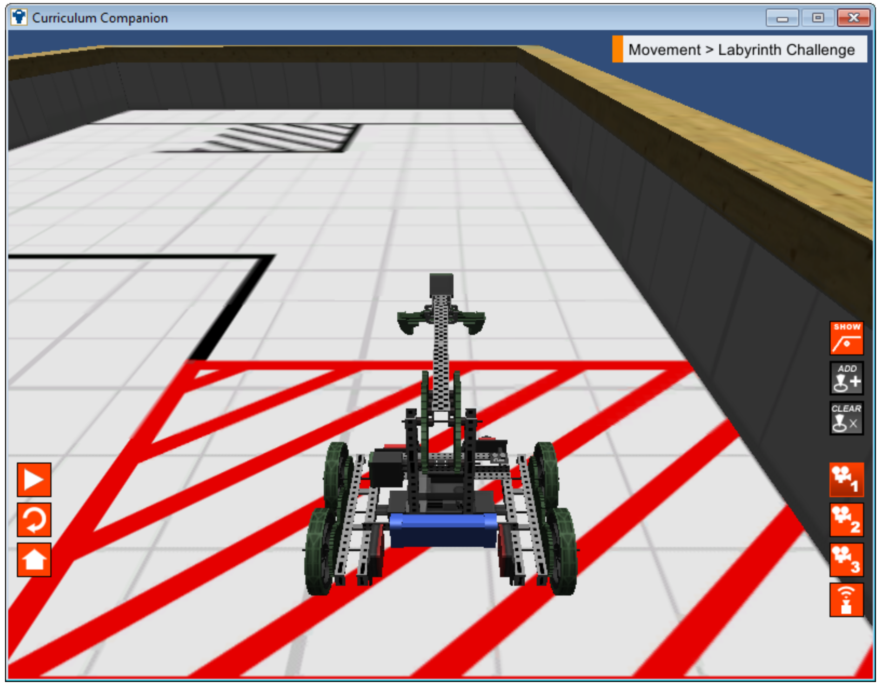
RobotC

Remote Control

Learning Objectives: Focusing on Virtual World with Physical Examples

- Understand Real-Time Joystick Mapping
- Understand how to use timers
- Understand how to incorporate buttons into controlling robot arms

Getting Started in RobotC



- // Comment
- task
- main()
- motor[]
- {}
- wait1Msec()
- ;
- =
- Header
- Code
- Compile
- Download
- Run

Learning Objectives

- Understand Motion
  - Motors: How they work and respond.
  - Fuses: Understand why they keep blowing
- Understand how to control Motors with a program

SuperQuest Salem

Motion

VEX Motion: Motors

- 2-Wire Motor 393
  - 100 RPM
  - Stall Torque 1.67 Nm
- Motor Controller: 2-Wire to 3-Wire
- Integrated Motor Encoder Sold Separately
  - Counts ticks
  - 627.2 Ticks per revolution

- High Speed Gearing (Comes with motor)
  - 160 RPM
  - Stall Torque 1.04 Nm
  - 392 Ticks per Revolution
- Turbo Gear Set (Sold Separately)
  - 240 RPM
  - Stall Torque 0.7 Nm
  - 261.333 Ticks per Revolution

393 Specifications

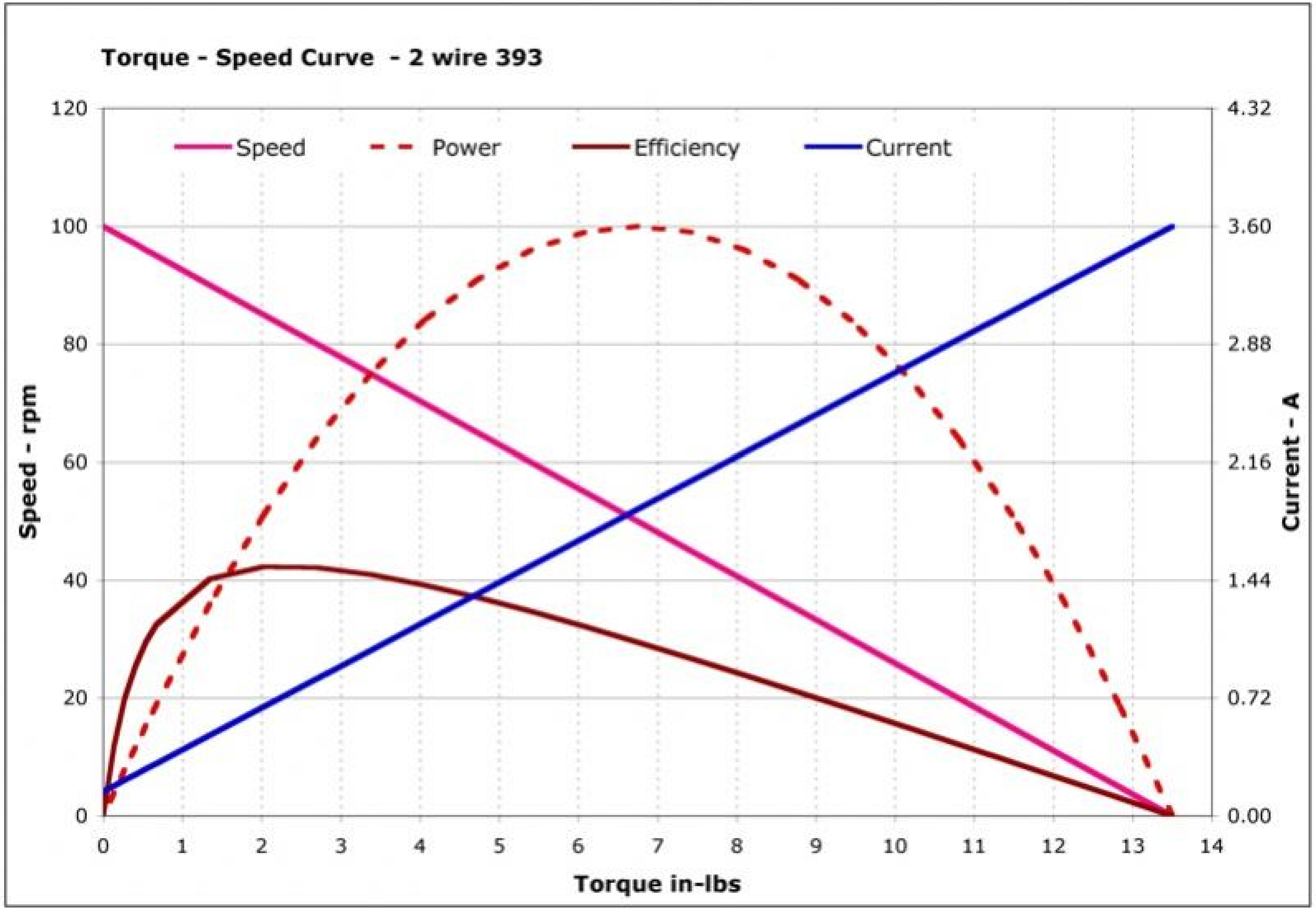
Max Current and Torque at 0 RPM

Max Power (Combination of Speed and Torque) at 50 RPM.

Max Efficiency. Out/In at 85 RPM.

speed rpm	torque in-lbs	power W	Power %	current A	power input W	Efficiency %
0	13.500	0.000	0.000	3.600	25.920	0.000
5	12.825	0.757	19.000	3.428	24.678	3.066
10	12.150	1.434	36.000	3.255	23.436	6.118
15	11.475	2.031	51.000	3.083	22.194	9.151
20	10.800	2.549	64.000	2.910	20.952	12.165
25	10.125	2.987	75.000	2.738	19.710	15.154
30	9.450	3.345	84.000	2.565	18.468	18.114
35	8.775	3.624	91.000	2.393	17.226	21.038
40	8.100	3.823	96.000	2.220	15.984	23.919
45	7.425	3.943	99.000	2.048	14.742	26.745
50	6.750	3.983	100.000	1.875	13.500	29.500
55	6.075	3.943	99.000	1.703	12.258	32.164
60	5.400	3.823	96.000	1.530	11.016	34.706
65	4.725	3.624	91.000	1.358	9.774	37.079
70	4.050	3.345	84.000	1.185	8.532	39.209
75	3.375	2.987	75.000	1.013	7.290	40.972
80	2.700	2.549	64.000	0.840	6.048	42.143
85	2.025	2.031	51.000	0.668	4.806	42.261
90	1.350	1.434	36.000	0.495	3.564	40.227
95	0.675	0.757	19.000	0.323	2.322	32.587
96	0.540	0.612	15.360	0.288	2.074	29.500
97	0.405	0.464	11.640	0.254	1.825	25.398
98	0.270	0.312	7.840	0.219	1.577	19.801
99	0.135	0.158	3.960	0.185	1.328	11.872
100	0.000	0.000	0.000	0.150	1.080	0.000
	Max Power	3.983		Max Efficiency		42.261

393 Torque – Speed Curve

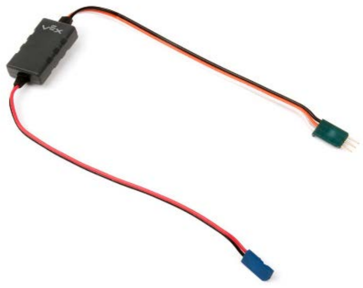


More 393 Motor Facts

- 3.6 Amp Stall Current
- Built in Thermal Fuse.
  - Will cut power when pulling 1.8A + for 7 + seconds.
    - Just wait for 10 seconds for fuse to cool.
  - Will trip faster with higher current or warmer temps.
- Designed to run continuously at 0.9 A.

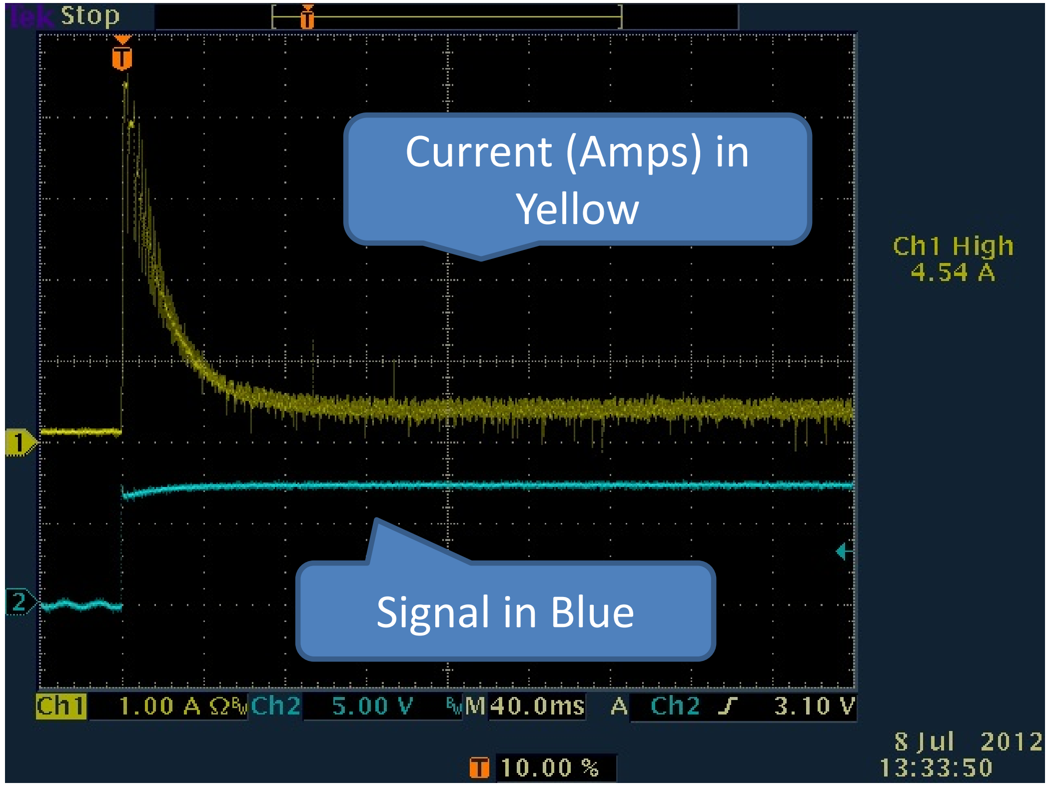
Cortex Thermal Fuses: Causes robot to stop moving

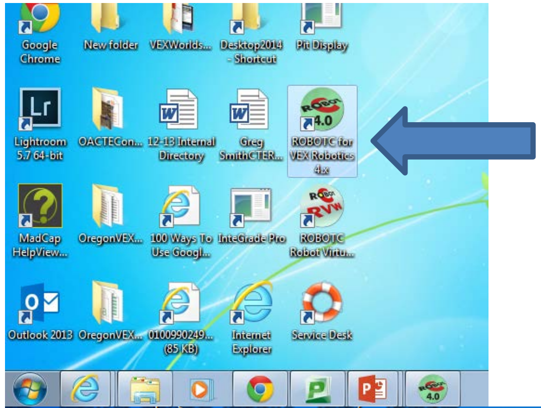
- 4 amp combined draw from ports 1-5
- 4 amp combined draw from ports 6-10
- 4 amp combined draw from 4 ports on Power Expander
- Motor Controller: Max Current: 3 amps at 8.5 V

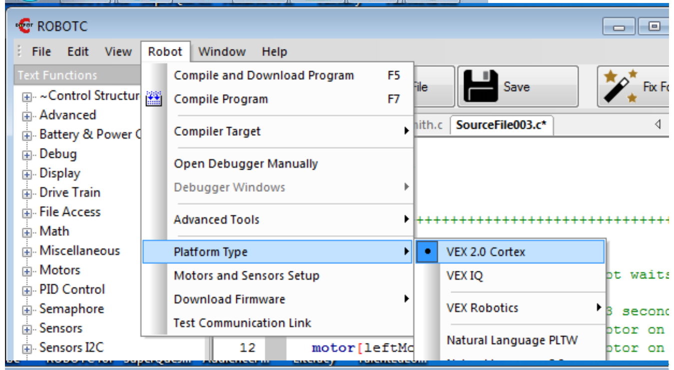


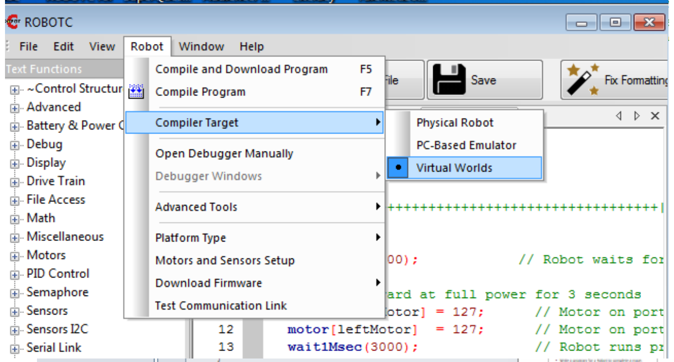
What happens when you floor it?

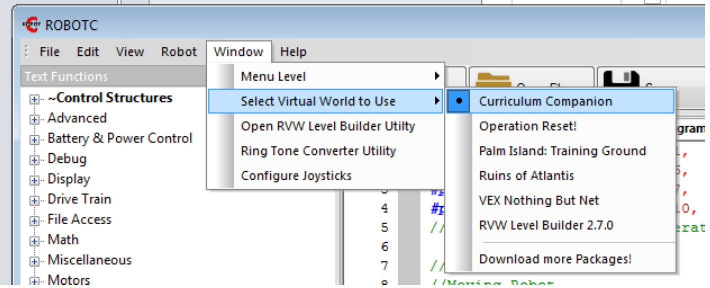
- Fuses you can blow
- Motor: 3.6 Amp
  - One Motor Stops
- Controller: 3 Amp
  - One motor stops
- Cortex Port: 4 amps combined with four other ports. Robot Stops





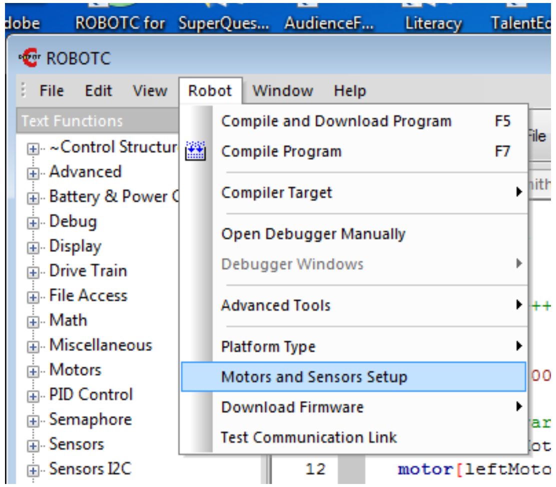






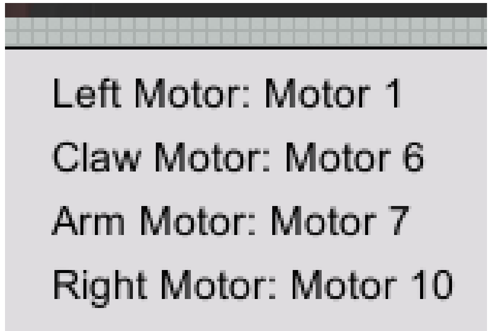
Getting Started

- Open RobotC
- Select VEX 2.0 Cortex Platform
  - Robot-> Platform ->VEX 2.0 Cortex
- Make the robot compile to Virtual Worlds
  - Robot-> Compiler Target -> Virtual Worlds
- Select Virtual World
  - Window->Select Virtual World to Use -> Curriculum Companion

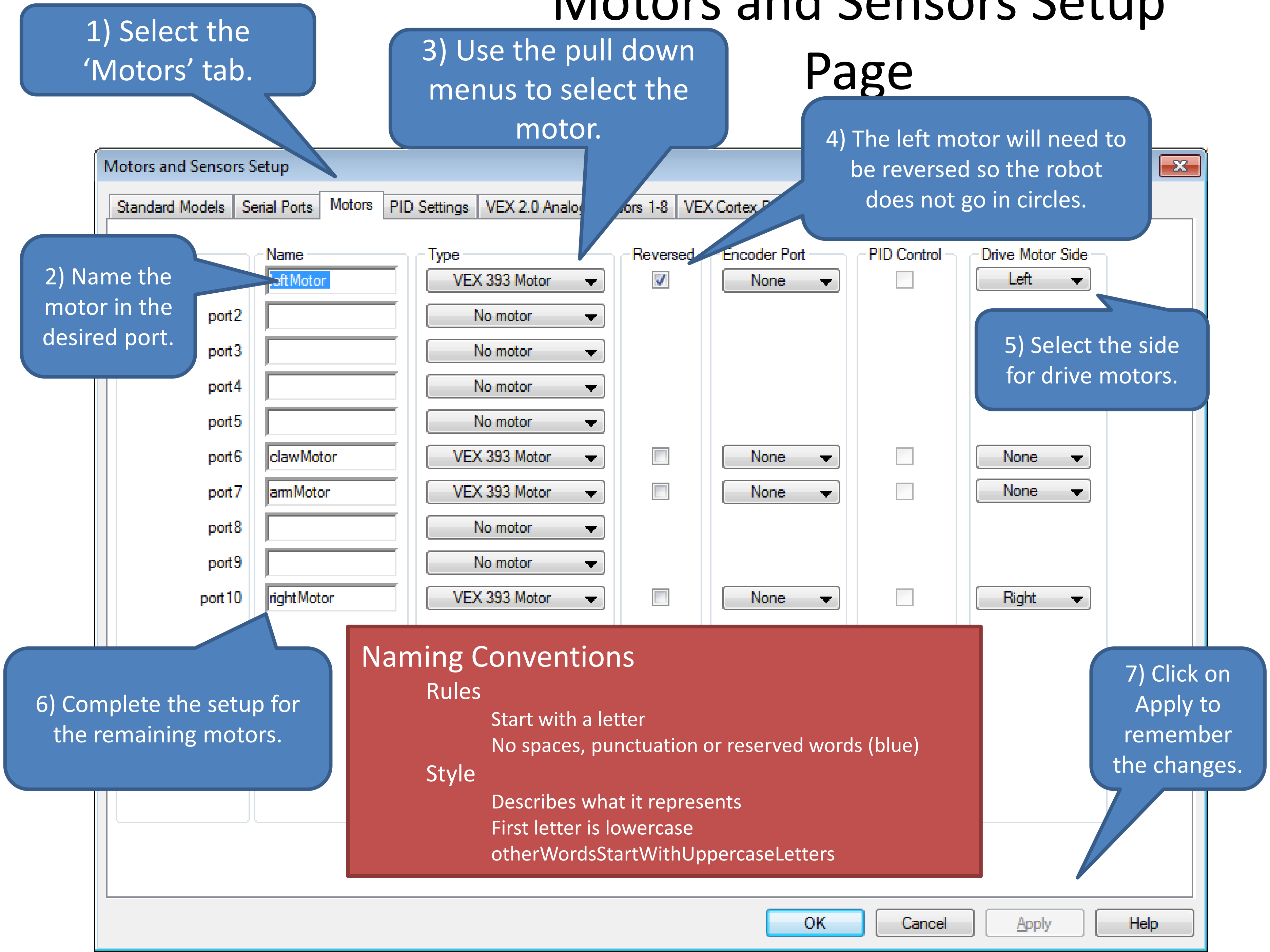


Configuring the Robot: Focus on Motors

- Robot -> Motors and Sensors Setup
- Select the motor
  - Currently can only purchase 393 Motors, also modify for internal gearing (high speed, turbo speed)
- Naming Convention
  - Rules
    - Start with a letter
    - No spaces, punctuation or reserved words (blue)
  - Style
    - Describes what it represents
    - First letter is lowercase
    - otherWordsStartWithUppercaseLetters
  - For these motors
    - leftMotor
    - clawMotor
    - armMotor
    - rightMotor



Motors and Sensors Setup Page



Naming Conventions

Rules

- Start with a letter
- No spaces, punctuation or reserved words (blue)

Style

- Describes what it represents
- First letter is lowercase
- otherWordsStartWithUppercaseLetters

Code the setup creates ‘pre-processor directives’

VEX Start Page | Moving ForwardSmith.c | SmithFirstProgramRobotc.c

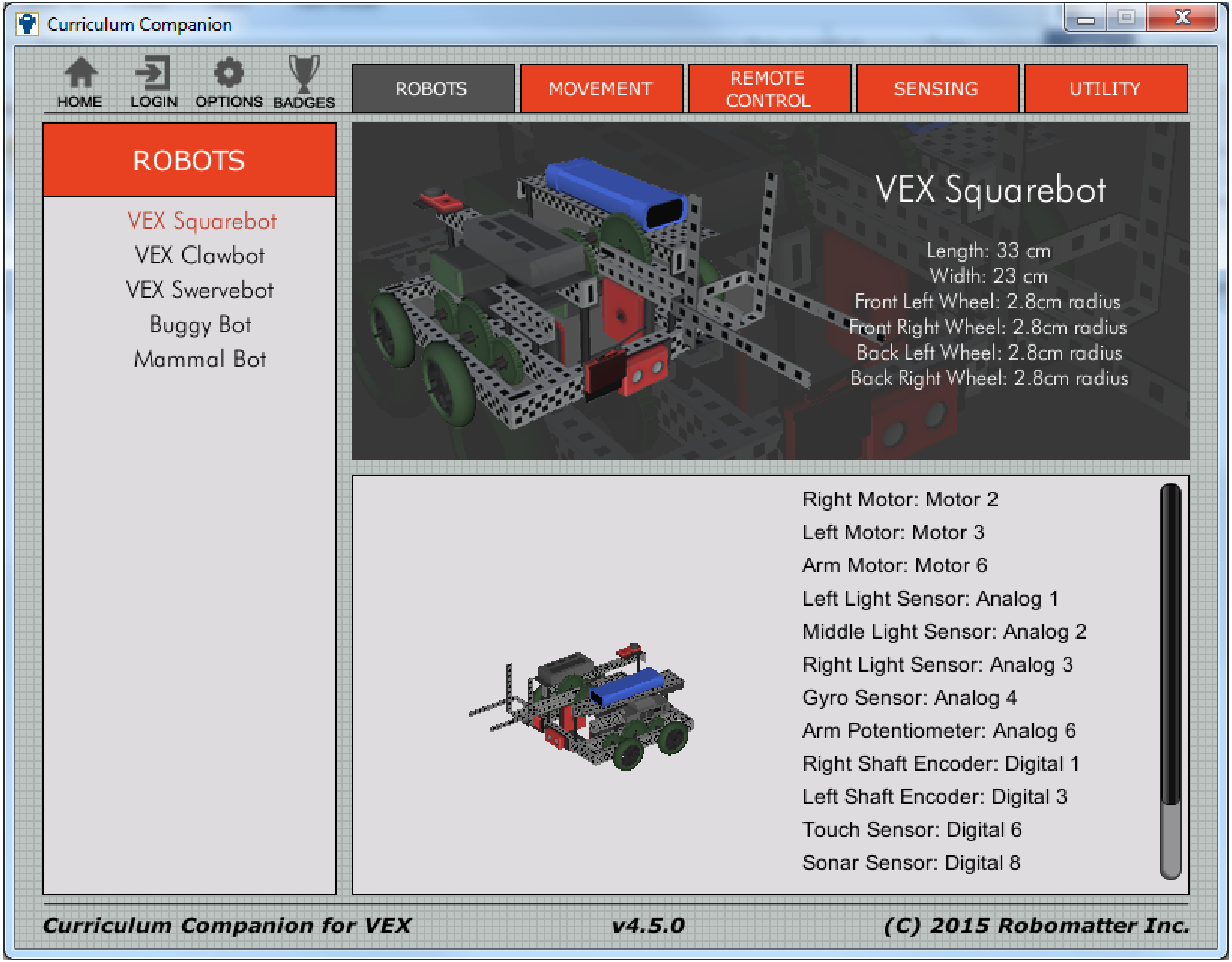
```
#pragma config(Motor,  port1,           leftMotor,     tmotorVex393_HBridge, openLoop, reversed, driveLeft)
#pragma config(Motor,  port6,           clawMotor,     tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port7,           armMotor,      tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port10,          rightMotor,    tmotorVex393_HBridge, openLoop, driveRight)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

//Greg Smith
//Moving Robot
//8-4-2015

```

Getting Started… Configuring the **motors** for Squarebot

Only configure the motors for now.



Looking at the Joysticks on the Remote: Physical Robot

Joystick Mapping: Physical

Channel	Left/Down	Middle	Right/Up
vexRT[Ch1]	-127…	0	…127
vexRT[Ch2]	-127…	0	…127
vexRT[Ch3]	-127…	0	…127
vexRT[Ch4]	-127…	0	…127

Joystick Mapping: Virtual

Note: If you copy-paste these into your program, you will need to retype in the “”.

//Place before task main()
#pragma debuggerWindows(“joystickSimple”);
#include “JoystickDriver.c”;

//Place inside the loop prior to joystick. Command
getJoystickSettings(joystick);

Channel	Left/Down	Middle	Right/Up
joystick.joy1_x2	-127…	0	…127
joystick.joy1_y2	-127…	0	…127
joystick.joy1_y1	-127…	0	…127
joystick.joy1_x1	-127…	0	…127

Accessing the Value for the Remote Commands

Physical

vexRT*[ChannelNumber]*

Virtual

```
//Place before task main()
#pragma debuggerWindows("joystickSimple");
#include "JoystickDriver.c";
```

//Place inside the loop prior to '***joystick.***' Command

getJoystickSettings(joystick);

joystick.*joy#_axis#*

Example Using the Remote Values to Drive the Motors

Physical Robot

```
task main ()
{

  while(true)
  {
    motor[leftMotor]  = vexRT[Ch3];
    motor[rightMotor] = vexRT[Ch2];
  }

}
```

These examples assume that the programmer labeled their motors leftMotor and rightMotor.

Can also send the value directly to the motor port.
motor[port3] = vexRT[Ch3];

Virtual World

```
#pragma debuggerWindows("joystickSimple");
#include "JoystickDriver.c"

task main()
{
  //Loop Forever
  while(true)
  {
    //Get the Latest joystick values
    getJoystickSettings(joystick);
    motor[leftMotor] = joystick.joy1_y1;
    motor[rightMotor] = joystick.joy1_y2;
  }
}
```

Online Time: Configure the motors and code the following

Make sure the motors are configured:
leftMotor, port 3, reversed
rightMotor, port 2

Physical Robot

```
task main ()
{

  while(true)
  {
    motor[leftMotor]  = vexRT[Ch3];
    motor[rightMotor] = vexRT[Ch2];
  }

}
```

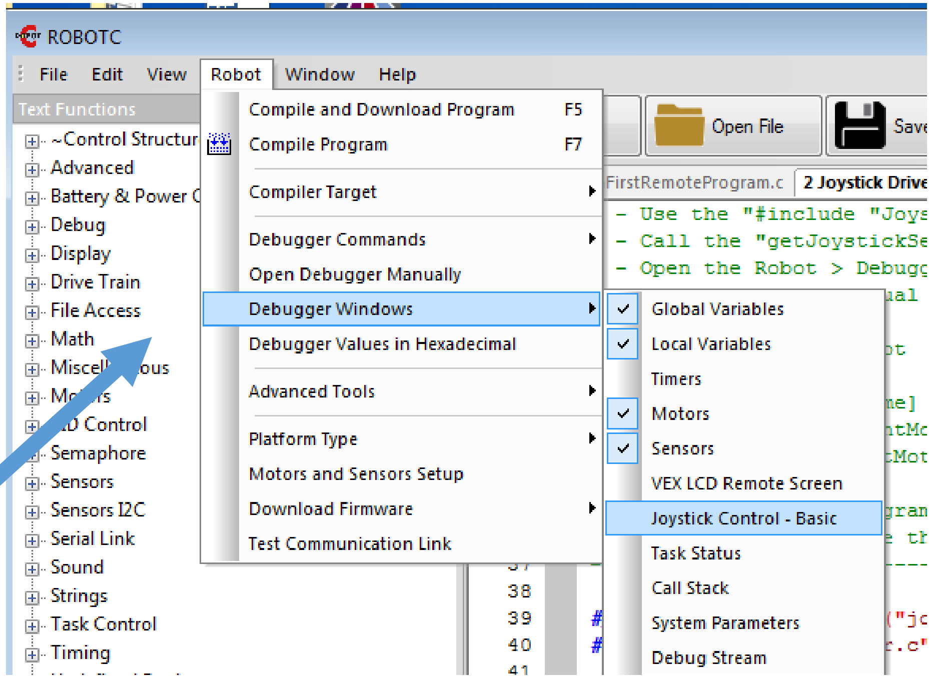
Virtual World

```
#pragma debuggerWindows("joystickSimple");
#include "JoystickDriver.c"

task main()
{
  //Loop Forever
  while(true)
  {
    //Get the Latest joystick values
    getJoystickSettings(joystick);
    motor[leftMotor] = joystick.joy1_y1;
    motor[rightMotor] = joystick.joy1_y2;
  }
}
```

Driving in the Virtual World

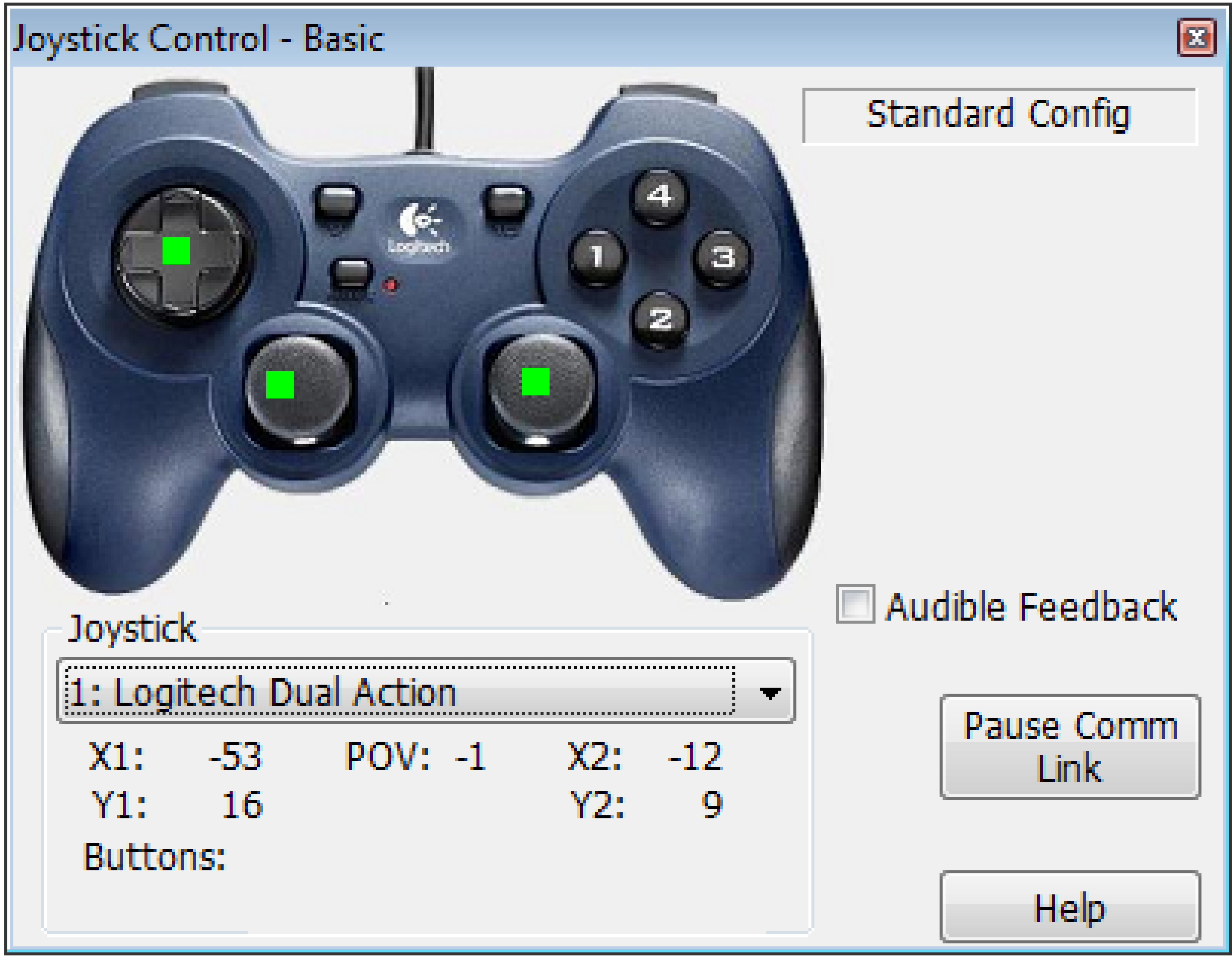
- Compile and Download the Program
- Select Virtual World (Utility – Huge Table is good for starters)
- Open ‘Joystick Control –Basic ‘ Debugger Window’
  - Robot-> Debugger Windows -> ‘Joystick Control – Basic’
- Refresh List if the remote does not show up.



Robot Creeping?

Robot Creeping

- Y1 and Y2 values might not go exactly to ‘0’ when you release the buttons which can cause your robot to creep.
- Can correct this in the code.
- Pseudo Code
  - If the joystick reading is close to 0, say +/- 20
    - Give a 0 power value to the motor
  - Else
    - Give the joystick reading to the motor



A Little RobotC Math to Help

RobotC Function	Description	Example
abs()	Finds the absolute value of a number	float x; x = abs(5-10);

pow()	Calculates a power	float x;
		x = pow(10,3);
		//Calculates and returns 10^3

sqrt()	Finds the square root of a number	float x; x = sqrt(8);

Physical:
Getting Rid of the Creep

```
task main ()
{
  int threshold = 20;

  while(true)
  {
    if(abs(vexRT[Ch3]) < threshold)
    {
      motor[leftMotor]  = 0 ;
    }
    else
    {
      motor[leftMotor]  = (vexRT[Ch3]);
    }

    if(abs(vexRT[Ch2]) < threshold)
    {
      motor[rightMotor] = 0;
    }
    else
    {
      motor[rightMotor] = (vexRT[Ch2]);
    }
  }
}
```

Using a variable to make threshold changes easier

Using the abs command to simplify the condition.
if (vexRT[Ch3] >(-threshold)) && (vexRT[Ch3] < (threshold))
Would give the same results.

Executes this line of code when the above condition is true.

Executes the commands in the ‘else’ when the above condition is false.

Do the same for the rightMotor

Virtual Getting Rid of the Creep

```
#pragma debuggerWindows("joystickSimple");
#include "JoystickDriver.c";

//+++++++++++++++++++++++++++++++++++++++++++++++++
task main ()
{
  int threshold = 20;

  while(true)
  {
    getJoystickSettings(joystick);
    if(abs(joystick.joy1_y1) < threshold)
    {
      motor[leftMotor]  = 0 ;
    }
    else
    {
      motor[leftMotor]  = joystick.joy1_y1;
    }

    if(abs(joystick.joy1_y2) < threshold)
    {
      motor[rightMotor] = 0;
    }
    else
    {
      motor[rightMotor] = joystick.joy1_y2;
    }
  }
}
```

Add the pragma directive and include file. If you copy and paste from the PowerPoint you will need to retype in the “”.

Add the getJoystickSettings(joystick); command inside the while loop.

Replaced vexRT(Ch3) with joystick.joy1_y1

Replaced vexRT(Ch2) with joystick.joy1_y2

More Control Options

- To fight motors timing out, you can modify the drive code to lower the power sent to the motors.
  - Go half-power
  - Create a fancy equation that maps remote input to output. Had some math wizzes that used a 5^{th} degree polynomial to provide more control when going slow.
  - Can put together a bunch of ‘stepped’ if elses to give different power values for different ranges of input values.

```
task main ()
{
  int threshold = 20;

  while(true)
  {
    if(abs(vexRT[Ch3]) < threshold)
    {
      motor[leftMotor]  = 0 ;
    }
    else
    {
      motor[leftMotor]  = (vexRT[Ch3])/2;
    }

    if(abs(vexRT[Ch2]) < threshold)
    {
      motor[rightMotor] = 0;
    }
    else
    {
      motor[rightMotor] = (vexRT[Ch2])/2;
    }
  }
}
```

Physical
No Creep,
Half Power = more control

Half Power

Virtual
No Creep Half Power:

```
#pragma debuggerWindows("joystickSimple");
#include "JoystickDriver.c";

//+++++++++++++++++++++++++++++++++++++++++++++| MA

task main ()
{
  int threshold = 20;

  while(true)
  {
    getJoystickSettings(joystick);
    if(abs(joystick.joy1_y1) < threshold)
    {
      motor[leftMotor]  = 0 ;
    }
    else
    {
      motor[leftMotor]  = joystick.joy1_y1 / 2;
    }

    if(abs(joystick.joy1_y2) < threshold)
    {
      motor[rightMotor] = 0;
    }
    else
    {
      motor[rightMotor] = joystick.joy1_y2 / 2;
    }
  }
}
```

Online Time: Test it on the Utilities -> Huge Table

Half Power

Buttons

- Learning Objectives
- Be able to use the buttons to control motors on your robot.
- Complete challenges that incorporate buttons.

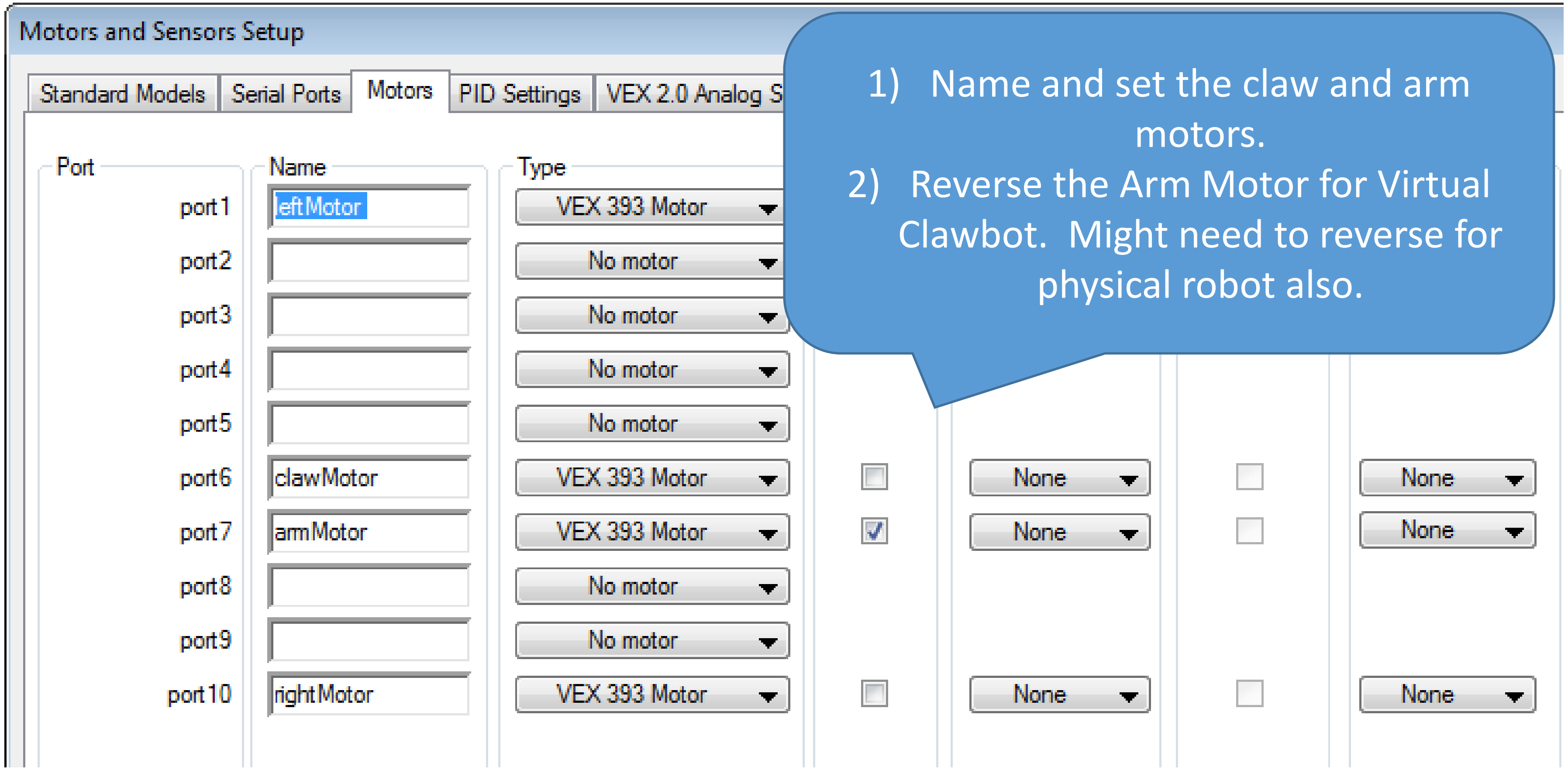
Joystick Buttons: Physical

Buttons return a value of ‘1’ when pushed and ‘0’ when not pushed

Button	Description	Example
5U	Top button on back left	vexRT[Btn5U]
5D	Bottom button, back left	vexRT[Btn5D]
6U	Top button, back right	vexRT[Btn6U]
6D	Bottom button, back right	vexRT[Btn6D]
7U	Button 7 up	vexRT[Btn7U]
7D	Button 7 down	vexRT[Btn7D]
7R	Button 7 right	vexRT[Btn7R]
7L	Button 7 left	vexRT[Btn7L]
8U	Button 8 up	vexRT[Btn8U]
8D	Button 8 down	vexRT[Btn8D]
8R	Button 8 right	vexRT[Btn8R]
8L	Button 8 left	vexRT[Btn8L]

Using the buttons to control the arm motor

- First we need to go to Motors and Sensors setup to configure the arm and claw motor.
- Clawbot
  - Arm: Port 7
  - Claw: Port 6
- Robot -> Motors and Sensors setup



1) Name and set the claw and arm motors.
2) Reverse the Arm Motor for Virtual Clawbot. Might need to reverse for physical robot also.

3) Click Apply and OK when finished.

Looking at Arm Control using buttons: Pseudo-Code

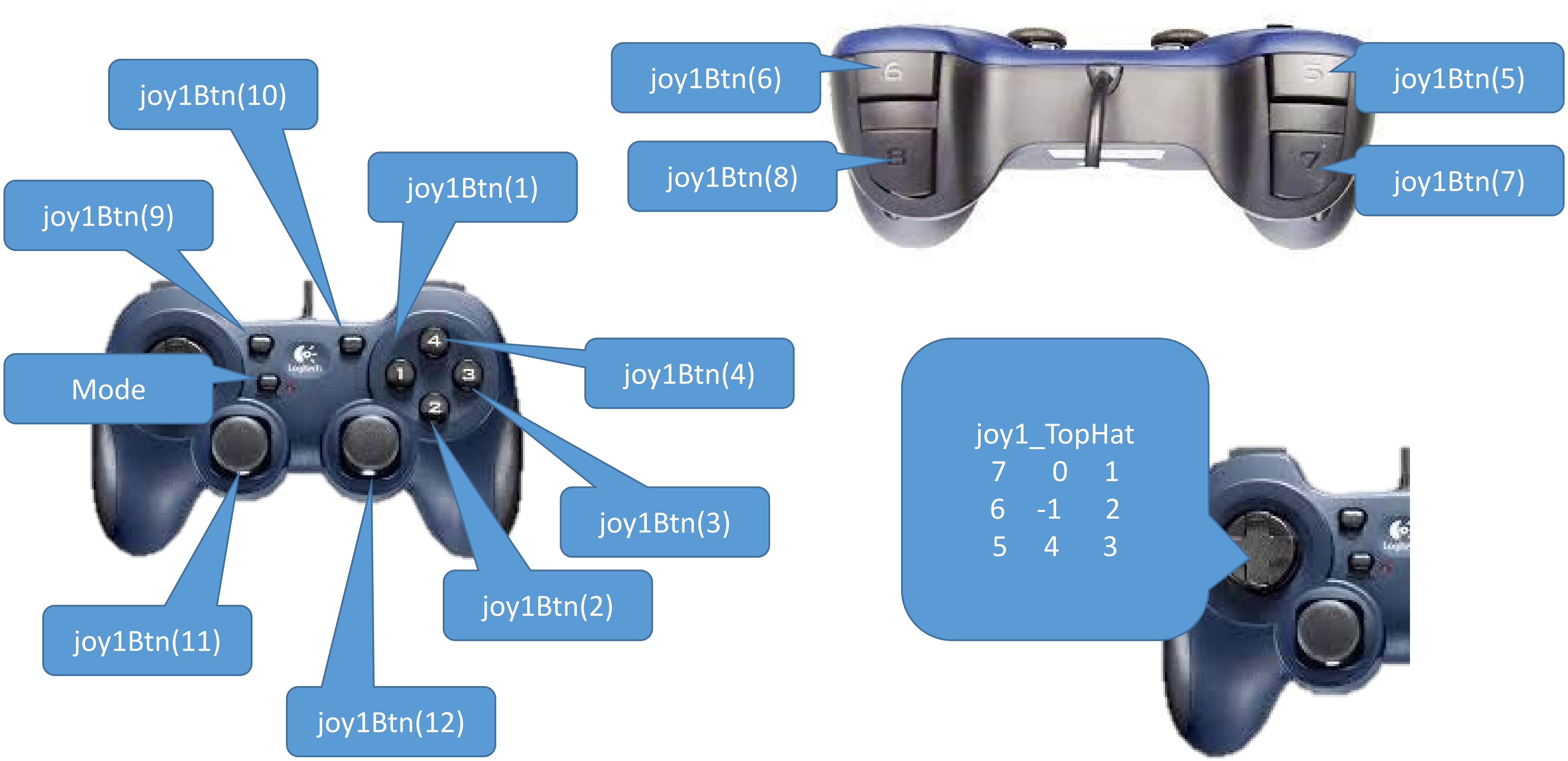
- If button 6U is pushed
  - raise the arm (Send a signal of 127)
- Else if button 6D is pushed
  - Lower the arm (Send a signal of -127)
- Else
  - Stop the arm (Send a signal of 0)

Looking at the Arm: Pseudo-Code to Code

- If button 6U is pushed
  - raise the arm (Send a signal of 127)
- Else if button 6D is pushed
  - Lower the arm (Send a signal of -127)
- Else
  - Stop the arm (Send a signal of 0)

Style Note: Indent between the {} to make the code easier to read.

Virtual World Buttons



Joystick Buttons Virtual World

Buttons return a value of ‘1’ when pushed and ‘0’ when not pushed, except the TopHat.

joy1_TopHat
7 0 1
6 -1 2
5 4 3

Button	Description	Example
1	Left	joy1Btn(1)
2	Bottom	joy1Btn(2)
3	Right	joy1Btn(3)
4	Top	joy1Btn(4)
5	Back, top left	joy1Btn(5)
6	Back, top right	joy1Btn(6)
7	Back, bottom left	joy1Btn(7)
8	Back, bottom right	joy1Btn(8)
9	Small button, top left	joy1Btn(9)
10	Small button, top right	joy1Btn(10)
11	Left joystick button	joy1Btn(11)
12	Right joystick button	joy1Btn(12)
TopHat	Returns values -1 (Not pushed) or 0, 1, … 7 depending on which part is pushed.	joystick.joy1_TopHat

Back to the Arm Movement Pseudo-Code but for Virtual Remote

- If button 6 is pushed
  - raise the arm (Send a signal of 127)
- Else if button 8 is pushed
  - Lower the arm (Send a signal of -127)
- Else
  - Stop the arm (Send a signal of 0)

Arm Pseudo-Code to Code: Virtual World

- If button 6 is pushed
  - raise the arm (Send a signal of 127)
- Else if button 8 is pushed
  - Lower the arm (Send a signal of -127)
- Else
  - Stop the arm (Send a signal of 0)

```
if(joy1Btn(6) == 1)
{
  motor[armMotor] = 127;
}
else if(joy1Btn(8) == 1)
{
  motor[armMotor] = -127;
}
else
{
  motor[armMotor] = 0;
}
```

Where does this code go?

Since you want the robot to continually check for the buttons being pressed, it needs to go inside the while(true) loop.

```
//+++++++++++++++++++++++++++++++++++++++++++++| MAIN |+++++++++++++++++
task main ()
{
  int threshold = 20;

  while(true)
  {
    getJoystickSettings(joystick);
    if(abs(joystick.joy1_y1) < threshold)
    {
      motor[leftMotor]  = 0 ;
    }
    else
    {
      motor[leftMotor]  = joystick.joy1_y1 / 2;
    }

    if(abs(joystick.joy1_y2) < threshold)
    {
      motor[rightMotor] = 0;
    }
    else
    {
      motor[rightMotor] = joystick.joy1_y2 / 2;
    }
        //Arm Control
    if(joy1Btn(6) == 1)       //If button 6 is pressed...
    {
      motor[armMotor] = 127;      //...raise the arm.
    }
    else if(joy1Btn(8) == 1)  //Else, if button 8 is pressed...
    {
      motor[armMotor] = -127;     //...lower the arm.
    }
    else                      //Else (neither button is pressed)...
    {
      motor[armMotor] = 0;       //...stop the arm.
    }
  }
}
```

Claw Motor (For clawbot)

- Pseudo Code
  - If the back, top, left button is pushed
    - Close the claw (127)
  - Else if the back-bottom-left button is pushed
    - Open the claw (-127)
  - Else
    - Leave the claw (0)

Virtual

```
//Claw Control
if(joy1Btn(5) == 1)
{
  motor[clawMotor] = 127;
}
else if(joy1Btn(7) == 1)
{
  motor[clawMotor] = -127;
}
else
{
  motor[clawMotor] = 0;
}
```

Physical

```
// Open-Close Claw
if(vexRT[Btn5U] == 1)
{
  motor[clawMotor] = 127;
}
else if(vexRT[Btn5D] == 1)
{
  motor[clawMotor] = -127;
}
else
{
  motor[clawMotor] = 0;
}
```