Topics

- Remote Control
  - Overview
  - Programming
  - Operator Assist
- Competition Programming

Remote Control Overview

V5 Wireless Controller (276-4820)

- The Wireless Controller allows you to remotely operate your robot.
  - Built-in LCD for real-time information
  - Supports VEXnet 3.0 and Bluetooth®
  - Integrated rechargeable battery
    - Charging takes ~1 Hour
    - Runs up to 10 hours without recharge

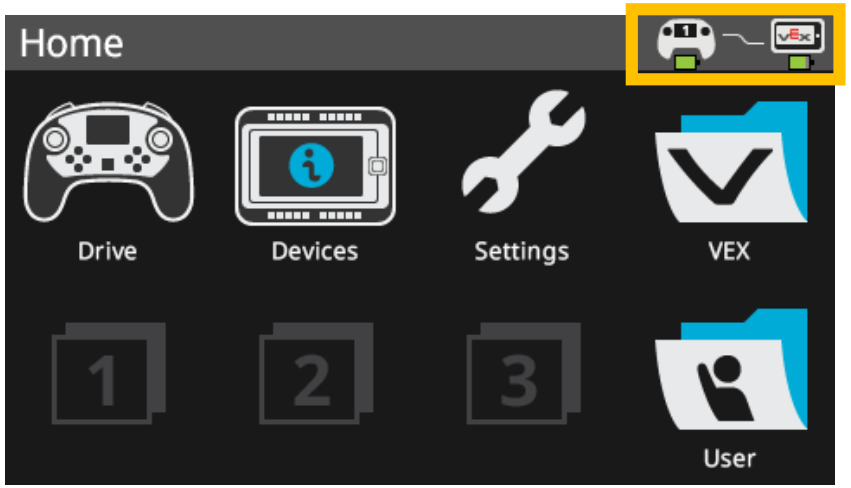
V5 Robot Radio (276-4830)

- The Robot Radio is the connection point between the Robot Brain and Wireless Controller.
  - VEXnet 3.0 supports 500 simultaneous robot channels
  - Supports Bluetooth®
  - LED indicator for linked, scanning and active modes

Pairing the Robot Brain and Wireless Controller – Part 1

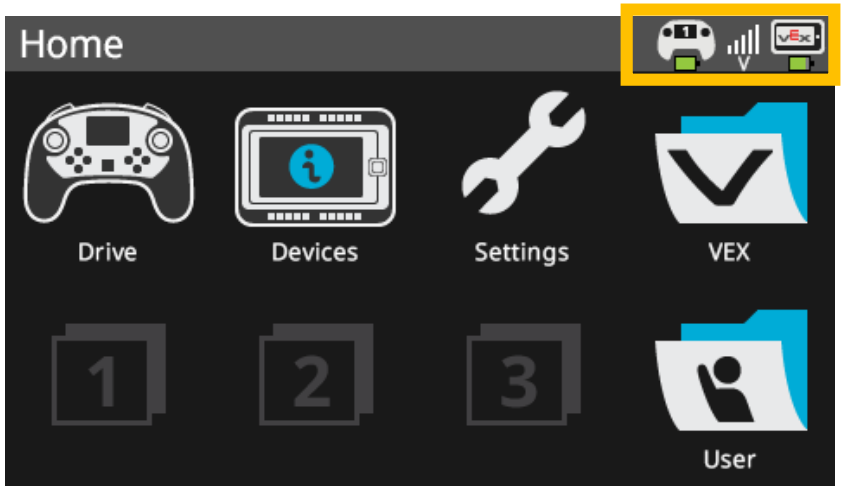
- Before programming or driving with the Wireless Controller, you must “pair” the Robot Brain with the Wireless Controller.

1. Connect the Wireless Controller and Robot Radio to the Robot Brain using Smart Cables.
2. Power on the Robot Brain.
3. Ensure that a “wired” connection has been created by checking the upper-right corner of the Robot Brain’s screen.



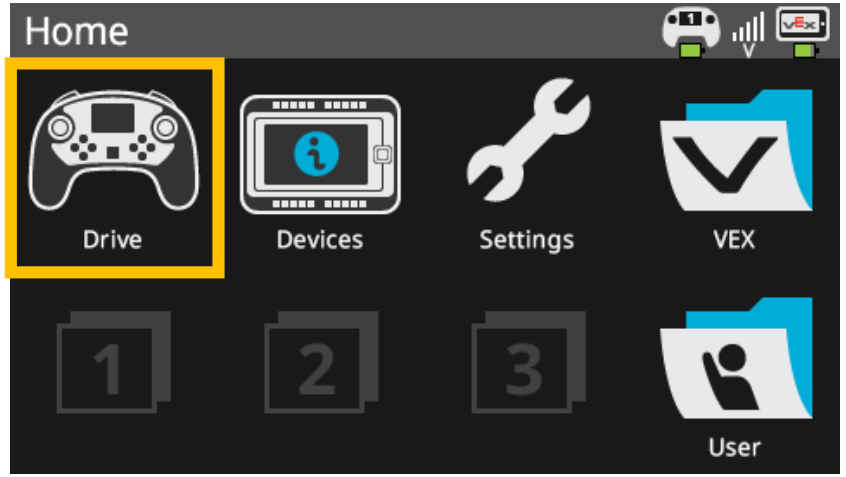
Pairing the Robot Brain and Wireless Controller – Part 1

1. Disconnect only the Wireless Controller from the Robot Brain. A searching **….** symbol will appear on the Robot Brain and Wireless Controller.
2. After a few seconds, the searching symbol should be replaced with signal bars once the devices have linked.



Using the Built-in **Drive** Program

- The V5 Robot Brain includes a built-in **Drive** program that enables flexible, out-of-the-box Remote Control – no programming necessary.
- This is good for early engagement and testing of robot mechanisms, but falls short in a few areas:
  - Meaningful Sensor Integration
  - VRC Autonomous Period



Remote Control Programming

Programming – **Axis#.position()**

```
Controller1.Axis2.position(percentUnits::pct)
```

- The Axis#.position() command allows you to access the values from the X and Y axis on the Left and Right Joystick.
- The number corresponds to the axis label on the physical Wireless Controller.

Programming – Tank Control, Direct Mapping

```
// define your global instances of motors and other devices here
motor LeftMotor(PORT1 );
motor RightMotor(PORT10, true );
controller Controller1 = vex::controller();

int main()
{
        Brain.Screen.print("Tank Control Program Started");
        Controller1.Screen.clearScreen();
        while(true)
        {
            //Drive Control
            //Set the left and right motor to spin forward using the controller's Axis positions as the velocity value.
            LeftMotor.spin(directionType::fwd, Controller1.Axis3.position(), velocityUnits::pct);
            Controller1.Screen.setCursor(1, 1); Controller1.Screen.print("Left Axis 3 %5d",Controller1.Axis3.position());
            RightMotor.spin(directionType::fwd, Controller1.Axis2.position(), velocityUnits::pct);
            Controller1.Screen.setCursor(2, 1); Controller1.Screen.print("Right Axis 2 %5d",Controller1.Axis2.position());
            task::sleep(20);
        }
}
```

Code Break: Enter and test the following.

The **Controller1.Screen…** commands are to show the values of the joysticks on the remote and are not necessary for programming the remote.

Programming – Tank Control, Indirect Mapping

- Adjusts the values from the Joysticks before providing them to the Motors.
- Allows for things such as lower top speeds, finer control, etc.
- In the example below, ½ of the joystick reading is sent to the motor

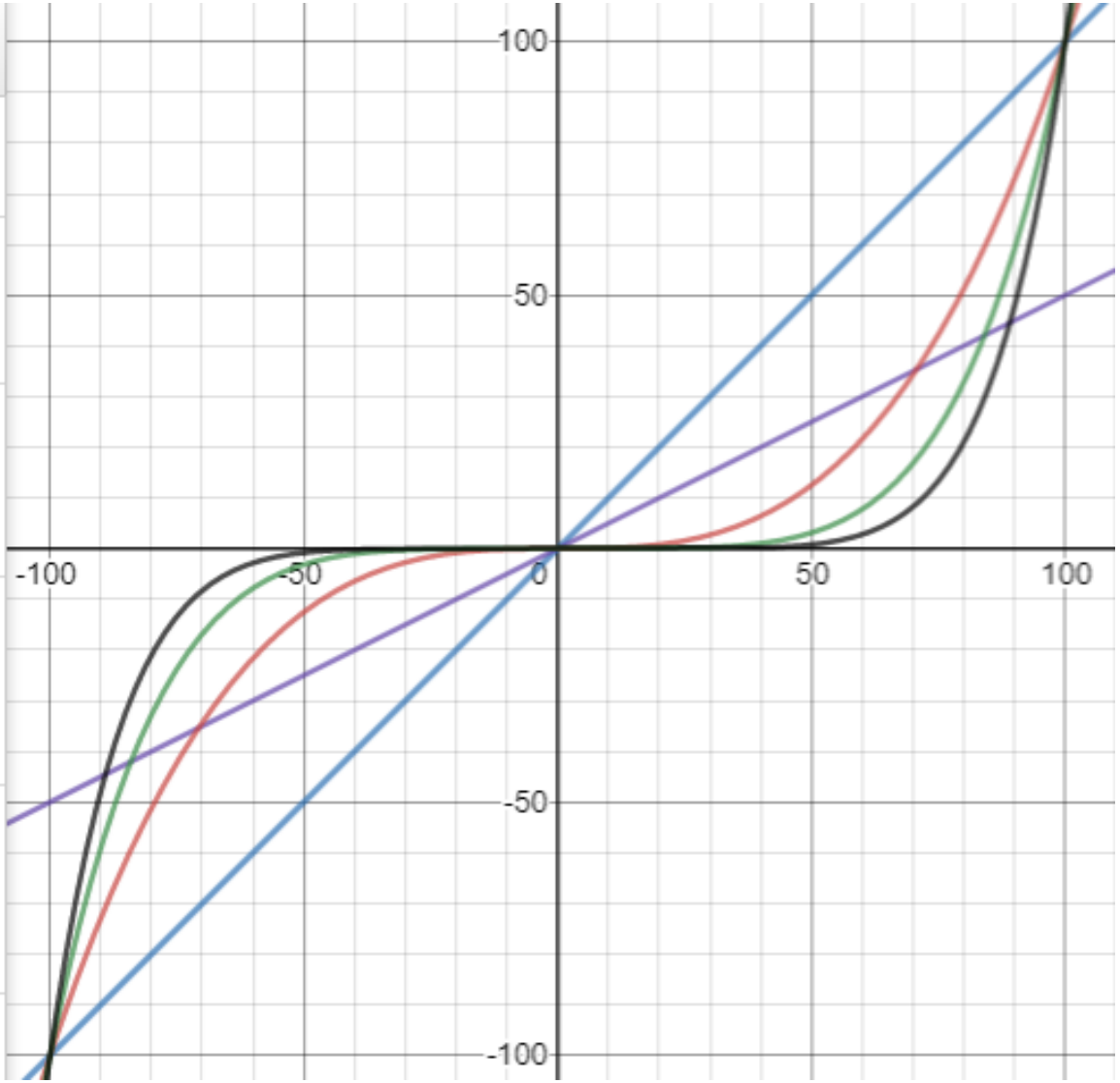
```
// define your global instances of motors and other devices here
motor LeftMotor(PORT1 );
motor RightMotor(PORT10, true );
controller Controller1 = vex::controller();

int main()
{
        while(true)
        {
            LeftMotor.spin(directionType::fwd, Controller1.Axis3.position()/2, velocityUnits::pct);
            RightMotor.spin(directionType::fwd, Controller1.Axis2.position()/2, velocityUnits::pct);
            task::sleep(20);
        }
}
```

Improved control at low speeds, but it also lowers the top speed.

Being able to use math to modify the values sent to the motors allows you to customize the controls to fit the driver.

Possible Equations for Mapping the Drive

1. $y = x$
2. $y = \frac{x}{2}$
3. $y = \frac{x^3}{100^2}$
4. $y = \frac{(x^5)}{(100^4)}$
5. $y = \frac{(x^7)}{(100^6)}$
6.



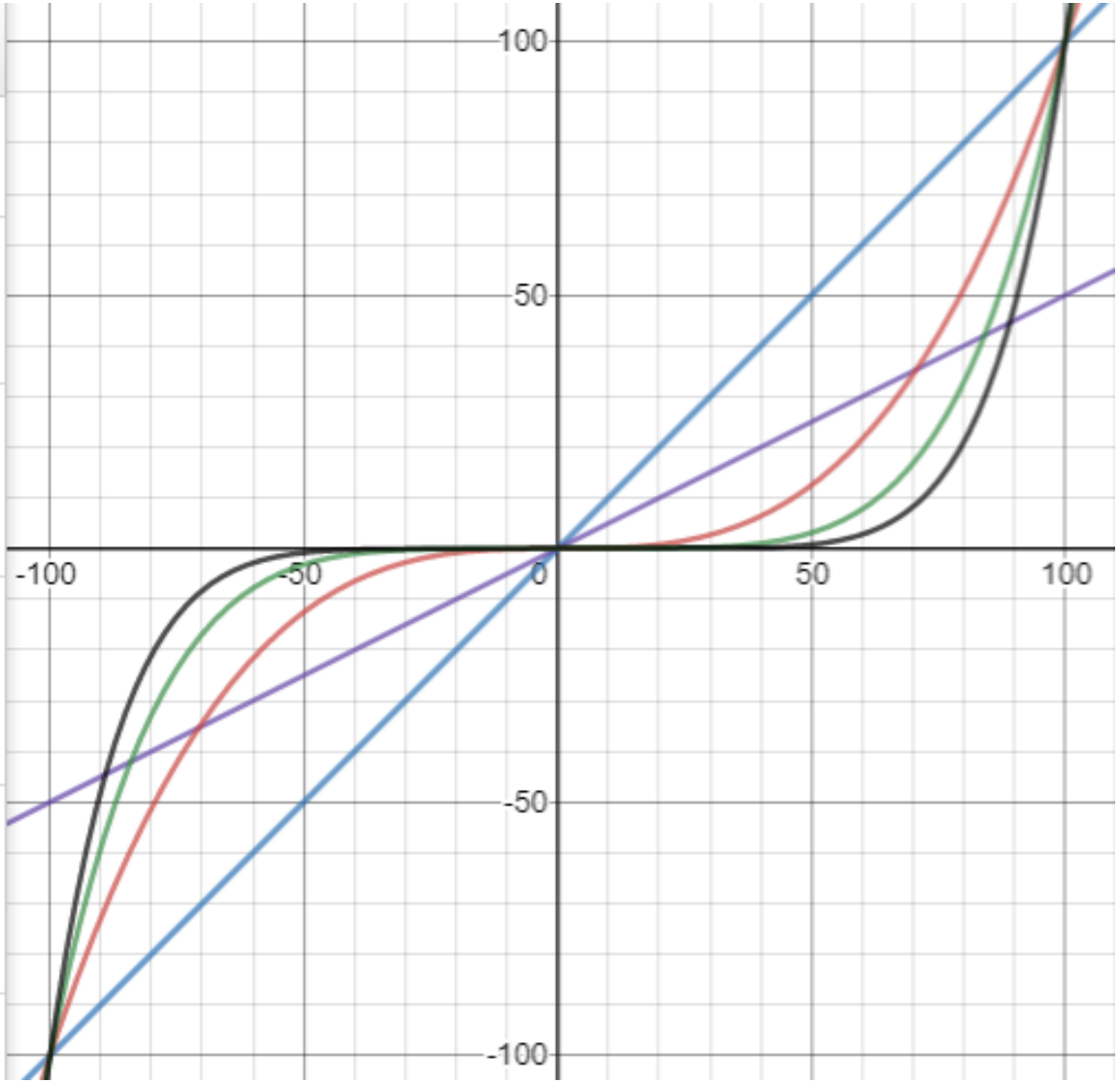
Math Tools to Help with Indirect Mapping

Idea	Math	VEXcode
Powers	y = x^n	y = power(x,b);
Absolute Value	y = |x|;	y = abs(x);
Square Root	y = √x	y = sqrt(x);

Note: Keep track of how big the values get when evaluating the math. Integers only go to about +/- 2 billion.
If you go beyond the largest integer is just cycles back to the –largest.
Counting
2,147,483,646 then 2,147,483,647 then -2,147,483,648 then -2,147,483,647

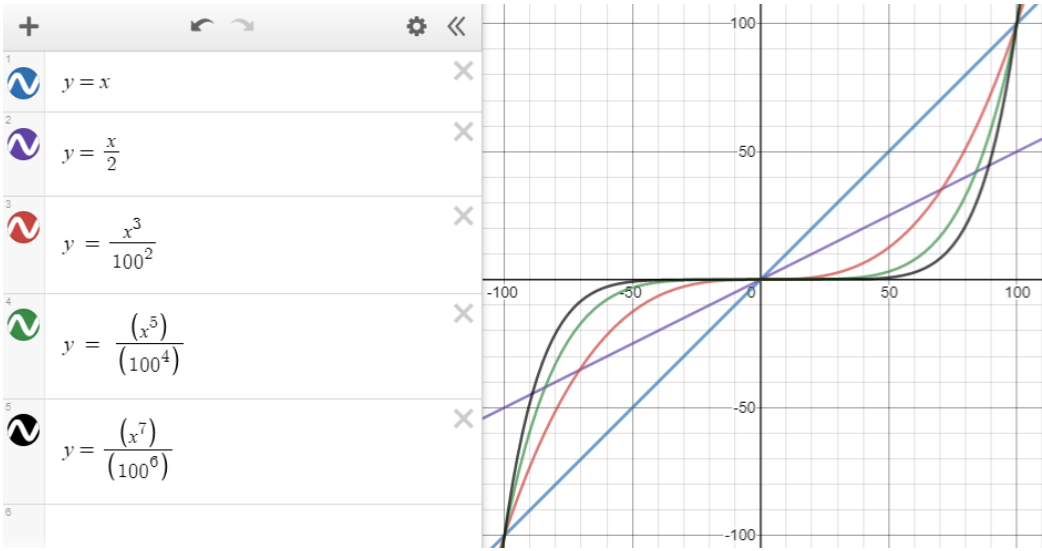
Possible Equations for Mapping the Drive

1. $y = x$
2. $y = \frac{x}{2}$
3. $y = \frac{x^3}{100^2}$
4. $y = \frac{(x^5)}{(100^4)}$
5. $y = \frac{(x^7)}{(100^6)}$
6.



Implementing motor = remote^3 / 100^2 with output to the controller's screen

Code Break. Enter and watch the values

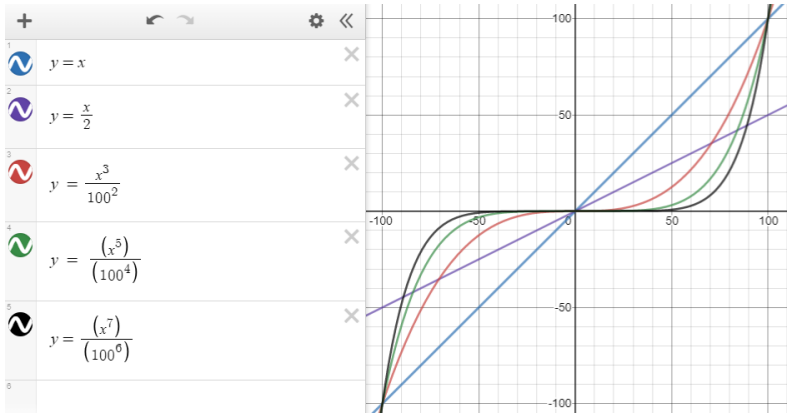



```
int main()
{
        // Display that the program has started to the screen.
        Brain.Screen.print("Tank Control Program Started");
        Controller1.Screen.clearScreen();
        int leftPower = 0; //Used to calculate the value sent to the left motor
        int rightPower = 0; //Used to calculate the value sent to the right motor
        while(true)
        {
                leftPower = Controller1.Axis3.position();
                leftPower = pow(leftPower, 3)/pow(100,2); // y = x^3/100^2
                rightPower = Controller1.Axis2.position();
                rightPower = pow(rightPower, 3)/pow(100,2); // y = x^3/100^2
                LeftMotor.spin(directionType::fwd, leftPower, velocityUnits::pct);
                Controller1.Screen.setCursor(1, 1); Controller1.Screen.print("Left Axis 3 %5d",leftPower);
                RightMotor.spin(directionType::fwd, rightPower, velocityUnits::pct);
                Controller1.Screen.setCursor(2, 1); Controller1.Screen.print("Right Axis 2 %5d", rightPower);
                task::sleep(20);
        }
}
```

Range still works since 100^3 will be the largest value calculated and 100^3 = 1000000 while is less than 2 billion

Implementing motor = remote^5 / 100^4 with output to the controller's screen

Code Break. Save as, modify and watch values.

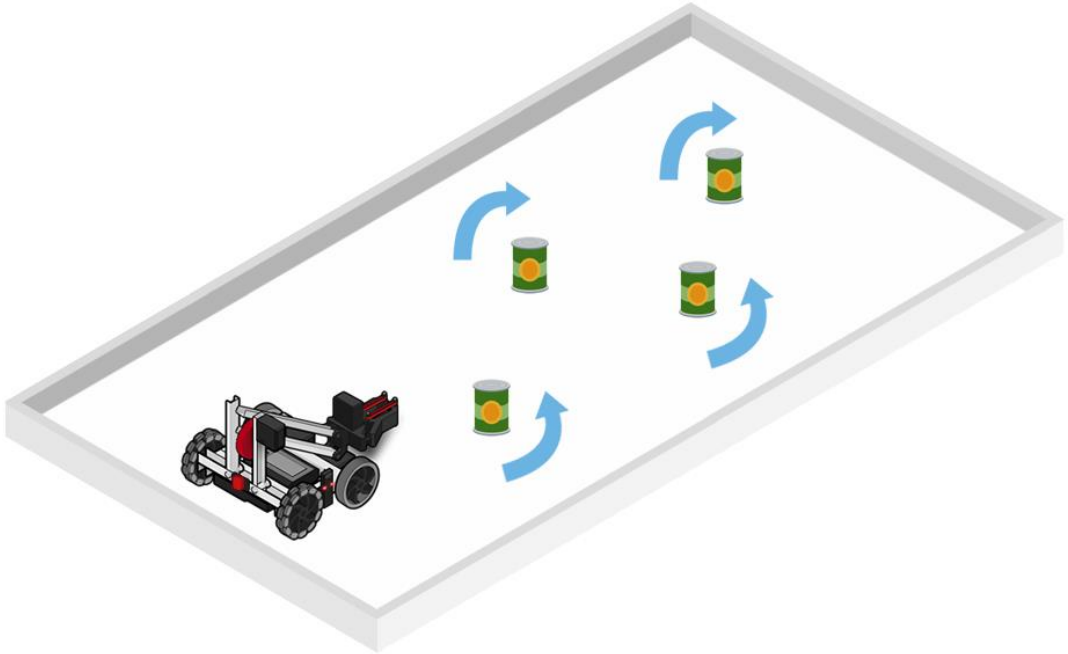



```
int main()
{
        Controller1.Screen.clearScreen();
        int leftPower = 0;
        int rightPower = 0;
        while(true)
         {
            leftPower = Controller1.Axis3.position();
             // y = (x^3/100^2)*(x^2/100^2) = x^5/100^4
            leftPower = pow(leftPower, 3)/pow(100,2)*pow(leftPower,2)/pow(100,2);

            rightPower = Controller1.Axis2.position();
            // y = (x^3/100^2)*(x^2/100^2) = x^5/100^4
            rightPower = pow(rightPower, 3)/pow(100,2)*pow(rightPower,2)/pow(100,2);

            LeftMotor.spin(directionType::fwd, leftPower, velocityUnits::pct);
            Controller1.Screen.setCursor(1, 1); Controller1.Screen.print("Left Axis 3 %5d",leftPower);

            RightMotor.spin(directionType::fwd, rightPower, velocityUnits::pct);
            Controller1.Screen.setCursor(2, 1); Controller1.Screen.print("Right Axis 2 %5d", rightPower);
            task::sleep(20);
        }
}
```

Needed to break down the equation since 100^5 > 2 billion

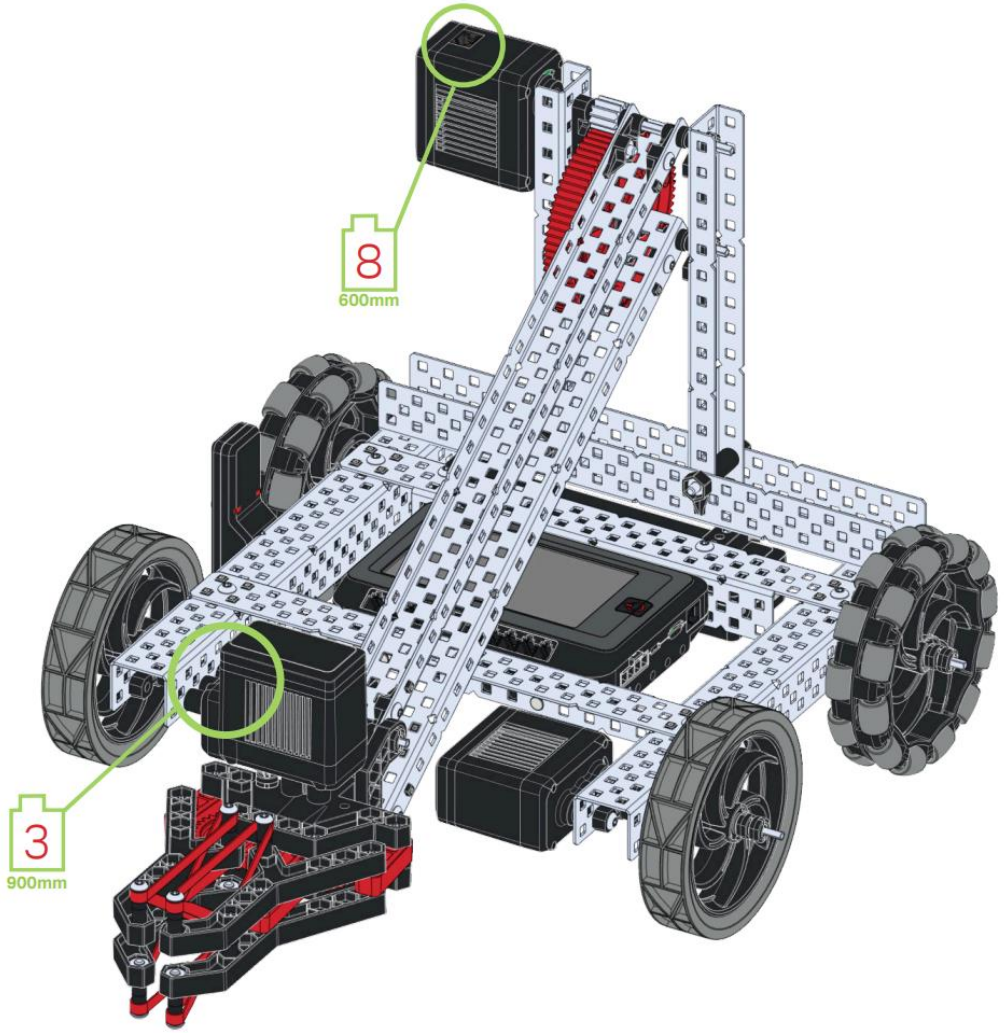
Driving Activity : RoboSlalom Challenge

- Practice driving through a slalom without hitting any of the objects.
- Try different “mappings” to see what best suits your driving style.
- Demo to instructor or classmates.



Robot Config – Arm and Claw Motors

- When programming the Arm and Claw, don’t forget to add their motors to the Robot Config!



Programming – `if() {}` Statements Review

```
if(/*condition*/){
    //commands
}
```

- An **`if() {}`** statement is similar to a **`while() {}`** loop, but it is **not a loop**!
- An **`if() {}`** statement checks the Boolean Condition between its parenthesis…
  - **IF** the condition is **true**, the robot **runs** the commands between the curly braces **once** and then Program Flow moves on in the program.
  - **IF** the condition is **false**, the robot **skips** the commands between the curly braces and Program Flow moves on in the program.

Programming – `if() {}` Statements with `else{}` Branches Review

```
if(/*condition*/){
    //commands
}
else{
    //other commands
}
```

- An `else{}` branch may be appended to an `if() {}` statement.
- The `else{}` branch does not have a Boolean Condition!
- If the condition of the `if() {}` statement is false, the commands that belong to the `else{}` branch are run.
  - One half of an `if-else` statement is **ALWAYS** run by the robot!

Programming – **`Button##.pressing()`**

```
Controller1.
```

- ButtonL1
- ButtonL2
- ButtonR1
- ButtonR2
- ButtonUp
- ButtonDown
- ButtonLeft
- ButtonRight
- ButtonX
- ButtonB
- ButtonY
- ButtonA

ButtonL1 - button&
vex::controller::ButtonL1

A button that represents the L1 button on the controller.

- The **`Button##.pressing()`** command allows you to check if the specified button is pressed (or true).
- The names of the buttons correspond to the labels on the Wireless Controller.

Programming - Raising the Arm

```
4   int main() {
5       while(1 == 1){
6           RightMotor.spin(directionType::fwd, Controller1.Axis2.position(percentUnits::pct), velocityUnits::pct);
7           LeftMotor.spin(directionType::fwd, Controller1.Axis3.position(percentUnits::pct), velocityUnits::pct);
8
9           //If Button R1 is being pressed...
10          if(Controller1.ButtonR1.pressing()){
11              //...raise the arm...
12              ArmMotor.spin(directionType::fwd, 50, velocityUnits::pct);
13          }
14          else{
15              //...else, stop and hold the arm.
16              ArmMotor.stop(brakeType::hold);
17          }
18      }
19  }
```

- Arm Control Code must be **in the same loop** as Driving Control code!
- The **`if-else`** statement is continuously revisited since it is embedded within the loop.

Programming - Arm Control – Bad - Shakey

```
//If Button R1 is being pressed...
if(Controller1.ButtonR1.pressing()){
    //...raise the arm...
    ArmMotor.spin(directionType::fwd, 50, velocityUnits::pct);
}
else{
    //...else stop and hold the arm at its current position.
    ArmMotor.stop(brakeType::hold);
}

//If Button R2 is being pressed...
if(Controller1.ButtonR2.pressing()){
    //...lower the arm...
    ArmMotor.spin(directionType::rev, 30, velocityUnits::pct);
}
else{
    //...else stop and hold the arm at its current position.
    ArmMotor.stop(brakeType::hold);
}
```

- Remember, one half of an **`if-else`** statement is always run!
  - Since these **`if-else`** statements control the same motor, if one set turns it on, the other will turn it off!

Programming - Arm Control - BETTER

```
//If Button R1 is being pressed...
if(Controller1.ButtonR1.pressing()){
    //...raise the arm...
    ArmMotor.spin(directionType::fwd, 50, velocityUnits::pct);
}
else{
    //If Button R2 is being pressed...
    if(Controller1.ButtonR2.pressing()){
        //...lower the arm...
        ArmMotor.spin(directionType::rev, 30, velocityUnits::pct);
    }
    else{
        //...else stop and hold the arm at its current position.
        ArmMotor.stop(brakeType::hold);
    }
}
```

- We can correct this by embedding the entire second **`if-else`** inside of the first else branch!
- Program Flow only reaches the **`.stop()`** if neither button is pressed!

Programming - Arm Control – BETTER, Condensed

```
//If Button R1 is being pressed...
if(Controller1.ButtonR1.pressing()){
    //...raise the arm...
    ArmMotor.spin(directionType::fwd, 50, velocityUnits::pct);
}
else if(Controller1.ButtonR2.pressing(){ //...else, if Button R2 is being pressed...
    //...lower the arm...
    ArmMotor.spin(directionType::rev, 30, velocityUnits::pct);
}
else{
    //...else stop and hold the arm at its current position.
    ArmMotor.stop(brakeType::hold);
}
```

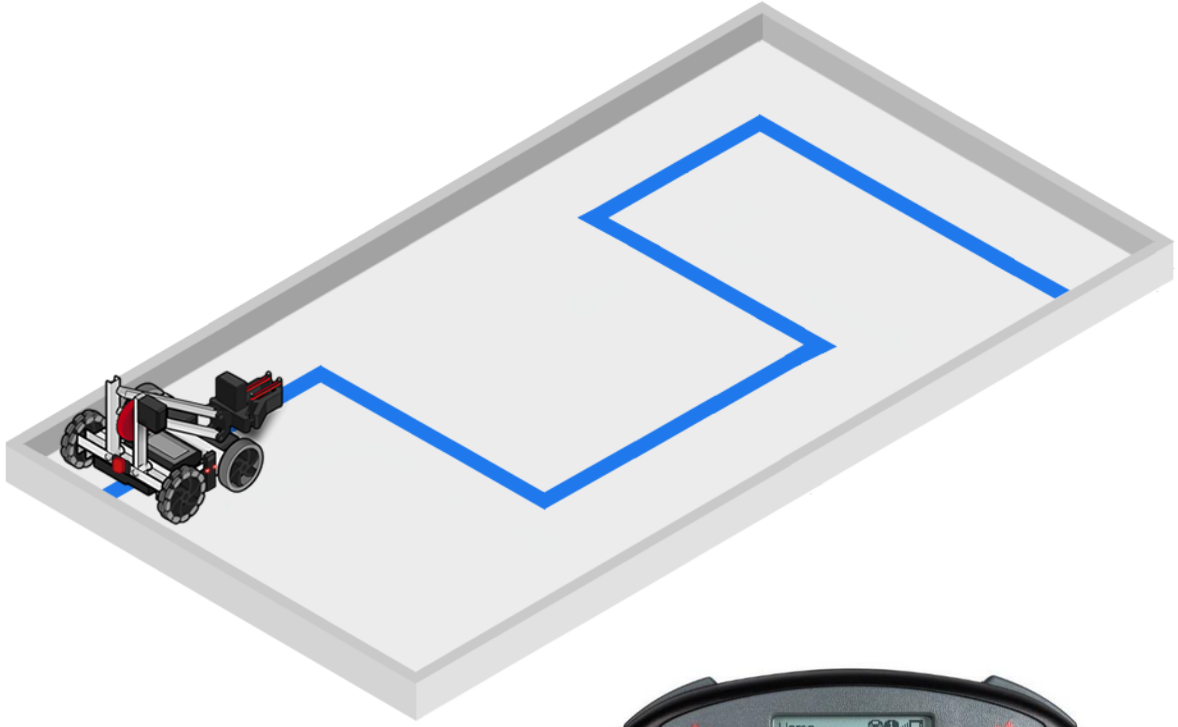
- This code can be condensed by taking advantage of a “trick” in C++.
  - If no curly braces are provided for a structure, that structure “owns” the next line of code or structure that appears.
  - We can eliminate the curly braces belonging to the first else branch to represent the code in this manner.

Full Remote Control

```
int main() {
    while(1 == 1){
        //Set the velocity of the Right Motor equal to the value of the Right Joystick Vertical Axis
        RightMotor.spin(directionType::fwd, Controller1.Axis2.position(percentUnits::pct), velocityUnits::pct);
        //Set the velocity of the Left Motor equal to the value of the Left Joystick Vertical Axis
        LeftMotor.spin(directionType::fwd, Controller1.Axis3.position(percentUnits::pct), velocityUnits::pct);

        //If Button R1 is being pressed...
        if(Controller1.ButtonR1.pressing()){
            //...raise the arm...
            ArmMotor.spin(directionType::fwd, 50, velocityUnits::pct);
        }
        else if(Controller1.ButtonR2.pressing(){ //...else, if Button R2 is being pressed...
            //...lower the arm...
            ArmMotor.spin(directionType::rev, 30, velocityUnits::pct);
        }
        else{
            //...else stop and hold the arm at its current position.
            ArmMotor.stop(brakeType::hold);
        }

         //If Button L1 is being pressed...
        if(Controller1.ButtonL1.pressing()){
            //...open the claw...
            ClawMotor.spin(directionType::fwd, 30, velocityUnits::pct);
        }
        else if(Controller1.ButtonL2.pressing(){ //...else, if Button L2 is being pressed...
            //...close the claw...
            ClawMotor.spin(directionType::rev, 30, velocityUnits::pct);
        }
        else{
            //...else stop and hold the claw at its current position.
            ClawMotor.stop(brakeType::hold);
        }
    }
}
```

Remote Control Activity: Complete the Button Control Challenge!

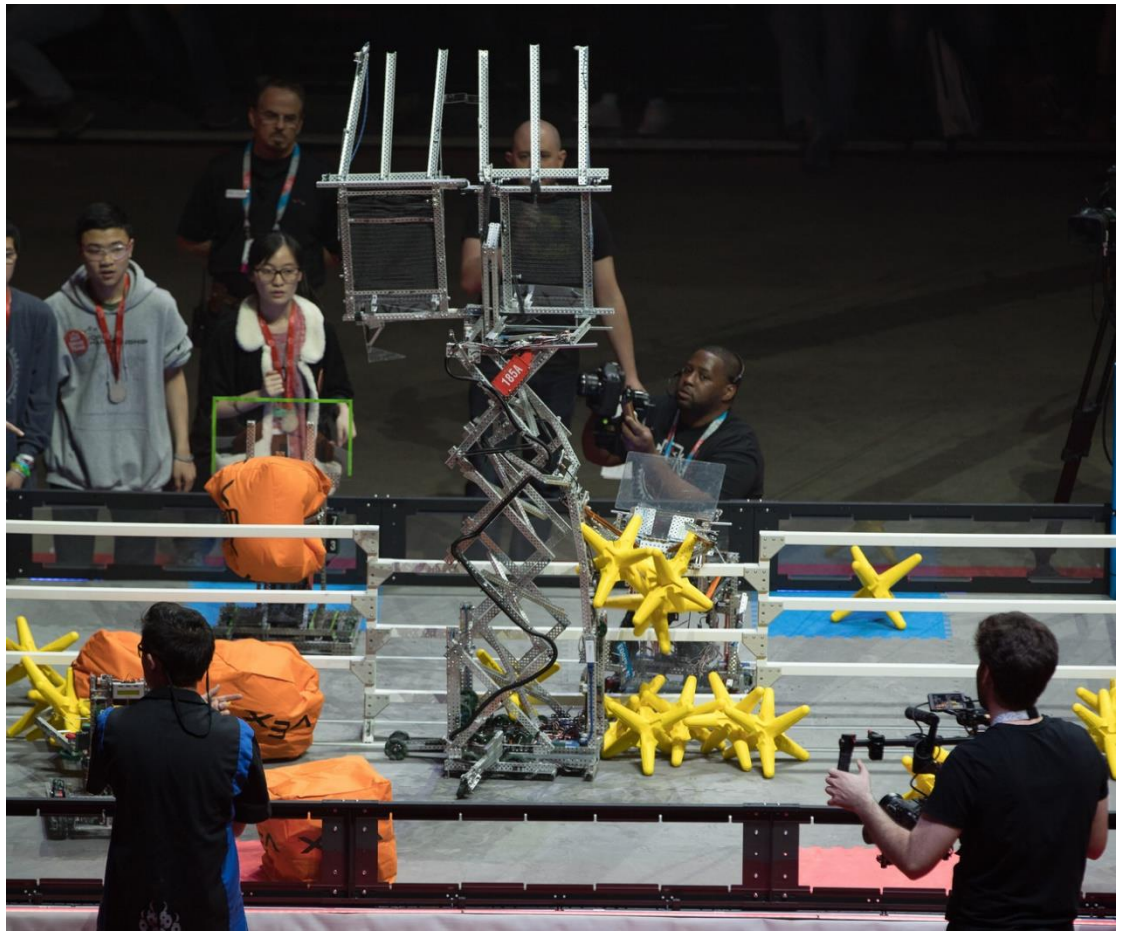
- Program the robot to move forward, backward, right, and left based on input only from the directional buttons on the Wireless Controller.
  - For example: “If the Up Button is pressed, move forward.”
- Drive your robot along a predetermined course based on your program.
- Don’t forget to Pseudocode how you will structure your program!



Operator Assist
Incorporating Sensors into Driving

Operator Assist

- Sensors are often thought of as tools for Autonomous only, but they can give incredible competitive advantage when blended with Remote Control!
- Factors such as limited visibility and mechanically complex control systems make the robot better suited to complete certain tasks.



Sensor Overview – Bumper Switch v2

- The Bumper Switch v2 reports to the Robot Brain is whether it is pressed in, versus released.
- The Bumper Switch v2 allows the robot to sense physical collisions:
  - Robot body bumps into an obstacle
  - Robot arm reaches a mechanical limit
- This sensor connects to 1 of the 8 Tri-Ports on the V5 Robot Brain.
- The sensor “cap” may be disconnected and other components may be used to create a more ideal “hit area”.

Connecting a Bumper Switch

- For these next activities, we recommend connecting:
  1. 1 Bumper Switch on the top-front of the chassis, so that the arm makes contact

Logical Operators

- The robot is able to make a decision based on multiple Boolean Conditions.
- Multiple Boolean Conditions may be combined in code using the **AND** and **OR** Logical Operators.
- The below "truth table" shows how a condition containing multiple Boolean Conditions is evaluated based on the individual conditions and Logical Operator:

Operator Syntax	Operator Meaning
&&	AND
||	OR
!	NOT

Condition 1	Condition 2	&& (AND)	|| (OR)
`false`	`false`	`false`	`false`
`true`	`false`	`false`	`true`
`false`	`true`	`false`	`true`
`true`	`true`	`true`	`true`

Operator Assist – Mechanical Stop

```
//If Button R1 is being pressed...
if(Controller1.ButtonR1.pressing()){
    //...raise the arm...
    ArmMotor.spin(directionType::fwd, 50, velocityUnits::pct);
}
//...else, if Button R2 is being pressed AND the Arm Bumper is NOT being pressed...
else if(Controller1.ButtonR2.pressing() && !ArmBumper.pressing()){
    //...lower the arm...
    ArmMotor.spin(directionType::rev, 30, velocityUnits::pct);
}
else{
    //...else stop and hold the arm at its current position.
    ArmMotor.stop(brakeType::hold);
}
```

- The Bumper Switch detects when the arm reaches the lowest point and prevents the motor from continuing to spin downward.
  - This is very helpful for reducing wear and tear and wasting energy.

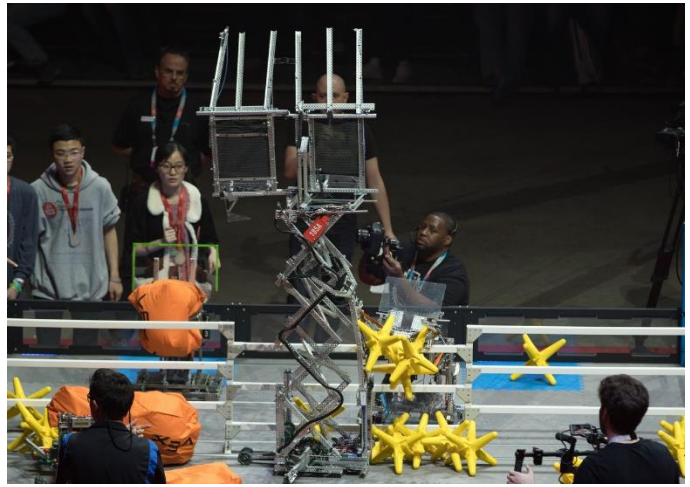
Operator Assist – Ideal Angle

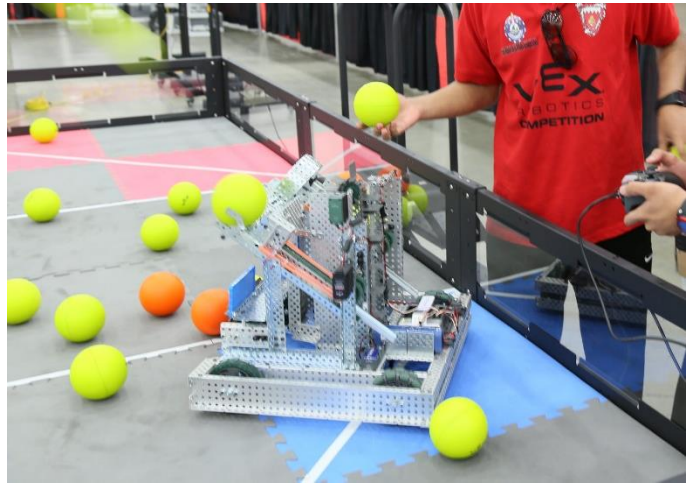
```
//If Button R1 is being pressed AND the Arm Motor Encoder has spun less than the max desired value...
if(Controller1.ButtonR1.pressing() && ArmMotor.rotation(rotationUnits::deg)<270){
    //...raise the arm...
    ArmMotor.spin(directionType::fwd, 50, velocityUnits::pct);
}
//...else, if Button R2 is being pressed AND the Arm Bumper is NOT being pressed...
else if(Controller1.ButtonR2.pressing() && !ArmBumper.pressing()){
    //...lower the arm...
    ArmMotor.spin(directionType::rev, 30, velocityUnits::pct);
}
else{
    //...else stop and hold the arm at its current position.
    ArmMotor.stop(brakeType::hold);
}
```

- The Motor Encoder detects when it has rotated to/past the desired amount and prevents the motor from continuing to spin upward.
  - This is very helpful for raising the arm to a specific position within its range of motion.
  - Note: The angle of the arm is affected by the value and arm gear train.

Operator Assist – Additional Ideas

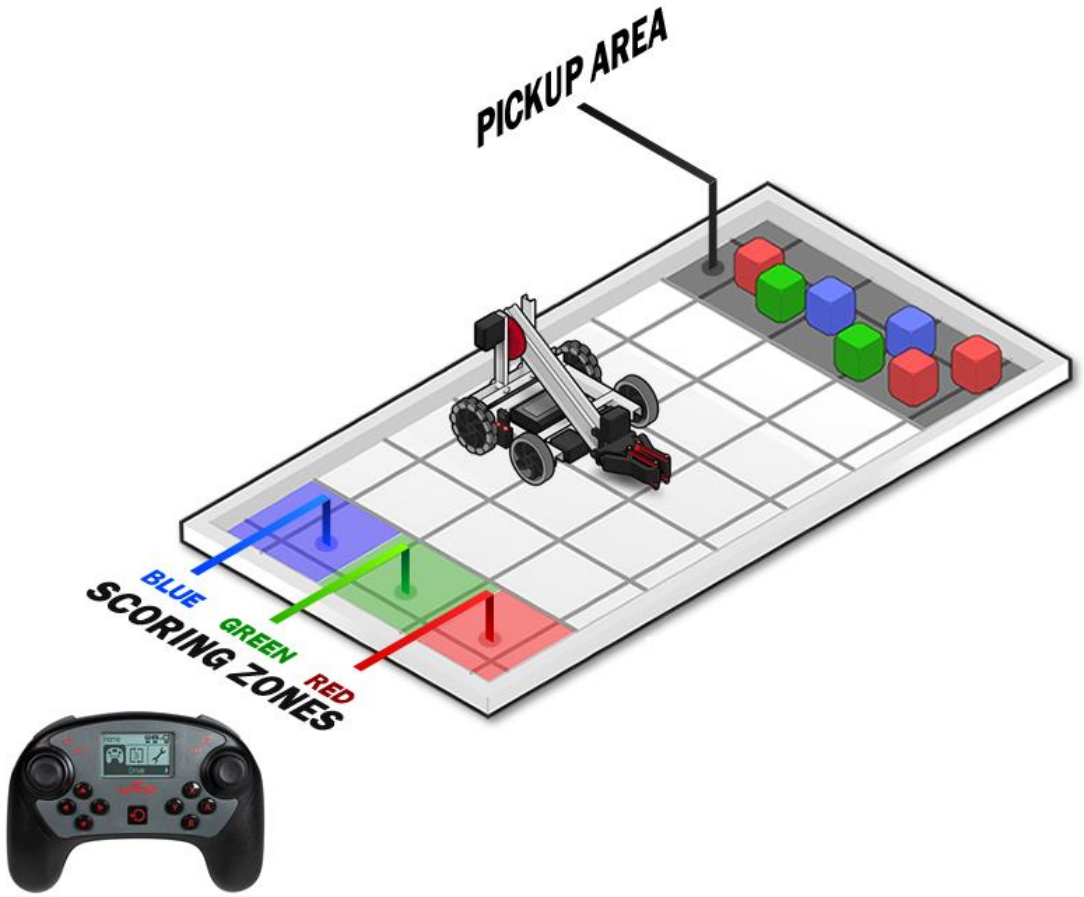
- Automatic Pickup
  - Drive forward to an object with the Ultrasonic Rangefinder or Vision Sensor and automatically pick it up, assigned to a Controller Button press.
- Maintain Equilibrium
  - In actuators where you have more than one motor, you can place a sensor on each (or use the built in encoders) to ensure that they are moving at the same rate.
- Pre-set Positions / Angles
  - If you are engineering a robot that needs to reach goals or hit targets at different heights/from different distances, you can program the Controller Buttons to have the actuator go to desirable presets.





Try the Operator Assist Challenge!

- Use remote control to sort 3 different colors of game objects from one side of a field to another.
- The human operator must stay on the side of the field with the scoring zones.
- See how many game objects you can score in 60 seconds!
- Customize the value mappings and operator assists in your code to suit your driving style!



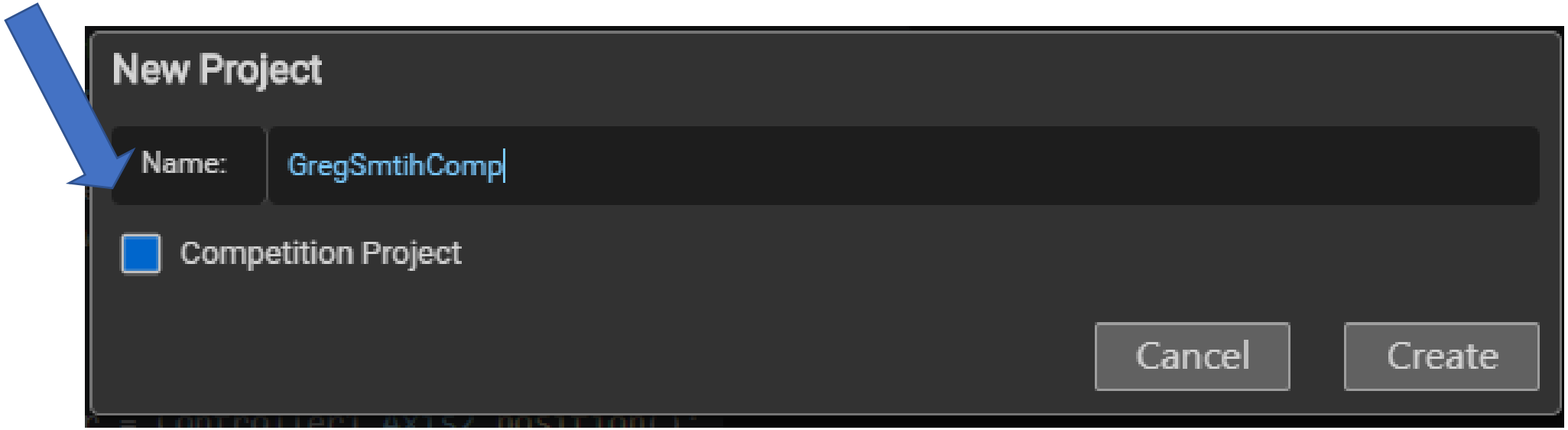
Competition Template

Field Control System

- At an official VEX Robotics Competition, teams must connect their Remote Controls to the Field Control System during matches.
- The Field Control System ensures that all teams start and stop their Autonomous and Driver Control periods at the same time.
- It also allows referees to disable a malfunctioning or misbehaving robot.
- For a robot to correctly respond to the Field Control System, it needs to be programmed using the Competition Template in VCS.

Using the Competition Template

- To write your own Autonomous and Driver Control code for the competition, make sure to program in the Competition Template!
- The Competition Template ensures that your robot is compliant to the VEX Field Control System.
- To start…
  - File
  - New
  - Check the Competition Project checkbox



Setup the motors, sensors, controller

- Need to tell VEXcode what you have on your robot.
- Place the configuration code before task main().
- Recommend putting it near where the vex::brain is configured.

```
/*
/*----------------------------------------------------------------------------
#include "vex.h"

using namespace vex;

// A global instance of vex::brain used for printing to the V5 brain screen
vex::brain       Brain;
controller Controller1 = vex::controller();
motor LeftMotor(PORT1 );
motor RightMotor(PORT10, true );
limit Limit1 = limit( Brain.ThreeWirePort.A );
// A global instance of vex::competition
vex::competition Competition;

// define your global instances of motors and other devices here
```

Pre-Autonomous

- The Pre-Autonomous (sometimes called pre-auton) portion runs before the Autonomous routine in a match.
- This section is perfect for:
  - Resetting Encoders
  - Calibrating the Gyro Sensor
  - Routine Selection (Advanced)
  - Initial positioning/orientation of the robot (still within legal dimensions)
- Do NOT use this section to move or drive the robot!

```
/*---------------------------------------------------------------------------*/
/*                          Pre-Autonomous Functions                         */
/*                                                                           */
/*  You may want to perform some actions before the competition starts.      */
/*  Do them in the following function.  You must return from this function   */
/*  or the autonomous and usercontrol tasks will not be started.  This       */
/*  function is only called once after the cortex has been powered on and    */
/*  not every time that the robot is disabled.                               */
/*---------------------------------------------------------------------------*/

void pre_auton( void ) {
  // All activities that occur before the competition starts
  // Example: clearing encoders, setting servo positions, ...

}
```

Autonomous

- Autonomous (sometimes called auton) portion runs at the beginning of a match, before the Driver Control portion.
- This section is perfect for:
  - Sensing game objects
  - Scoring game objects autonomously
  - Strategically positioning the robot before Driver Control
- Do NOT use this section to drive the robot using remote control signals!
- Copy and paste autonomous code in this section.

```
/*----------------------------------------------------------------------------*/
/*                                                                            */
/*                              Autonomous Task                               */
/*                                                                            */
/*  This task is used to control your robot during the autonomous phase of    */
/*  a VEX Competition.                                                        */
/*                                                                            */
/*  You must modify the code to add your own robot specific commands here.    */
/*----------------------------------------------------------------------------*/

void autonomous( void ) {
  // ..........................................
  // Insert autonomous user code here.
  // ..........................................

}
```

Place autonomous code here.

Example where the robot drives forward for 3 revolutions.

```
void autonomous( void ) {
  // ..........................................
  // Insert autonomous user code here.
  // ..........................................
  LeftMotor.rotateFor(3, rotationUnits::rev, 50, velocityUnits::pct, false);
  RightMotor.rotateFor(3, rotationUnits::rev, 50, velocityUnits::pct);

}
```

Testing your Autonomous code and/or running programming Skills from your remote

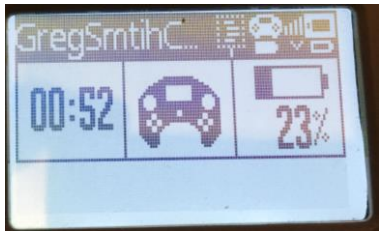
From the Computer.

1. Put your autonomous code into the void **autonomous(void) {}** section of the Competition Template.
2. Download the program to the vex brain.
3. Disconnect from the computer.

Testing your Autonomous code and/or running programming Skills from your remote (Continued)

From the Remote

1. Use the < > keys to go to **Programs** and click **'A'** to select
2. Use the < > keys to go to **Competition** and click '**A'** to select
3. Use the < > keys to go to **Programming Skills** and click **'A'** to select.
4. Click **'A'** to start the countdown.
    1. The screen will count down from '3' and run the code from the autonomous section of you Competition Template program
5. To **exit** early hit the **power button** on the remote.



Driver Control

- The Driver Control (sometimes called User Control or Tele-Op) portion runs after the Autonomous routine in a match.
- Copy and paste your driving code inside the while () {} loop.
- Any commands to be repeated should be kept within the same `while(1){}` loop.

```
void usercontrol( void ) {
  // User control code here, inside the loop
  while (1) {
    // This is the main execution loop for the user control program.
    // Each time through the loop your program should update motor + servo
    // values based on feedback from the joysticks.

    // ..........................................................................
    // Insert user code here. This is where you use the joystick values to
    // update your motors, etc.
    // ..........................................................................

    vex::task::sleep(20); //Sleep the task for a short amount of time to prevent wasted resources.
  }
}
```

Place your driving code in this section.

Example driving code. No arm or claw.

```
void usercontrol( void ) {
  // User control code here, inside the loop
  while (1) {
    LeftMotor.spin(directionType::fwd, Controller1.Axis3.position(), velocityUnits::pct);
    RightMotor.spin(directionType::fwd, Controller1.Axis2.position(), velocityUnits::pct);
    vex::task::sleep(20); //Sleep the task for a short amount of time to prevent wasted resources.
  }
}
```

Testing your User Control code and/or running Driving Skills from your remote

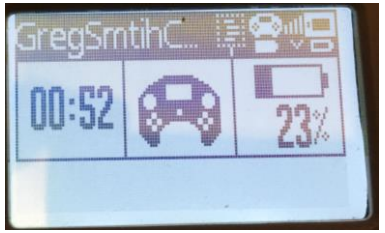
From the Computer.

1. Put your driving code into the void **usercontrol(void) {}** section of the Competition Template.
2. Download the program to the vex brain.
3. Disconnect from the computer.

Testing your User Control code and/or running Driving Skills from your remote(Continued)

From the Remote

1. Use the < > keys to go to **Programs** and click **'A'** to select
2. Use the < > keys to go to **Competition** and click '**A**' to select
3. Use the < > keys to go to **Driver Skills** and click **'A'** to select.
4. Click **'A'** to start the countdown.
    1. The screen will count down from '3' and run the code from the autonomous section of you Competition Template program
5. To **exit** early hit the **power button** on the remote.



DO NOT MODIFY int main() ON THE COMPETITION TEMPLATE!!!

- Generally, the `int main()` section of the Competition Template should not be modified.
- This section:
  - Is where program execution begins
  - Is responsible for initializing the other sections of the template
  - Keeps the entire program running for the duration of the match

```
//
// Main will set up the competition functions and callbacks.
//
int main() {
    //Set up callbacks for autonomous and driver control periods.
    Competition.autonomous( autonomous );
    Competition.drivercontrol( usercontrol );

    //Run the pre-autonomous function.
    pre_auton();

    //Prevent main from exiting with an infinite loop.
    while(1) {
      vex::task::sleep(100);//Sleep the task for a short amount of time to prevent wasted resources.
    }

}
```

Testing with a VEXnet Switch

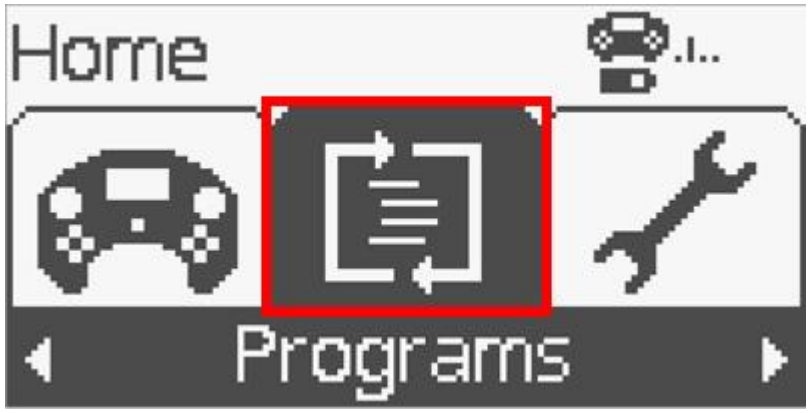
- A VEXnet Competition Switch allows you to verify that your V5 robot will work as expected when you connect to an actual competition field.
- The VEXnet Competition Switch allows you to toggle between Enable/Disable and Driver/Autonomous to ensure that your robot is field compliant.
- **<u>Running the Robot Using the VEXnet Switch</u>**
  - Start with the switch set to ‘DISABLED’ so the program does not start running.
  - Plug the remote into the switch
  - Start running the program from the remote (Programs -> Select Program)
  - Now you use the switch to select DRIVER/AUTONOMOUS and ENABLE/DISABLE.
  - You can connect four robots to this switch and use it to run matches.

Connecting to Field Control for Competition

- When competing in a VEX Robotics Competition, your robot will need to be connected to the Field Control System (FCS).
- The FCS determines when the robot starts, and when it is in Autonomous versus Driver Control mode.
- When you arrive at the competition field:
  1. Position your robot (turned off) on the field
  2. Connect your V5 Controller to the FCS through a Driver Interface using the Ethernet cable
  3. Turn both the V5 Controller and V5 Brain on and allow them to connect wirelessly
  4. Start the built-in Drive program or one of your own user programs written with the Competition Template.

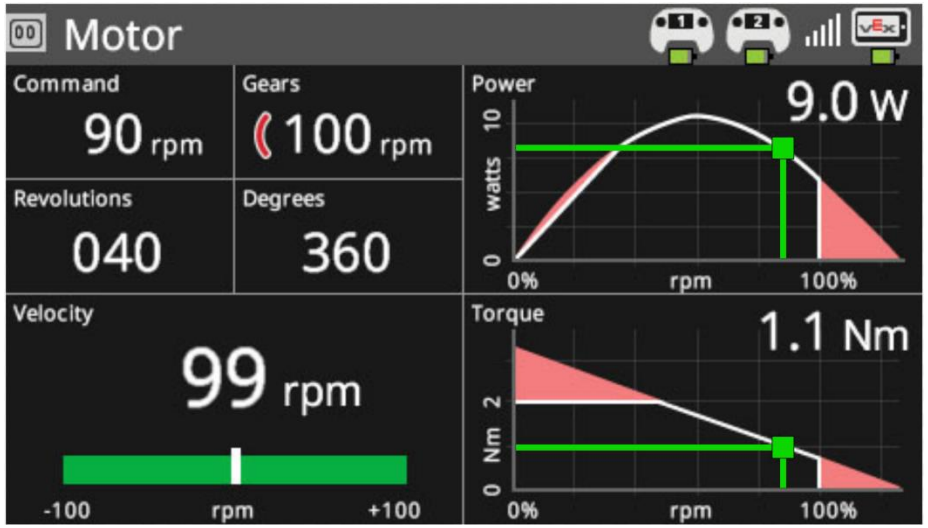
Starting your Program
Important Reminder!

- The VEX V5 allows you to store up to 8 User Programs!
  - This means that **you need to remember to choose which user program to start** (or the built-in Drive Program), and **start it**!
  - Any user program you want to choose must be based on the Competition Template.
  - You can choose which program to run through the V5 Brain screen or the V5 Controller screen.



Pro-tip: Testing Devices

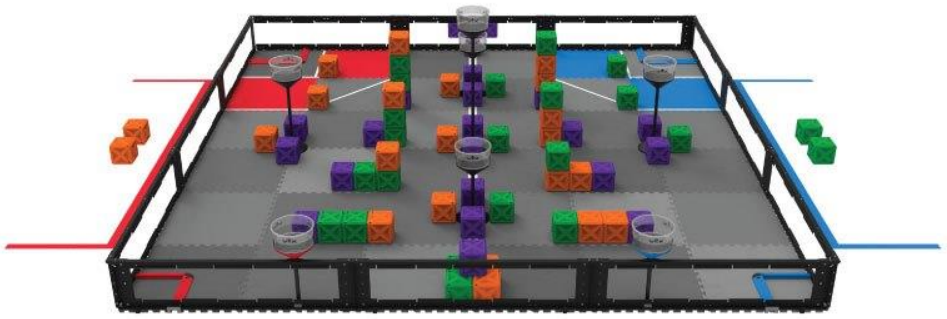
- Prior to the competition, you can use the Device Info screen on the V5 Brain to verify that all of the devices on your robot are properly connected and functioning.
- Tapping on individual devices such as Smart Motors allows you to check their status.
  - The Motor screen also allows you to provide power to the selected motor, ensuring that the motor is working properly in your drivetrain/



- **Tapping on the triangle symbol will take you to a screen where you can check the status of any 3-wire devices connected to your V5 Brain.**

Programming Skills/ Driving Skills Challenge

- Using the modified Tower Takeover Field
  - **Strategize, practice and prepare to complete a 1 minute Driving Skills Challenge**
  - **Strategize, practice and prepare to complete a 1 minute Programming Skills Challenge**
- Tower Takeover Website
- https://www.vexrobotics.com/vexedr/competition/vrc-current-game



Scoring:

Each Green Cube Scored in a goal

1 point + 1 point for every Green Cube Placed in Towers

Each Orange Cube Scored in a goal

1 point + 1 point for every Orange Cube Placed in Towers

Each Purple Cube Scored in a goal

1 point + 1 point for every Purple Cube Placed in Towers

Autonomous Bonus (Not in Skills Challenge)

6 points

.

Driving Skills Match – A Driving Skills Match consists of a sixty (60) second Driver Controlled Period. There is no Autonomous Period. Teams can elect to end their run early, however this will count as an official run.

Programming Skills Match – A Programming Skills Match consists of a sixty (60) second Autonomous Period. There is no Driver Controlled Period. Teams can elect to end their run early, however this will count as an official run.

The Robot Skills Challenge

Teams will be ranked based on their combined score in the two types of Matches. The playing field will be set up exactly the same as a normal VEX Robotics Competition Tower Takeover Match.

Robot Skills Challenge Definitions:

<RSC1> In Robot Skills Matches, all Goal Zones and Alliance Towers considered to be the same color for the purposes of any Alliance-specific rules or definitions.

a. Robots may **start on either side of the field.**

b. Robots may Score Cubes in **any color** of Goal Zone for points.

c. Robots may utilize **either Alliance Tower** for Placing Cubes.

<RSC2> In Robot Skills Matches**, all Cubes are considered to be the same color**.

<RSC3> Prior to the start of Robot Skills Matches, the **Robot must use its one (1) Cube available as a Preload.**

Alliances receive points for each Cube that is Scored in their Goal Zones at the end of a Match.

The point value of each Cube is defined by the number of Cubes that are Placed.

# of Cubes Placed in Towers	Point value for each cube on the field
0	1
1	2
2	3
3	4
4	5
5	6
6	7

Troubleshooting

- help.vex.com has many V5 specific setup and troubleshooting procedures which are very helpful!
- It's a good idea to visit the site and learn these procedures even before the need arises at the competition!