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)
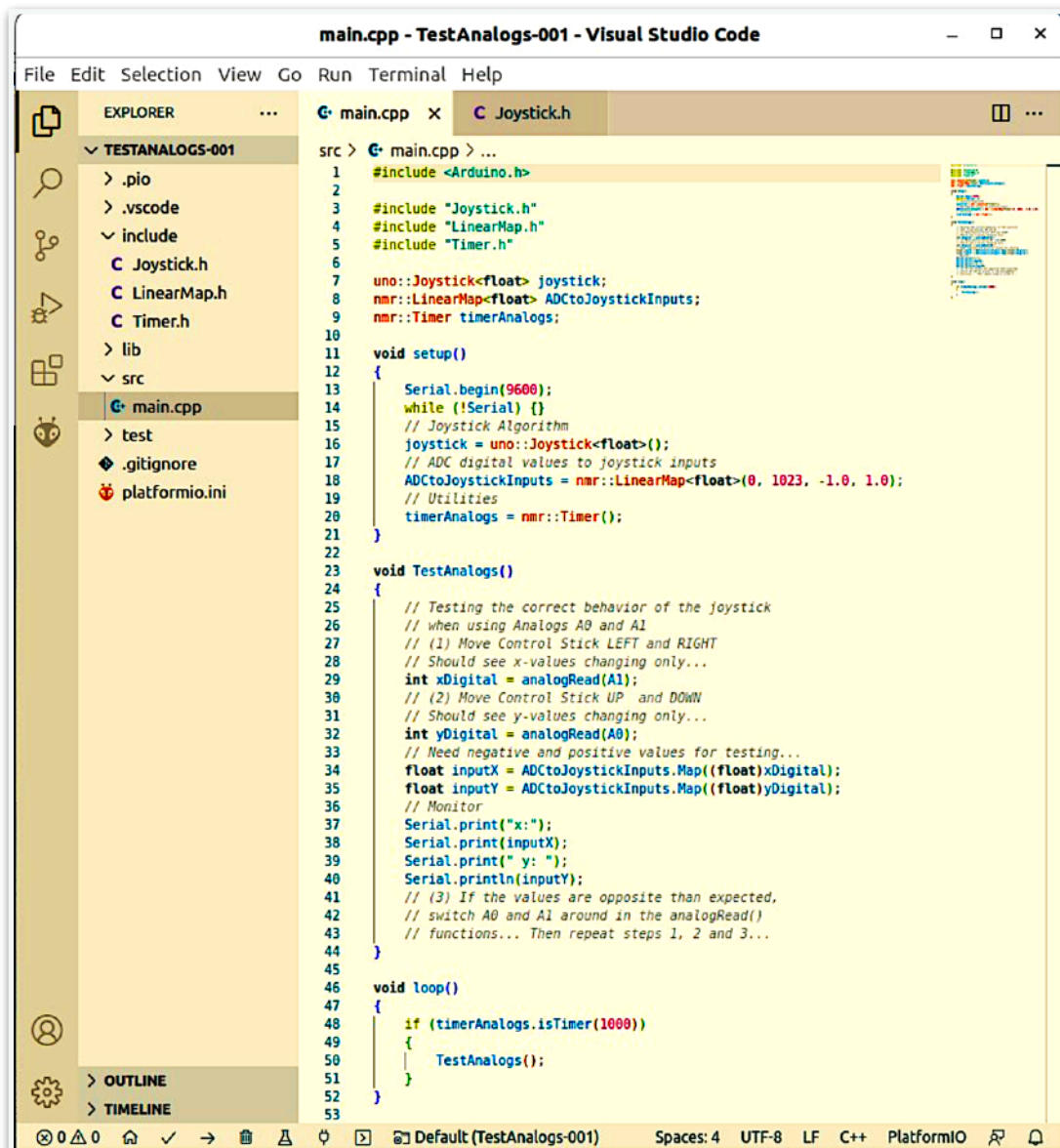
Parts listed may not be Pb Lead-Free RoHS Safe…
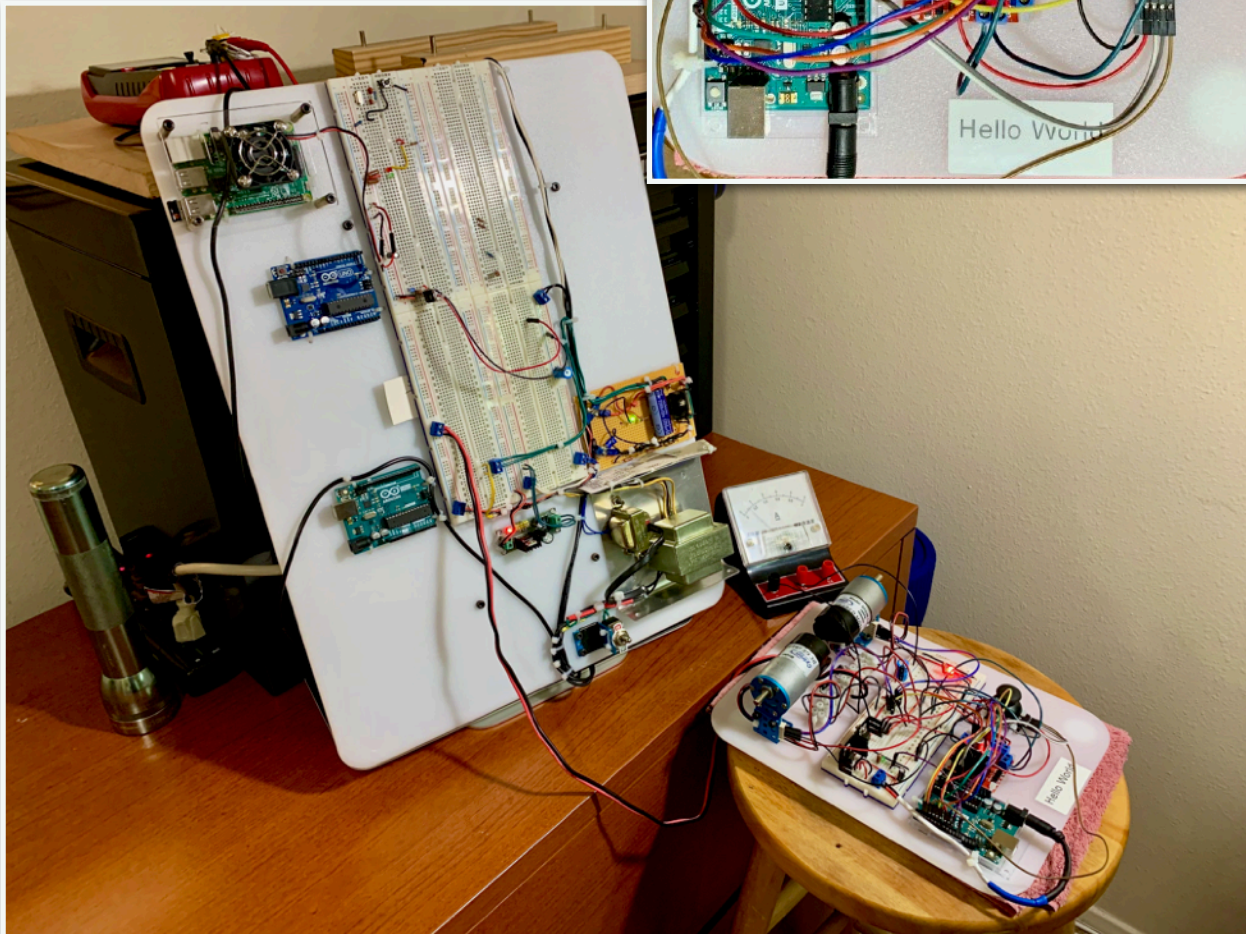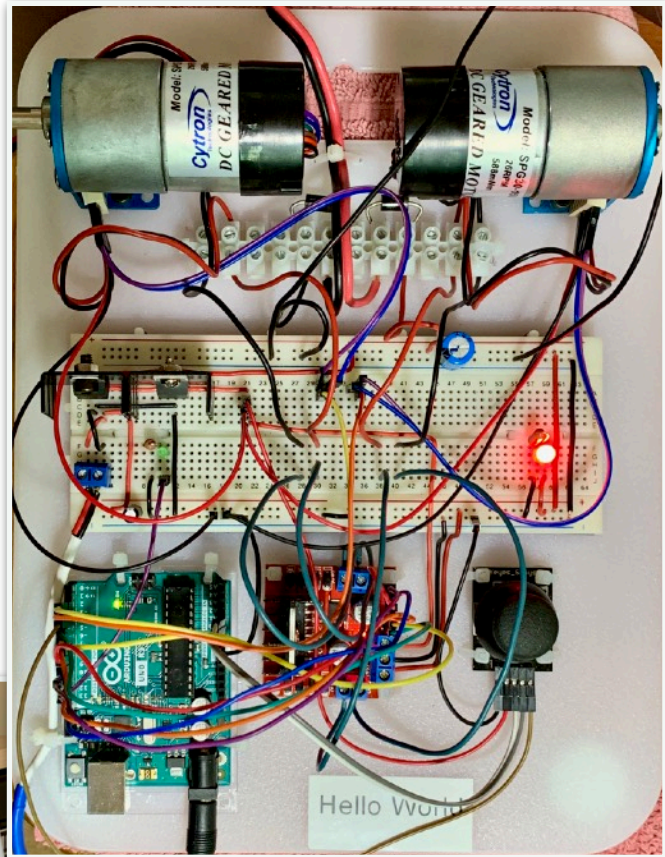
Introduction

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 - TestAnalogs-001 - Visual Studio Code                    _  □  ×

File  Edit  Selection  View  Go  Run  Terminal  Help

 EXPLORER          ···      main.cpp ×    C Joystick.h                         ···

 v TESTANALOGS-001          src > main.cpp > ...
    > .pio                  1   #include <Arduino.h>
    > .vscode               2
    v include               3   #include "Joystick.h"
       C Joystick.h         4   #include "LinearMap.h"
       C LinearMap.h        5   #include "Timer.h"
       C Timer.h            6
    > lib                   7   uno::Joystick<float> joystick;
    v src                   8   nmr::LinearMap<float> ADCtoJoystickInputs;
       main.cpp             9   nmr::Timer timerAnalogs;
    > test                  10
    .gitignore              11  void setup()
    platformio.ini          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      timerAnalogs = nmr::Timer();
                            21  }
                            22
                            23  void TestAnalogs()
                            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 (timerAnalogs.isTimer(1000))
                            49      {
                            50          TestAnalogs();
                            51      }
                            52  }
                            53

 ⊗0⚠0  ⌂  ✓  →  ☰  ⅄  ⌀  ▷    Default (TestAnalogs-001)    Spaces: 4  UTF-8  LF  C++  PlatformIO
```

not horizontal. The TestAnalogs() 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
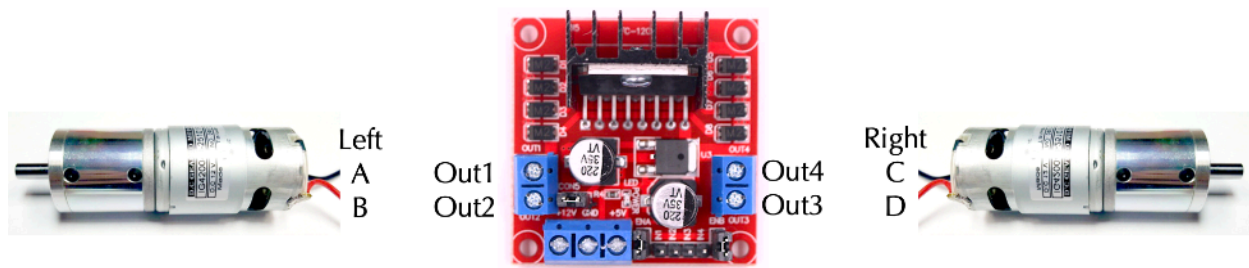
8 Combinations	Left Motor	L298N		Right Motor	BIT	3-Bits	BOOL
1	A	Out1	Out3	C	0	000	FFF
	B	Out2	Out4	D			
2	A	Out2	Out3	C	1	001	FFT
	B	Out1	Out4	D			
3	A	Out1	Out4	C	2	010	FTF
	B	Out2	Out3	D			
4	A	Out2	Out4	C	3	011	FTT
	B	Out1	Out3	D			
5	A	Out3	Out1	C	4	100	TFF
	B	Out4	Out2	D			
6	A	Out3	Out2	C	5	101	TFT
	B	Out4	Out1	D			
7	A	Out4	Out1	C	6	110	TTF
	B	Out3	Out2	D			
8	A	Out4	Out2	C	7	111	TTT
	B	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**.

The **Bits** method is placed with the initialization of the L298N Class in the of 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 **S**implified **J**oystick **U**no **L**298N Repository are two folders **SJUL** and **JULE** inside the **src** folder is where the examples can be located in the *main.cpp* file.