• Stensat Group LLC assumes no responsibility and/or liability for the use of the kit and documentation.

• There is a 90 day warranty for the Quad-Bot kit against component defects. Damage caused by the user or owner is not covered.
  – Warranty does not cover such things as over tightening nuts on standoffs to the point of breaking off the standoff threads, breaking wires off the motors, causing shorts to damage components, powering the motor driver backwards, plugging the power input into an AC outlet, applying more than 9 volts to the power input, dropping the kit, kicking the kit, throwing the kit in fits of rage, unforeseen damage caused by the user/owner or any other method of destruction.

• If you do cause damage, we can sell you replacement parts or you can get most replacement parts from online hardware distributors.

• This document can be copied and printed and used by individuals who bought the kit, classroom use, summer camp use, and anywhere the kit is used. Stealing and using this document for profit is not allowed.

• If you need to contact us, go to www.stensat.org and click on contact us.
In this lesson, a joystick controller is built to control the movement of the ROV kit. The block diagram shows all the major components and how they are all connected.

- The joystick provides the input to the processor board.
- Batteries provide power to everything.
- The processor board controls the motor drivers which are outputs.
- The motor drivers provide power to the ROV motors.
Components

- The joystick controller consists of an analog joystick and four push buttons.
- There are pins for connecting the joystick to the processor board.
Push Buttons

- Push buttons are switches that complete a circuit connection when pressed.
- The top figure on the right shows a circuit with a battery, a push button and a light bulb. When the push button is not pressed, the circuit is not completed and the light bulb does not light.
- When the push button is pressed, the circuit is completed and the light bulb lights up.
• The joystick consists of two variable resistors also called potentiometers.

• A potentiometer is a resistor with a wiper that taps into the resistor. Think of the potentiometer as a voltage divider where the tap is the divided voltage.

• The Equation below is used to determine the output voltage of the voltage divider circuit.

\[
V_{out} = \frac{V \cdot R_2}{R_1 + R_2}
\]
The joystick potentiometer is used as a voltage divider. The resistance of the potentiometer is $R_1 + R_2$ which is 10,000 ohms or 10 Kohms. The tap adjusts the values of $R_1$ and $R_2$ but the total will always be 10 Kohms.

At the mid point, $R_1$ and $R_2$ will each be 5 Kohms. The output voltage will be 2.5 volts.

$$V_{out} = \frac{V R_2}{R_1 + R_2}$$
Motor Control

- Dual H-Bridge Driver is used to control the motors. It uses four transistors to control the polarity of the voltage supplied to the motor. The transistors are used as switches turning on and off. Below shows the H-bridge driver circuit and the current flows.
To make the motor turn on one direction, two switches need to be turned on to let power get to the motor. One switch connects the positive side of the battery to one side of the motor and another switch connects the negative side to the other side of the motor.
Flip all the switches to the opposite position and the motor turns in reverse. Notice the polarity signs on the motor switched sides.
• The motor controller is the interface between the motors and the processor board. It has circuitry to allow control of the motors and can handle the high currents required to operate the motors. The processor board cannot directly power the motors. The controller is capable of providing the needed current and is used as the interface.
The motor controller module consists of two H-bridge drivers to control two motors.

The circuit side is shown at the top right. The square block in the center contains the two motor drivers.

The bottom picture shows the signal names next to the pins.

Power is supplied at pins GND and VIN.

Control signals for each motor are A1 IN, A2 IN, and B1 IN, B2 IN.

The motors connect to the pins marked OUT.

The other pins are not used.
How the H-Bridge Driver Works

- This drawing shows how the H-Bridge driver works. Only one is shown.
- There are two signals that control the direction and operation. Control logic decodes the two signals and turns on the appropriate switches to control the motor. The drawing shows the condition of AIN1 and AIN2 set to logic zero.
How the H-Bridge Driver Works

- When AIN1 is set to logic 1, the motor drives in the forward direction.
- You will notice that setting AIN1 = 1, and AIN2=0 turns on two signals that turn on the two switches.
How the H-Bridge Driver Works

- When AIN1 is set to logic 1, the motor drives in the reverse direction.
- You will notice that setting AIN1 = 0, and AIN2=1 turns on two signals that turn on the two switches.
How the H-Bridge Driver Works

- When you set both AIN1 and AIN2 to logic 1, you get a breaking action.
- This turns on the two bottom switches which shorts the motor connections together. The inductance created by the motor turning in one direction will power the motor to turn in the opposite direction. It causes the motor to slow down quickly.
Below is the circuit board where the joystick and push buttons will be mounted and soldered.

- The component order of soldering will be from shortest to tallest.
- The yellow blocks are where the push buttons are mounted.
- The blue block is where the pins get soldered.
- The red block is where the joystick is mounted.
• First, the pins need to be cut up into smaller sets.
• Cut the pins up so that there is a strip of 5 pins and two strips of 2 pins.
Soldering the Pins

- Solder the pins into place.
Installing the Push Buttons

- Insert the push buttons as shown. They have to be installed in a specific orientation as shown in the picture.
Installing the Joystick

- Align the pins with the holes on the circuit board and insert the joystick. Don't force it too hard. If it doesn't press in, check for a pin that is bent out of place a little and straighten it.
Installing the Knob

- Look at the inside bottom of the joystick knob. The center hole is rectangular. Align that with the joystick and press fit into place.

Align the two when installing knob
Mounting Joystick Board

- Next is to mount the joystick board onto the processor electronics plate.
- Remove the solderless breadboard.
- Install four ½ inch screws from the bottom and secure with nuts as shown.
Mounting Joystick Board

- Install the joystick board on the screws and secure with four nuts.
Wiring the Joystick

- Use the provided jumpers that have female connectors on both ends.
- Connect the +5V from the joystick to 5V on the processor board at the ANALOG signals.
- Connect the GND from the joystick to the GND on the processor board.
Wiring the Joystick

- Connect Y signal from the joystick to Analog port 1.
- Connect X signal from the joystick to Analog port 2.
Wiring the Joystick

- Connect DOWN signal from the joystick to digital port D4.
- Connect the UP signal from the joystick to digital port D2.
- This completes the wiring of the joystick. The other pins are not used.
An Analog-to-Digital Converter or ADC is a device that generates a number based on the voltage level it measures.

The ADC on the processor board can measure a voltage range from zero to 5 volts. If a higher voltage needs to be measured, the voltage divider circuit could be used to reduce the voltage to 5 volts or less.

The ADC is 10 bits. This gives a numerical range of zero to 1023. It is a linear relationship to the voltage range of 0 to 5 volts.

- The ADC will generate a value of 0 for 0 volts, 1023 for 5 volts, and 511 for 2.5 volts. The resolution of the ADC is 5/1023 or 0.00489 volts per bit.
- Voltage the ADC measures can be calculated by the equation
  \[ V = \frac{\text{ADC value}}{1023} \times 5.0 \]

ADC value is the number the ADC generates.
The code to the right measures the voltage of the potentiometer and displays the ADC value.

With the joystick at center position, the potentiometer is at the mid point which should be at the mid voltage of 2.5 volts since 5 volts is being supplied.

Move the joystick around and observe the values.

The first column number is the x value and corresponds with the side to side movement of the joystick.

The second column number is the y value and corresponds with the forward and back movement.

```cpp
void setup()
{
    Serial.begin(9600);
}

void loop()
{
    int x, y;
    x = analogRead(0);
    y = analogRead(1);
    Serial.print(x, DEC);
    Serial.print(" ");
    Serial.println(y, DEC);
    delay(50);
}
```
Code to Measure Joystick Position

- Move the joystick all the way forward and observe the y value. It should be 1023.
- Move the joystick all the way back and observe the y value. It should be 0.
- Move the joystick all the way to the left and observe the x value. It should be 1023.
- Move the joystick all the way to the right and observe the x value. It should be 0.
- Remember the positions and values. They will be used soon.

```cpp
void setup()
{
    Serial.begin(9600);
}

void loop()
{
    int x,y;
    x = analogRead(0);
    y = analogRead(1);
    Serial.print(x,DEC);
    Serial.print(" ");
    Serial.println(y,DEC);
    delay(50);
}
```
Code to Detect the Push Buttons

- Start a new program. This new program will detect when the UP and DOWN push buttons are pressed.
- In the `setup()` function, digital pins 2 and 4 are configured as inputs with pullups. What this does is it connects resistors inside the processor chip to the digital pins so if there is no connection, the digital pins will detect a logic 1, HIGH which is 5 volts.
- When the push buttons are pressed, the digital pins are connected to ground which is logic 0.

```cpp
void setup()
{
    Serial.begin(9600);
    pinMode(2, INPUT_PULLUP);
    pinMode(4, INPUT_PULLUP);
}

void loop()
{
    int up, down;
    up = digitalRead(2);
    down = digitalRead(4);
    Serial.print(up);
    Serial.print(" ");
    Serial.println(down);
    delay(50);
}
```
Motor Control

- With the joystick board installed and tested, now it is time to get the joystick to control the ROV.
- Shown is the motor driver board. It includes 4 H-bridge drivers grouped in pairs.
- There are a set of pins for input and output.
- There is also a terminal block for the output.
Motor Control

- Marked on the left is H-BRIDGE 1. This contains two motors drivers marked A and B.
- On the right is H-Bridge 2 which also contains two motor drivers marked A and B.
On the left side are two sets of power input pins. This allows power to be daisy chained to other devices if needed.
Mount the Motor Driver

- Install the four stand-offs onto the four screws that secure the processor board.
Mount the Motor Driver

- Before installing the motor driver, connect a jumper wire from the **BAT** pin of the processor board. This needs to be done now since it will not be accessible when the motor driver board is mounted.

- Connect another jumper wire to the **GND** pin next to the **BAT** pin.
Mounting the Motor Driver

- Secure the motor driver board on top of the standoffs.
- Secure with nuts.
Mounting the Motor Driver

- The following digital pins will be assigned to the motors
  - Vertical motors are controlled by pins 6, 9
  - Left horizontal motor is controlled by pins 5, 3
  - Right horizontal motor is controlled by pins 11, 10
Wiring the Motor Driver

- Follow the drawing in connecting wires.
- First, connect the BAT pin on the processor board to +V on the motor driver board.
- Second, connect the GND pin on the processor board to the GND on the motor driver board.
Wiring the Motor Driver

- Connect Digital pin 3 to A1 of H-Bridge 2.
- Connect Digital pin 5 to A2 of H-Bridge 2.
- Connect Digital pin 6 to B2 of H-Bridge 2.
- Connect Digital pin 9 to B1 of H-Bridge 2.
- Connect Digital pin 10 to A1 of H-Bridge 2.
- Connect Digital pin 11 to A2 of H-Bridge 2.
Completed Wiring

- This completes the wiring of the motor driver to the processor board. Next is connecting the ROV to the motor driver.
- Disconnect the tether from the ROV hand controller.
Complete Wiring

- Connect the tether as shown.
- The A driver for H-Bridge 2 will control one horizontal motor.
- The B driver for H-Bridge 2 will control the vertical motor.
- The A driver for H-Bridge 1 will control the other horizontal motor.
- We do not need to worry about motor polarity or which horizontal motor goes where. It can be fixed in software.
We will start the software by first controlling the vertical motor. The vertical motor is to be controlled by the push buttons. The **UP** button will control the state of digital pin 6. The **DOWN** button will control the state of digital pin 9.

```cpp
void setup() {
    pinMode(2, INPUT_PULLUP);
    pinMode(4, INPUT_PULLUP);
}

void loop() {
    int up, down;
    up = digitalRead(2);
    down = digitalRead(4);
    if(up == 0) digitalWrite(6, HIGH);
    else digitalWrite(6, LOW);
    if(down == 0) digitalWrite(9, HIGH);
    else digitalWrite(9, LOW);
}
```
• Notice how the pins are configured in `setup()`.

• They are configured as inputs with the addition of pullup.

• The switch when pressed, connects the ground to the input pin. When not pressed, the input pin is not connected to anything. Microcontrollers do not like that as the state of the input is not known.

• `INPUT_PULLUP` sets the input to 5 volts when the button is not pressed.

```cpp
void setup()
{
    pinMode(2, INPUT_PULLUP);
    pinMode(4, INPUT_PULLUP);
}
```
In the **loop()** function, the **UP** button is checked to see if it is pressed. If it is pressed, the result value is 0.

The **if()** statement does the comparison. if variable **up** is equal to 0 then set digital pin 6 **HIGH**.

If the statement comparison is not true, then digital pin 6 is set **LOW**.

```cpp
void setup()
{
    pinMode(2,INPUT_PULLUP);
    pinMode(4,INPUT_PULLUP);
}

void loop()
{
    int up,down;
    up = digitalRead(2);
    down = digitalRead(4);
    if(up == 0) digitalWrite(6,HIGH);
    else digitalWrite(6,LOW);
    if(down == 0) digitalWrite(9,HIGH);
    else digitalWrite(9,LOW);
}
```
The same comparison is used for the **DOWN** button.

The **DOWN** button control digital pin 9.
Simple Software Control

- Below shows how digital pins 6 and 9 are connected to the motor driver.
- You can review how the motor driver operates based on signal inputs.

![Diagram showing connections between control logic, motor, and battery.]
Simple Software Control

- Upload and run the program. See if pressing either push button makes the vertical motor operate.
- Try pressing both buttons at the same time and see what happens.

```c
void setup()
{
    pinMode(2, INPUT_PULLUP);
    pinMode(4, INPUT_PULLUP);
}

void loop()
{
    int up, down;
    up = digitalRead(2);
    down = digitalRead(4);
    if(up == 0) digitalWrite(6, HIGH);
    else digitalWrite(6, LOW);
    if(down == 0) digitalWrite(9, HIGH);
    else digitalWrite(9, LOW);
}
```
Simple Software Control

- If the propeller is going in the wrong direction, simply swap the variables **up** and **down** in the **if()** statements.

- This flips the spinning direction of the vertical propellers.

```cpp
void setup()
{
  pinMode(2, INPUT_PULLUP);
  pinMode(4, INPUT_PULLUP);
}

void loop()
{
  int up, down;
  up = digitalRead(2);
  down = digitalRead(4);
  if (up == 0) digitalWrite(6, HIGH);
  else digitalWrite(6, LOW);
  if (down == 0) digitalWrite(9, HIGH);
  else digitalWrite(9, LOW);
}
```
Simple Joystick Control

- We will do simple joystick control of the horizontal motors.

- If the joystick is pushed all the way forward, the ROV will move forwards.

- Digital pins 3 and 5 control one horizontal motor and digital pins 10 and 11 control the other horizontal motor.

```cpp
void loop()
{
  int up, down;
  int x, y;
  up = digitalRead(2);
  down = digitalRead(4);
  if(up == 0) digitalWrite(6, HIGH);
  else digitalWrite(6, LOW);
  if(down == 0) digitalWrite(9, HIGH);
  else digitalWrite(9, LOW);
  x = analogRead(0);
  y = analogRead(1);
  if(y == 0) {
    digitalWrite(5, HIGH);
    digitalWrite(3, LOW);
    digitalWrite(11, HIGH);
    digitalWrite(10, LOW);
  } else {
    digitalWrite(5, LOW);
    digitalWrite(3, LOW);
    digitalWrite(11, LOW);
    digitalWrite(10, LOW);
  }
}
```
Simple Joystick Control

- Test the code out and see if the ROV moves forward.
- If the ROV turns swap one of the motor digital pins such as 3 and 5 or 10 and 11.
- If the ROV moves backwards then swap both 3 and 5 and 10 and 11.

```cpp
void loop()
{
    int up,down;
    int x,y;
    up = digitalRead(2);
    down = digitalRead(4);
    if(up == 0) digitalWrite(6,HIGH);
    else digitalWrite(6,LOW);
    if(down == 0) digitalWrite(9,HIGH);
    else digitalWrite(9,LOW);
    x = analogRead(0);
    y = analogRead(1);
    if(y == 0) { // go forward
        digitalWrite(3,HIGH);
        digitalWrite(5,LOW);
        digitalWrite(10,HIGH);
        digitalWrite(11,LOW);
    } else { 
        // stop
        digitalWrite(3,LOW);
        digitalWrite(5,LOW);
        digitalWrite(10,LOW);
        digitalWrite(11,LOW);
    }
}
```
Simple Joystick Control

- To go in reverse, another test of \texttt{y} must be made. The highlighted area shows a new \texttt{if()} statement inserted in front of the \texttt{else} statement.

- To go into reverse, reverse the digital pins as shown.

```c
if(y == 0) { // go forward
digitalWrite(3,HIGH);
digitalWrite(5,LOW);
digitalWrite(10,HIGH);
digitalWrite(11,LOW);
} else if(y == 1023) { // go reverse
digitalWrite(5,HIGH);
digitalWrite(3,LOW);
digitalWrite(11,HIGH);
digitalWrite(10,LOW);
} else { // stop
digitalWrite(3,LOW);
digitalWrite(5,LOW);
digitalWrite(10,LOW);
digitalWrite(11,LOW);
}
```
Simple Joystick Control

- Next step is to add turning. It will look the same as to the right except replace `y` with `x`.
- You will also notice the **HIGH** and **LOW** states are changed. This makes one motor turn the opposite of the other.
- Add the code highlighted.
- Test it out and see if the ROV turns right. If it turns left, then swap 3 and 5 and swap 10 and 11.

```c
if(y == 0) { // go forward
  digitalWrite(3,HIGH);
  digitalWrite(5,LOW);
  digitalWrite(10,HIGH);
  digitalWrite(11,LOW);
} else if(y == 1023) { // go reverse
  digitalWrite(5,HIGH);
  digitalWrite(3,LOW);
  digitalWrite(11,HIGH);
  digitalWrite(10,LOW);
} else { // stop
  digitalWrite(3,LOW);
  digitalWrite(5,LOW);
  digitalWrite(10,LOW);
  digitalWrite(11,LOW);
}
```

```c
if(x == 0) { // turn right
  digitalWrite(3,HIGH);
  digitalWrite(5,LOW);
  digitalWrite(10,LOW);
  digitalWrite(11,HIGH);
}
```
Simple Joystick Control

- To turn left, \( x \) is compared with 1023. If true then turn the ROV left.

- The highlighted code is the same as the previous but with the digital pins swapped to run the motors in the opposite direction.

- Add the highlighted code to the program.

- Test out the program and operate the ROV. Make any digital pin changes necessary to move and turn in the expected directions.

```cpp
} 
if(x == 0) { 
    digitalWrite(3,HIGH);
    digitalWrite(5,LOW);
    digitalWrite(10,LOW);
    digitalWrite(11,HIGH);
} else if(x == 1023) { 
    digitalWrite(5,HIGH);
    digitalWrite(3,LOW);
    digitalWrite(11,LOW);
    digitalWrite(10,HIGH);
} 
```
Complete Simple Joystick Control Program

```c
void setup()
{
  pinMode(2, INPUT_PULLUP);
  pinMode(4, INPUT_PULLUP);
}

void loop()
{
  int up, down;
  int x, y;
  up = digitalRead(2);
  down = digitalRead(4);
  if (up == 0) digitalWrite(6, HIGH);
  else digitalWrite(6, LOW);
  if (down == 0) digitalWrite(9, HIGH);
  else digitalWrite(9, LOW);
  x = analogRead(0);
  y = analogRead(1);
  if (y == 0) {
    // go forward
    digitalWrite(5, HIGH);
    digitalWrite(3, LOW);
    digitalWrite(11, HIGH);
    digitalWrite(10, LOW);
  } else if (y == 1023) {
    // turn left
    digitalWrite(5, HIGH);
    digitalWrite(3, LOW);
    digitalWrite(11, LOW);
    digitalWrite(10, HIGH);
  } else {
    // stop
    digitalWrite(5, LOW);
    digitalWrite(3, LOW);
    digitalWrite(11, LOW);
    digitalWrite(10, LOW);
  }
}      
if(x == 0) {  // turn right
  digitalWrite(3, HIGH);
  digitalWrite(5, LOW);
  digitalWrite(10, LOW);
  digitalWrite(11, HIGH);
} else if(x == 1023) {
  // turn left
  digitalWrite(5, HIGH);
  digitalWrite(3, LOW);
  digitalWrite(11, LOW);
  digitalWrite(10, HIGH);
}
```
Proportional Joystick Control

• The previous program used the joystick like an on/off switch. The motors were either turned on or off. This section will use the position of the joystick to control the speed and direction of the motors. This will provide finer control of the ROV. You may have noticed the turns are fast and most likely faster than you like.
Proportional Joystick Control

- The joystick will be used for only the horizontal motors and control the forward and reverse and left and right turns. The push buttons will continue to be used for vertical movement since it is much slower.
- The trick will be converting X-Y joystick position to tank drive control.
- The Y axis of the joystick will control the forward and reverse direction and speed of the ROV.
- The X axis of the joystick will control the turning left and right of the ROV.
- Start a new program.
Proportional Speed Control

- Configure the digital pins in the `setup()` function.
- The same digital pins will be used to control the same motors.
- The same two digital pins will be used to detect the push buttons.

```cpp
void setup() {
    Serial.begin(9600);
    pinMode(3, OUTPUT);
    pinMode(5, OUTPUT);
    pinMode(6, OUTPUT);
    pinMode(9, OUTPUT);
    pinMode(10, OUTPUT);
    pinMode(11, OUTPUT);
    pinMode(2, INPUT_PULLUP);
    pinMode(4, INPUT_PULLUP);
}
```
The map() function remaps a given number from one range to another range.

The plot on the right shows how a number is mapped from one range to another.

- The x axis is the input range that goes from 0 to 1023.
- The y axis is the output range that goes from 0 to 255.
- The input value is 700. The output will be the value that intersects the line on the y axis.
- The output will be 174.
The `map()` function has the following format:

\[ y = \text{map}(x, \text{lowX}, \text{highY}, \text{lowY}, \text{highY}); \]

- The values are only integers and can be negative.
Proportional Speed Control

- First operation is to get the analog measurements from the joystick.
- Next is to adjust the values so the variables will go negative when the joystick is pushed one direction and positive in the other direction. This will let us calculate the direction of the motors.
- The `map()` function converts one range to another range of numbers.
- Variable `y` will control the speed and forward and reverse direction.
- Variable `x` will control the left and right turning.

```c
void loop() {
    int x = analogRead(0);
    int y = analogRead(1);
    x = map(x, 0, 1023, -255, 255);
    y = map(y, 0, 1023, -255, 255);
}
```
Proportional Speed Control

- Now it is time to mix the direction and speed portions.
- First the speed control axis is flipped so the ROV moves in the correct direction.

```cpp
void loop() {
    int x = analogRead(0);
    int y = analogRead(1);
    x = map(x, 0, 1023, -255, 255);
    y = map(y, 0, 1023, -255, 255);
    y = -y;
    int left = x + y;
    int right = y - x;
}
```
max() Function

• The max() function compares two numbers and returns the largest number.
• The function format is this:  $y = \text{max}(a,b)$;
• Next is to calculate if the results are greater than 255. This is done because the motor speed control has a range of 0 to 255. If the calculation results are greater than 255, then the motors will not operate properly.

```cpp
void loop() {
  int x = analogRead(0);
  int y = analogRead(1);
  x = map(x, 0, 1023, -255, 255);
  y = map(y, 0, 1023, -255, 255);
  int left = x + y;
  int right = y - x;
  float leftscale = left / 255.0;
  leftscale = abs(leftscale);
  float rightscale = right / 255.0;
  float maxscale = max(leftscale, rightscale);
  maxscale = max(1, maxscale);
}```
constrain() Function

- The constrain() function tests a value against a range and returns either the value being tested or the range limits if the value is beyond the range.
- The function format is: \( y = \text{constrain}(x, \text{low}, \text{high}) \);
- \( x \) is the value being tested.
- \( \text{low} \) is the lower end of the range.
- \( \text{high} \) is the upper end of the range.
Proportional Speed Control

- Next is to calculate the final values for the left and right motor speeds.
- The constrain() function compares the variable with the range of values. If the variable is within the range, the result is the variable. If the variable is larger than either limit, the limit is returned.

```cpp
void loop() {
  int x = analogRead(0);
  int y = analogRead(1);
  x = map(x, 0, 1023, -255, 255);
  y = map(y, 0, 1023, -255, 255);
  int left = x + y;
  int right = y - x;
  left = constrain(left, -255, 255);
  right = constrain(right, -255, 255);
}
```
Proportional Speed Control

- Now it is time to control the motors with the results in `left` and `right` variables.
- Variable `left` has a range of -255 to 255.
- A positive value is forward and a negative value is reverse.
- Going forward, pin 5 is set to the speed and pin 3 is set to 0.

```java
void loop() {
  int x = analogRead(0);
  int y = analogRead(1);
  x = map(x, 0, 1023, -255, 255);
  y = map(y, 0, 1023, -255, 255);
  y = -y;
  int left = x + y;
  int right = y - x;
  left = constrain(left, -255, 255);
  right = constrain(right, -255, 255);
  if (left > 0) {
    analogWrite(5, left);
    analogWrite(3, 0);
  } else if (left < 0) {
    left = -left;
    analogWrite(5, 0);
    analogWrite(3, left);
  }
}
```
Proportional Speed Control

- To go in reverse, pin 5 is set to 0 and pin 3 is set to the value of `left`.
- `AnalogWrite()` can only use values of 0 to 255 so the variable `left` needs to be made positive. This is done by `left = -left`.

```cpp
void loop() {
  int x = analogRead(0);
  int y = analogRead(1);
  x = map(x, 0, 1023, -255, 255);
  y = map(y, 0, 1023, -255, 255);
  y = -y;
  int left = x + y;
  int right = y - x;
  left = constrain(left, -255, 255);
  right = constrain(right, -255, 255);
  if(left > 0) {
    analogWrite(5, left);
    analogWrite(3, 0);
  } else if(left < 0) {
    left = -left;
    analogWrite(5, 0);
    analogWrite(3, left);
  }
}
```
Proportional Speed Control

- The same code is used for the right motor.

```cpp
right = constrain(right,-255,255);
if(left>0) {
    analogWrite(5,left);
    analogWrite(3,0);
} else if(left < 0) {
    left = -left;
    analogWrite(5,0);
    analogWrite(3,left);
}
if(right>0) {
    analogWrite(11,right);
    analogWrite(10,0);
} else if(right < 0) {
    right = -right;
    analogWrite(11,0);
    analogWrite(10,right);
}
```
Proportional Speed Control

- Next, the code to operate the vertical motors need to be added.

- It doesn't change from the first program.

- A delay is inserted at the end. This means the motor speed control is updated 10 times a second. If the delay was not included, the constant updating of the speed control would not give the motors a chance to operate. The digital port is reset every time `analogWrite()` is executed.

```cpp
right = constrain(right, -255, 255);
if(left > 0) {
  analogWrite(5, left);
  analogWrite(3, 0);
} else if(left < 0) {
  left = -left;
  analogWrite(5, 0);
  analogWrite(3, left);
}
if(right > 0) {
  analogWrite(11, right);
  analogWrite(10, 0);
} else if(right < 0) {
  right = -right;
  analogWrite(11, 0);
  analogWrite(10, right);
}
int up =digitalRead(2);
int down =digitalRead(4);
if(up == 0) digitalWrite(6, HIGH);
else digitalWrite(6, LOW);
if(down == 0) digitalWrite(9, HIGH);
else digitalWrite(9, LOW);
delay(100);
```
Proportional Speed Control

• This completes the coding for the speed control. Test out the ROV and the controller.

• If the ROV turns in the opposite direction, add the following right after the `analogRead()` functions: $x = -x$; Same can be done with the y variable if the ROV is going backwards when it is supposed to go forwards.

• If the ROV is turning when it is supposed to go straight, go back in the code and swap the appropriate pins for the motor that is going in the wrong direction.

• This is one simple way to use a joystick to operate a tank style drive which is controlling left and right side motors separately. There are other algorithms that are more complex. This simple algorithm is good enough to control the ROV.