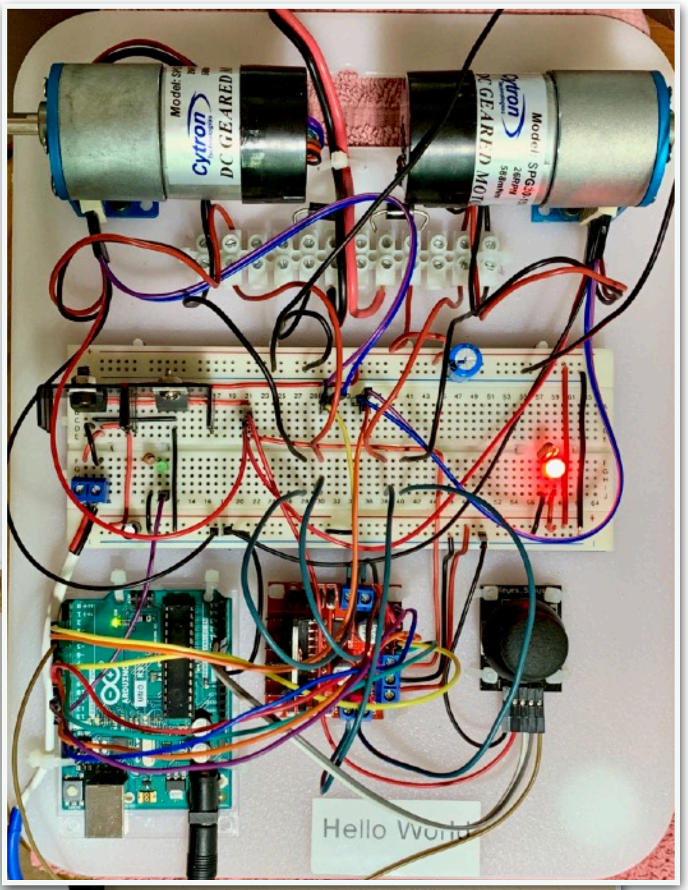
1009 L298N Supplemental

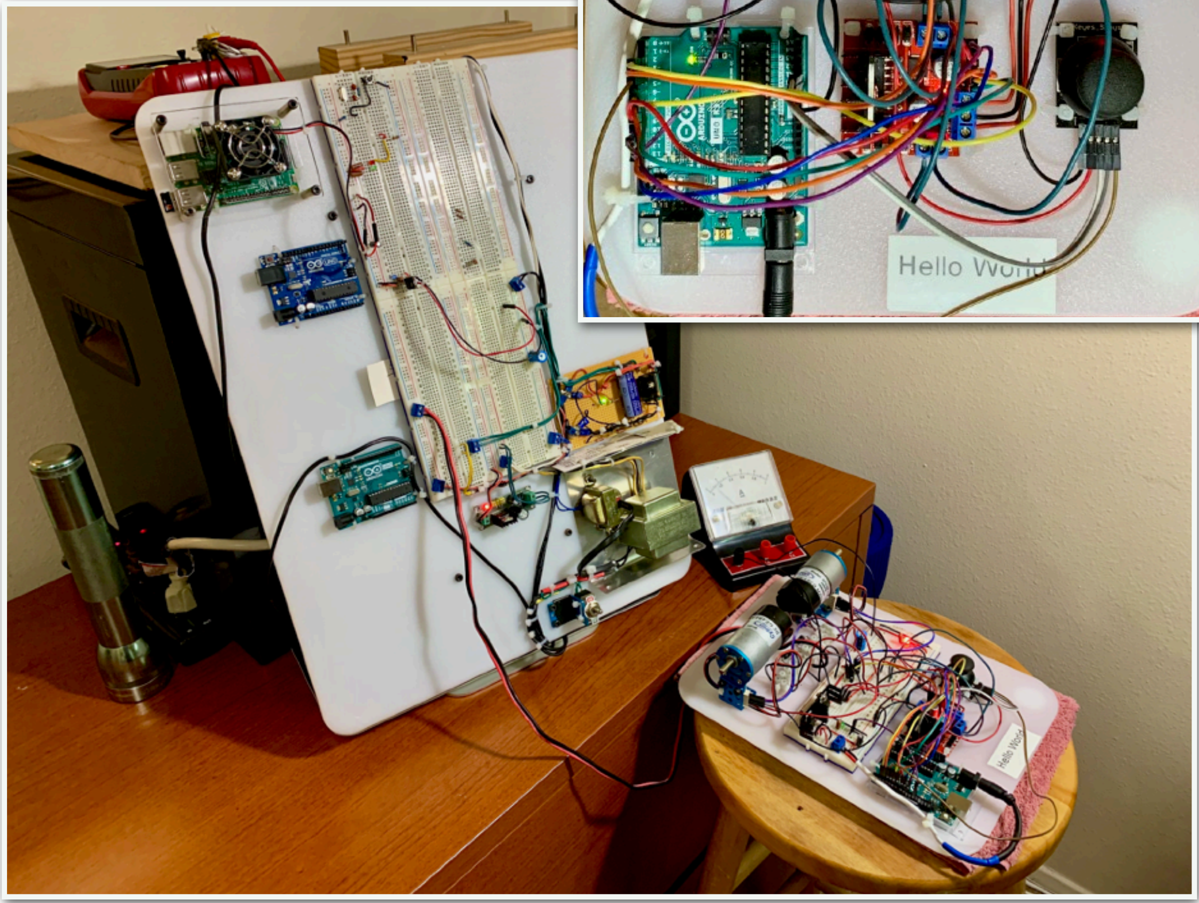
How to use the Joystick Algorithm and the L298N Class - An Experimental Teaching Guide

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…



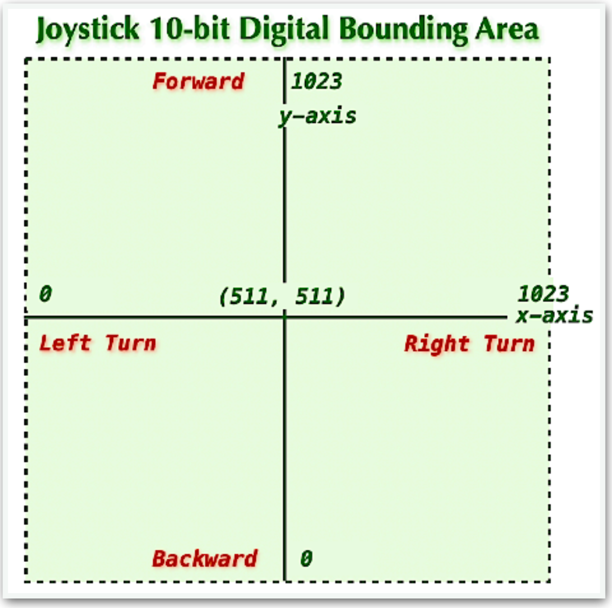


Introduction

The problem is physically wiring all this together as shown. There are eight different ways of connecting wires from the two Motors to the L298N Motor Driver Controller Board. 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.[1] On the other hand, setting-up the horizontal (x) and vertical (y) data can be a little tricky. Sometimes the x-values are not horizontal. Even worse, the digital values are reversed from 1023 to zero.



The Joystick Algorithm depends on how the inputs are read from Arduino's analog readings A0 and A1 when using the Joystick Class *UpdateInputs(InputX, InputY)* Method. In the Digital Bounding Area, the x-axis is ordered from the left as zero through to the right as 1023. The x-digital analog reading has to be in the same left to right order. You have to observe these values from the *Serial.print()* method while debugging your code. If x-digital reads in the reverse order then use the following equation:

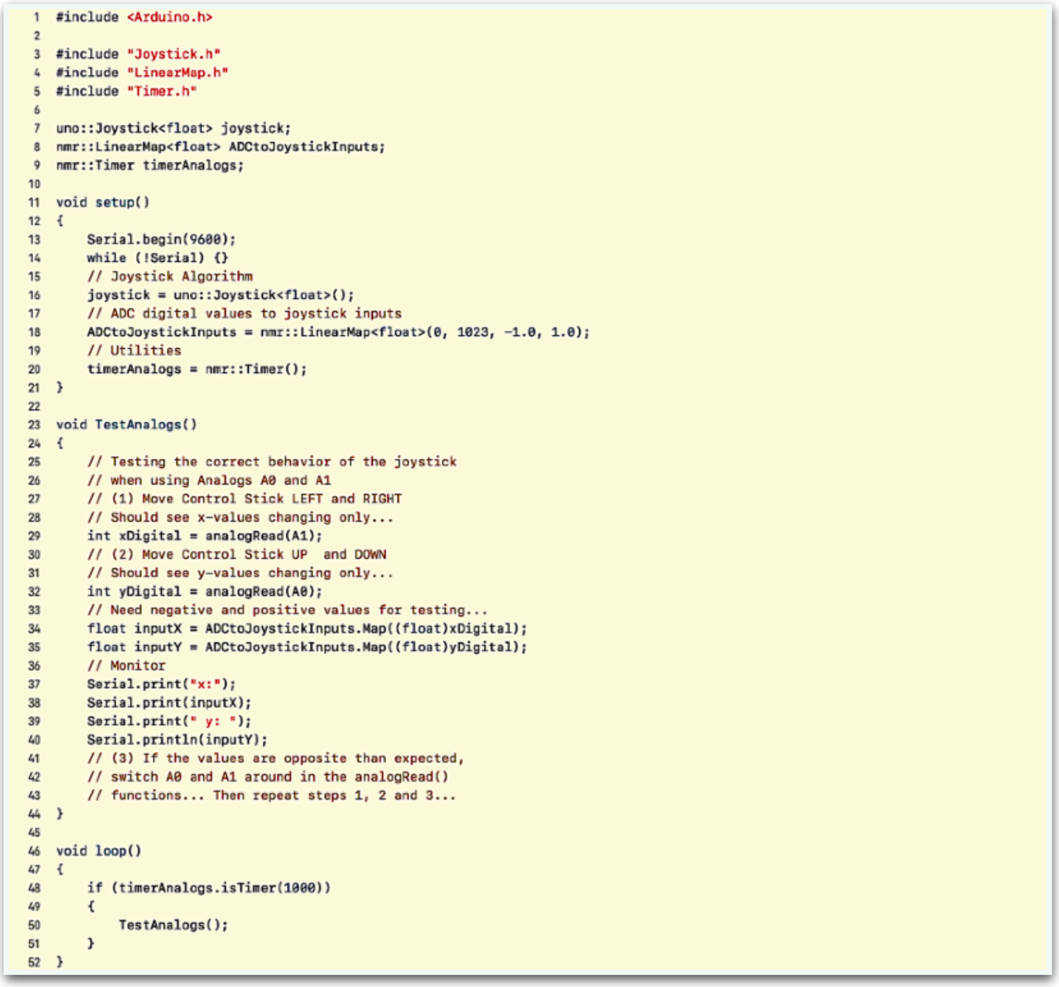
```
int xDigital = 1023 - analogRead(A1);
```

1. The author has not tested all the possibilities of the L298N and Joystick code and devices… The goal of the AI-IPP C++ compiled project is to auto-produce a C++ code into a single simplex of the L298N and Joystick code based on the Bits() method…

This is the same with the y-digital analog reading, it is ordered from the backward as zero through to the forward as 1023. The y-digital analog reading has to be in the same backward to forward order. Again these values can be observed from the *Serial.print()* method. If y-digital reads in the reverse order then use the following equation:

```
int yDigital = 1023 - analogRead(A0);
```

The TestAnalogs() function shows how to do this. Simply follow the steps… Read the details within the function below… 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 testing the wiring setup among the L298N driver and motors…

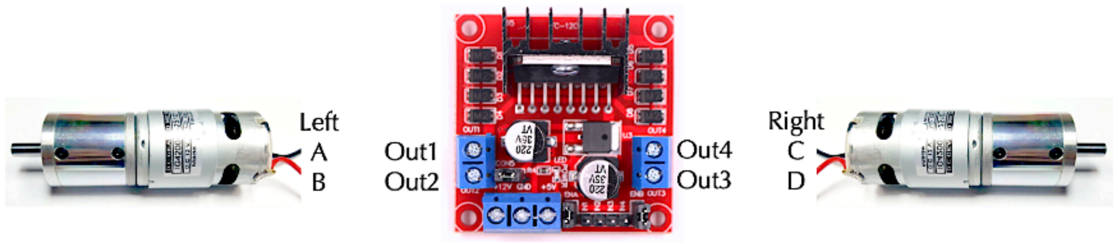


```
#include <Arduino.h>

#include "Joystick.h"
#include "LinearMap.h"
#include "Timer.h"

uno::Joystick<float> joystick;
nmr::LinearMap<float> ADCtoJoystickInputs;
nmr::Timer timerAnalogs;

void setup()
{
    Serial.begin(9600);
    while (!Serial) {}
    // Joystick Algorithm
    joystick = uno::Joystick<float>();
    // ADC digital values to joystick inputs
    ADCtoJoystickInputs = nmr::LinearMap<float>(0, 1023, -1.0, 1.0);
    // Utilities
    timerAnalogs = nmr::Timer();
}

void TestAnalogs()
{
    // Testing the correct behavior of the joystick
    // when using Analogs A0 and A1
    // (1) Move Control Stick LEFT and RIGHT
    // Should see x-values changing only...
    int xDigital = analogRead(A1);
    // (2) Move Control Stick UP  and DOWN
    // Should see y-values changing only...
    int yDigital = analogRead(A0);
    // Need negative and positive values for testing...
    float inputX = ADCtoJoystickInputs.Map((float)xDigital);
    float inputY = ADCtoJoystickInputs.Map((float)yDigital);
    // Monitor
    Serial.print("x:");
    Serial.print(inputX);
    Serial.print(" y: ");
    Serial.println(inputY);
    // (3) If the values are opposite than expected,
    // switch A0 and A1 around in the analogRead()
    // functions... Then repeat steps 1, 2 and 3...
}

void loop()
{
    if (timerAnalogs.isTimer(1000))
    {
        TestAnalogs();
    }
}
```

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 they 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.

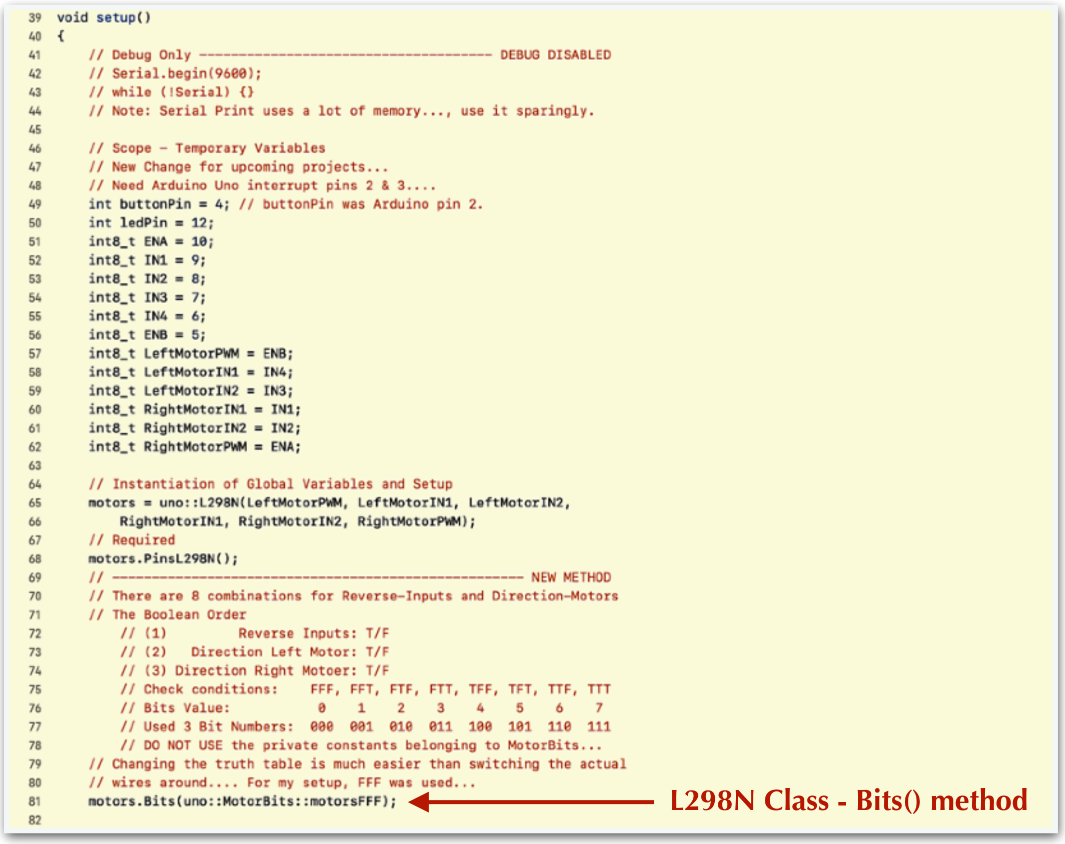
There are 8 combinations but 4 of the combinations involving the L298N seem equal at least until the polarities of each motor is considered. The colored bars indicate those combinations being similar. Once the correct combination is selected, the motors will behave in an expected manner while using the joystick algorithm. These combinations are independent of the analog readings from A0 and A1 and if one believes that the analog readings might be the problem, switching A0 and A1 might break the algorithm's functionality.



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			

Connecting wires is a hassle. Do not physically pull and plug wires among the L298N and motors until it works. Choose a single wiring setup. Test the setup using the Bits() method…

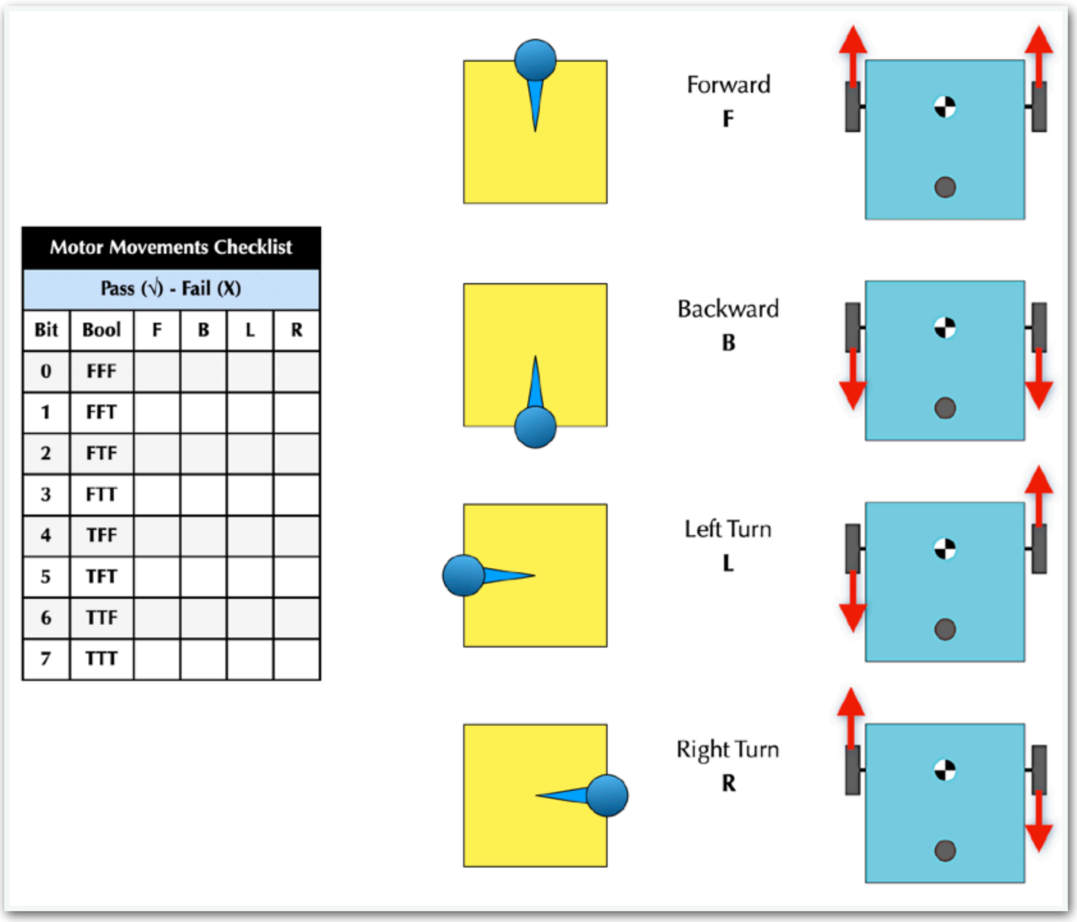
In the L298N Class, there is a method called ***Bits***. The programmer can enter a constant of a combination of three boolean values as shown on the far right of the table. This method will be further explained. Although these values may not match exactly to the motors and L298N depicted in the software, 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***.



```
void setup()
{
    // Debug Only ---------------------------------- DEBUG DISABLED
    // Serial.begin(9600);
    // while (!Serial) {}
    // Note: Serial Print uses a lot of memory..., use it sparingly.

    // Scope - Temporary Variables
    // New Change for upcoming projects...
    // Need Arduino Uno interrupt pins 2 & 3....
    int buttonPin = 4; // buttonPin was Arduino pin 2.
    int ledPin = 12;
    int8_t ENA = 10;
    int8_t IN1 = 9;
    int8_t IN2 = 8;
    int8_t IN3 = 7;
    int8_t IN4 = 6;
    int8_t ENB = 5;
    int8_t LeftMotorPWM = ENB;
    int8_t LeftMotorIN1 = IN4;
    int8_t LeftMotorIN2 = IN3;
    int8_t RightMotorIN1 = IN1;
    int8_t RightMotorIN2 = IN2;
    int8_t RightMotorPWM = ENA;

    // Instantiation of Global Variables and Setup
    motors = uno::L298N(LeftMotorPWM, LeftMotorIN1, LeftMotorIN2,
        RightMotorIN1, RightMotorIN2, RightMotorPWM);
    // Required
    motors.PinsL298N();
    // --------------------------------------------- NEW METHOD
    // There are 8 combinations for Reverse-Inputs and Direction-Motors
    // The Boolean Order
        // (1)          Reverse Inputs: T/F
        // (2)    Direction Left Motor: T/F
        // (3) Direction Right Motoer: T/F
        // Check conditions:    FFF, FFT, FTF, FTT, TFF, TFT, TTF, TTT
        // Bits Value:           0    1    2    3    4    5    6    7
        // Used 3 Bit Numbers:  000  001  010  011  100  101  110  111
        // DO NOT USE the private constants belonging to MotorBits...
    // Changing the truth table is much easier than switching the actual
    // wires around.... For my setup, FFF was used...
    motors.Bits(uno::MotorBits::motorsFFF);

```

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. 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.



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				

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 20230120A.
Article 1009 L298N Supplemental is a work in progress…