

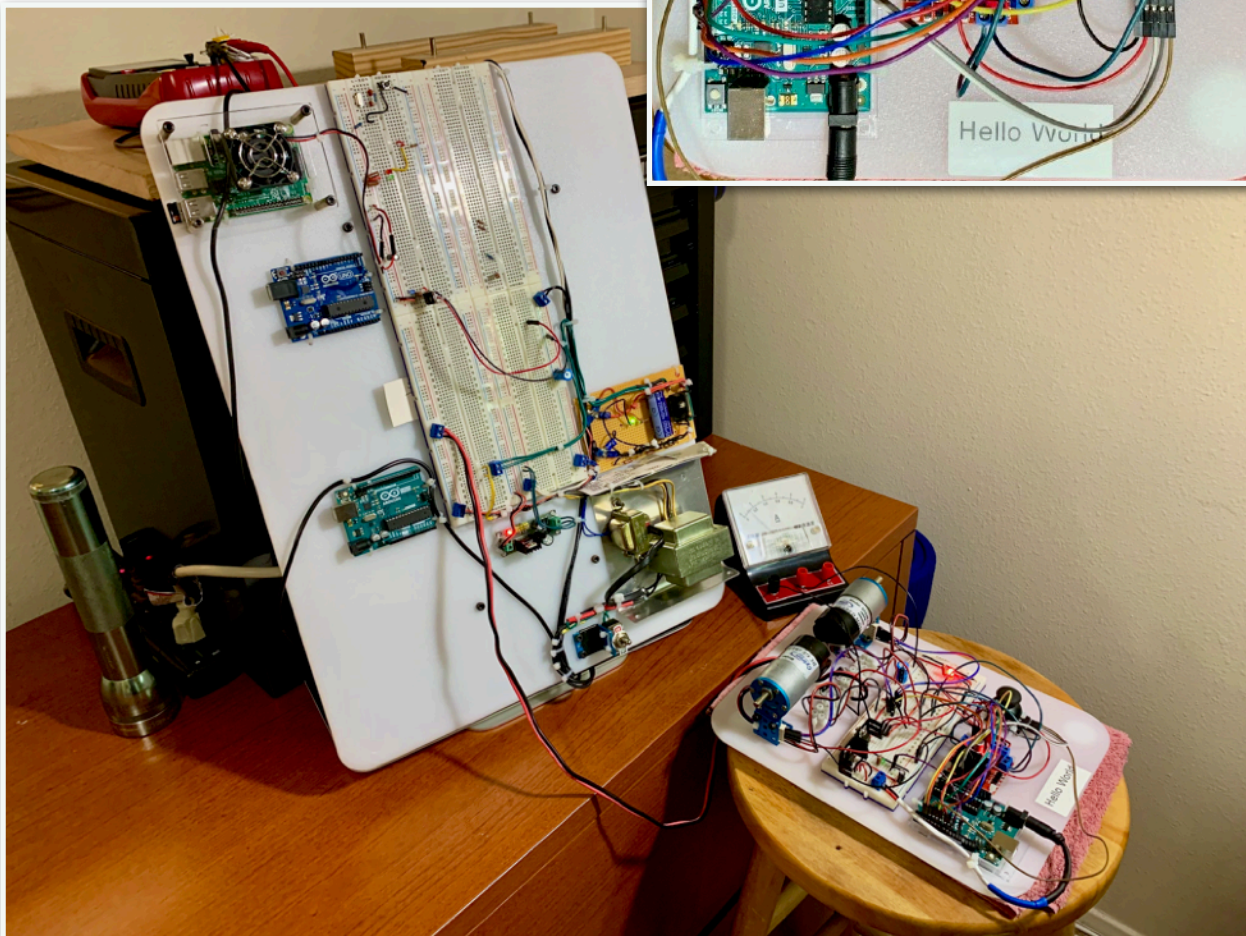
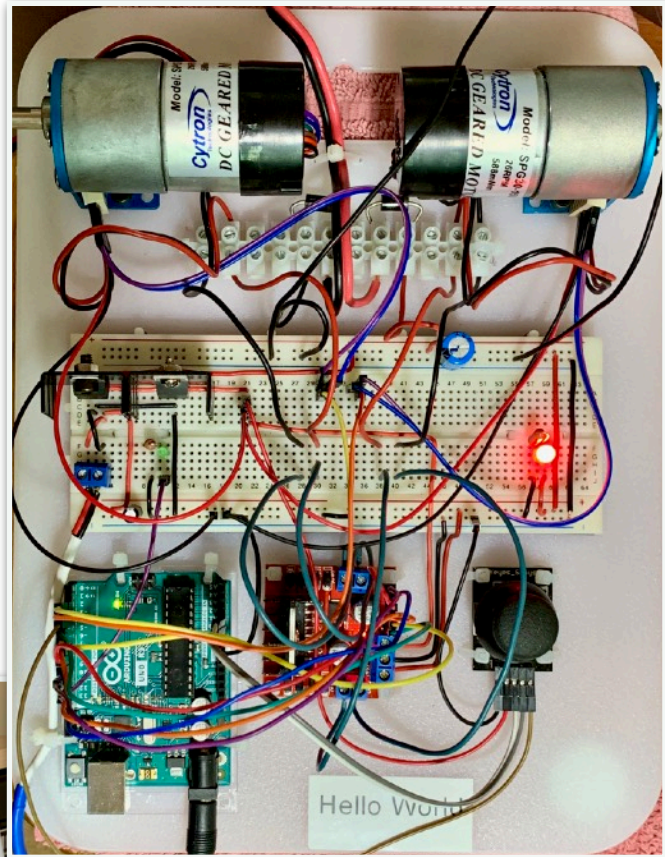
1009 L298N Supplemental

How to use the Joystick Algorithm and the L298N Class - A Teaching Experimental Guide

Manifest

- Cytron Technologies 12V DC Motors SPG30-150K 26 RPM (The higher gear ratio or lower rpm gives a higher torque value and also gives a higher encoder count.)
- **Thumb Joystick** (with 5-connections - Amazon online store)
- **Arduino Uno R3** (Arduino online store)
- **L298N Motor Driver Controller Board** (Red is the color... Amazon)
- 7805 5-Volt Regulator (Amazon)
- 7809 9-Volt Regulator (Amazon)
- 18VAC Transformer 2 to 3 Amp 115/230VAC Primary (Completely Enclosed in a Metal Shielding with a 3 wire output with center tap... This is a popular hobby transformer... Try Amazon.)
- Breadboard with passive electronic components. (Amazon)

Parts listed may not be Pb Lead-Free RoHS Safe...

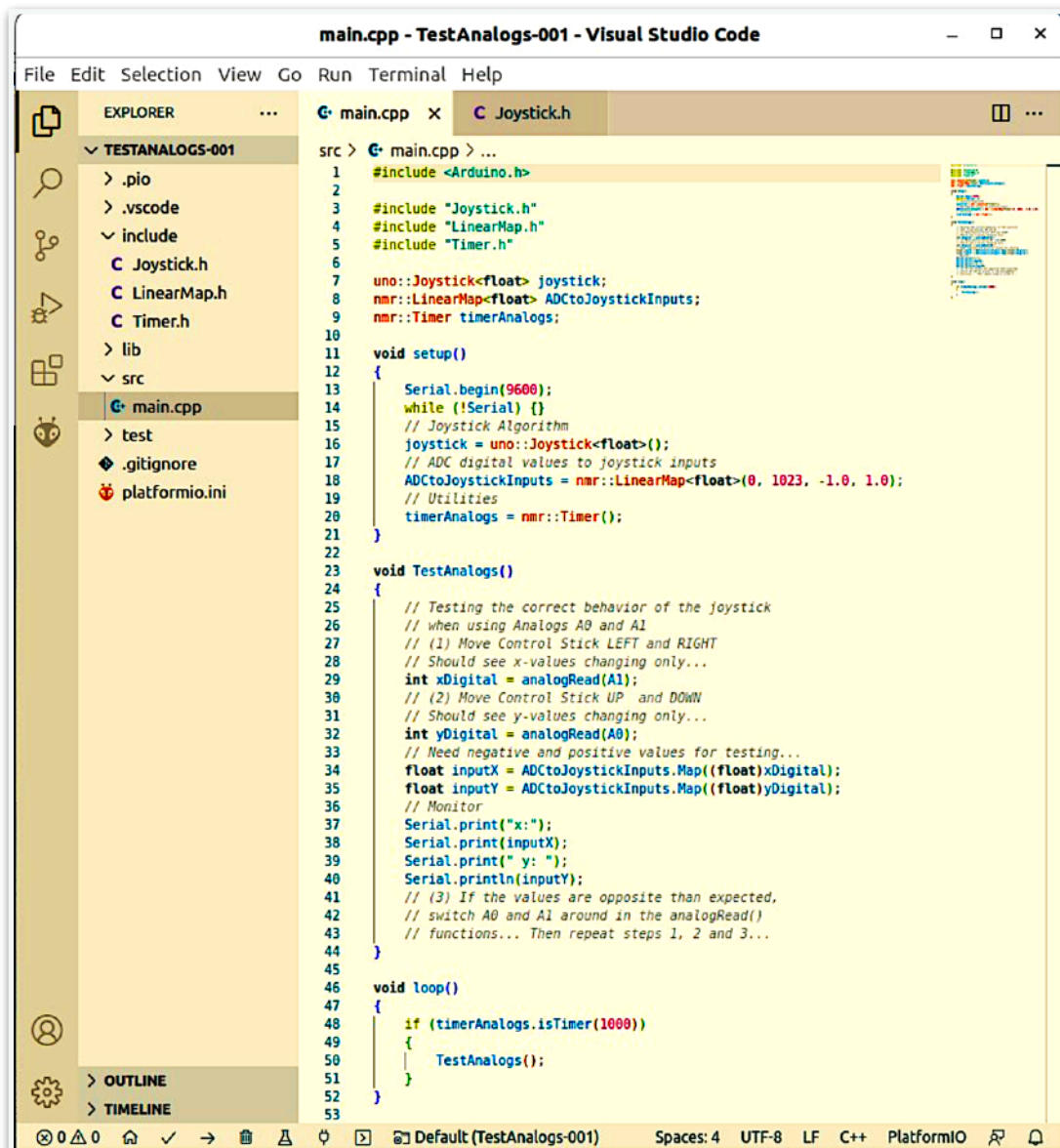


Introduction

The problem is wiring all this together as shown. There are eight different ways of connecting wires from the L298N Motor Driver Controller Board to the two Motors. There are two different ways of connecting wires from the Thumb Joystick to the Arduino Uno. All of which can change the behavior of the motors output.

Joystick

Before wiring the motors and the L298N, setup the Thumb Joystick and the Arduino Uno. When positioning the joystick permanently, its orientation can be whatever position. On the other hand, setting-up the horizontal (x) and vertical (y) data can be a little tricky. Sometimes the x-values are



```
main.cpp - TestAnalog-001 - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
TESTANALOGS-001
  .pio
  .vscode
  include
    Joystick.h
    LinearMap.h
    Timer.h
  lib
  src
    main.cpp
  test
  .gitignore
  platformio.ini

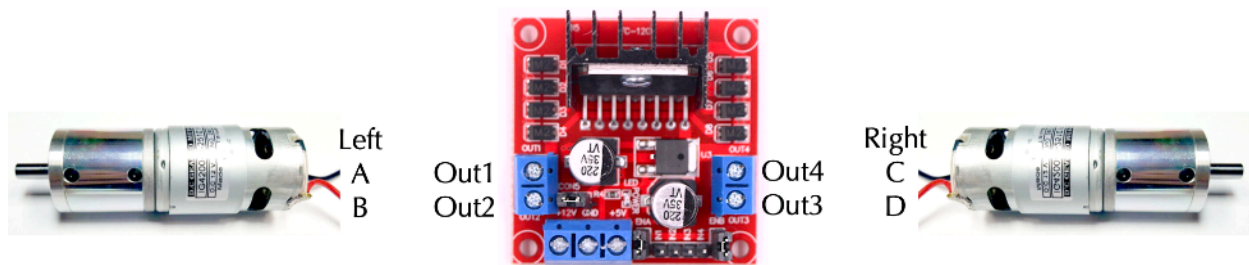
src > main.cpp > ...
1  #include <Arduino.h>
2
3  #include "Joystick.h"
4  #include "LinearMap.h"
5  #include "Timer.h"
6
7  uno::Joystick<float> joystick;
8  nmr::LinearMap<float> ADCtoJoystickInputs;
9  nmr::Timer timerAnalog;
10
11 void setup()
12 {
13   Serial.begin(9600);
14   while (!Serial) {}
15   // Joystick Algorithm
16   joystick = uno::Joystick<float>();
17   // ADC digital values to joystick inputs
18   ADCtoJoystickInputs = nmr::LinearMap<float>(0, 1023, -1.0, 1.0);
19   // Utilities
20   timerAnalog = nmr::Timer();
21 }
22
23 void TestAnalog()
24 {
25   // Testing the correct behavior of the joystick
26   // when using Analogs A0 and A1
27   // (1) Move Control Stick LEFT and RIGHT
28   // Should see x-values changing only...
29   int xDigital = analogRead(A1);
30   // (2) Move Control Stick UP and DOWN
31   // Should see y-values changing only...
32   int yDigital = analogRead(A0);
33   // Need negative and positive values for testing...
34   float inputX = ADCtoJoystickInputs.Map((float)xDigital);
35   float inputY = ADCtoJoystickInputs.Map((float)yDigital);
36   // Monitor
37   Serial.print("x:");
38   Serial.print(inputX);
39   Serial.print(" y: ");
40   Serial.println(inputY);
41   // (3) If the values are opposite than expected,
42   // switch A0 and A1 around in the analogRead()
43   // functions... Then repeat steps 1, 2 and 3...
44 }
45
46 void loop()
47 {
48   if (timerAnalog.isTimer(1000))
49   {
50     TestAnalog();
51   }
52 }
53
```

not horizontal. The TestAnalog() function shows how to do this. Simply follow the steps... Read the fine details within the function above... Once mastered, personally it was easy to remember... The joystick testing and setup is important and it should be completed before attempting to wire the motors and the L298N. Once the setup is complete never never touch the wires nor the analog readings. Moving onto the bigger project...

L298N

When it comes to wiring the Motors and the L298N, this becomes a nightmare with the first experimentation by young engineers and even more so when the decide to switch around the analogs A0 and A1 thinking that it might be the problem. This is especially true when getting the motors to behave correctly with the joystick algorithm.

There are 8 combinations but 4 of the combinations involving the L298N seem equal at least until the polarities of each motor is considered. The colored bars indicate those combinations being



8 Combinations		Left Motor		L298N		Right Motor	BIT	3-Bits	BOOL
1	Orange	A	B	Out1	Out3	C	0	000	FFF
				Out2	Out4	D			
2	Blue	A	B	Out2	Out3	C	1	001	FFT
		B	A	Out1	Out4	D			
3	Blue	A	B	Out1	Out4	C	2	010	FTF
		B	A	Out2	Out3	D			
4	Orange	A	B	Out2	Out4	C	3	011	FTT
		B	A	Out1	Out3	D			
5	Teal	A	B	Out3	Out1	C	4	100	TFF
		B	A	Out4	Out2	D			
6	Pink	A	B	Out3	Out2	C	5	101	TFT
		B	A	Out4	Out1	D			
7	Pink	A	B	Out4	Out1	C	6	110	TTF
		B	A	Out3	Out2	D			
8	Teal	A	B	Out4	Out2	C	7	111	TTT
		B	A	Out3	Out1	D			

similar. Once the correct combination is selected, the motors will behave in an expected manner while using the joystick algorithm. These combinations are independent of the analogs A0 and A1 rather or not if the analogs were switched around.

In the L298N Class, there is a method called **Bits**. The programmer or engineer can enter a combination of three boolean values as shown on the far right of the table. This method will be further explained. Although these values may not match exactly to the motors and L298N depicted in the software, it demonstrates their usefulness. Notice how the bit number, the binary bits, and the boolean values which are equal compare with one another. Try experimenting with the Bitwise.h Class as a stand alone program. These boolean values are used in the L298N Class with the method called **Bits**.

```

L298N.h - SJUL-004 - Visual Studio Code
File Edit Selection View Go Run Terminal Help

main.cpp  C L298N.h  C Joystick.h  Workspace Trust  platformio.ini

include > C L298N.h
140 pinMode(L_RightMotorIN1, OUTPUT);
141 pinMode(L_RightMotorIN2, OUTPUT);
142 pinMode(L_RightMotorPWM, OUTPUT);
143 PowerDownL298N();
144 }
145
146 // Follow instructions in Supplemental Article...
147 void L298N::Bits(MotorBits bitsValue)
148 {
149     int localBitsValue = (int)bitsValue - (int)MotorBits::motorsFFF;
150     // There are 8 combinations for Reverse-Inputs and Direction-Motors
151     // The Boolean Order
152     // (1) Reverse Inputs: T/F (Bit-2)
153     // (2) Direction Left Motor: T/F (Bit-1)
154     // (3) Direction Right Motor: T/F (Bit-0)
155     // Check conditions: FFF, FFT, FTF, FTT, TFF, TFT, TTF, TTT
156     // Local-Bits-Value: 0 1 2 3 4 5 6 7
157     // Used 3 Bit Numbers: 000 001 010 011 100 101 110 111
158     // DO NOT USE the private constants belonging to MotorBits...
159     // Changing the truth table is much easier than switching the actual
160     // wires around on the L298N module.... For my setup, FFF was used...
161     //
162     // Change the Direction Right Motor (Bit-0)
163     L_bitRightFlag = L_bWise.IsBitNumberSetToBitsValue((int)MotorBits::private_bitZero, localBitsValue);
164     // Change the Direction Left Motor (Bit-1)
165     L_bitLeftFlag = L_bWise.IsBitNumberSetToBitsValue((int)MotorBits::private_bitOne, localBitsValue);
166     // Reverse L298N Inputs (Bit-2)
167     L_bitReverseInputsFlag = L_bWise.IsBitNumberSetToBitsValue((int)MotorBits::private_bitTwo, localBitsValue);
168 }
169
170 // Used with a timer within loop()
171 void L298N::UpdateL298N(int PWM_LeftMotor, int PWM_RightMotor, bool PowerMotors = false)
172 {
173     // Reverse L298N Inputs (Bit-2 Flag)
174     if (L_bitReverseInputsFlag)
175     {
176         L_PWM_LeftMotor = PWM_LeftMotor;
177         L_PWM_RightMotor = PWM_RightMotor;
178     }
179 }

```

The **Bits** method is placed with the initialization of the L298N Class in the Arduino's **setup()** function in the **main.cpp** file. There are examples in how this method is used. Someone might ask the question as to which comes first. When reading from left to right, **bit-2** is on the left, then **bit-1** and the last is **bit-0**. Study the method above. The Simplified Joystick Uno L298N Repository are two folders **SJUL** and **JULE** inside the **src** folder is where the examples can be located in the **main.cpp** file.

L298N Class Bits() Method and Motor Movements Checklist

Use the **Simplified Joystick Uno L298N** in the Repository in the **SJUL** code folder at *MageMCU* at *Github* for the *Motors Movement Test*. In the *Motor Movements Checklist*, each line represents a single test beginning at Bit-0. Begin with the **MotorBits::motorsFFF** as shown below in the code as `motors.Bits(uno::MotorBits::motorsFFF)`. As a reminder, the wiring shown above in the images may not have the exact wiring in your project. Therefore, `motorsFFF` may not work as it did for me.

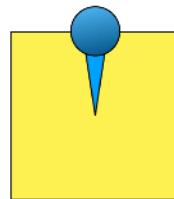
```

69 // ----- NEW METHOD
70 // There are 8 combinations for Reverse-Inputs and Direction-Motors
71 // The Boolean Order
72 // (1) Reverse Inputs: T/F
73 // (2) Direction Left Motor: T/F
74 // (3) Direction Right Motor: T/F
75 // Check conditions: FFF, FFT, FTF, FTT, TFF, TFT, TTF, TTT
76 // Bits Value:      0  1  2  3  4  5  6  7
77 // Used 3 Bit Numbers: 000 001 010 011 100 101 110 111
78 // DO NOT USE the private constants belonging to MotorBits...
79 // Changing the truth table is much easier than switching the actual
80 // wires around.... For my setup, FFF was used...
81 motors.Bits(uno::MotorBits::motorsFFF);

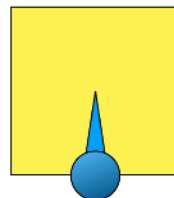
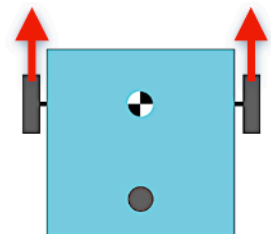
```

L298N Class - Bits() method

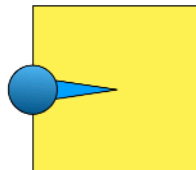
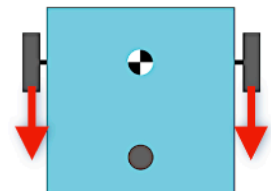
Motor Movements Checklist					
Pass (✓) - Fail (X)					
Bit	Bool	F	B	L	R
0	FFF				
1	FFT				
2	FTF				
3	FTT				
4	TFF				
5	TFT				
6	TTF				
7	TTT				



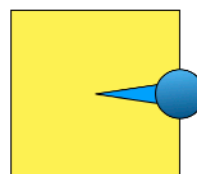
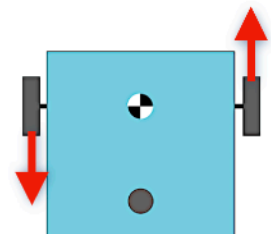
Forward
F



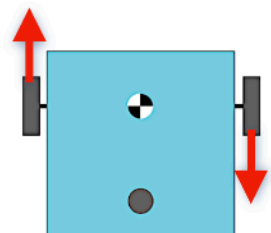
Backward
B



Left Turn
L



Right Turn
R



While testing for the correct boolean values, check off each item along each line for **F**, **B**, **L** and **R**. These letters represent the movement test for the motors. While testing, move the joystick as shown to test the motors movement for the correct motor directions as shown with the red arrows. If any of these movements fail, then the test for that particular boolean values also failed. Therefore proceed to the next line to test the next boolean values. Proceed until a test passes.

Last Note for Completeness

The revised code for both the *Joystick Class* and the *N298N Class* has not been fully tested. For example, the two added Getter Methods, *IsLeftForward()* and *IsRightForward()* may not output correctly depending on the orientation position of the Thumb Joystick. Also depending on the orientation of the Thumb Joystick, the analog readings for **A0** and **A1** may have to be set with a negative value. Usually I prefer testing new code over time to work out these possible *enigmatic* bugs. Please email me with any problems you might have with code. Thank you.

Carpenter Software
Jesse Carpenter
See Contact Info at carpentersoftware.com
published 20230115... Edited 20230119.
Article 1009 L298N Supplemental is a work in progress...
ALL RIGHTS RESERVED.

