

7b. Keep the robot connected
Unplugging will disable you from using the Start button, which is essential for this test.

7c. Press the Start button
This will restart the robot and run the program. Do not release the joysticks while performing this step.

7d. Observe robot behavior
Observe the behavior of the robot. Try moving the sticks around while the robot is moving. Record your observations in your notes.
Control Mapping Values and Axes (cont.)

7d. Hold joysticks in “down” position
Hold both joysticks in the “down” position. This will cause both vexRT[ch2] and vexRT[ch3] to take on values at or around -127.

7c. Press the Start button
This will restart the robot and run the program. Do not release the joysticks while performing this step.

7f. Observe robot behavior
Observe the robot’s behavior. Again, try moving the sticks while the robot is running.
Record your observations.

7g. Power down
Turn off the robot and Transmitter to save battery power while we review our observations.

End of Section
The robot responds to Transmitter commands, but only once at the very beginning of the program. The motors do not update again to reflect changes in the joystick positions after that. Continue on to the next lesson, where you will learn about the while() loop, a control structure that can help to make sure the motor powers are continually updated to match the joystick values.

Helpful Hints
Because the program is still incomplete, the robot will not respond to additional commands after it has started moving. However, if your robot does not respond even to the initial command, make sure that you have the sticks held in position WHILE turning on the robot. If you still do not observe the desired results, please see the Radio Troubleshooting Guide.
Control Mapping while() loop

In this lesson, you will learn what a while() loop is and how it works.

Your robot has the ability to respond to Transmitter input only once, when the motor command runs. Consequently, your Transmitter control over the robot is gone after the first split second of the program (when the motor command runs). In order to maintain continued control, the motors need to continue being updated with values from the Transmitter. That is, the motor values must be updated repeatedly.

The sample code snippet below allows the robot to respond to Transmitter input continually by repeating the motor commands over and over. Each time a motor command runs, it refreshes the value in the motor to match whatever was coming from the Transmitter at the time. As this happens over and over, the motor powers keep changing, always keeping up with the latest Transmitter values.

```
while(SensorValue(bumper) == 0)
{
    motor[port3] = vexRT[ch3];
    motor[port2] = vexRT[ch2];
}
```

Reading this statement out loud tells you pretty much exactly what it does:

“While the sensor value of the Bumper is equal to zero, set the power on motor ports 3 and 2 to match the joystick values.”

The program will repeatedly set the motor powers to match the current Transmitter values. The motor lines should be familiar to you. The repetition in the program, however, is done by a structure called a while() loop.

When the program reaches most commands, it runs them, and then moves on. When the program reaches a while() loop, however, it steps inside the loop, and stays there as long as the while() loop decides that it should. The commands that are contained inside the loop will repeat over and over as long as the program remains inside the loop.

The programmer specifies in advance under what conditions the program should remain in the loop, and what commands the robot should repeat while inside the loop.

The while() loop therefore has three parts, in order:
- The word while
- The condition enclosed in parentheses ( )
- A body of commands enclosed in curly braces { }
Control Mapping while() loop (cont.)

while
A while() loop always starts with the word “while”.

The (condition)
The statement in parentheses specifies the condition(s) under which the loop continues repeating. These conditions are specified in the form of a true-or-false statement such as the one in the example above, “The sensor value of the Bumper Switch is equal to zero”. The statement is either true (the value IS zero) or it is false (the value IS NOT zero).

The true (or false) value of the statement determines whether the loop will continue or end. As long as the condition is true, the while loop will continue to run. If the condition becomes false, the loop will end and the program will move on to the commands that come after it.

The {commands}, sometimes called the {body}
These are the commands that are run while the condition is true. The commands inside the braces are run in order. When they have all been run, the program goes back to check the condition again. If the (condition) is still true, the loop continues and the {commands} are run again. In the code shown above, the {commands} are to update both of the robot’s motors to match the Transmitter values, and the program will do that as long as the Bumper Switch remains unpressed.

<table>
<thead>
<tr>
<th>When the bumper is...</th>
<th>...the SensorValue is...</th>
<th>Therefore, the condition (SensorValue(bumper) == 0) is...</th>
<th>...and the while loop will...</th>
<th>...causing the robot to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not pressed</td>
<td>0</td>
<td>true</td>
<td>repeat</td>
<td>Update the motor powers</td>
</tr>
<tr>
<td>Pressed</td>
<td>1</td>
<td>false</td>
<td>end</td>
<td>Stop updating the motor powers</td>
</tr>
</tbody>
</table>

End of Section

The while() loop allows the program to make a decision about program flow, based on a true-or-false statement. It works by checking to see if a (condition) is true. Then, if it is true, it runs a group of {commands} and loops back to recheck the (condition). If the (condition) ever stops being true, the while() loop skips over the {commands}, and moves on to the next section of the program.

In the sample above, the robot repeatedly updates the motor powers to match the Transmitter input while the Bumper remains unpressed. When the bumper is pressed, the robot exits the loop and moves on to the rest of the program.
In this lesson, you will use an infinite while() loop to achieve real-time control of your robot’s motors.

In the original program, your robot was able to set its motor powers based on the Transmitter joystick values, but only once at the beginning of the program, when the motor commands set the motors to run at a power level based on the joystick positions. In order to make the motors update continuously, the motor commands need to be repeated more than just once. For this, we will use a while() loop.

1. Open your RadioControl program from the previous lesson.

2. Add an infinite while() loop to your program.

```c
1       task main()
2           {
3                      bIfiAutonomousMode = false;
4                      bMotorReflected[port2] = 1;
5                      motor[port3] = vexRT[Ch3];
6                      motor[port2] = vexRT[Ch2];
7                      wait1Msec(2000);
8                      while(1 == 1)
9                          {
10                             
11                             }
12           }
13
14
15
16 }
```

== symbol

The == symbol in ROBOTC means "is equal to", and is used in conditions to check whether two numbers are equal.

A condition like (x == y) is true if x and y are equal, and false if they are not.

2. Add this code

An infinite while() loop uses a (condition) that is always true, like 1 == 1 (1 is equal to 1).
Control Mapping  Real-time Control (cont.)

3. Move the motor commands into the `while` loop's `{body}`.

```c
while(1 == 1) {
    motor[port3] = vexRT[Ch3];
    motor[port2] = vexRT[Ch2];
    wait1Msec(2000);
}
```

3a. Highlight this code
Highlight both `motor` commands and the `wait1Msec` command.

3b. Cut highlighted code
Select Edit > Cut to remove the highlighted code from the program and put it on the clipboard.

3c. Place cursor here
Place your cursor on the line between the `while` loop's `{curly braces}`.

3d. Paste
Select Edit > Paste to put the clipboard code back into the program.

4. Add an infinite `while()` loop to your program.

```c
while(1 == 1) {
    motor[port3] = vexRT[Ch3];
    motor[port2] = vexRT[Ch2];
    wait1Msec(2000);
}
```

4. Delete this code
The robot will repeat the `motor` commands as long as the loop continues running (forever) so the old 2-second delay is no longer necessary.
Control Mapping  Real-time Control  (cont.)

5. Save your program, compile and download it to the robot, and run it.

5a. Save your program
Select File > Save.

5b. Compile and Download
Make sure your robot is on and that the robot is plugged in with the USB cable. Then, choose Robot > Compile and Download Program.

6. Turn the Transmitter on.

6. Turn Transmitter ON
Control Mapping  Real-time Control (cont.)

7. Run the program and use the joysticks to control the robot’s movement.

7a. Run the Program

7b. Optional
Unplug the robot and place it on the board.

7c. Move Transmitter joysticks
Observe the robot’s reaction as you move the joysticks in different combinations.

7d. Turn Transmitter OFF
Always remember to turn the transmitter off after use to conserve battery power.

End of section

You now have remote control of your robot through the Transmitter joysticks! Continue on to the next lesson to learn more about how to make the robot move in specific ways and to improve the robot’s low-speed performance to help it maneuver through the tight quarters of the minefield.
Control Mapping  Mapping Functions

In this lesson, you will learn how the relationship between a joystick position value and a motor’s output power can be expressed as a function, and how modifying this “mapping” function can produce different results.

1. Let’s begin by visualizing how the joystick movements are affecting the robot’s motors. “Block up” your robot by holding it up in the air or placing it on top of something that prevents its wheels from touching the ground.

   1a. Elevate the robot
   Use a book or box to suspend the robot in the air so that its gears and wheels can turn freely without actually moving the robot.

   1b. Turn Transmitter ON

   1c. Turn the robot off then on
   The program runs (this should be the RadioControl program from the previous lesson).

   1d. Move left joystick
   Push the left joystick on the Transmitter up, then down. Observe how the robot responds.
Control Mapping  Mapping Functions (cont.)

Checkpoint

The line above, `motor[port3] = vexRT[ch3];` sets the motor’s power based on the Transmitter’s left stick vertical position. When you press forward on the stick, the Transmitter generates a value. That value is used as the motor’s power setting. Pressing forward on the stick gives a high value (around +127) to the motor, which turns forward at full speed in response (motor power 127). Leaving the stick in the neutral position generates a value of zero, which causes the motor to brake (motor power 0). Pulling back on the stick give a low value (around –127) to the motor, which makes it spin backward (motor power –127). Stick positions in-between make the motor move at corresponding in-between speeds as well.

This tying of input (joystick) values to output (motor) values is called a mapping. Since the raw input values are directly used as output values without modification, this specific setup is called a direct mapping.

```
motor[port3] = vexRT[ch3];
```

Move the right Transmitter joystick and observe the effect on the robot’s motors.
An identical mapping relationship is established between the right Transmitter stick and the right robot motor. Both the left and right motors have their power levels set directly from the Transmitter joysticks.

You know which motor powers are needed to perform each of the behaviors listed below in step #2. Since motor powers and stick positions are equivalent thanks to the direct mapping we have set up, what combination of left and right stick positions would create these motions?

2. Find the combination of joystick inputs needed to produce all the basic movement and turning behaviors:
   - Forward at full speed
   - Forward at half speed
   - Left turn in place
   - Right turn in place
   - Stop

motor[port3] = vexRT[ch3];
**Control Mapping**  
*Mapping Functions (cont.)*

1c. **Run the program**  
Turn the robot off, then on.  
The program will run.

1d. **Use the sticks to drive the robot**  
Make sure that you can perform all the important maneuvers listed on the previous page.

1e. **Turn the Transmitter OFF**
### Control Mapping Mapping Functions (cont.)

<table>
<thead>
<tr>
<th>Movement</th>
<th>Left Motor Power</th>
<th>Right Motor Power</th>
<th>Left Stick Vertical Position</th>
<th>Right Stick Vertical Position</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td>127</td>
<td>127</td>
<td>![Image2]</td>
<td>![Image3]</td>
</tr>
<tr>
<td><img src="image4" alt="Diagram" /></td>
<td>63</td>
<td>63</td>
<td>![Image5]</td>
<td>![Image6]</td>
</tr>
<tr>
<td><img src="image7" alt="Diagram" /></td>
<td>-127</td>
<td>127</td>
<td>![Image8]</td>
<td>![Image9]</td>
</tr>
<tr>
<td><img src="image10" alt="Diagram" /></td>
<td>127</td>
<td>-127</td>
<td>![Image11]</td>
<td>![Image12]</td>
</tr>
<tr>
<td><img src="image13" alt="Diagram" /></td>
<td>0</td>
<td>0</td>
<td>![Image14]</td>
<td>![Image15]</td>
</tr>
</tbody>
</table>
Control Mapping  Mapping Functions (cont.)

Checkpoint

Simple enough, right? Just figure out what motor power each wheel needs, then push the sticks to the corresponding positions.

Use the code that you’ve written to drive around the minefield and approach the mines without knocking them over. How does the robot handle at low speeds? You are probably noticing that it is difficult to make some of the more precise maneuvers. Remember, in the Mine Retrieval challenge, you will need to move exactly into position to pick up the mines without knocking over their trigger stems.

We can gain better low-speed control by changing the mapping to sacrifice high-speed performance in favor of more precise low-speed handling. Let’s change things so that the joysticks will “map” their full range of movement to cover only half of the possible motor speeds. Pressing the stick all the way forward will make the robot move only half of full speed. This will allow you to use the same joystick values, but to have them be less “sensitive” because they will be covering a smaller range of motor values. We can achieve this by having the robot assign the motors to run at half the value of the joystick input.

Mapping Graphs
Red shows the original mapping (motor = joystick). Blue shows the modified mapping (motor = joystick/2). If you pick a joystick input (X-value), find the corresponding point, then look up its motor output power (Y-value), you can see that the motor output is half of the input value.

3. Change the code so that the motors powers are set to only half the joystick value.

```cpp
while(1 == 1)
{
    motor[port3] = vexRT[Ch3]/2;
    motor[port2] = vexRT[Ch2]/2;
}
```

3a. Modify this code
Divide the Transmitter joystick values by two, then assign the resulting value to the motors.
Control Mapping  Mapping Functions  (cont.)

3b. Save the program  
Choose File > Save to save your program.

3c. Compile and Download  
Choose Robot > Compile and Download.

3d. Turn Transmitter ON

3e. Run the Program
End of section

You have changed the control mapping so that the joystick’s full range of motion is now spread out over only half the range of motor speeds. This tradeoff allows you more precise control of the robot at low speeds, at the expense of maximum speed performance.

A final note about mappings. While our change was simple, there are many more mappings possible. Some of them are also simple (motor = joystick/4), while others may be much more complex and bring additional benefits (motor = joystick2/127). You should feel free to explore these additional possibilities until you find one that makes it easy to maneuver the robot.