Left Sidebar

Get manual control of all the motors on QuadBot

Overview

Single Motor Control

Read button

Read button pressed

Count button presses

Modulo Function

Bringing it all together

Difficulty rating: 4

Fun Rating: 3

Time taken: 25 mins

Supporting activity - Motor Moving

Motor Tweaking

Get manual control over all the motors and be able to tweak them individually using the dial



Making the motors move in code is great, but it would be quite hard to do things like make robot poses that way. Let’s add manual control using the motors.

**Single Motor Control**

To control one motor is quite simple. We read the value of the dial, map it to between 0 and 180, then write it to the motor, like below...

#include <QuadBot.h> *//Include the QuadBot Library*

int dial = 0; //A variable to hold the motor number

Int angle = 45; //A variable to hold the desired angle

void setup(){

QuadBot.begin() *//Initialize QuadBot*

}

void loop(){

dial = QuadBot.readDial(); //Read the value of the dial and store it

angle = map(dial,0,1023,0,180); //Map the dial value to maximum 180 and store it

QuadBot.setMotor(0, angle); //Set the angle of the motor to the mapped dial value

}

Simple enough, let’s add some button control in there for good measure!

**Read button**

Ok there’s 8 motors, so every time the button is pressed we want to increment a motor variable. To do that we need some way of detected when the button was pressed. This is a bit different to just reading it’s current state. We now need to know it’s current state, and it’s previous state, to see if it was pressed.

Here’s some simple code to do that.

int newButtonState = LOW;

int oldButtonState = LOW;

void setup() {

QuadBot.begin();

}

void loop() {

newButtonState = QuadBot.readButton(); //Read the value of button

if (newButtonState != oldButtonState) { //If the button is different to its previous value...

Serial.println("SOMETHING HAPPENED"); //Print “SOMETHING HAPPENED” to the serial monitor

}

delay(50);

oldButtonState = newButtonstate; //Assign the current button state to be the old button state for the next loop.

}

So if you ran that and opened a serial monitor, you’ll see that QuadBot says “SOMETHNG HAPPENED” any time the button is pressed or released.

**Read button pressed**

Let’s add in an extra if statement to detect only when the button is pressed…

int newButtonState = LOW;

int oldButtonState = LOW;

void setup() {

QuadBot.begin();

}

void loop() {

newButtonState = QuadBot.readButton(); //Read the value of button

if (newButtonState != oldButtonState) { //If the button is different to its previous value…

if(newButtonState == LOW){

Serial.println("BUTTON PRESSED"); //Print “BUTTON PRESSED” to the serial monitor

}

}

delay(50);

oldButtonState = newButtonstate; //Assign the current button state to be the old button state for the next loop.

}

Now we can detect a button press, let’s increment a counter every time the button is pressed.

**Count button presses**

With an incremental counter we can now count how many times the button as been pressed.

int newButtonState = LOW;

int oldButtonState = LOW;

int numberOfPresses = 0;

void setup() {

QuadBot.begin();

}

void loop() {

newButtonState = QuadBot.readButton(); //Read the value of button

if (newButtonState != oldButtonState) { //If the button is different to its previous value…

if(newButtonState == LOW){ //If the button has just been pressed, the value should be LOW

numberOfPresses++; //Increment the number of presses

}

}

delay(50);

oldButtonState = newButtonstate; //Assign the current button state to be the old button state for the next loop.

Serial.println(numberOfPresses); //Print out how mnay times the button has been pressed.

}

**Modulo Function**

Ok, you’re nearly there, great job! One problem though. If you kept pressing the button, the numberOfPresses value would just keep increasing. However to use it to set the motor, we want the value to “wrap around” back to 0 when it reaches value 7. How can we do that?

Well there’s a handy function called modulo that will handle this for us. It will cause our value to wrap around at 7 back to 0. Here we go...

How does the module function work? Read our basic maths tutorial.

int newButtonState = LOW;

int oldButtonState = LOW;

int numberOfPresses = 0;

int motorNumber = 0;

void setup() {

QuadBot.begin();

}

void loop() {

newButtonState = QuadBot.readButton(); //Read the value of button

if (newButtonState != oldButtonState) { //If the button is different to its previous value…

if(newButtonState == LOW){ //If the button has just been pressed, the value should be LOW

numberOfPresses++; //Increment the number of presses

}

}

delay(50);

oldButtonState = newButtonstate; //Assign the current button state to be the old button state for the next loop.

motorNumber = numberOfPresses%8 //Take the modulo 8 of numberOfPresses

Serial.println(motorNumber); //Print out how mnay times the button has been pressed.

}

**Bringing it all together**

Now it’s simple a case of adding in our previous code to set the motor.

int newButtonState = LOW;

int oldButtonState = LOW;

int numberOfPresses = 0;

int motorNumber = 0;

int dial;

int angle;

void setup() {

QuadBot.begin();

}

void loop() {

newButtonState = QuadBot.readButton(); //Read the value of button

if (newButtonState != oldButtonState) { //If the button is different to its previous value…

if(newButtonState == LOW){ //If the button has just been pressed, the value should be LOW

numberOfPresses++; //Increment the number of presses

}

}

delay(50);

oldButtonState = newButtonstate; //Assign the current button state to be the old button state for the next loop.

motorNumber = numberOfPresses%8 //Take the modulo 8 of numberOfPresses

dial = QuadBot.readDial(); //Read the value of the dial and store it

angle = map(dial,0,1023,0,180); //Map the dial value to maximum 180 and store it

QuadBot.setMotor(motorNumber, angle); //Set the angle of the motor to the mapped dial value

}

And voila! You can now use the dial and button to set specific manual poses for QuadBot, good job!

**Conclusion**

This program was a bit more complex than all the others we’ve dealt with so far. This is because it stores previous data and uses memory. We’ll see this will become much more important in our upcoming activities.