Left Sidebar

Create a simple synth with QuadBot

Overview

Tone output

Frequencies

Synthin’ it together

Button Control

Conclusion

Difficulty rating: 2

Fun Rating: 2

Time taken: 15 mins

Supporting activity - QuadBot Colour

Dimmer Control

Use your analog input skills to make funky sounds on QuadBot.



We’ll combine the dial and the buzzer on QuadBot to output a buzz with varying frequency, just like a basic synth machine!

**Tone Output**

When a voltage signal of varying frequency is applied to the buzzer it will output a sound at that frequency. We can do this one QuadBot using the buzz() function like this…

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

int freq = 4000; // Declare a variable to store the frequency value

void setup(){

QuadBot.begin() *//Initialize QuadBot*

}

void loop(){

buzz(freq); //Send a signal of the desired frequency to the buzzer

}

**Frequencies**

Frequency of a sound is basically its pitch; how high or low it sounds. A higher frequency means the sound is higher, a lower frequency means the sound is lower. For humans, our speaking range is 85 Hz to 255 Hz. Our audible range is massive though from 20Hz to 20,000Hz, but most older folks can only hear up to 10,000Hz



That means we must select values between 50Hz and 10,000Hz so we can hear QuadBot sing!

**Synthin’ it together**

Ok let’s link the dial and buzzer together and make our synth, just like this…

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

int freq; // Declare a variable to store the frequency value

int dial; //A variable to store the value of the dial

void setup(){

QuadBot.begin() *//Initialize QuadBot*

}

void loop(){

dial = readDial(); //Read the value of the dial

freq = map(dial,0,1023,50,10000); //Calculate frequency by mapping the dial between 0 and 10000.

buzz(freq); //Send a signal of the desired frequency to the buzzer

}

**Button Control**

In this final step let’s make QuadBot output only when the button is pressed, so we can make our own musical tunes! We’ll add the if/else logic into the previous example we had…

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

int freq; // Declare a variable to store the frequency value

int dial; //A variable to store the value of the dial

void setup(){

QuadBot.begin() *//Initialize QuadBot*

}

void loop(){

dial = readDial(); //Read the value of the dial

freq = map(dial,0,1023,50,10000); //Calculate frequency by mapping the dial between 0 and 10000.

if(QuadBot.readButton()){

buzz(freq); //Send a signal of the desired frequency to the buzzer

}

else{

buzz(0); //Send out no signal to the buzzer

}

}

**Conclusion**

And there you have it! Some simple code and you’ve turned QuadBot into a synth! Now let’s move on to motors!

The microcontroller reads this voltage as a number between 0 and 1023. 0 volts means 0, and 5 volts means 1023. This might seem a bit strange, and it’s down to how microcontrollers store the voltage information.

Why does a microcontroller read an analog voltage as a number between 0 and 1023? Read our Input and Output tutorial.

Now we know that let’s look at how to actually use this…

**Reading the dial**

We can use the readDial() function from the QuadBot.h library to return the current value of the dial, like this…

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

int dial; *//Declare a variable to store the value of the dial*

void setup(){

QuadBot.begin() *//Initialize QuadBot*

}

void loop(){

dial = QuadBot.readDial() /*/Read the value of the dial and store it in “dial”*

}

The readDial() function returns the value from the dial (between 0 and 1023) We can use this variable in our code to make decisions.

To use this value to control the LED brightness, we’ll need to give the value into the QuadBot.setLEDs(R,G,B) function.

There’s a problem though. We know that readDial() returns a value between 0 and 1023, however we know that setLEDs() expects values for R,G and B between 0 and 255! To solve this problem we can use the map() function...

**The Map Function**

The Map function is one of those sort of functions you just can’t live without. It’s built into the Arduino environment and you can read about it in detail Arduino’s page [here](https://www.arduino.cc/en/reference/map). The map function takes one number from one range, and scales it into another number from another range.

To understand this, let’s take an example. Say I’ve got a grade of 0.73 out of a range between 0 and 1. How can I express that as a grade between 0 and 100. This is simple, we can just multiple the original grade by 100. So 0.73 between 0 and 1 is equivalent to 73 between 0 and 100. That’s a very simple mapping.

But how can we do this for more complicated mappings?

* Map 34 from range 5-85 to range 0 to 100
* Map 0.1 from range -4 to 4 to range 80 to 90
* Map 512 from range 0 to 1023 to range 0 to 255

We could go through the math, but we don’t have to, we can use the Arduino map() function. The function takes five inputs and returns the new, mapped value. We use it like this…

int newVal = map(oldVal, oldLow, oldHigh, newLow, newHigh);

So to map value “dial” in the range of 0 to 1023, to the range of 0 to 255 we do the following.

map(dial, 0, 1023, 0, 255);

**Combining what we know**

Let’s bring it all together to create our dimmer switch. Let’s read the dial, map the value, then use it to set the LEDs…

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

int dial; *//Declare a variable to store the value of the dial*

int level; // Declare a variable to store the brightness value

void setup(){

QuadBot.begin() *//Initialize QuadBot*

}

void loop(){

dial = QuadBot.readDial() *//Read the value of the dial and store it in “dial”*

level = map(dial, 0,1023, 0,255); //map the value to bet

setLEDs(level, level, level); //Set the brightness according to the level

}

And hurrah! You’ve now created a white dimmer lamp with QuadBot. Try using some maths to increase/decrease the brightness values for R, G and B. That way you can create a colored dimmer lamp!

**Conclusion**

Now you’re getting somewhere! Well done, you’ve combined input with output and are ready to move on to some more advanced concepts, go you!