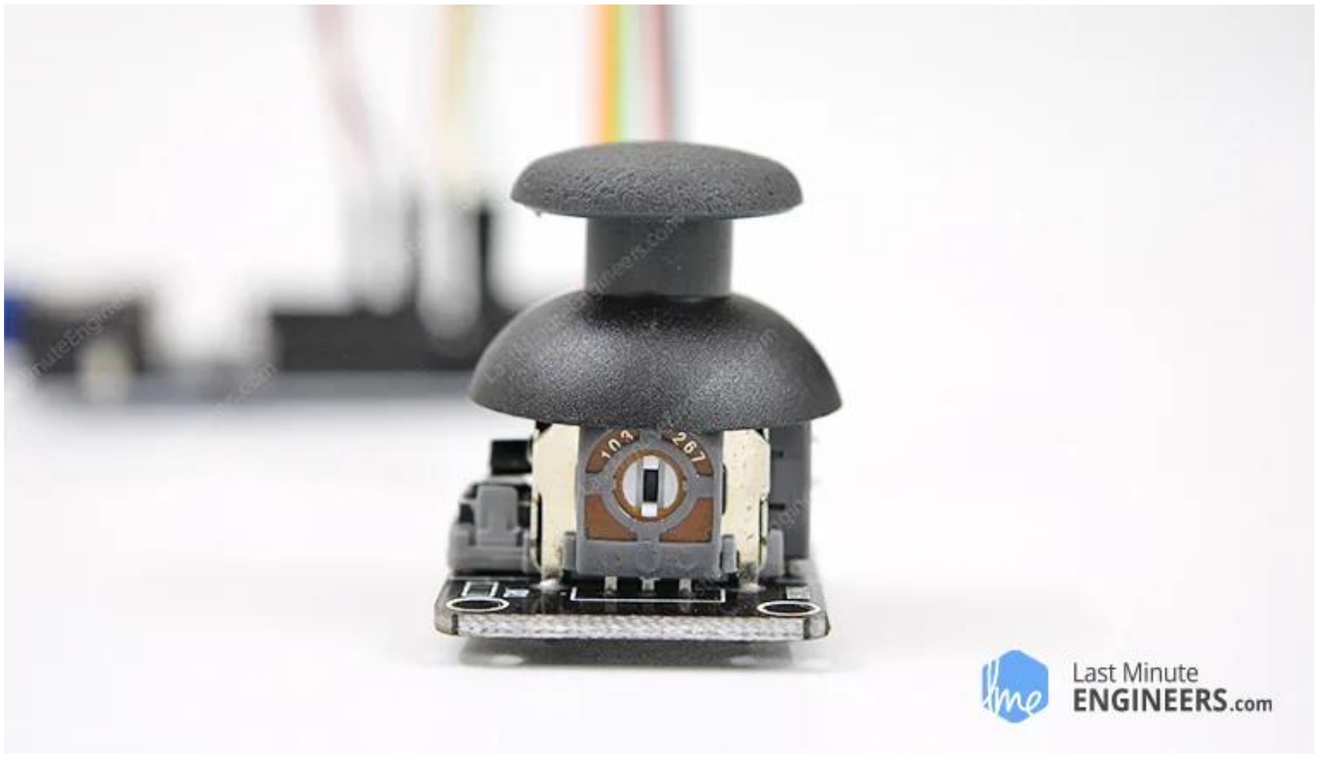
JOYSTICK MODULE



When you hear the term “thumb joystick,” the first image that probably pops into your mind is a game controller. While thumb joysticks are definitely common in gaming, in the world of DIY electronics, they open up a whole universe of exciting possibilities

You might use one to control the movements of a robot or rover you’ve built, allowing it to move forward, backward, or turn with intuitive thumb movements. Or perhaps you’d like to create a system for smoothly controlling camera movements, where gentle pushes of the joystick result in slow pans or tilts.

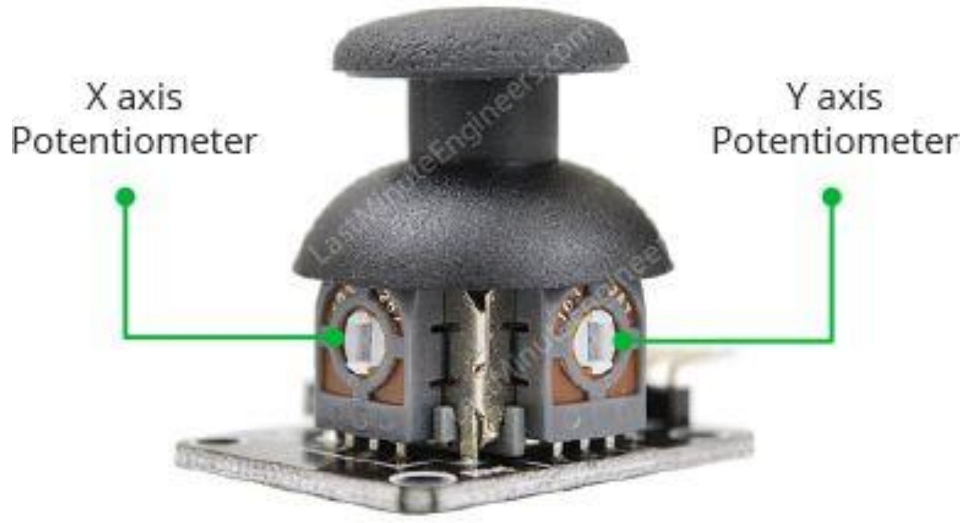
Let’s explore how these joystick modules actually work and how you can connect them to an Arduino for your own projects.

Hardware Overview

If you've ever played games on a PlayStation controller, you'll instantly recognize this thumb joystick. It looks and feels very similar to the ones Sony uses in their controllers. What makes this joystick special is that it's "self-centering" and "spring-loaded." This means that whenever you stop pushing it and let go, it automatically bounces back to the center position all by itself. The joystick also has a comfortable cup-shaped knob that fits your thumb perfectly, just like the thumbsticks on gaming controllers.

- ### Potentiometers

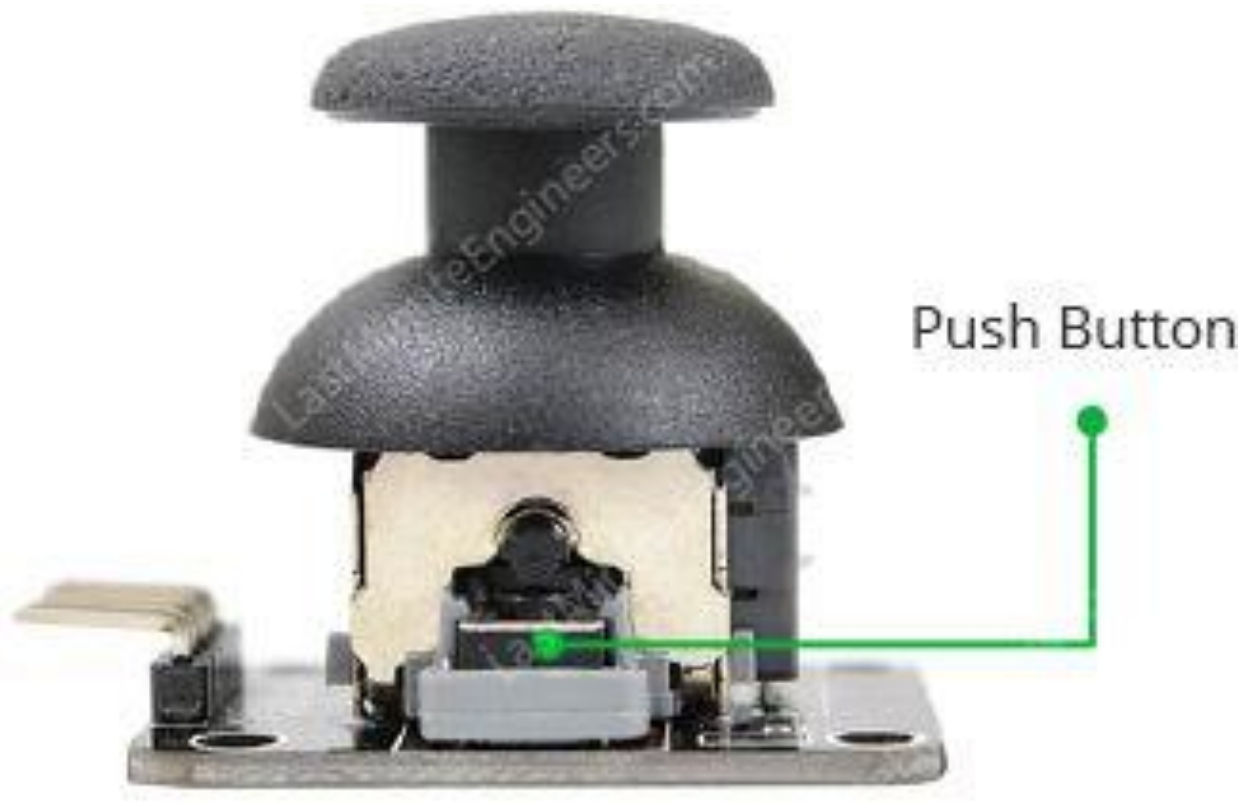
At the heart of how this joystick works are two 5K potentiometers. If you look at the joystick module closely, you'll notice two gray box-shaped components on either side—these are the potentiometers. As you move the joystick, you'll notice that each one only responds to movement in one direction. The potentiometer on one side only detects when you move the stick left or right (X-axis movement), while the other only detects when you move it up or down (Y-axis movement).



These potentiometers are connected to the joystick through a mechanical system called a "gimbal mechanism", which we will discuss a bit later. This clever arrangement ensures that when you move the joystick diagonally, it properly separates that movement into the correct amounts of horizontal and vertical signals.

- **Momentary Pushbutton switch**

Another cool feature of this joystick is the built-in momentary pushbutton switch—it's that small black box you can see on one side of the joystick module. This switch activates whenever you press down directly on the joystick knob.



If you look closely, you'll notice that pressing the knob causes a small lever to push down on the switch. This clever design ensures that the lever can operate the switch no matter what position the joystick is in.

This means you can push the joystick in any direction and still be able to click it.

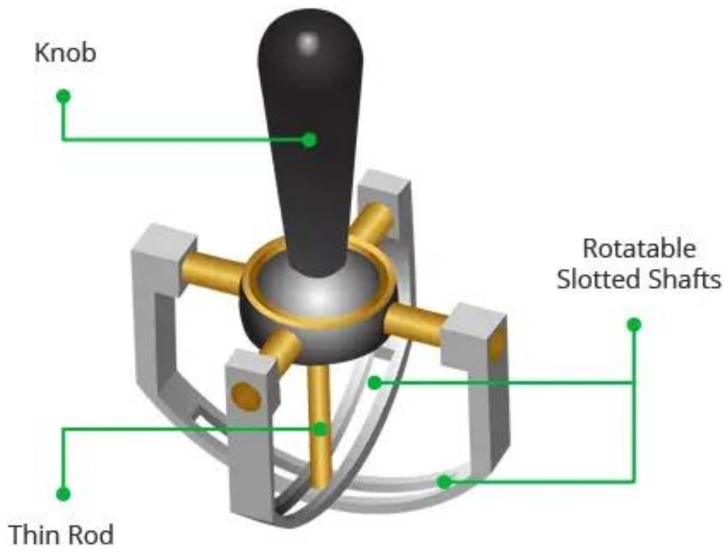
Now, let's explore the thumb joystick module more deeply and understand how it actually works.

How does the Thumb Joystick Module Work?

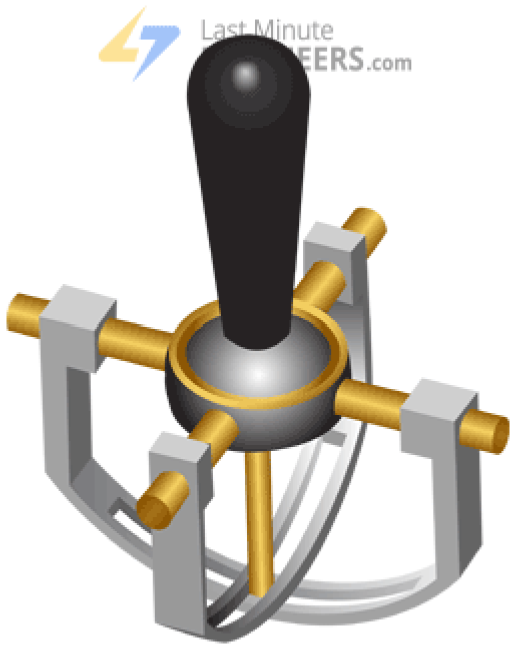
It's really fascinating how a joystick can accurately convert even tiny movements of your fingertips into electrical signals that a computer or microcontroller can understand. This precision comes from the joystick's clever design, which uses two potentiometers and something called a gimbal mechanism.

Gimbal Mechanism

The gimbal mechanism is what allows the joystick to move smoothly in multiple directions. When you move the joystick, the thumb handle moves a thin rod that sits between two rotatable slotted shafts. These shafts form the gimbal system. One shaft controls movement along the X-axis (left and right), while the other controls movement along the Y-axis (up and down).

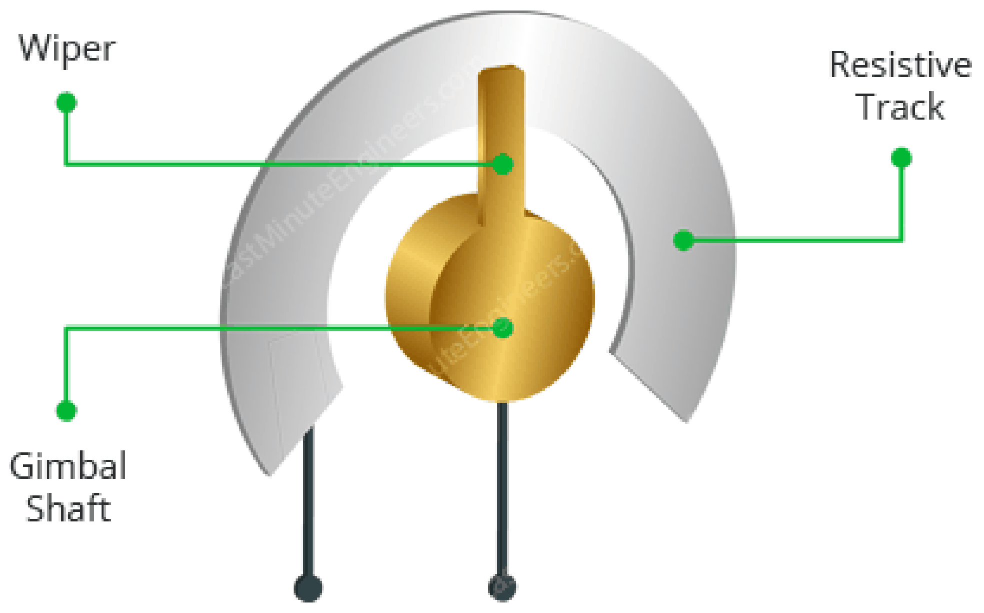


When you tilt the joystick forward or backward, the Y-axis shaft rotates. When you move it left or right, the X-axis shaft rotates instead. And here’s where it gets really interesting—when you push the joystick diagonally, both shafts rotate simultaneously.



Now, how does the joystick convert these physical movements into electrical signals that your Arduino can understand? The answer lies in two special potentiometers.

Each of the rotating shafts in the joystick connects to its own potentiometer. As you move the joystick, the shafts rotate, which causes the potentiometers to rotate as well. Inside each potentiometer is a contact arm (wiper) that slides along a resistive track. When the joystick moves all the way in one direction, this contact arm moves to one end of the track. When you move the joystick in the opposite direction, the arm moves to the other end.



This changing position creates a changing resistance. By measuring these potentiometer values, the joystick's exact position can be determined.

- **Reading analog values from Joystick**

The joystick outputs an analog signal with a voltage between 0 and VCC—that represents its position. As you move the joystick along the X-axis from one extreme to the other, the output voltage changes from 0V to VCC. The same happens for the Y-axis. When the joystick is in its center (or resting) position, the output voltage is about half of the supply voltage.

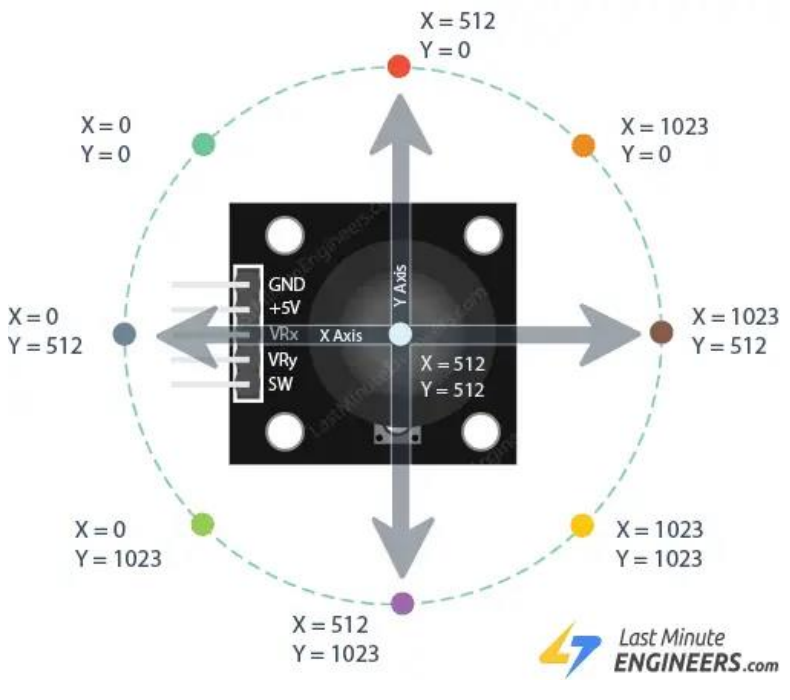
This changing voltage can be read by the Analog-to-Digital Converter (ADC) in a microcontroller to figure out exactly where the joystick is positioned.

When using an Arduino, which has a 10-bit ADC resolution, the analog readings for each axis can range from 0 to 1023. So as you move the joystick along the X-axis, the readings change in a predictable pattern. When you push it all the way to the left, the X-axis reads a value close to 0. When the joystick sits in its center or neutral position, the X-axis reads approximately 512. And when you push the joystick all the way to the right, the X-axis reads close to 1023.

The Y-axis works in exactly the same way. When you push the joystick all the way up, the Y-axis reads a value close to 0. In the center position, it reads around 512. And when you push the joystick all the way down, the Y-axis reads close to 1023.

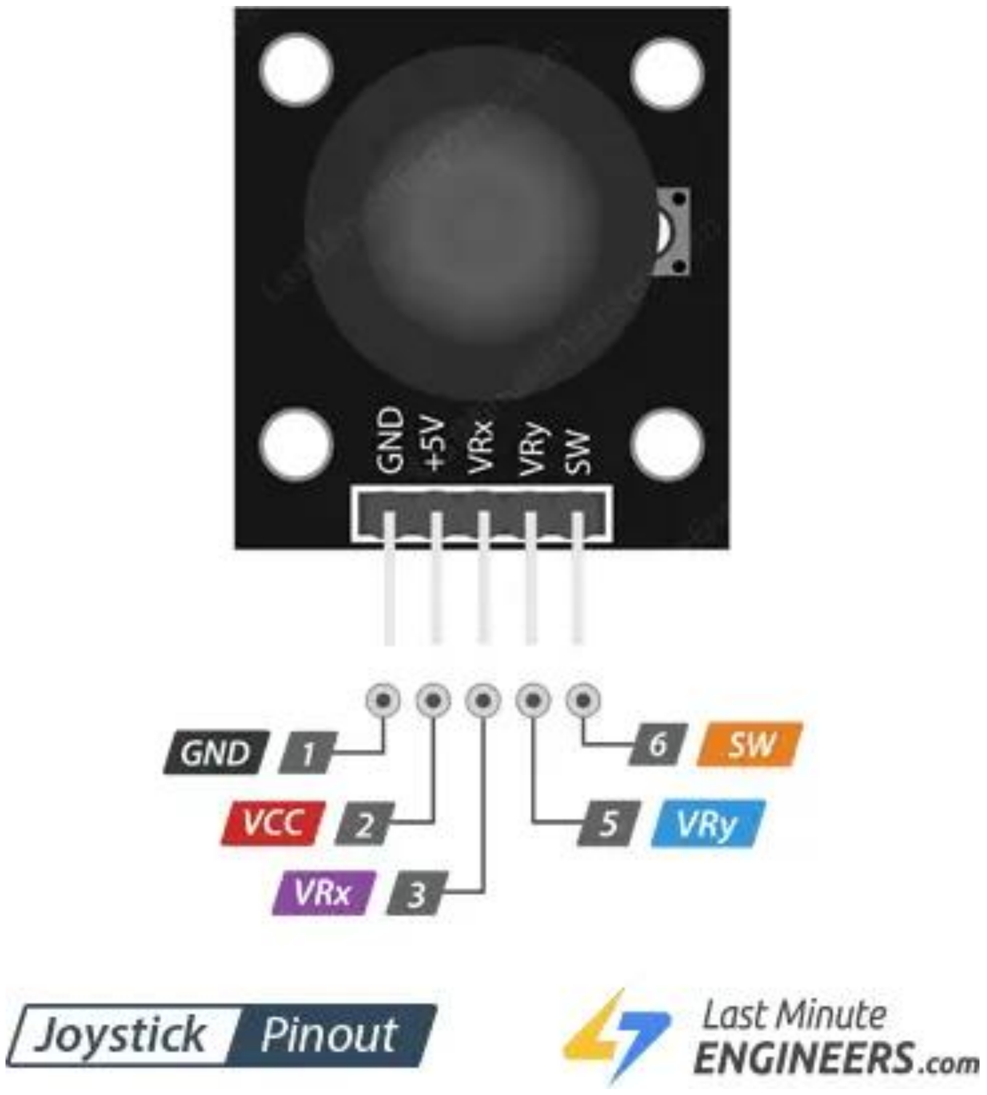
When you move the joystick diagonally, both X and Y values change at the same time. By reading both the X and Y values together, your program can determine the exact position of the joystick at any moment. For example, if X = 512 and Y = 1023, you know the joystick is being pushed straight down. If X = 0 and Y = 0, it's being pushed to the top-left corner.

The figure below illustrates the different values you can expect as you move the joystick in various directions.



Thumb Joystick Module Pinout

Let’s look at what each pin on the thumb joystick module does:



GND is the ground pin.

VCC powers the joystick module. Connect it to your positive power supply – usually 5V or 3.3V, depending on what your project needs.

VRx pin provides an analog output signal that corresponds to the horizontal (left-right) movement of the joystick. When you move the joystick all the way to the left, this pin outputs close to 0 volts. When you move it all the way to the right, it outputs a voltage close to whatever your VCC is (5V or 3.3V). When the joystick sits in the middle without being touched, this pin outputs approximately half of your VCC voltage (around 2.5V if you're using 5V power).

VRy pin works just like VRx, but for the vertical (up-down) movement of the joystick. Moving the joystick all the way up gives you close to 0 volts, while moving it all the way down might give you the maximum voltage (VCC). Again, when the joystick is centered and not being touched, this pin outputs about half of your VCC voltage.

SW pin connects to a built-in pushbutton switch inside the joystick. This button activates when you press down on the joystick knob itself. An important thing to understand about this pin is that, by default, it floats, meaning it doesn't automatically give clear HIGH or LOW signals on its own. To make the switch work properly, you need to use a pull-up resistor. This can be either the internal pull-up resistor in your Arduino (which you can enable in your code) or an external resistor you add to your circuit. With the pull-up in place, the pin will normally read as HIGH, and when you press down on the joystick, it will change to LOW.

Wiring a Thumb Joystick Module to an Arduino

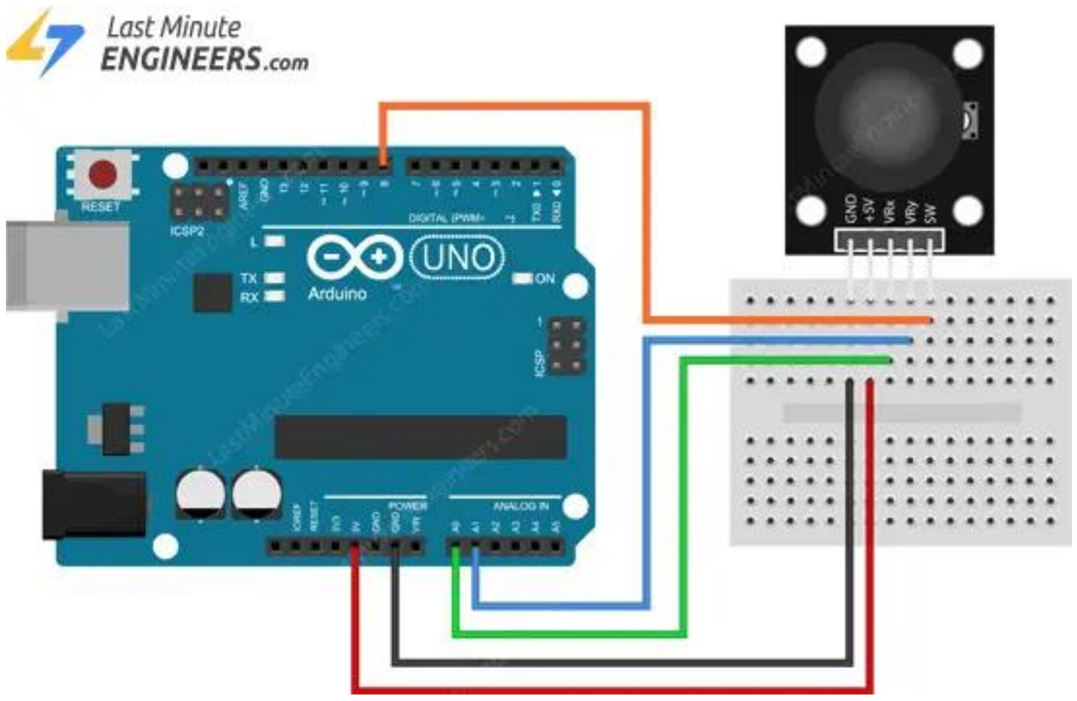
Let's connect the joystick module to the Arduino. It's pretty straightforward once you know which wires go where.

First, we'll connect the signals from the joystick to the Arduino. Connect the VRx pin (X-axis) to the Arduino's analog pin A0, and connect the VRy pin (Y-axis) to the Arduino's analog pin A1. To detect when the joystick's pushbutton is pressed, connect the SW pin to Arduino digital pin D8.

Next, we'll hook up the power. Connect the VCC pin to the Arduino's 5V output, and connect the GND pin to the Arduino's ground.

JOYSTICK MODULE	**ARDUINO**
GND	GND
VCC	5V
VRX	A0
VRY	A1
SW	8

The figure below shows how to wire the thumb joystick module to the Arduino.



Once everything is wired up correctly, you can begin writing code to read the joystick's position and detect button presses.

Arduino Example 1 – Reading the Joystick

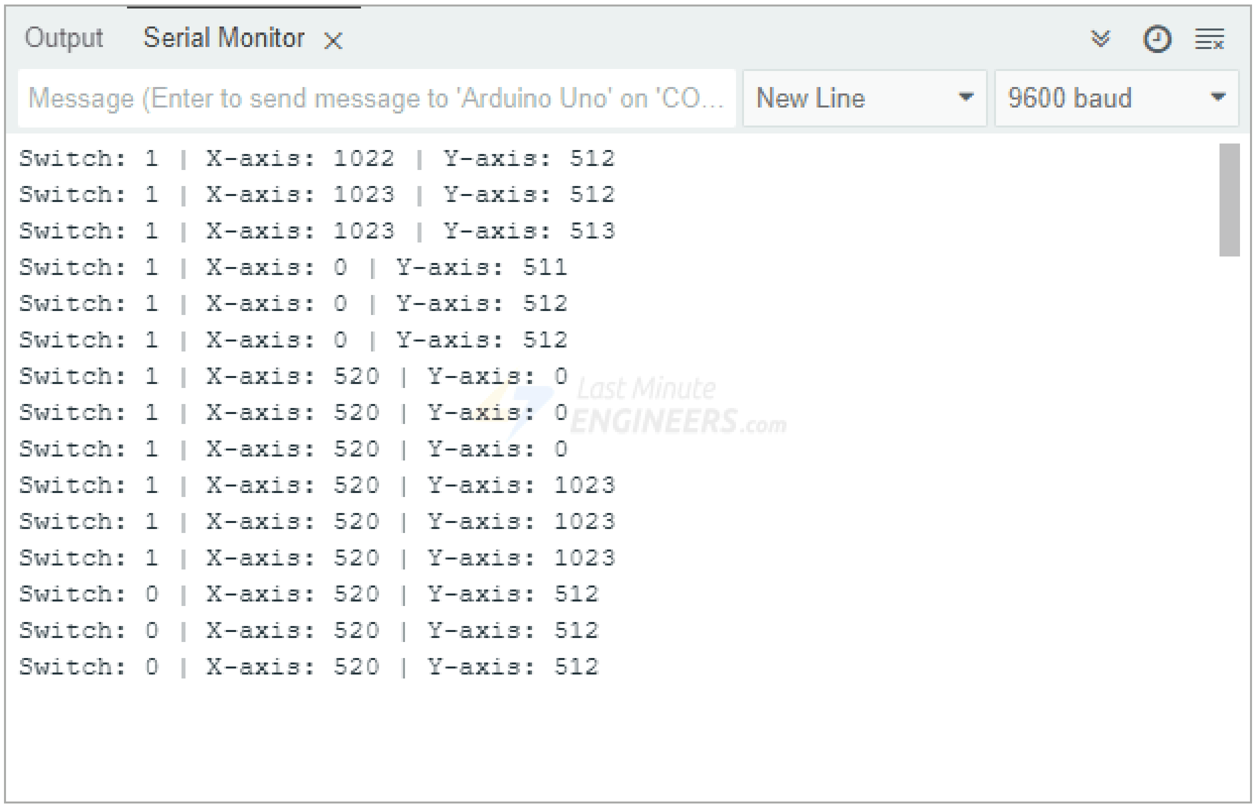
Here's a simple Arduino sketch that reads all the inputs from your joystick and displays them on your computer screen through the serial monitor.

```
// Arduino pin numbers
const int SW_pin = 8; // digital pin connected to switch output
const int X_pin = 0; // analog pin connected to X output
const int Y_pin = 1; // analog pin connected to Y output

void setup() {
  pinMode(SW_pin, INPUT);
  digitalWrite(SW_pin, HIGH);
  Serial.begin(9600);
}

void loop() {
  Serial.print("Switch:  ");
  Serial.print(digitalRead(SW_pin));
  Serial.print(" | ");
  Serial.print("X-axis: ");
  Serial.print(analogRead(X_pin));
  Serial.print(" | ");
  Serial.print("Y-axis: ");
  Serial.print(analogRead(Y_pin));
  Serial.println(" | ");
  delay(200);
}
```

Once you upload the sketch, you should see the following output appear on the serial monitor.



```
Switch: 1 | X-axis: 1022 | Y-axis: 512
Switch: 1 | X-axis: 1023 | Y-axis: 512
Switch: 1 | X-axis: 1023 | Y-axis: 513
Switch: 1 | X-axis: 0 | Y-axis: 511
Switch: 1 | X-axis: 0 | Y-axis: 512
Switch: 1 | X-axis: 0 | Y-axis: 512
Switch: 1 | X-axis: 520 | Y-axis: 0
Switch: 1 | X-axis: 520 | Y-axis: 0
Switch: 1 | X-axis: 520 | Y-axis: 0
Switch: 1 | X-axis: 520 | Y-axis: 1023
Switch: 1 | X-axis: 520 | Y-axis: 1023
Switch: 1 | X-axis: 520 | Y-axis: 1023
Switch: 0 | X-axis: 520 | Y-axis: 512
Switch: 0 | X-axis: 520 | Y-axis: 512
Switch: 0 | X-axis: 520 | Y-axis: 512
```

Last Minute ENGINEERS.com

Code Explanation:

At the beginning, we tell the Arduino which pins are connected to our joystick. We're using digital pin 8 for the button that's built into the joystick. For tracking the joystick's movement, we use analog pins A0 and A1. Pin A0 measures left and right movement (X-axis), while pin A1 measures up and down movement (Y-axis).

```
// Arduino pin numbers
const int SW_pin = 8; // digital pin connected to switch output
const int X_pin = 0; // analog pin connected to X output
const int Y_pin = 1; // analog pin connected to Y output
```

In the setup() function, we prepare the button to be read properly. We first configure the button pin as an input. We also activate Arduino's internal pull-up resistor using `digitalWrite(pin, value)` function. This makes sure that when the button isn't pressed, the Arduino reads a value of 1 (HIGH), and when it is pressed, it reads a value of 0 (LOW). Finally, we start communication between the Arduino and the computer so we can see the joystick readings in the Serial Monitor.

```
pinMode(SW_pin, INPUT);
digitalWrite(SW_pin, HIGH);
Serial.begin(9600);
```

In the loop() function, we continuously read the joystick's button state and position, then display the values on the Serial Monitor. First, we check if the button is pressed by using `digitalRead(SW_pin)`. If the function returns 1, it means the button is not pressed, and if it returns 0, it means the button is being pressed.

Next, we read the joystick's position on the X and Y axes using `analogRead(X_pin)` and `analogRead(Y_pin)`. These functions give us values between 0 and 1023, depending on the joystick's movement. The closer the value is to 0, the more the joystick is pushed in one direction, while a value near 1023 means it's pushed in the opposite direction. These readings are then printed on the Serial Monitor in an organized way, so you can easily see how the values change as you move the joystick.

Finally, we add a short delay of 200 milliseconds before taking the next set of readings to prevent the display from updating too fast, making it easier to observe the joystick's behavior.

```
Serial.print("Switch:  ");
Serial.print(digitalRead(SW_pin));
Serial.print(" | ");
Serial.print("X-axis: ");
Serial.print(analogRead(X_pin));
Serial.print(" | ");
Serial.print("Y-axis: ");
Serial.print(analogRead(Y_pin));
Serial.println(" | ");
delay(200);
```

REFERENCE:

https://lastminuteengineers.com/joystick-interfacing-arduino-processing/