Joystick-Uno-L298N

by Jesse Carpenter

Contents (by sections)

Revised
September 17, 2021

Carpenter
Software

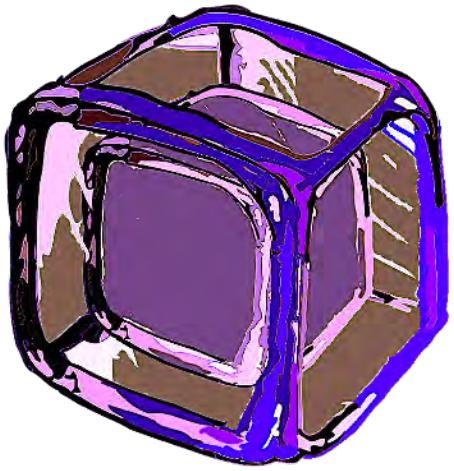
Introduction Robotics *Joystick-Uno-L298N*

AVR Bare-Metal C/C++ Programming

Bare metal refers to a computer executing instructions directly on logic hardware without loading a separate operating system. The C/C++ programming language remains a common practice in embedded systems, where microcontrollers often boot directly into a single large block from a single-purpose software often called firmware. Programing an AVR centers around the 8-bit Atmel (Microchip) family specifically the Atmega328P. The board we're using here is the Arduino Uno.

Computer Science

The practical approach to hands-on computer science is often done in a laboratory environment. In the lab, notes are often taken to answer an inquiry into connected devices while aiming to implement such devices for example in a differential drive mobile robots (DDMR). The laboratory might be part of undergraduate curriculum at a major university like MIT or it might be a simple study desk at home. The most important tool in the laboratory is the Lab-Notebook.

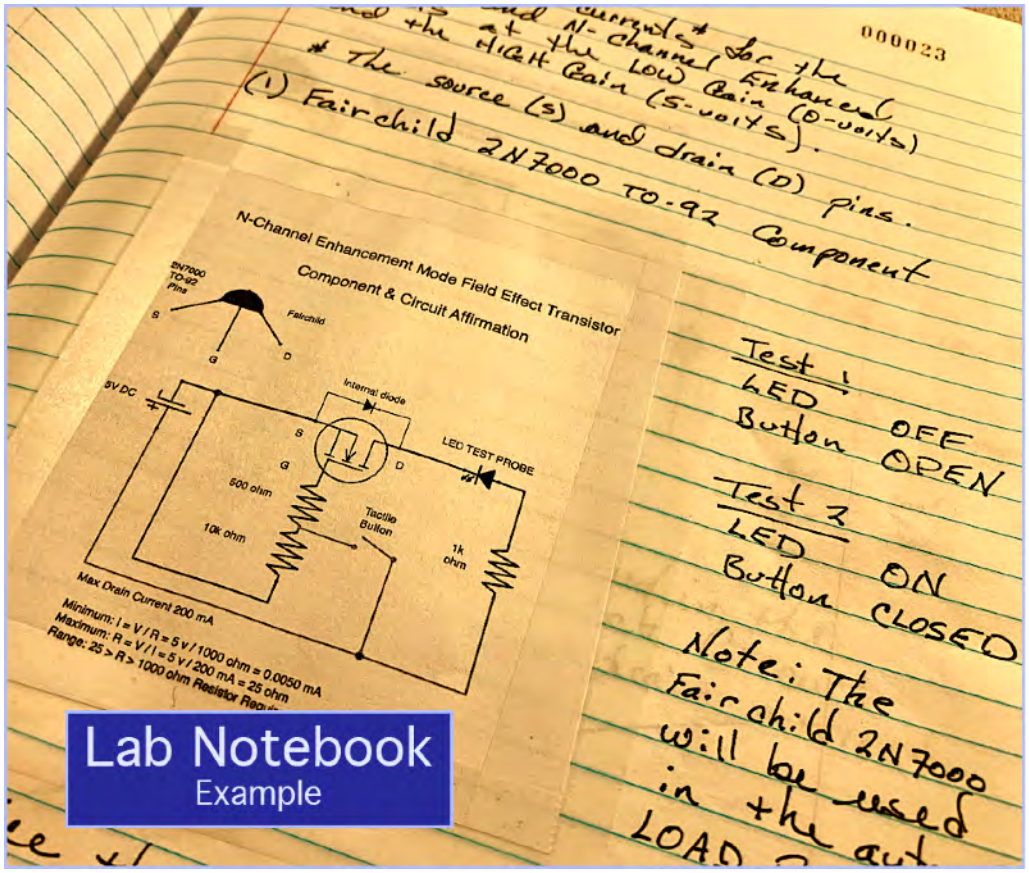


imperceptible black box

Black Box

The imperceptible black box sitting there on the lab bench. *What is it?* It is a phenomenon unmeasurable by the five senses. The use of the scientific equipment may be required to investigate such a black box. An electronic multimeter for example that is used in part of an investigation called the scientific method. The multimeter, an inexpensive device can be used to break open that imperceptible black box starting from a simple question like what can be measured from a passive electronic component like a resistor. Or better, imagine the scientific equipment required into the inquiries on quantum computing (another imperceptible black box). One might observe from an ohmmeter that the resistance measured was 50 ohms and the empirical value was recorded with at

most 3 significant digits in the lab notebook. The lab notebook while using the best practices of the scientific method prevents repeating the same experiment, answering the same question because the lab notebook is a permanent record. It is a reference to be used to expand upon into other experiments to come…



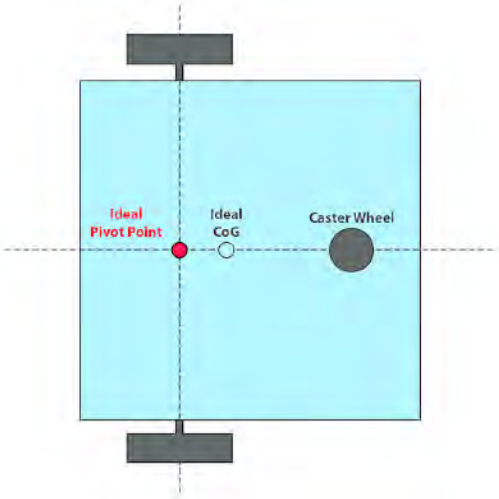
Lab Notebook

In the lab notebook example, the diagram of the 2N7000 Component & Circuit Affirmation was simply inquired by using an LED test probe. Open-circuit, LED off. Close-circuit, LED on. Upon closer scrutiny, the diagram can be referenced later for several key facts. For example, what is the difference between a limiting and a pull-down resistor and on the device silhouette, where are the actual connections for the Source (S), Gate (G), and Drain (D). Incorrect 2N7000 connections can be observed by a burnt electronic smell which is quite common in a lab. An observation by many who would like to avoid and who usually gets a standing applaud by their peers. The lab notebook is an important reference and legal document. It may contain hundreds of experiments like the Component & Circuit Affirmation lab test which are always referenced by scientist, engineer or hobbyist.

Differential Drive Mobile Robot

The robot has two independently driven wheels in front and one unpowered omnidirectional wheel in the rear. The driven wheels are motorized standard wheels with two degrees of freedom with its rotation around the wheel's axle and the contact point with the surface. The DDMR has a third wheel called a caster wheel with two degrees of freedom with its rotation around an offset steering joint.[1]

The chassis, the base frame of the robot, is a square structure or wider to reduce differential (sliding) friction by the driven wheels and any caster drag while turning. The ideal Center of



Gravity (CoG) is not necessarily along the horizontal plane of the chassis but does sit behind the Ideal Pivot Point (IPP). With the CoG close to the wheels axle line, most of its weight distribution of the robot is held by the driven wheels and not the caster. The IPP sits along the driven wheels axle line and the center line of the robot. The assumption that the environment for the robot's locomotion is on a flat and smooth surface.

C/C++ Programming Language

Code for embedded software is typically written in C/C++ programming language. Many beginners often begin with the popular Arduino integrated development environment (IDE) software and their line of Arduino boards with the Arduino Uno being their starter board. The IDE uses both the C and C++ language. The Arduino Uno is an open-source microcontroller board based on the Microchip ATmega328P microcontroller.

1. Roland Siegwart, Allah R. Nourbakhsh, Davide Scaramuzza, *Introduction to Autonomous Mobile Robot* (The MIT Press, Cambridge, Massachusetts, London, England; 2ed; 2004) Chapter 2 Locomotion.

Microcontroller Unit (MCU)

The ATmega328 is a single-chip microcontroller created by Atmel in the megaAVR family that has a modified Harvard architecture 8-bit RISC processor core. The microcontroller combines 32 KB in-system programming (ISP), also called in-circuit serial programming (ICSP) flash memory with read-while-write capabilities, 1 KB electrically erasable programmable read-only memory (EEPROM), 2 KB static random-access memory (SRAM), 23 general-purpose I/O lines, 32 general-purpose working registers, 3 flexible timer/counters with compare modes, internal and external interrupts, serial programmable universal asynchronous receiver-transmitter (USART), a byte-oriented 2-wire serial interface, serial peripheral interface (SPI) serial port, 6-channel 10-bit analog/digital (A/D) converter, programmable watchdog timer with internal oscillator, and 5 software-selectable power-saving modes. The device operates between 1.8 and 5.5 volts. The device achieves throughput approaching one (1) microprocessor without interlocked pipelined stages (MIPS) per megahertz (MHz).

For the the beginner, the technology can be overwhelming. The datasheet alone contains well over 200 pages. The one advantage using Arduino Uno with the ATmega328P microcontroller over other Arduino's boards is that the ATmega328P microcontroller can be replaced from its socket if some unfortunate electrical mishap occurs.

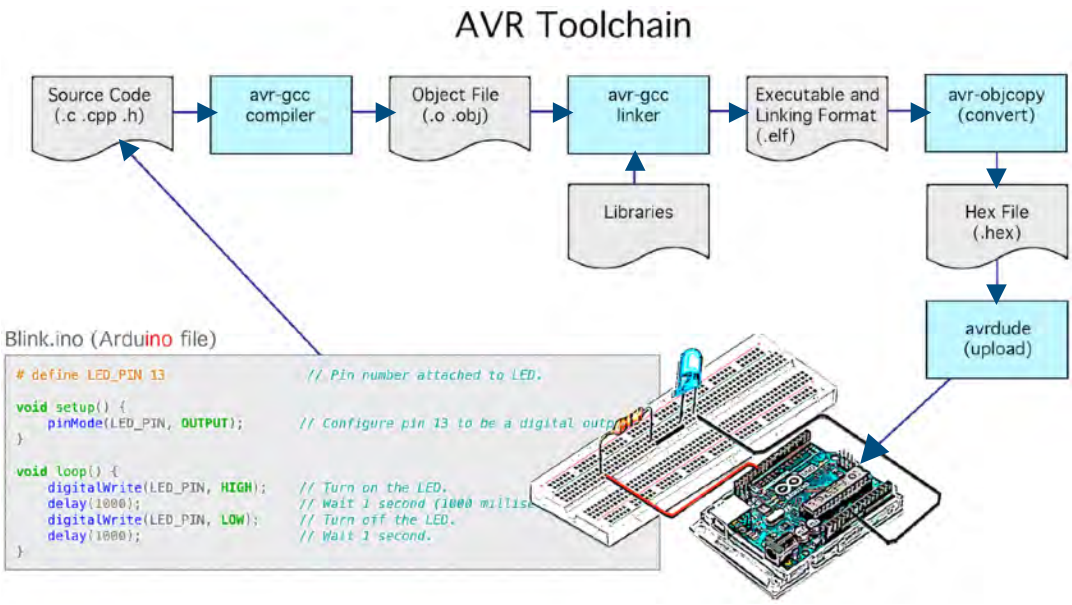
Integrated Development Environment

Arduino IDE is a cross-platform application for Windows, macOS, Linux. The IDE is used to write and upload programs called sketches (.ino) to Arduino compatible boards. Many hobbyists today uses a Linux computer like Raspberry Pi 4 (RP4) and its operating system, the Raspberry OS. The Arduino Uno board with its IDE while using it with the RP4 is very inexpensive.

AVR Toolchain

There is not a single *tool* that provides everything needed to develop software for the AVR. It takes many tools working together. Collectively, the group of tools are commonly called a toolchain, as the tools are chained together to produce the final executable application for the AVR microcontroller. The tools are built for the AVR target, the actual program names are prefixed with *avr* such as the executable compiler avr-gcc.[2] Few of the tools used under the hood by Arduino IDE are the *avr-gcc* compiler and the *avrdude* uploading tool. Lucky for us, Arduino IDE does all this out of the users sight making it seamlessly easy.

2. AVR Libc webpage, *avr-libc 2.0.0 - standard C library for AVR-GCC* (https://www.nongnu.org/avr-libc/user-manual/overview.html) visited August 2021.



Manifest

The following list of robotic components is the starting point. Once mastered, the novice is ready for bigger and better gadgets.

- **Thumb Joystick**

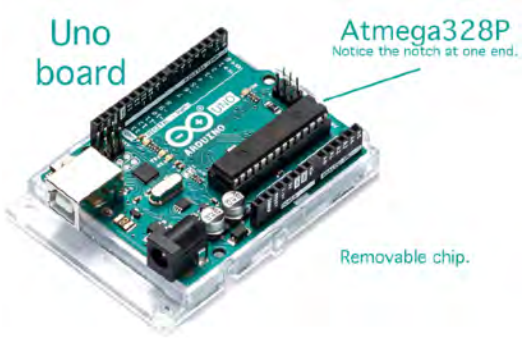
  The analog 2-axis thumb joystick with button by Makerfabs. The joystick has two analog (*10k potentiometers*), you'll need two analog (*ADC*) reading pins on your microcontroller to determine X and Y.

  Connections

  1. GND: (-) ground negative lead from 7805.
  2. 5V: (+) positive lead from 7805.
  3. X: (***A0***) Uno Analog Pin.
  4. Y: (***A1***) Uno Analog Pin.
  5. SW: (***2***) Switch Uno Digital Input Pin.

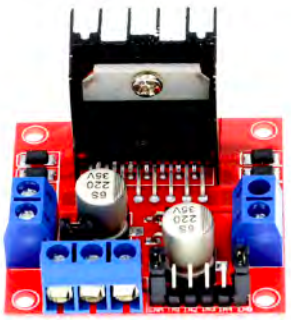
- **MCU - (*Atmega328P*)**

  Arduino Uno Board.
  - The USB connection from a Linux computer like Raspberry Pi 4 (*RP4*) using Raspberry OS using Arduino IDE to Arduino Uno.
  - The 5V USB can be supplied by RP4, but it's recommended to use Arduino Uno's 9V power supply pin.



- Notice in the Electronics Lab image, there is the Uno-Black-Wire connecting a ground to all components in the circuits. Although it is hard to see from the image between the 5V bus and the 14V bus, the buses are grounded together by a small black wire. The reason for Uno's ground connections would make a complete circuit with the Arduino Pins. See image and schematic.

- **L298N Motor Driver Module**



The L298N is a dual full-bridge motor driver designed to accept standard TTL logic levels.

Connections

Power Output Stage

The L298N integrates two power output stages (***A*** & ***B***). The power output stage is a bridge configuration and its outputs can drive an inductive load in common or **differential** mode, depending on the **state** of the inputs

Input State

Each bridge is driven by means of four gates the input of which are ***In1***, ***In2***, ***EnA*** and ***In3***, ***In4***, ***EnB***. The ***In*** inputs are used to set the bridge *states* with the ***En*** input used for **pulse width modulation** (PWM). PWM is a digital pulse length that varies a logic voltage from 0 to 5V. Thus PWM is used to adjust the motor speed. All inputs are Transistor-Transistor Logic (TTL) compatible. TTL is exactly what the Atmega328P uses…

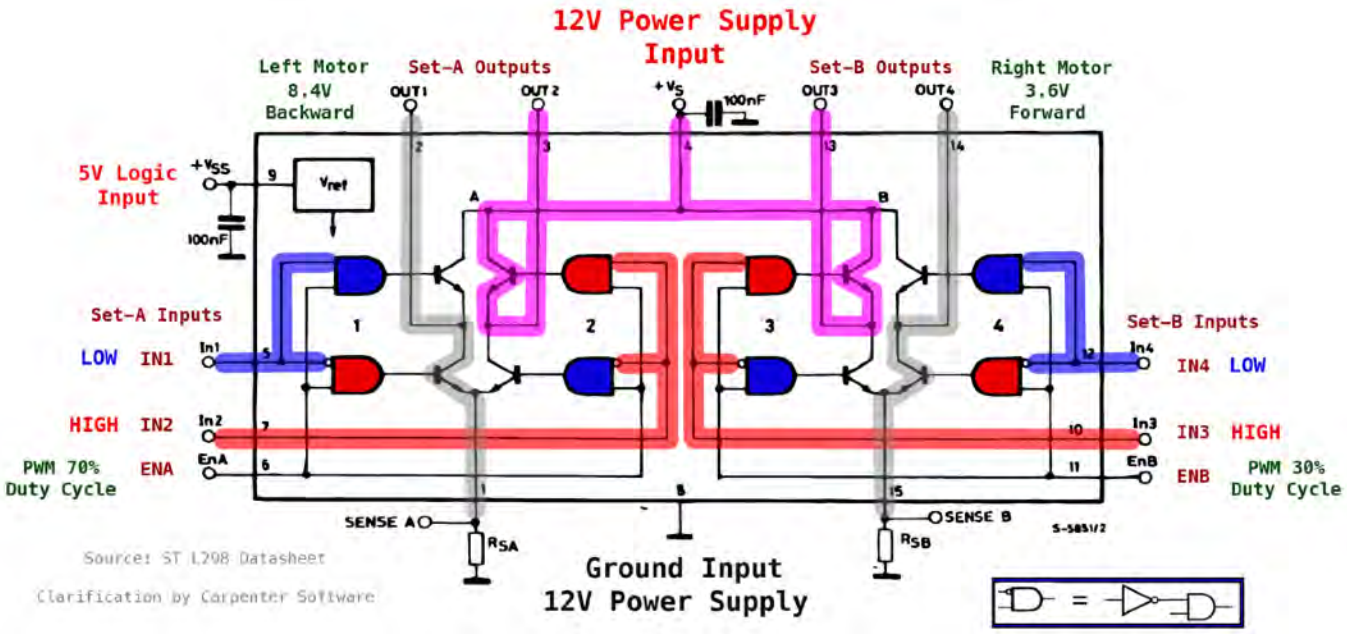
- **Two Geared Motors** 12V with ***brackets*** (*Gear Ratio around 1:75*) fastened to cheap Walmart cutting board with machine screws.

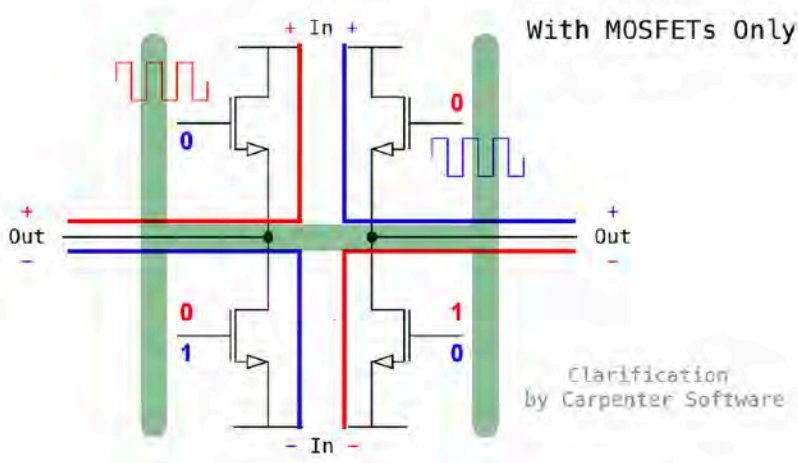
2 12V DC Motors with gears (approx. gear ratio 1:75

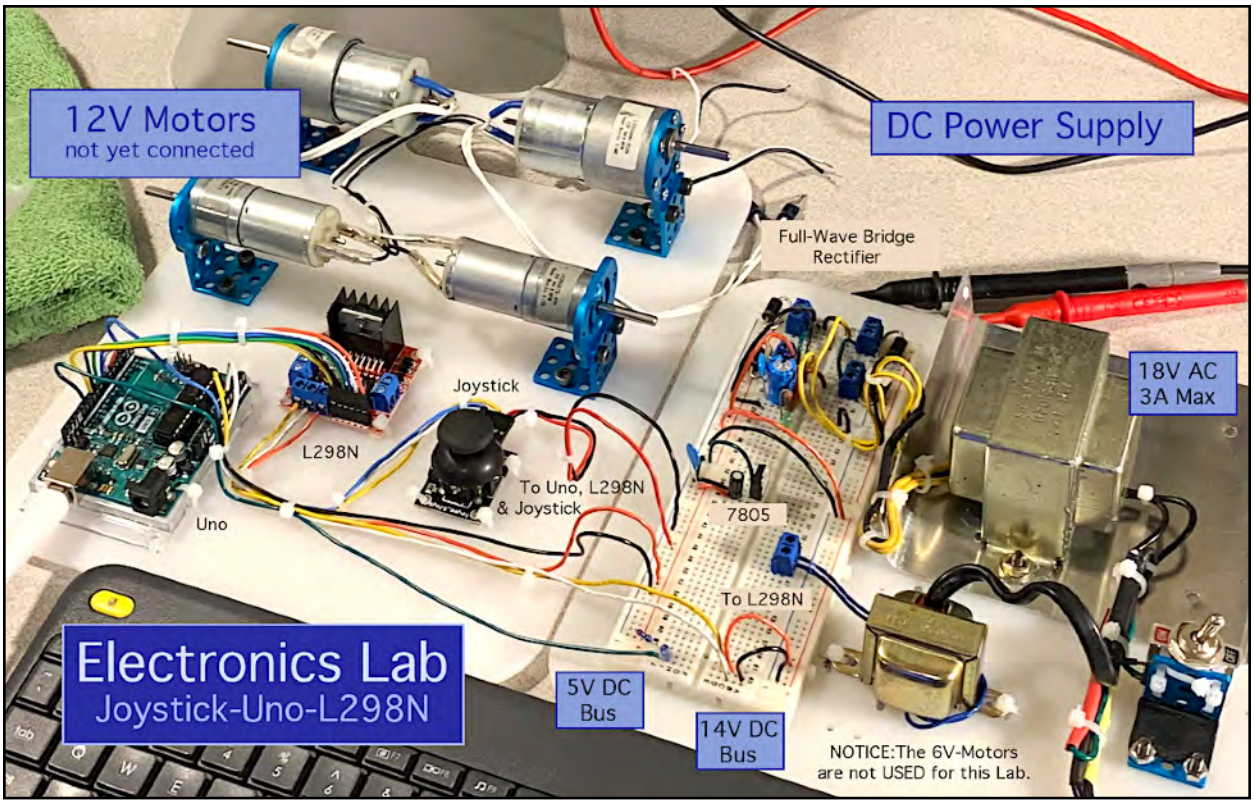
- **Power Supply** 12V DC *(2-3A max)* with breadboard and passive components (*Electronics Outlet*). Notice that many components are fasten to a lab board (*a cheap cutting board from Walmart*) held on by wire-ties (*also from Walmart*).

L298N & Typical H-Bridge Circuit

An H-bridge is an electronic circuit that switches the polarity of a voltage applied to a load. These circuits are often used in DC motors to run forwards or backwards. The typical graphical representation of an H-bridge is shown as a circuit. An H-bridge is built with four solid-state switches (MOSFET) controlled by 8-AND (with 4-Input-Inverter) logic gates. The L298N is a dual-channel H-Bridge motor driver capable of driving a pair of DC motors.





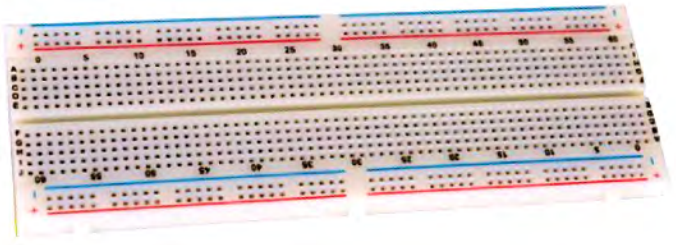


Electronics Lab Board

The programmer does not really need a robot, so let's do this in a simpler way by building something cheaper called an electronics lab board (by attaching the components using a cheap cutting board). Notice the wire-ties holding the components onto the board. See image.

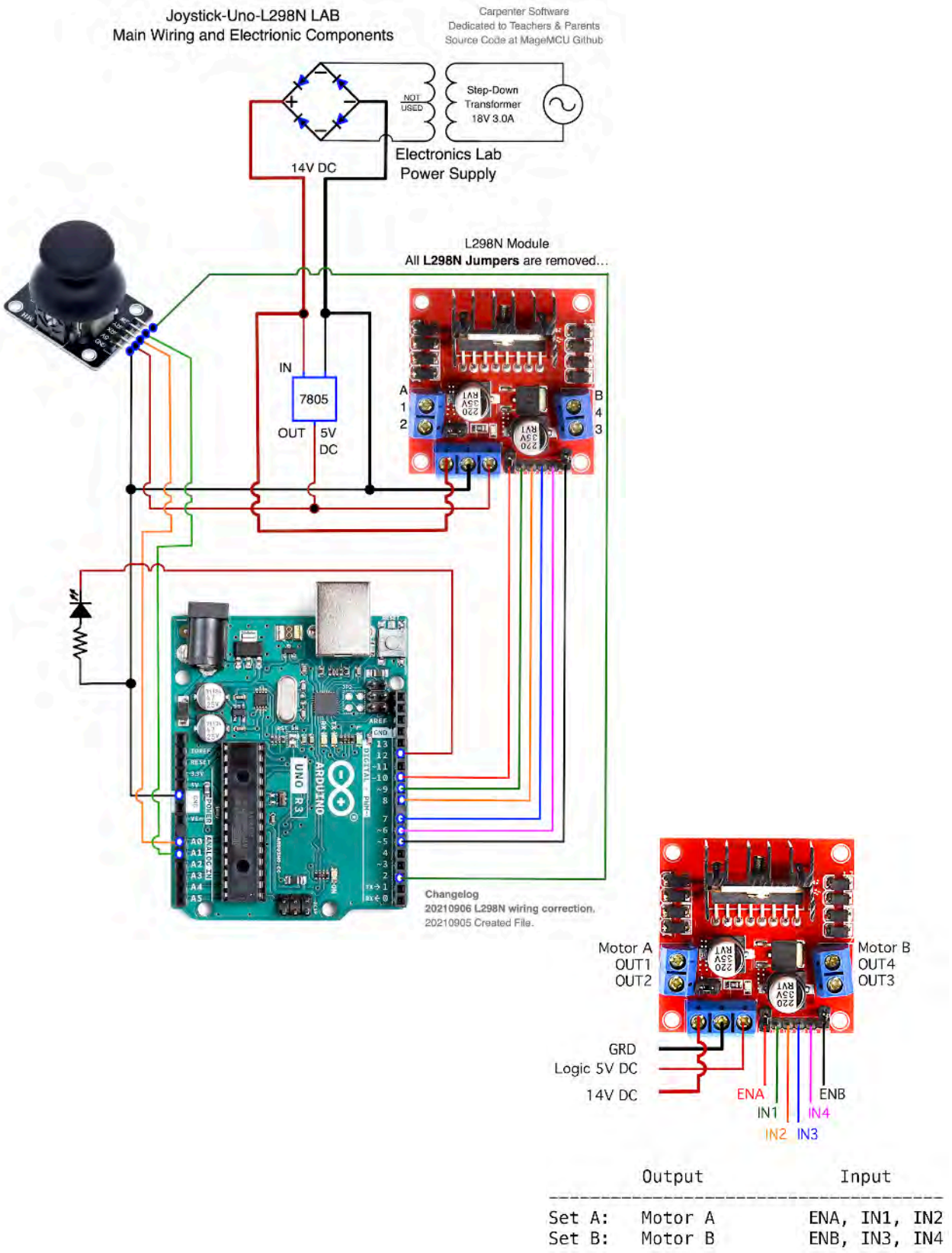
Breadboard, Wires & Electronics Active & Passive Components

Breadboards are one of the most fundamental pieces when learning how to build circuits. This board has 4 power supply buses which could supply *3.3*, *5*, *6* & *9* volts at the same time. Each bus could share the same ground connection.



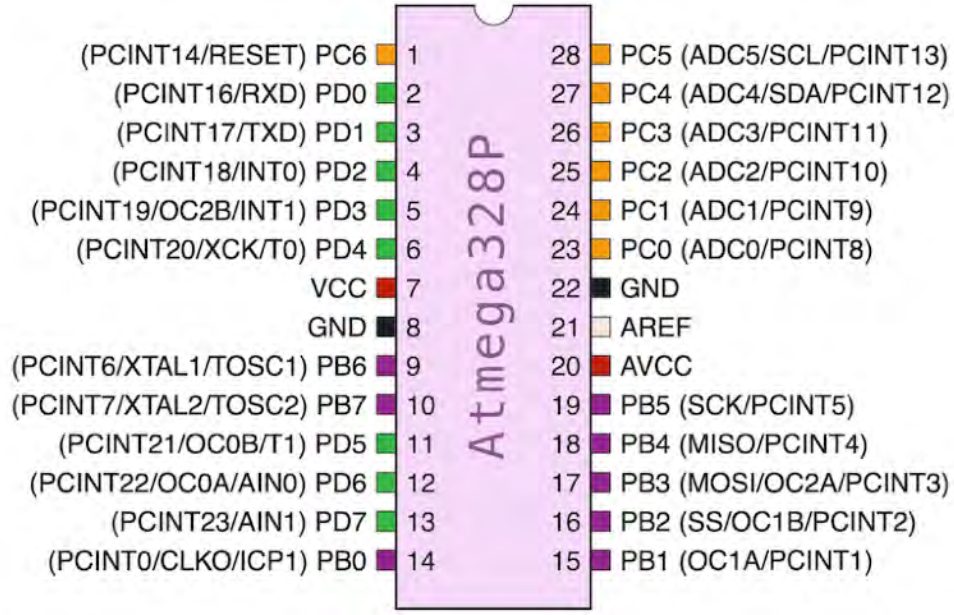
If one is new to electronics, search through the internet or research through the public library or buy a book.

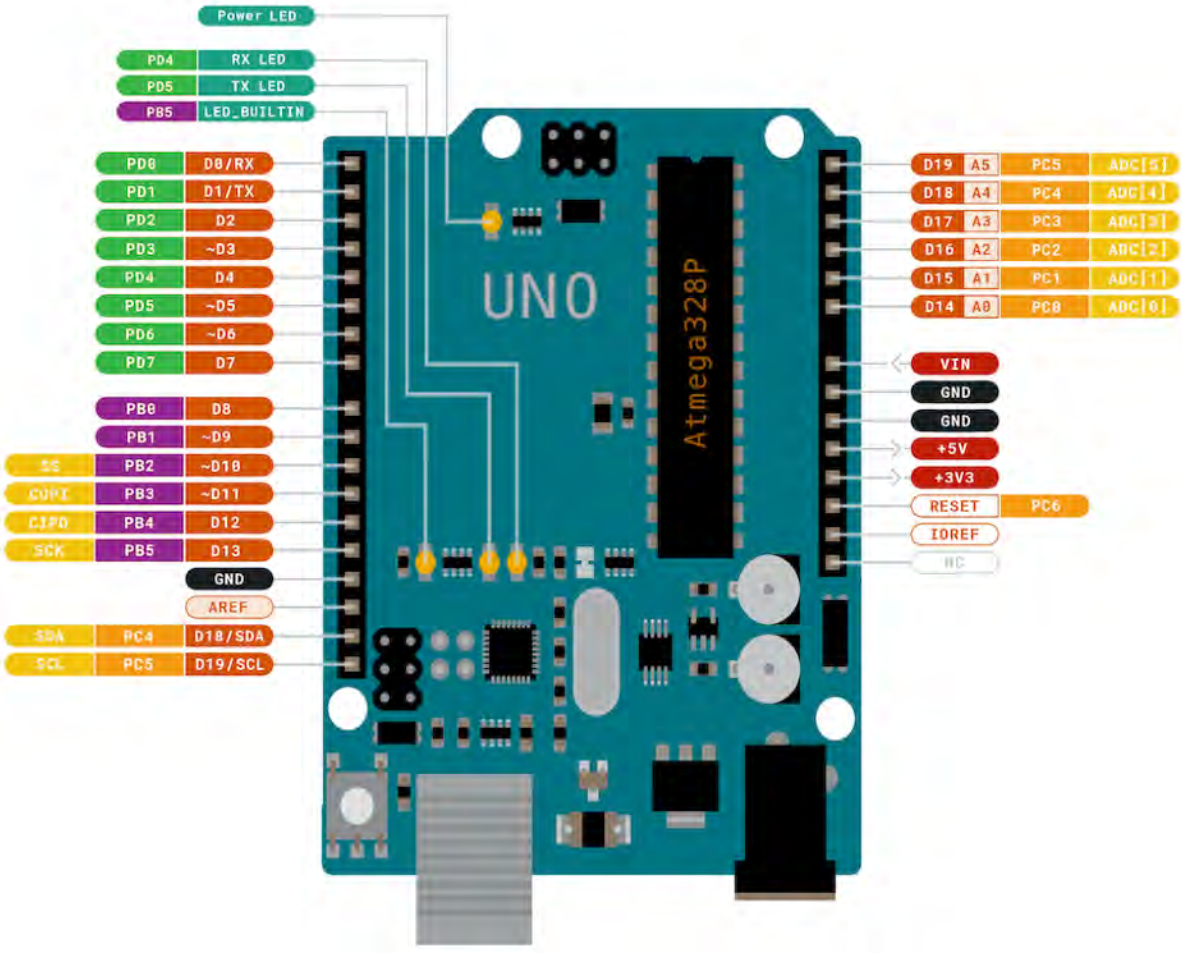
The Joystick-Uno-L298N Wiring Diagram



Output		Input
Set A:	Motor A	ENA, IN1, IN2
Set B:	Motor B	ENB, IN3, IN4

Atmega328P & Uno Pin Mapping





Where to find the Source Code

Software titled ***Joystick Uno L298N*** by MageMCU at Github.

DISCLAIMER (Carpenter Software & Github)

Any Information, including the Electronic Schematics and the Software (AI_ES_S) is provided as is, without any representation or warranty of any kind, either express or implied, including without limitation any representations or endorsements regarding the use of, the results of, or performance of the AI_ES_S, its appropriateness, accuracy, reliability, or correctness.

The entire risk as to the use of AI_ES_S is assumed by Licensee (MIT).

Carpenter Software LC (AUTHOR) does not assume liability for the use of this AI_ES_S. In no event will the AUTHOR be liable for additional direct or indirect damages including any lost profits, lost savings, or other incidental or consequential damages arising from any mishap, or the use or inability to use these AI_ES_S, even if AUTHOR has been advised of the possibility of such damages.

The MageMCU GitHub account is a living workspace used by the AUTHOR for the AUTHOR and if by consequence the AI_ES_S is used by a Licensee then the Licensee is solely responsible.

Parents and Teachers

Parents and Teachers: This Github Account is restricted to minors. When working with electricity of any kind, they should be directed and instructed by a responsible mentor. A USER (Licensee) must be at least 13 years of age when using GitHub.

Minors (Children) and Teenagers

All children and teenagers, please consult with your parent (your guardian, your mentor or your teacher) before experimenting with any of these electronic circuits. Never work with electricity alone. Electricity Kills.

The code may be download by MageMCU at GitHub. The manifest chosen contain some of the most inexpensive items. The circuit has been tested over several years. Anyone who wants to volunteer to teach, please educate our children for the world to come…

–Jesse Carpenter

Carpenter Software
revision 20210917 edited…
revision 20210906 Jesse Carpenter
ALL RIGHTS RESERVED.

```
//
// Carpenter Software
// File: main.cpp (release version)
// Date: 20210830
// Folder: joystick_L298N_debug
// Purpose: Github Depository (MageMCU)
//
// Student Copy...
//
// Hardware (For further information, see Debug Version)
//
// RAM:   [          ]   3.4% (used 69 bytes from 2048 bytes)
// Flash: [==        ]  15.2% (used 4890 bytes from 32256 byt
//
// MIT LICENSE
//

#include <Arduino.h>
// Using Adruino Uno

#include "L298N.h"
#include "Joystick.h"
#include "Timer.h"
#include "Button.h"

// Declaration of GLOBAL VARIABLES
uno::L298N mtr;
uno::Joystick stk;
uno::Timer tmrMotors;
uno::Button btnMotors;

void setup()
{
    // - Scope - Temp Variables
    int buttonPin = 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
    mtr = uno::L298N(LeftMotorPWM, LeftMotorIN1, LeftMotorIN2,
                     RightMotorIN1, RightMotorIN2, RightMotorPWM);
    mtr.SetupPinsL298N();
    stk = uno::Joystick();
    tmrMotors = uno::Timer();
    btnMotors = uno::Button(buttonPin, ledPin);
}

void updateAnalog()
{
    if (btnMotors.isButtonOn())
    {
        int xAnalog = analogRead(A1);
        int yAnalog = analogRead(A0);
        stk.UpdateInputs(xAnalog, yAnalog);

        int left = stk.GetOutputLeftInteger(255);
        int right = stk.GetOutputRightInteger(255);
```

```
        mtr.updateL298N(left, right);
    }
}

void loop()
{
    btnMotors.updateButton();
    if (tmrMotors.isTimer(50))
    {
        updateAnalog();
    }
}
```

```
//
// Carpenter Software
// File: Joystick.h (release version)
// Date: 20210830
// Folder: joystick_L298N_debug
// Purpose: Github Depository (MageMCU)
//
// Student Use...
//
// Hardware (For further information, see Debug Version)
//
// RAM:   [          ]   3.4% (used 69 bytes from 2048 bytes)
// Flash: [==        ]  15.2% (used 4890 bytes from 32256 byt
//
// MIT LICENSE
//

#ifndef __UNO_JOYSTICK_H__
#define __UNO_JOYSTICK_H__

#include <Arduino.h>

#define _ZERO 0
#define _MIDPOINT_LO 511
#define _MIDPOINT_HI 512
#define _1023 1023

namespace uno
{
  class Joystick
  {
  private:
    bool _ready;

    int _processX;
    int _processY;

    double _normalLeft;
    double _normalRight;
    int _cntCalibration21;
    int _xCenterOffset;
    int _yCenterOffset;
    int _xDynamicOffset;
    int _yDynamicOffset;

    void _dataJoystick();
    bool _centerCalibrated();
    void _alignRawInput();
    void _shiftFromZeroToMidpoint();
    void _adjustToDynamicFriction();
    void _joystick();

  public:
    Joystick();
    bool UpdateInputs(int rawX, int rawY);
    int GetOutputLeftInteger(int maxL298N_PWM);
    int GetOutputRightInteger(int maxL298N_PWM);
    double GetOutputLeft();
    double GetOutputRight();
  };
  Joystick::Joystick()
  {
    _dataJoystick();
  }

  double Joystick::GetOutputLeft()
  {
    return _normalLeft;
```

```
}

double Joystick::GetOutputRight()
{
  return _normalRight;
}

int Joystick::GetOutputLeftInteger(int maxL298N_PWM)
{
  return (int)(_normalLeft * (double)maxL298N_PWM);
}

int Joystick::GetOutputRightInteger(int maxL298N_PWM)
{
  return (int)(_normalRight * (double)maxL298N_PWM);
}

bool Joystick::UpdateInputs(int const rawX, int const rawY)
{
  _processX = rawX;
  _processY = rawY;
  if (_cntCalibration21 < 22)
    _ready = false;
  else
    _ready = true;
  if (_centerCalibrated())
  {
    _alignRawInput();
    _shiftFromZeroToMidpoint();
    _adjustToDynamicFriction();
    _joystick();
  }
  return _ready;
}

void Joystick::_dataJoystick()
{
  _ready = false;
  _cntCalibration21 = 0;
}

boolean Joystick::_centerCalibrated()
{
  if (_cntCalibration21 <= 21)
  {
    _xCenterOffset = ((_xCenterOffset + _processX) / 2) + 1;
    _yCenterOffset = ((_yCenterOffset + _processY) / 2) + 1;
    _cntCalibration21++;
    return false;
  }
  return true;
}

void Joystick::_alignRawInput()
{
  int _x = _processX;
  int _y = _processY;
  if (_x >= _xCenterOffset)
  {
    _x = (int)map(_x, _xCenterOffset, _1023, _MIDPOINT_HI, _1023);
  }
  else if (_x < _xCenterOffset)
  {
    _x = (int)map(_x, _ZERO, (_xCenterOffset - 1), _ZERO, _MIDPOINT_LO);
  }

  if (_y >= _yCenterOffset)
  {
```

```
    _y = (int)map(_y, _yCenterOffset, _1023, _MIDPOINT_HI, _1023);
  }
  else if (_y < _yCenterOffset)
  {
    _y = (int)map(_y, _ZERO, (_yCenterOffset - 1), _ZERO, _MIDPOINT_LO);
  }
  _processX = _x;
  _processY = _y;
}
void Joystick::_shiftFromZeroToMidpoint()
{
  _processX -= _MIDPOINT_LO;
  _processY -= _MIDPOINT_LO;
}

void Joystick::_adjustToDynamicFriction()
{
  int const _centerTolerance = 5;
  int _x = _processX;
  int _y = _processY;

  _xDynamicOffset = 260;
  if (_x >= _centerTolerance)
  {
    _x = (int)map(_x, 0, 512, _xDynamicOffset, 512);
  }
  else if (_x < -_centerTolerance)
  {
    _x = (int)map(_x, 0, -511, -_xDynamicOffset, -511);
  }
  else
  {
    _x = 0;
  }
  _yDynamicOffset = 260;
  if (_y >= _centerTolerance)
  {
    _y = (int)map(_y, _ZERO, _MIDPOINT_HI, _yDynamicOffset, _MIDPOINT_HI);
  }
  else if (_y < -_centerTolerance)
  {
    _y = (int)map(_y, _ZERO, -_MIDPOINT_LO, -_yDynamicOffset, -_MIDPOINT_LO);
  }
  else
  {
    _y = 0;
  }
  _processX = _x;
  _processY = _y;
}

void Joystick::_joystick()
{
  double const _tol = 0.04;

  double const _inputX = (double)_processX / (double)_MIDPOINT_HI;
  double const _inputY = (double)_processY / (double)_MIDPOINT_HI;

  double _x = 0;
  double _y = 0;
  double delta = abs(abs(_inputX) - abs(_inputY));
  if (abs(_inputX) > _tol && abs(_inputY) > _tol)
  {
    if (abs(_inputX) <= abs(_inputY))
    {
      if (_inputX > _ZERO && _inputY > _ZERO)
      {
        _x = _inputY;
```

```
          _y = delta;
        }
        else if (_inputX > _ZERO && _inputY < _ZERO)
        {
          _x = -delta;
          _y = _inputY;
        }
        else if (_inputX < _ZERO && _inputY > _ZERO)
        {
          _x = delta;
          _y = _inputY;
        }
        else if (_inputX < _ZERO && _inputY < _ZERO)
        {
          _x = _inputY;
          _y = -delta;
        }
      }
      else if (abs(_inputX) > abs(_inputY))
      {
        if (_inputX > _ZERO && _inputY > _ZERO)
        {
          _x = _inputX;
          _y = -delta;
        }
        else if (_inputX > _ZERO && _inputY < _ZERO)
        {
          _x = delta;
          _y = -_inputX;
        }
        else if (_inputX < _ZERO && _inputY > _ZERO)
        {
          _x = -delta;
          _y = -_inputX;
        }
        else if (_inputX < _ZERO && _inputY < _ZERO)
        {
          _x = _inputX;
          _y = delta;
        }
      }
    }
    else if (abs(_inputX) > _tol && abs(_inputY) < _tol)
    {
      _x = _inputX;
      _y = -_inputX;
    }
    else if (abs(_inputX) < _tol && abs(_inputY) > _tol)
    {
      _x = _inputY;
      _y = _inputY;
    }
    else if (abs(_inputX) < _tol && abs(_inputY) < _tol)
    {
      _x = 0;
      _y = 0;
    }

    _normalLeft = _x;
    _normalRight = _y;
  }
}
#endif
```

```
//
// Carpenter Software
// File: L298N.h (release version)
// Date: 20210830
// Folder: joystick_L298N_debug
// Purpose: Github Depository (MageMCU)
//
// Student Use...
//
// Hardware (For further information, see Debug Version)
//
// RAM:   [          ]   3.4% (used 69 bytes from 2048 bytes)
// Flash: [==        ]  15.2% (used 4890 bytes from 32256 byt
//
// MIT LICENSE
//

#ifndef __UNO_L298N_H__
#define __UNO_L298N_H__

#include <Arduino.h>
// Using Adruino Uno

namespace uno
{
    class L298N
    {
    private:
        uint8_t _ZERO;

        uint8_t _LeftMotorPWM;
        uint8_t _LeftMotorIN1;
        uint8_t _LeftMotorIN2;

        uint8_t _RightMotorIN1;
        uint8_t _RightMotorIN2;
        uint8_t _RightMotorPWM;

        int _PWM_LeftMotor;
        int _PWM_RightMotor;

        void _setDirectionPins();
        void _powerDownL298N();

    public:
        L298N();
        L298N(uint8_t LeftMotorPWM,
              uint8_t LeftMotorIN1,
              uint8_t LeftMotorIN2,
              uint8_t RightMotorIN1,
              uint8_t RightMotorIN2,
              uint8_t RightMotorPWM);
        void SetupPinsL298N();
        void updateL298N(int UnoPWM_ToENA, int UnoPWM_ToENB);
    };

    L298N::L298N()
    {
        _ZERO = 0;
        _LeftMotorPWM = 10;
        _LeftMotorIN1 = 9;
        _LeftMotorIN2 = 8;
        _RightMotorIN1 = 7;
        _RightMotorIN2 = 6;
        _RightMotorPWM = 5;
    }

    L298N::L298N(uint8_t LeftMotorPWM,
```

```
                uint8_t LeftMotorIN1,
                uint8_t LeftMotorIN2,
                uint8_t RightMotorIN1,
                uint8_t RightMotorIN2,
                uint8_t RightMotorPWM)
    {
        _ZERO = 0;
        _LeftMotorPWM = LeftMotorPWM;
        _LeftMotorIN1 = LeftMotorIN1;
        _LeftMotorIN2 = LeftMotorIN2;
        _RightMotorIN1 = RightMotorIN1;
        _RightMotorIN2 = RightMotorIN2;
        _RightMotorPWM = RightMotorPWM;
    }

    void L298N::updateL298N(int PWM_LeftMotor, int PWM_RightMotor)
    {
        _PWM_LeftMotor = PWM_LeftMotor;
        _PWM_RightMotor = PWM_RightMotor;

        _setDirectionPins();

        analogWrite(_LeftMotorPWM, abs(PWM_LeftMotor));
        analogWrite(_RightMotorPWM, abs(PWM_RightMotor));
    }

    void L298N::_powerDownL298N()
    {
        digitalWrite(_LeftMotorIN1, LOW);
        digitalWrite(_LeftMotorIN2, LOW);
        digitalWrite(_RightMotorIN1, LOW);
        digitalWrite(_RightMotorIN2, LOW);
    }

    void L298N::SetupPinsL298N()
    {
        pinMode(_LeftMotorPWM, OUTPUT);
        pinMode(_LeftMotorIN1, OUTPUT);
        pinMode(_LeftMotorIN2, OUTPUT);
        pinMode(_RightMotorIN1, OUTPUT);
        pinMode(_RightMotorIN2, OUTPUT);
        pinMode(_RightMotorPWM, OUTPUT);
        _powerDownL298N();
    }

    void L298N::_setDirectionPins()
    {
        if (_PWM_LeftMotor > _ZERO)
        {
            digitalWrite(_LeftMotorIN1, LOW);
            digitalWrite(_LeftMotorIN2, HIGH);
        }
        else if (_PWM_LeftMotor < _ZERO)
        {
            digitalWrite(_LeftMotorIN2, LOW);
            digitalWrite(_LeftMotorIN1, HIGH);
        }
        else
        {
            digitalWrite(_LeftMotorIN1, LOW);
            digitalWrite(_LeftMotorIN2, LOW);
        }

        if (_PWM_RightMotor > _ZERO)
        {
            digitalWrite(_RightMotorIN1, LOW);
            digitalWrite(_RightMotorIN2, HIGH);
        }
```

```
        else if (_PWM_RightMotor < _ZERO)
        {
            digitalWrite(_RightMotorIN2, LOW);
            digitalWrite(_RightMotorIN1, HIGH);
        }
        else
        {
            digitalWrite(_RightMotorIN1, LOW);
            digitalWrite(_RightMotorIN2, LOW);
        }
    }
}

#endif
```

```
//
// Carpenter Software
// File: Button.h (release version)
// Date: 20210830
// Folder: joystick_L298N_debug
// Purpose: Github Depository (MageMCU)
//
// Student Use...
//
// Hardware (For further information, see Debug Version)
//
// RAM:   [          ]   3.4% (used 69 bytes from 2048 bytes)
// Flash: [==        ]  15.2% (used 4890 bytes from 32256 byt
//
// MIT LICENSE
//
// Timer Class was adapted from Arduino's
// Debounce Sketch Example.

#ifndef __UNO_BUTTON_H__
#define __UNO_BUTTON_H__

#include <Arduino.h>
// Using Adruino Uno

#define DEBOUNCE_TIME 75

namespace uno
{
  class Button
  {
  private:
    int _ledPin;
    int _ledState;
    int _buttonPin;
    int _buttonState;
    int _lastButtonState;
    unsigned long _lastDebounceTime;
    unsigned long _debounceDelay;

    void _data();
    void _pins();
    void _button();

  public:
    Button();
    Button(int buttonPin);
    Button(int buttonPin, int ledPin);

    bool isButtonOn();
    void updateButton();
  };

  // Constructor
  Button::Button()
  {
    _pins();
    _data();
  }

  // Constructor
  Button::Button(int buttonPin)
```

```
{
  _pins();
  _data();
  _buttonPin = buttonPin;
}

// Constructor
Button::Button(int buttonPin, int ledPin)
{
  _pins();
  _data();
  _buttonPin = buttonPin;
  _ledPin = ledPin;
}

// PUBLIC method: Is the Button On = HIGH
bool Button::isButtonOn()
{
  return _ledState == HIGH;
}

// PUBLIC method: updateButton (used in the Arduino loop() function)
void Button::updateButton()
{
  _button();
}

// Private Method
void Button::_data()
{
  _buttonPin = 2;
  _ledPin = 13;
  // Begin the current state LOW of the output pin
  _ledState = LOW;
  // the previous reading from the input pin
  _lastButtonState = HIGH;
  _lastDebounceTime = 0;
  _debounceDelay = DEBOUNCE_TIME;
}

// Private Method
void Button::_pins()
{
  pinMode(_buttonPin, INPUT);
  pinMode(_ledPin, OUTPUT);
}

// Private Method
void Button::_button()
{
  int currentState = digitalRead(_buttonPin);
  long currentTime = millis();

  // If the switch changed, due to noise or pressing:
  if (currentState != _lastButtonState)
  {
    // reset the debouncing timer
    _lastDebounceTime = currentTime;
  }

  // whatever the reading is at, it's been there for longer than the debounce
  // delay, so take it as the actual current state:
```

```
    if ((currentTime - _lastDebounceTime) > _debounceDelay)
    {
      // if the button state has changed:
      if (currentState != _buttonState)
      {
        // Add the current reading from the input pin
        _buttonState = currentState;

        // only toggle the LED if the new button state is HIGH
        if (_buttonState == HIGH)
        {
          _ledState = !_ledState;
        }
      }
    }

    // set the LED:
    digitalWrite(_ledPin, _ledState);

    // save the reading. Next time through the loop, it'll be the lastButtonState:
    _lastButtonState = currentState;
  }
}
#endif
```

```
//
// Carpenter Software
// File: Timer.h (release version)
// Date: 20210830
// Folder: joystick_L298N_debug
// Purpose: Github Depository (MageMCU)
//
// Student Use...
//
// Hardware (For further information, see Debug Version)
//
// RAM:   [          ]   3.4% (used 69 bytes from 2048 bytes)
// Flash: [==        ]  15.2% (used 4890 bytes from 32256 byt
//
// MIT LICENSE
//

#ifndef __UNO_TIMER_H__
#define __UNO_TIMER_H__

#include <Arduino.h>
// Using Adruino Uno

namespace uno
{
  class Timer
  {
  private:
    long _elapsedTime;
    long _lastMillisecond;

  public:
    Timer();
    void resetTimer();
    boolean isTimer(int incrementedTime);
  };

  Timer::Timer()
  {
    _elapsedTime = -1;
    _lastMillisecond = 0;
  }

  void Timer::resetTimer()
  {
    _elapsedTime = -1;
  }

  boolean Timer::isTimer(int incrementedTime)
  {
    long currentTime = millis();
    if (_elapsedTime == -1)
    {
      _elapsedTime = currentTime + (long)incrementedTime;
    }
    else if (currentTime > _elapsedTime)
    {
      _elapsedTime = currentTime + (long)incrementedTime;
      return true;
    }
    return false;
  }
}
#endif
```

```
//
// Carpenter Software
// File: MAIN-  main.cpp (debug version)
// Folder: joystick_L298N_debug
// Purpose: Github Depository (MageMCU)
//
// Teacher's and Parent's copy...
// Dedicated to Parents and Teachers of the USA.
// Teachers Note: The _BEGIN_DEBUG is defined (see
//                DEBUG.h file), also MAIN _ANALOG_
//                is defined for debugging the
//                analog readings. For safety
//                reasons while debugging, the
//                motors are conditioned not to
//                operate. Be warned. the Student's
//                copy has no safety feature.
//
// Hardware
// MCU: Atmega328P (Microchip)
// * Board: Arduino Uno (Italy)
// Thumb Joystick: (China)
// Motor Driver: L298N (ST)
// * Motor Module: (China)
// 12V Geared Motors (ratio 1:75) (China)
// Note: motors: current peaks on average 1.14 amps
//       for static friction, and it levels down to
//       about 0.29 amps for dynamic friction. When
//       joystick with hands-off (center position),
//       is about 0.10 amps. The robot was sitting
//       on stand with no weight on wheels, otherwise,
//       the electrical currents would have increased.
// Battery 12V - APX1250 (12V 5.0Ah) Sealed Lead-Acid
// Note: battery: with five years design life (using
//       now for 7 years.) Requires special battery
//       charger from APEX.
//
// By Jesse Carpenter (Github as MageMCU)
//
// CHANGELOG
// August 30, 2021 - debug comments
// August 24, 2021 - editing
// June 19, 2021 - programming debug version
//
// Atmega328P Total Usgae
// RAM : [==] 15.3 % (used 314 bytes from 2048 bytes)
// Flash : [==] 21.2 % (used 6846 bytes from 32256 bytes)
//
// MIT LICENSE
//

#include <Arduino.h>
#include "L298N.h"
#include "Joystick.h"
#include "Timer.h"
#include "Button.h"
#include "DEBUG.h"
```

```
// Using Adruino Uno
// GLOBAL VARIABLES
uno::L298N mtr;
uno::Joystick stk;
uno::Timer tmrMotors;
uno::Timer tmrMainDebug;
uno::Button btnMotors;

void setup()
{
// Debug
#ifdef _dM_
    Serial.begin(9600);
    while (!Serial)
    {
    }
    Serial.println("Serial 9600");
#endif

    // STUB --RADIO-CONTROL---       TODO
    // BY-PASS THUMB-JOYSTICK

    // STUB --I2C--                  TODO
    // MOVE TO PROTOBOARD
    // AS SLAVE DEVICE AS MOTOR
    // MANAGER - THERE IS ROOM FOR
    // A SERVO...

    // To make it less difficult to get the code up and running,
    // use L298N Assignments as discussed below. I have found it
    // much easier to PROGRAM the code whether than re-wiriing
    // the hardware.
    //
    // Record the Wired Connections from Uno to L298N and
    // from L298N to the Motors. OTHERWISE, JOYSTICK CODE
    // WILL BREAK. EVERYTHING IS PROGRAMMED AROUND THE JOYSTICK
    // CODE.
    //
    // (1) WIRING ARDUINO UNO to L298N...
    // ENA-Set
    // int8_t ENA = Uno-Pin? ;
    // int8_t IN1 = Uno-Pin? ;
    // int8_t IN2 = Uno-Pin ?;
    // ENB-Set
    // int8_t IN3 = Uno-Pin? ;
    // int8_t IN4 = Uno-Pin? ;
    // int8_t ENB = Uno-Pin? ;

    // (2) L298N to MOTORS...
    // Set ENA
    //      (a) L298N-Motor-Outs(1 & 2) matches to ENA.
    //      (b) Assigned to (Left/Right)__________ Motor.
    //      (c) The Motor assigned gets set (ENA, IN1, IN2).
    // Set ENB
    //      (a) L298N-Motor-Outs(3 & 4) matches to ENB.
    //      (b) Assigned to(Left/Right) __________ Motor.
    //      (c) The Motor assigned gets set (ENB, IN3, IN4).
```

```
    // (3a) Based on (1 & 2), is this set (ENA/ENB)_____________?
    // Instead of using (Uno-Pin), use assigned variables above.
    // int8_t LeftMotorPWM = Uno-Pin? ;
    // int8_t LeftMotorIN1 = Uno-Pin? ;
    // int8_t LeftMotorIN2 = Uno-Pin? ;

    // (3b) ) Based on (1 & 2), is this set (ENA/ENB)_____________?
    // Instead of using (Uno-Pin), use assigned variables above.
    // int8_t RightMotorIN1 = Uno-Pin? ;
    // int8_t RightMotorIN2 = Uno-Pin? ;
    // int8_t RightMotorPWM = Uno-Pin? ;

    // (4) L298N Instantiation
    // mtr = uno::L298N(LeftMotorPWM, LeftMotorIN1, LeftMotorIN2,
    //                  RightMotorIN1, RightMotorIN2, RightMotorPWM);

    // EXAMPLE
    // SCOPE - ALL DECLARED VARIABLE HERE ARE LOST ONCE OUT OF SCOPE...
    // An example (based on author's (robot) hardware setup.)
    // Set ENA
    int8_t ENA = 10;
    int8_t IN1 = 9;
    int8_t IN2 = 8;
    // Set ENB
    int8_t IN3 = 7;
    int8_t IN4 = 6;
    int8_t ENB = 5;

    // (2) L298N to MOTORS...
    // ENA-Set
    //      (a) L298N-Motor-Outs(1 & 2) matches to ENA-Set.
    //      (b) Assigned to (Left/Right)**RIGHT** Motor.
    //      (c) The Motor assigned gets set (ENA, IN1, IN2).
    // ENB-Set
    //      (a) L298N-Motor-Outs(3 & 4) matches to ENB-Set.
    //      (b) Assigned to(Left/Right) **LEFT** Motor.
    //      (c) The Motor assigned gets set (ENB, IN3, IN4).

    // (3a) Based on (1 & 2), is this set (ENA/ENB) **ENB** ?
    int8_t LeftMotorPWM = ENB;
    int8_t LeftMotorIN1 = IN4;
    int8_t LeftMotorIN2 = IN3;
    // (3b) ) Based on (1 & 2), is this set (ENA/ENB) **ENA** ?
    int8_t RightMotorIN1 = IN1;
    int8_t RightMotorIN2 = IN2;
    int8_t RightMotorPWM = ENA;

    // L298N Instantiation (mtr was declared as a global variable. See abvoe.)
    mtr = uno::L298N(LeftMotorPWM, LeftMotorIN1, LeftMotorIN2,
                     RightMotorIN1, RightMotorIN2, RightMotorPWM);
    // L298N Setup
    mtr.SetupPinsL298N();

    // Joystick setup
    stk = uno::Joystick();
    // NOTICE: Joystick Calibration
```

```
    // Hands-Off Stick..., about
    // a second, press Arduino reset to
    // re-calibrate. HANDS-OFF...

    // Timer setup
    tmrMotors = uno::Timer();
    tmrMainDebug = uno::Timer();

    // Button setup
    // buttonPin 2
    // ledPin 12
    btnMotors = uno::Button(2, 12);
}
// BEGIN ////////////////////////////////////////////////////////////////// DEBUG
void _debug(int xAnalog, int yAnalog, int left, int right)
{
#ifdef _dM_
#ifdef _ANALOG_
    // USE SERIAL (9600) to Debug.
    //
    //  (STEP-1) ---- MAIN ANALOG -------
    // ------------------------------------------------------------------
    // HARDWARE usually solved with the software.
    // ------------------------------------------------------------------
    // SOFTWARE
    // Joystick Movements ADC Readings
    // from 0 to 1023...
    // The ideal analog readings may not display as shown yet the
    // readings should closely match the ideal values.
    // ----------------------------------------------------------
    // Center (Hands-Off) ------ Center between 0 and 1023.
    // (x, y) = (511, 511).
    // ----------------------------------------------------------
    // Up (Forward)    ---------- y-changes in (x, y).
    // (x, y) = (511, 1023)(+) or ((511, 0)(-) (where x is constant).
    // ----------------------------------------------------------
    // Down (Backward) ---------- y-changes in (x, y).
    // (x, y) = (511, 0)(+) or (511, 1023)(-) (where x is constant).
    // ----------------------------------------------------------
    // Left            ---------- x-changes in (x, y).
    // (x, y) = (0, 511)(+) or (1023, 511)(-) (where y is constant).
    // ----------------------------------------------------------
    // Right           ---------- x-changes in (x, y).
    // (x, y) = (1023, 511)(+) or (0, 511)(-) (where y is constant).
    // ----------------------------------------------------------
    // If (x, y) changes are reversed, exchange the analogRead() from (A0)
    // to (A1) or the other way around inside the main.cpp file.
    // (+) Good...
    // (-) May have to reverse either (x = 1023 - x) or the same with (y).
    // if the readings are OK, then go to MAIN JOYSTICK.
    // SEE DEBUG.h file.

    Serial.print("MAIN(xA,yA)=(");
    Serial.print(xAnalog);
    Serial.print(" , ");
    Serial.print(yAnalog);
    Serial.println(")");
```

```
#endif

#ifdef _JOYSTICK_
    //  (STEP-2) ---- MAIN JOYSTICK -------
    // ------------------------------------------------------------------
    // SOFTWARE - ALGORITHM (NO HARDWARE)
    // ------------------------------------------------------------------
    // Normalized Joystick Values from -1 to 1...
    // Hands-Off          ---------
    // [L, R] = [0, 0]   Octant 100 means [0, 0]
    // Up (Forward)       ---------
    // [L, R] = [1, 1]   Octant 101 means [0, R]
    // Down (Backward)    ---------
    // [L, R] = [-1, -1] Octant 101 means [0, R]
    // Left               ---------
    // [L, R] = [-1, 1]  Octant 110 means [L, 0]
    // Right              ---------
    // [L, R] = [1, -1]  Octant 110 means [L, 0]
    // Starting from Right Joystick Position, octants
    // should read CCW(1 to 8). Ignore octant values.
    // Should work based on MAIN ANALOG DEBUG (+), otherwise
    // in DEBUG.h, under HEADERS define _dJ_ and undefine
    // all others under MAIN definitions.
    // If OK - Go To MAIN N298N. SEE DEBUG.h
    //
    // EXAMPLES from Serial 9600
    //     (Analog)- transformed >[Joystick Normalized]   (octant)
    // Hands-Off
    // MAIN (xA,yA)=(534, 524)->[nL,nR]=[0.00, 0.00] from octant(100)
    // Up (Forward)
    // MAIN (xA,yA)=(534, 1023)->[nL,nR]=[1.00, 1.00] from octant(101)
    // Down (Backward)
    // MAIN (xA,yA)=(534, 0)->[nL,nR]=[-1.00, -1.00] from octant(101)
    // Left
    // MAIN (xA,yA)=(0, 524)->[nL,nR]=[-1.00, 1.00] from octant(110)
    // Right
    // MAIN (xA,yA)=(1023, 524)->[nL,nR]=[1.00, -1.00] from octant(110)

    int octant = stk.GetOctantCondition();
    double normalLeft = stk.GetOutputLeft();
    double normalRight = stk.GetOutputRight();
    Serial.print("MAIN (xA,yA)=(");
    Serial.print(xAnalog);
    Serial.print(", ");
    Serial.print(yAnalog);
    Serial.print(")->[nL,nR]=[");
    Serial.print(normalLeft);
    Serial.print(", ");
    Serial.print(normalRight);
    Serial.print("] from octant(");
    Serial.print(octant);
    Serial.println(")");
#endif

#ifdef _L298N_
    // (STEP-3) ----MAIN L298N-- -----
    // ------------------------------------------------------------------
```

```
// SOFTWARE. Easy where Re-wiring the hardware can be a hassle.
// ---------------------------------------------------------------
// The default constructor L298N() assigns the Uno and L298N Pins as:
// (Use this as a reference.)
// L298N::L298N()
// {
//     ...
//     _LeftMotorPWM = 10; // Arduino Pin 10  wired to L298N Pin ENA
//     _LeftMotorIN1 = 9;  // Arduino Pin 9  wired to L298N Pin IN1
//     _LeftMotorIN2 = 8;  // Arduino Pin 8  wired to L298N Pin IN2
//     _RightMotorIN1 = 7; // Arduino Pin 7  wired to L298N Pin IN3
//     _RightMotorIN2 = 6; // Arduino Pin 6  wired to L298N Pin IN4
//     _RightMotorPWM = 5; // Arduino Pin 5  wired to L298N Pin ENB
//      ...
// }

// The other L298N() constructor allows for assignment of arduino pins:
// (Of course use the same pins as the default but assign one of the
// two sets of pins to the other motor.)
// Assign Arduino Pins to Constructor
// Assign Arduino Pins to Constructor
// Assign Arduino Pins to Constructor
// L298N::L298N(uint8_t LeftMotorPWM,
//              uint8_t LeftMotorIN1,
//              uint8_t LeftMotorIN2,
//              uint8_t RightMotorIN1,
//              uint8_t RightMotorIN2,
//              uint8_t RightMotorPWM)
// {
//     ...
//     _LeftMotorPWM = LeftMotorPWM; // Uno-PWM-To-L298N-PWM
//     _LeftMotorIN1 = LeftMotorIN1; // Uno-Output-To-L298N-Digital-Input
//     _LeftMotorIN2 = LeftMotorIN2; // Uno-Output-To-L298N-Digital-Input
//     _RightMotorIN1 = RightMotorIN1; // Uno-Output-To-L298N-Digital-Input
//     _RightMotorIN2 = RightMotorIN2; // Uno-Output-To-L298N-Digital-Input
//     _RightMotorPWM = RightMotorPWM; // Uno-PWM-To-L298N-PWM
//     ...
// }
//
// Trouble-shooting:
// (1) Add Uno's Pin Numbers to Instantiate mtr = uno::L298N():
// Example: mtr = uno::L298N(10, 9, 8, 7, 6, 5);
// This is exactly the same as L298N() without the integers.
// (2) If the Left Motor in connected to ENB, invert the Order of
// Uno Pins as follows:
//     mtr = uno::L298N(5, 6, 7, 8, 9, 10);
// (3) Position the Joystick in the Up (Forward). If both motors
// move forward, then OK. Otherwise, if the left moves back. swap
// LeftIn1 and LeftIn2.
// Example: mtr = uno::L298N(5, 7, 6, 8, 9, 10);
// Notice: Just remember that the pin order for L298N() is associated
// from the left motor to the right motor. If not, this will break
// Joystick Algorithm.
// Therefore:
// L298N(LeftMotorPWM, LeftMotorIN1,  LeftMotorIN2,
//       RightMotorIN1, RightMotorIN2, RightMotorPWM)
// where LeftMotorPWM & RightMotorPWM are either ENA or ENB.
```

```
    // If LeftMotorPWM is ENB, then LeftMotorIN1 & LeftMotorIN2 must be
    // either IN3 or IN4 which belong to ENB.

    int in1 = mtr.GetIN1();
    int in2 = mtr.GetIN2();
    int in3 = mtr.GetIN3();
    int in4 = mtr.GetIN4();
    Serial.print("MAIN Left: ");
    Serial.print(left);
    Serial.print(" -> <IN1, IN2> = <");
    Serial.print(in1);
    Serial.print(", ");
    Serial.print(in2);
    Serial.println(">");
    Serial.print("MAIN Right: ");
    Serial.print(right);
    Serial.print(" -> <IN3, IN4> = <");
    Serial.print(in3);
    Serial.print(", ");
    Serial.print(in4);
    Serial.println(">");
    Serial.println("");

    // For those who want to play with the hardware
    // ----------------------------------------------------------------
    // The L298N Motor-OUTS will change voltage polarity depending
    // on the Joystick Position. Connect the Motor Leads to the L298N.
    // When the Joystick is held in the forward position, observe
    // whether either motor rotates forward. If not, then simply swap
    // leads for the left and/or the right motor.
    // ----------------------------------------------------------------
    //                       Left Motor               Right Motor
    // Arduino Pins           10  9   8                7   6   5
    // L298N Pins            ENA IN1 IN2              IN3 IN4 ENB
    // L298N Motor-outs       OUT1  OUT2               OUT3  OUT4
    // Joystick Forward     yes-OK/no-swap            yes-OK/no-swap
    // ----------------------------------------------------------------
    // Define MAIN L298N in the DEBUG.h file. Everthing else is undefined.
    // The debug output shows how the L298N inputs behave relative either to
    // the left or the right motor. The furstration is knowing which is the
    // software issue or which is the hardware issue. Always start with the
    // Hardware. The hardware is easy.
    // ----------------------------------------------------------------
    // L298N MOTOR MODULE BOARD - TECHNICAL NOTES
    // ----------------------------------------------------------------
    // 12V Geared Ratio 1:75 Motors (China)
    // 12V Battery (USA)
    // NOTICE 1: JUMPER - If Battery is greater then 12V, then
    //                    remove jumper. (1) Jumper-PLUGGED-IN. The purpose
    //                    for the jumper is to supply 5V to the L298N Logic
    //                    circuits. (2) Jumper-PULLED-OFF. You have to supply
    //                    the L298N logic 5V either directly from the Arduino
    //                    or an external battery 5V regulator supply. (See 5V Out).
    // NOTICE 2: IF MOTORS BEGIN TO MOVE ON THEIR OWN, PRESS UNO-RESET BUTTON
    //           FOR RE-CALIBRATION TO CENTER JOYSTICK. HANDS-OFF JOYSTICK WHEN
    //           CALIBRATING.
    //                        ------------------------
```

```
    //                        |     |   L298N   |     |
    //                        |     |           |     |
    //                        |     -------------     |
    //              LEFT-MOTOR |----                 ----| RIGHT-MOTOR
    //                   OUT1  | O |                 | O |  OUT4
    //                   OUT2  | O | jumper          | O |  OUT3
    //                         |-------------      ----|
    //                         |    | O O O |  A1234B  |<- LOGIC A - ENA
    //                          -------------------------         1 - IN1
    //                       Battery(+)(-)(5V Out)                2 - IN2 etc.
    //
    // ---------------------------------------------------------------------
    // SOFTWARE
    // ---------------------------------------------------------------------
    // Software Examples.
    // LOW = 0, HIGH = 1..., MAX PWM 255
    // Serial 9600 - LAB-RESULTS                |  INTERPRETATION
    // JOYSTICK CENTER POSITION - HANDS OFF ---- MOTOR-OUTS - MULTIMETER
    // MAIN Left: 0 -> <IN1, IN2> = <0, 0>      | 1(-) 2(-)   No Voltage
    // MAIN Right: 0 -> <IN3, IN4> = <0, 0>     | 3(-) 4(-)   No Voltage
    // JOYSTICK UP POSITION - FORWARD ----------------CHECK-TWICE-[jc]--
    // MAIN Left: 255 -> <IN1, IN2> = <0, 1>    | 1(-) 2(+) +14V Forward
    // MAIN Right: 255 -> <IN3, IN4> = <0, 1>   | 3(-) 4(+) +14V Forward
    // JOYSTICK DOWN POSITION - BACKWARD --------------CHECK-TWICE-[jc]--
    // MAIN Left: -253 -> <IN1, IN2> = <1, 0>   | 1(+) 2(-) -14V Backward
    // MAIN Right: -253 -> <IN3, IN4> = <1, 0>  | 3(+) 4(-) -14V Backward
    // JOYSTICK LEFT POSITION            --------------CHECK-TWICE-[jc]--
    // MAIN Left : -254 -> <IN1, IN2> = <1, 0>  | 1(+) 2(-) -14V Backward
    // MAIN Right : 254 -> <IN3, IN4> = <0, 1>  | 1(-) 2(+) +14V Forward
    // JOYSTICK RIGHT POSITION            -------------CHECK-TWICE-[jc]--
    // MAIN Left : 255 -> <IN1, IN2> = <0, 1>   | 1(-) 2(+) +14V Forward
    // MAIN Right : -255 -> <IN3, IN4> = <1, 0> | 1(+) 2(-) -14V Backward
    // ---------------------------------------------------------------------
    // NOTICE 3: If there are issues with the motor behavior and
    //           if the debug show consistent values shown above, then
    //           the problem is not the software but it is the hardware.
    // ---------------------------------------------------------------------
    // NOTICE 4: L298N INPUT WITH <1, 1> VALUES ARE NOT NECESSARY.
    // ---------------------------------------------------------------------
    // NOTICE 5: L298N does not use negative values but before the
    // software math absolute abs() on the PWM Left & PWM Right values,
    // the negative values are used here for debug. The signs on
    // the Left & Right tells which direction the motors are heading.
    // The angle-brackets are the HIGH & LOW for the L298N Input pins.
    // Again this is not easy, but you are almost there.
    ////////////////////////////////////////////////////////////////////
    // Final note: These three steps ought to solve most issues,
    // otherwise read the code and start debugging.
    ////////////////////////////////////////////////////////////////////
    // The orientation of the joystick or the L298N board as presented
    // relative to the MCU or motors may not be the same, but this guide
    // can still be used to help.
    ////////////////////////////////////////////////////////////////////
#endif
#endif
}
// END ////////////////////////////////////////////////////////////////// DEBUG
```

```
void updateAnalog(bool debugMotors)
{
    // ---- ANALOG -------
    // Must Read ADC while
    // JoystickCalibrating Center...
    //    * KEEP-IN-MIND *
    // * HANDS-OFF-OF-JOYSTICK *
    //
    // Thumb-Joystick Orientation
    // sometimes give unexpected
    // analog readings. Therefore
    // use DEBUG.h (undefine everything
    // except MAIN ANALOG) to test
    // analog as described in _debug() in
    // main.cpp file.
    // * x-analog reads from 0 to 1023 *
    int xAnalog = analogRead(A1);
    // Uncomment if x-analog reads from 1023 to 0.
    // xAnalog = _1023 - xAnalog;
    // * y-analog reads from 0 to 1023 *
    int yAnalog = analogRead(A0);
    // Uncomment if analog reads from 1023 to 0.
    // yAnalog = _1023 - yAnalog;
    // For furhter details see _debug() ANALOG.

    // Joystick-IN
    //
    stk.UpdateInputs(xAnalog, yAnalog);
    // Joystick-OUT - L298N PWM 255
    int const left = stk.GetOutputLeftInteger(255);
    int const right = stk.GetOutputRightInteger(255);

    if (debugMotors)
    {
        if (tmrMainDebug.isTimer(2500))
        {
            // Scope TRUE Variable
            bool debugL298N = true;

            // Update Motors
            mtr.updateL298N(debugL298N, left, right);
            _debug(xAnalog, yAnalog, left, right);
            // Motors Off - Safety Reasons
            // Notice: While debugging, the motors
            // are conditioned not to operate.
            // See L298N updateL298N().
            // Evething works as though if the
            // motors were active...
        }
    }
    else if (btnMotors.isButtonOn())
    {
        // Scope FALSE Variable
        bool debugL298N = false;

        // Update Motors
```

```
        mtr.updateL298N(debugL298N, left, right);
    }
}

void loop()
{
    btnMotors.updateButton();
    // 50ms timer cycles at 20Hz
    // Enough time for motors...
    if (tmrMotors.isTimer(50))
    {
#ifdef _dM_ // See _debug() & DEBUG.h
        // Scope TRUE Variable
        bool debugMotors = true;
        updateAnalog(debugMotors);
#else
        // Scope FALSE Variable
        bool debugMotors = false;
        updateAnalog(debugMotors);
#endif
    }
}
```

```
//
// Carpenter Software
// File: Class Joystick.h (debug version)
// Folder: joystick_L298N_debug
// Purpose: Github Depository (MageMCU)
//
// Teacher's and Parent's copy...
// Dedicated to Parents and Teachers of the USA.
// Teachers Note: The _BEGIN_DEBUG is defined (see
//                DEBUG.h file), also MAIN _ANALOG_
//                is defined for debugging the
//                analog readings. For safety
//                reasons while debugging, the
//                motors are conditioned not to
//                operate. Be warned. the Student's
//                copy has no safety feature.
//
// Hardware
// MCU: Atmega328P (Microchip)
// * Board: Arduino Uno (Italy)
// Thumb Joystick: (China)
// Motor Driver: L298N (ST)
// * Motor Module: (China)
// 12V Geared Motors (ratio 1:75) (China)
// Note: motors: current peaks on average 1.14 amps
//       for static friction, and it levels down to
//       about 0.29 amps for dynamic friction. When
//       joystick with hands-off (center position),
//       is about 0.10 amps. The robot was sitting
//       on stand with no weight on wheels, otherwise,
//       the electrical currents would have increased.
// Battery 12V - APX1250 (12V 5.0Ah) Sealed Lead-Acid
// Note: battery: with five years design life (using
//       now for 7 years.) Requires special battery
//       charger from APEX.
//
// By Jesse Carpenter (Github as MageMCU)
//
// CHANGELOG
// August 30, 2021 - debug comments
// August 24, 2021 - editing
// June 19, 2021 - programming debug version
//
// Atmega328P Total Usgae
// RAM : [==] 15.3 % (used 314 bytes from 2048 bytes)
// Flash : [==] 21.2 % (used 6846 bytes from 32256 bytes)
//
// MIT LICENSE
//

#ifndef __UNO_JOYSTICK_H__
#define __UNO_JOYSTICK_H__

#include <Arduino.h>
#include "DEBUG.h"

// INPUT ADC READINGS
#define _ZERO 0
#define _MIDPOINT_LO 511
#define _MIDPOINT_HI 512
#define _1023 1023

// Using Adruino Uno
namespace uno
{
  class Joystick
  {
  private:
```

```
    // Properties ////////////////////////////////
    bool _ready;
    // Temp Debug Value
    int _octant;

    // Raw ADC Values
    int _rawX;
    int _rawY;
    // _1_alignRawInput()
    int _alignX;
    int _alignY;
    // _2_shiftFromZeroToMidpoint()
    int _shiftX;
    int _shiftY;
    // _3_adjustToDynamicFriction()
    int _dynamicX;
    int _dynamicY;
    // _4_convertToDouble
    double _doubleX;
    double _doubleY;
    // Motors Speeds
    double _normalLeft;
    double _normalRight;

    // Calibrate RAW Analog Data
    int _cntCalibration21;
    // Offset Joystick Center
    int _xCenterOffset;
    int _yCenterOffset;
    // Offset Static Friction
    int _xDynamicOffset;
    int _yDynamicOffset;
    // Methods ///////////////////////////////////
    void _dataJoystick();
    // Center Joystick
    boolean _centerCalibrated();
    // Align Raw Data
    void _1_alignRawInput();
    // Shift To Zero
    void _2_shiftFromZeroToMidpoint();
    // Adjust for static friction
    void _3_adjustToDynamicFriction();
    // Convert from integer to double
    void _4_convertToDouble();
    // Calculate Motors Speeds
    void _5_joystick();
    // Setup for Constructors
    void _debugConstructor(/* Debug Parameters */);
    void _allDebug(/* Debug Parameters */);

  public:
    // Constructor
    Joystick();
    // Methods ///////////////////////////////////
    // Debug Octants
    bool Ready();
    int GetOctantCondition();
    // Input - Analog Readings - ADC
    void UpdateInputs(int rawX, int rawY);
    double GetRawX();
    double GetRawY();
    // Output Motors Speeds with maximum PWM value
    // Arduino Uno & L298N = 255
    int GetOutputLeftInteger(int maxL298N_PWM);
    int GetOutputRightInteger(int maxL298N_PWM);
    // Output - Motors Speeds (for debug)
    double GetOutputLeft();
    double GetOutputRight();
```

```
};

// PUBLIC ////////////////////////////////////////////
Joystick::Joystick()
{
  _dataJoystick();
}

bool Joystick::Ready()
{
  return _ready;
}

int Joystick::GetOctantCondition()
{
  // Used for Debugging
  return _octant;
}

double Joystick::GetRawX()
{
  return _rawX;
}

double Joystick::GetRawY()
{
  return _rawY;
}

double Joystick::GetOutputLeft()
{
  return _normalLeft;
}

double Joystick::GetOutputRight()
{
  return _normalRight;
}

int Joystick::GetOutputLeftInteger(int maxL298N_PWM)
{
  return (int)(_normalLeft * (double)maxL298N_PWM);
}

int Joystick::GetOutputRightInteger(int maxL298N_PWM)
{
  return (int)(_normalRight * (double)maxL298N_PWM);
}

void Joystick::UpdateInputs(int const rawX, int const rawY)
{
  // Assign Inputs
  _rawX = rawX;
  _rawY = rawY;

  // Ready (or not here we go)
  if (_cntCalibration21 < 22)
    _ready = false;
  else
    _ready = true;

#ifdef _dJ_
  if (_cntCalibration21 == 0)
    Serial.println("*** Calibrating ***");
#endif

  if (_centerCalibrated()) // Simple 21 count two point averaging
  {
```

```
      // Keep each step simple.
      _1_alignRawInput();           // IN - Align Center to MIDPOINT
      _2_shiftFromZeroToMidpoint(); // IN - Shift Center to ZERO
      _3_adjustToDynamicFriction(); // IN - Jump over static friction
      _4_convertToDouble();         // IN - Convert Integers to Doubles
      _5_joystick();                // OUT - Transform to normalized output
      _allDebug();
    }
  }

  // PRIVATE ////////////////////////////////////////
  void Joystick::_dataJoystick()
  {
    _ready = false;
    // Calibration and Offsets
    _cntCalibration21 = 0;
    _debugConstructor(/* Debug Arguments */);
  }

  boolean Joystick::_centerCalibrated()
  {
    // Simple 21 count two point averaging
    if (_cntCalibration21 <= 21)
    {
      _xCenterOffset = ((_xCenterOffset + _rawX) / 2) + 1;
      _yCenterOffset = ((_yCenterOffset + _rawY) / 2) + 1;

#ifdef _dJ_
      Serial.print(_cntCalibration21);
      Serial.print(" _xC:");
      Serial.print(_xCenterOffset);
      Serial.print(" _yC:");
      Serial.println(_yCenterOffset);
#endif

      _cntCalibration21++;
      return false;
    }

    // Increment Value Remains at 22..., unless
    // MCU Reset.
    return true;
  }

  void Joystick::_1_alignRawInput()
  {
    int _x = _rawX;
    int _y = _rawY;

    if (_x >= _xCenterOffset)
    {
      // _x = (int)_util.Map(_x, _xCenterOffset, _1023, _MIDPOINT_HI, _1023);
      _x = (int)map(_x, _xCenterOffset, _1023, _MIDPOINT_HI, _1023);
    }
    else if (_x < _xCenterOffset)
    {
      // _x = (int)_util.Map(_x, _ZERO, (_xCenterOffset - 1), _ZERO, _MIDPOINT_LO);
      _x = (int)map(_x, _ZERO, (_xCenterOffset - 1), _ZERO, _MIDPOINT_LO);
    }

    if (_y >= _yCenterOffset)
    {
      // _y = (int)_util.Map(_y, _yCenterOffset, _1023, _MIDPOINT_HI, _1023);
      _y = (int)map(_y, _yCenterOffset, _1023, _MIDPOINT_HI, _1023);
    }
    else if (_y < _yCenterOffset)
    {
      // _y = (int)_util.Map(_y, _ZERO, (_yCenterOffset - 1), _ZERO, _MIDPOINT_LO);
```

```
      _y = (int)map(_y, _ZERO, (_yCenterOffset - 1), _ZERO, _MIDPOINT_LO);
    }

    _alignX = _x;
    _alignY = _y;
  }
  void Joystick::_2_shiftFromZeroToMidpoint()
  {
    // Offset to Center from 511 to zero
    _shiftX = _alignX - _MIDPOINT_LO;
    _shiftY = _alignY - _MIDPOINT_LO;
    // Center should now be (0,0)
  }

  void Joystick::_3_adjustToDynamicFriction()
  {
    int const _centerTolerance = 5;
    int _x = _shiftX;
    int _y = _shiftY;

    //  ------------------------------------------------------
    //  (Experimental Step)
    //  Calibrate by stepping over static friction (whinning
    //  motors) to dynamic friction (when motors actually start
    //  moving the robot under full load weight). The threshold
    //  values ought to be considered for other smooth solid
    //  surfaces (sss) like carpet. (Whinning motors uses a
    //  lot of energy.)
    //  RECORD digital ADC reading values as follows:
    //  ------------------------------------------------------
    //  motor      -      static      -      dynamic
    //                   whinning         movement for sss
    //  ------------------------------------------------------
    //  _xDynamicOffset (-)
    //                  (+)
    //  _yDynamicOffset (-)
    //                  (+)
    //  ------------------------------------------------------
    //
    // _xDynamicOffset = _centerTolerance; // Comment-out _centerTolerance and replace
with
    // emperical dynamic value..., otherwise do not
    // touch
    _xDynamicOffset = 260; // Author's  (suggest you change this)
    if (_x >= _centerTolerance)
    {
      // _x = (int)_util.Map(_x, 0, 512, _xDynamicOffset, 512);
      _x = (int)map(_x, 0, 512, _xDynamicOffset, 512);
    }
    else if (_x < -_centerTolerance)
    {
      // _x = (int)_util.Map(_x, 0, -511, -_xDynamicOffset, -511);
      _x = (int)map(_x, 0, -511, -_xDynamicOffset, -511);
    }
    else
    {
      _x = 0;
    }
    //
    // _yDynamicOffset = _centerTolerance; // Comment-out _centerTolerance and replace
with
    // emperical dynamic value..., otherwise do not
    // touch.
    _yDynamicOffset = 260; // Author's  (suggest you change this)
    if (_y >= _centerTolerance)
    {
      // _y = (int)_util.Map(_y, _ZERO, _MIDPOINT_HI, _yDynamicOffset, _MIDPOINT_HI);
      _y = (int)map(_y, _ZERO, _MIDPOINT_HI, _yDynamicOffset, _MIDPOINT_HI);
```

```
    }
    else if (_y < -_centerTolerance)
    {
      // _y = (int)_util.Map(_y, _ZERO, -_MIDPOINT_LO, -_yDynamicOffset,
-_MIDPOINT_LO);
      _y = (int)map(_y, _ZERO, -_MIDPOINT_LO, -_yDynamicOffset, -_MIDPOINT_LO);
    }
    else
    {
      _y = 0;
    }

    _dynamicX = _x;
    _dynamicY = _y;
  }

  void Joystick::_4_convertToDouble()
  {
    _doubleX = (double)_dynamicX / (double)_MIDPOINT_HI;
    _doubleY = (double)_dynamicY / (double)_MIDPOINT_HI;
  }

  void Joystick::_5_joystick()
  {
    double const _tol = 0.04;
    // Assign Processed Doubles
    double const _inputX = _doubleX;
    double const _inputY = _doubleY;
    // The Method uses DOUBLE TYPE so
    // the method should be tested for
    // performance. OK...
    double _x = 0;
    double _y = 0;

    // (1) Delta
    // Absolute difference of the absolute values of the inputs.
    double delta = abs(abs(_inputX) - abs(_inputY));

    // (2) Conditionals
    if (abs(_inputX) > _tol && abs(_inputY) > _tol)
    {
      // Y-Dominator
      if (abs(_inputX) <= abs(_inputY))
      {
        if (_inputX > _ZERO && _inputY > _ZERO)
        {
          _x = _inputY;
          _y = delta;
          // Octant 2
          _octant = 2;
        }
        else if (_inputX > _ZERO && _inputY < _ZERO)
        {
          _x = -delta;
          _y = _inputY;
          // Octant 7
          _octant = 7;
        }
        else if (_inputX < _ZERO && _inputY > _ZERO)
        {
          _x = delta;
          _y = _inputY;
          // Octant 3
          _octant = 3;
        }
        else if (_inputX < _ZERO && _inputY < _ZERO)
        {
          _x = _inputY;
```

```
          _y = -delta;
          // Octant 6
          _octant = 6;
        }
        // X-Dominator
      }
      else if (abs(_inputX) > abs(_inputY))
      {
        if (_inputX > _ZERO && _inputY > _ZERO)
        {
          _x = _inputX;
          _y = -delta;
          // Octant 1
          _octant = 1;
        }
        else if (_inputX > _ZERO && _inputY < _ZERO)
        {
          _x = delta;
          _y = -_inputX;
          // Octant 8
          _octant = 8;
        }
        else if (_inputX < _ZERO && _inputY > _ZERO)
        {
          _x = -delta;
          _y = -_inputX;
          // Octant 4
          _octant = 4;
        }
        else if (_inputX < _ZERO && _inputY < _ZERO)
        {
          _x = _inputX;
          _y = delta;
          // Octant 5
          _octant = 5;
        }
      }
    }
    else if (abs(_inputX) > _tol && abs(_inputY) < _tol)
    {
      _x = _inputX;
      _y = -_inputX;
      // input x is greater than zero and input y is zero
      _octant = 110;
    }
    else if (abs(_inputX) < _tol && abs(_inputY) > _tol)
    {
      _x = _inputY;
      _y = _inputY;
      // input x is zero and y is greater than zero
      _octant = 101;
    }
    else if (abs(_inputX) < _tol && abs(_inputY) < _tol)
    {
      _x = 0;
      _y = 0;
      // input x and y are zero
      _octant = 100;
    }

    _normalLeft = _x;
    _normalRight = _y;
  }

  void Joystick::_debugConstructor(/* Debug Parameters */)
  {
#ifdef _dJ_
```

```
    // Used simply whether or not
    // Constructor Instantiated Joystick CLASS
    Serial.println("Joystick");
// Write Serial Prints Here...
#endif
  }

  void Joystick::_allDebug(/* Debug Parameters */)
  {
#ifdef _dJ_
    Serial.println("*** Joystick ***");
    // Write Serial Prints Here...

    Serial.println("ANALOG IN");
    Serial.print(" X:");
    Serial.print(_rawX);
    Serial.print(" Y:");
    Serial.println(_rawY);

#ifdef _ALIGN_
    Serial.println("1 ALIGN IN");
    Serial.print(" X:");
    Serial.print(_alignX);
    Serial.print(" Y:");
    Serial.println(_alignY);
#endif
#ifdef _SHIFT_
    Serial.println("2 SHIFT IN");
    Serial.print(" X:");
    Serial.print(_shiftX);
    Serial.print(" Y:");
    Serial.println(_shiftY);
#endif
#ifdef _DYNAMIC_
    Serial.println("3 DYNAMIC IN");
    Serial.print(" X:");
    Serial.print(_dynamicX);
    Serial.print(" Y:");
    Serial.println(_dynamicY);
#endif
#ifdef _DOUBLE_
    Serial.println("4 DOUBLE IN");
    Serial.print(" X:");
    Serial.print(_doubleX);
    Serial.print(" Y:");
    Serial.println(_doubleY);
#endif
#ifdef _NORMAL_
    // Joystick NORMALIZED Readings
    // from -1 to 1...
    // OK if Test Values.
    // Hands-Off          ---------
    // (x, y) = (0, 0)    not-Octant 100 = (0, 0)
    // Up (Forward)       ---------
    // (x, y) = (1, 1))   not-Octant 101 = (0, y)
    // Down (Backward)    ---------
    // (x, y) = (-1, -1)) not-Octant 101 = (0, y)
    // Left               ---------
    // (x, y) = (-1, 1))  not-Octant 110 = (x, 0)
    // Right              ---------
    // (x, y) = (1, -1))  not-Octant 110 = (x, 0)
    // Starting Right, octants should read CCW(1 to 8).
    // Octant values - ignore
    // If OK - Go To L298N
    // SEE DEBUG.h

    Serial.println("5 MOTORS OUT");
    Serial.print(" Octant:");
```

```
    Serial.print(_octant);
    Serial.print(" X:");
    Serial.print(_normalLeft);
    Serial.print(" Y:");
    Serial.println(_normalRight);
#endif
#endif
  }
}
#endif
```

```
//
// Carpenter Software
// File: Class L298N.h (debug version)
// Folder: joystick_L298N_debug
// Purpose: Github Depository (MageMCU)
//
// Teacher's and Parent's copy...
// Dedicated to Parents and Teachers of the USA.
// Teachers Note: The _BEGIN_DEBUG is defined (see
//                DEBUG.h file), also MAIN _ANALOG_
//                is defined for debugging the
//                analog readings. For safety
//                reasons while debugging, the
//                motors are conditioned not to
//                operate. Be warned. the Student's
//                copy has no safety feature.
//
// Hardware
// MCU: Atmega328P (Microchip)
// * Board: Arduino Uno (Italy)
// Thumb Joystick: (China)
// Motor Driver: L298N (ST)
// * Motor Module: (China)
// 12V Geared Motors (ratio 1:75) (China)
// Note: motors: current peaks on average 1.14 amps
//       for static friction, and it levels down to
//       about 0.29 amps for dynamic friction. When
//       joystick with hands-off (center position),
//       is about 0.10 amps. The robot was sitting
//       on stand with no weight on wheels, otherwise,
//       the electrical currents would have increased.
// Battery 12V - APX1250 (12V 5.0Ah) Sealed Lead-Acid
// Note: battery: with five years design life (using
//       now for 7 years.) Requires special battery
//       charger from APEX.
//
// By Jesse Carpenter (Github as MageMCU)
//
// CHANGELOG
// August 30, 2021 - debug comments
// August 24, 2021 - editing
// June 19, 2021 - programming debug version
//
// Atmega328P Total Usgae
// RAM : [==] 15.3 % (used 314 bytes from 2048 bytes)
// Flash : [==] 21.2 % (used 6846 bytes from 32256 bytes)
//
// MIT LICENSE
//

#ifndef __UNO_L298N_H__
#define __UNO_L298N_H__

#include <Arduino.h>
#include "DEBUG.h"

// Using Adruino Uno
namespace uno
{
    class L298N
    {
    private:
        uint8_t _ZERO;

        uint8_t _LeftMotorPWM;
        uint8_t _LeftMotorIN1;
        uint8_t _LeftMotorIN2;
```

```
        uint8_t _RightMotorIN1;
        uint8_t _RightMotorIN2;
        uint8_t _RightMotorPWM;

        // Debug Variables
        int _PWM_LeftMotor;
        int _PWM_RightMotor;
        // used as boolean
        uint8_t _IN1_Flag;
        uint8_t _IN2_Flag;
        uint8_t _IN3_Flag;
        uint8_t _IN4_Flag;

        // Private method
        void _setDirectionPins();
        void _powerDownL298N();
        void _debugConstructor(/* Debug Parameters */);
        void _allDebug(/* Debug Parameters */);

    public:
        L298N();
        L298N(uint8_t LeftMotorPWM,
              uint8_t LeftMotorIN1,
              uint8_t LeftMotorIN2,
              uint8_t RightMotorIN1,
              uint8_t RightMotorIN2,
              uint8_t RightMotorPWM);

        // Getter Methods
        int GetIN1();
        int GetIN2();
        int GetIN3();
        int GetIN4();
        // Place inside setup() function
        void SetupPinsL298N();

        // Place inside loop() function
        void updateL298N(bool debugMotors, int UnoPWM_ToENA, int UnoPWM_ToENB);
    };

    // Default Constructor
    L298N::L298N()
    {
        _ZERO = 0;

        _LeftMotorPWM = 10; // Arduino Pin 10  wired to L298N Pin ENA
        _LeftMotorIN1 = 9;  // Arduino Pin 9  wired to L298N Pin IN1
        _LeftMotorIN2 = 8;  // Arduino Pin 8  wired to L298N Pin IN2
        _RightMotorIN1 = 7; // Arduino Pin 7  wired to L298N Pin IN3
        _RightMotorIN2 = 6; // Arduino Pin 6  wired to L298N Pin IN4
        _RightMotorPWM = 5; // Arduino Pin 5  wired to L298N Pin ENB

        _debugConstructor(/* Debug Arguments */);
    }

    // Assign Arduino Pins to Constructor
    L298N::L298N(uint8_t LeftMotorPWM,
                 uint8_t LeftMotorIN1,
                 uint8_t LeftMotorIN2,
                 uint8_t RightMotorIN1,
                 uint8_t RightMotorIN2,
                 uint8_t RightMotorPWM)
    {
        _ZERO = 0;
        // LEFT SET
        _LeftMotorPWM = LeftMotorPWM; // Uno-PWM-To-L298N-PWM
        _LeftMotorIN1 = LeftMotorIN1; // Uno-Output-To-L298N-Digital-Input
        _LeftMotorIN2 = LeftMotorIN2; // Uno-Output-To-L298N-Digital-Input
```

```
    // RIGHT SET
    _RightMotorIN1 = RightMotorIN1; // Uno-Output-To-L298N-Digital-Input
    _RightMotorIN2 = RightMotorIN2; // Uno-Output-To-L298N-Digital-Input
    _RightMotorPWM = RightMotorPWM; // Uno-PWM-To-L298N-PWM

    _debugConstructor(/* Debug Arguments */);
}

int L298N::GetIN1()
{
    return (int)_IN1_Flag;
}

int L298N::GetIN2()
{
    return (int)_IN2_Flag;
}

int L298N::GetIN3()
{
    return (int)_IN3_Flag;
}

int L298N::GetIN4()
{
    return (int)_IN4_Flag;
}

// Adjust  Motors
void L298N::updateL298N(bool debugMotors, int PWM_LeftMotor, int PWM_RightMotor)
{
    // Debug with Negative Values
    _PWM_LeftMotor = PWM_LeftMotor;
    _PWM_RightMotor = PWM_RightMotor;

    // L298N_DirectionPins requires negative numbers
    _setDirectionPins();

    // L298N does not use negative values
    // If both Input Pair Pins (IN1 & IN2, IN3 & IN4) are
    // all set to LOW then the motors will not RUN..., no matter
    // what the motor speed values might be.
    // See L298N_DirectionPins() method for details.

    if (debugMotors)
    {
        _allDebug();
    }
    else // Motors Off - Safety Reasons
    {
        // ENA Motor
        analogWrite(_LeftMotorPWM, abs(PWM_LeftMotor));
        // ENB Motor
        analogWrite(_RightMotorPWM, abs(PWM_RightMotor));
    }
}

void L298N::_powerDownL298N()
{
    digitalWrite(_LeftMotorIN1, LOW);
    _IN1_Flag = LOW;
    digitalWrite(_LeftMotorIN2, LOW);
    _IN2_Flag = LOW;
    digitalWrite(_RightMotorIN1, LOW);
    _IN3_Flag = LOW;
    digitalWrite(_RightMotorIN2, LOW);
    _IN4_Flag = LOW;
}
```

```
    void L298N::SetupPinsL298N()
    {
        // INTERFACE: Motors
        pinMode(_LeftMotorPWM, OUTPUT);
        pinMode(_LeftMotorIN1, OUTPUT);
        pinMode(_LeftMotorIN2, OUTPUT);
        pinMode(_RightMotorIN1, OUTPUT);
        pinMode(_RightMotorIN2, OUTPUT);
        pinMode(_RightMotorPWM, OUTPUT);
        //
        _powerDownL298N();
    }

    void L298N::_setDirectionPins()
    {
        // Center Tolerance is adjusted in Joystick Method
        // in the Manager Class (Header and Code Files)
        //
        // Note: IN1 and IN2 wheel direction can be changed
        //       by changing the conditionals "<" and ">"...,
        //       the order of the digitalWrite().
        //       Much easier changing code then wires on a motor.
        //
        // WARNING: Never set paired 1&2 values to HIGH at
        //          the same time. ALways set the FIRST-INPUT-PIN
        //          to LOW before setting the following PIN to HIGH.
        //          This will prevent two PINS from being HIGH at
        //          once.
        // This also includes IN3 and IN4 wheel direction pins below...

        // LEFT Motor ////////////////////////////////
        if (_PWM_LeftMotor > _ZERO)
        {
            digitalWrite(_LeftMotorIN1, LOW);
            _IN1_Flag = LOW;
            digitalWrite(_LeftMotorIN2, HIGH);
            _IN2_Flag = HIGH;
        }
        else if (_PWM_LeftMotor < _ZERO)
        {
            digitalWrite(_LeftMotorIN2, LOW);
            _IN2_Flag = LOW;
            digitalWrite(_LeftMotorIN1, HIGH);
            _IN1_Flag = HIGH;
        }
        else
        {
            digitalWrite(_LeftMotorIN1, LOW);
            _IN1_Flag = LOW;
            digitalWrite(_LeftMotorIN2, LOW);
            _IN2_Flag = LOW;
        }

        // RIGHT Motor ////////////////////////////////
        if (_PWM_RightMotor > _ZERO)
        {
            digitalWrite(_RightMotorIN1, LOW);
            _IN3_Flag = LOW;
            digitalWrite(_RightMotorIN2, HIGH);
            _IN4_Flag = HIGH;
        }
        else if (_PWM_RightMotor < _ZERO)
        {
            digitalWrite(_RightMotorIN2, LOW);
            _IN4_Flag = LOW;
            digitalWrite(_RightMotorIN1, HIGH);
            _IN3_Flag = HIGH;
```

```
        }
        else
        {
            digitalWrite(_RightMotorIN1, LOW);
            _IN3_Flag = LOW;
            digitalWrite(_RightMotorIN2, LOW);
            _IN4_Flag = LOW;
        }
    }

    void L298N::_debugConstructor(/* Debug Parameters */)
    {
#ifdef _dL_
        // Used simply whether or not
        // Constructor Instantiated L298N CLASS
        Serial.println("L298N");
        // Write Serial Prints Here...
#endif
    }

    void L298N::_allDebug(/* Debug Parameters */)
    {
        // The problem debugging L298N is the
        // confusion between Hardware and Software...
        // See L298N in the _debug() in the main.cpp
        // file before play with this code.

        // you want to DEBUG?
#ifdef _SOMETHING_
        // Serial Print Statements.
#endif
    }
}

#endif
```

```
//
// Carpenter Software
// File: Class Button.h (debug version)
// Folder: joystick_L298N_debug
// Purpose: Github Depository (MageMCU)
//
// Teacher's and Parent's copy...
// Dedicated to Parents and Teachers of the USA.
// Teachers Note: The _BEGIN_DEBUG is defined (see
//                DEBUG.h file), also MAIN _ANALOG_
//                is defined for debugging the
//                analog readings. For safety
//                reasons while debugging, the
//                motors are conditioned not to
//                operate. Be warned. the Student's
//                copy has no safety feature.
//
// Hardware
// MCU: Atmega328P (Microchip)
// * Board: Arduino Uno (Italy)
// Thumb Joystick: (China)
// Motor Driver: L298N (ST)
// * Motor Module: (China)
// 12V Geared Motors (ratio 1:75) (China)
// Note: motors: current peaks on average 1.14 amps
//       for static friction, and it levels down to
//       about 0.29 amps for dynamic friction. When
//       joystick with hands-off (center position),
//       is about 0.10 amps. The robot was sitting
//       on stand with no weight on wheels, otherwise,
//       the electrical currents would have increased.
// Battery 12V - APX1250 (12V 5.0Ah) Sealed Lead-Acid
// Note: battery: with five years design life (using
//       now for 7 years.) Requires special battery
//       charger from APEX.
//
// By Jesse Carpenter (Github as MageMCU)
//
// CHANGELOG
// August 30, 2021 - debug comments
// August 24, 2021 - editing
// June 19, 2021 - programming debug version
//
// Atmega328P Total Usgae
// RAM : [==] 15.3 % (used 314 bytes from 2048 bytes)
// Flash : [==] 21.2 % (used 6846 bytes from 32256 bytes)
//
// MIT LICENSE
//
// Timer Class was adapted from Arduino's
// Debounce Sketch Example.

#ifndef __UNO_BUTTON_H__
#define __UNO_BUTTON_H__

#include <Arduino.h>

#define DEBOUNCE_TIME 75

namespace uno
{
  class Button
  {
  private:
    int _ledPin;
    int _ledState;
    int _buttonPin;
    int _buttonState;
```

```
    int _lastButtonState;
    unsigned long _lastDebounceTime;
    unsigned long _debounceDelay;

    void _data();
    void _pins();
    void _button();

    void _debugConstructor(/* Debug Parameters */);
    void _allDebug(/* Debug Parameters */);

  public:
    Button();
    Button(int buttonPin);
    Button(int buttonPin, int ledPin);

    bool isButtonOn();
    void updateButton();
  };

  // Constructor
  Button::Button()
  {
    _pins();
    _data();
  }

  // Constructor
  Button::Button(int buttonPin)
  {
    _pins();
    _data();
    _buttonPin = buttonPin;
  }

  // Constructor
  Button::Button(int buttonPin, int ledPin)
  {
    _pins();
    _data();
    _buttonPin = buttonPin;
    _ledPin = ledPin;
  }

  // PUBLIC method: Is the Button On = HIGH
  bool Button::isButtonOn()
  {
    return _ledState == HIGH;
  }

  // PUBLIC method: updateButton (used in the Arduino loop() function)
  void Button::updateButton()
  {
    _button();

    _allDebug(/* Debug Arguments */);
  }

  // Private Method
  void Button::_data()
  {
    _buttonPin = 2;
    _ledPin = 13;
    // Begin the current state LOW of the output pin
    _ledState = LOW;
    // the previous reading from the input pin
    _lastButtonState = HIGH;
    _lastDebounceTime = 0;
```

```
    _debounceDelay = DEBOUNCE_TIME;
  }

  // Private Method
  void Button::_pins()
  {
    pinMode(_buttonPin, INPUT);
    pinMode(_ledPin, OUTPUT);
  }

  // Private Method
  void Button::_button()
  {
    int currentState = digitalRead(_buttonPin);
    long currentTime = millis();

    // If the switch changed, due to noise or pressing:
    if (currentState != _lastButtonState)
    {
      // reset the debouncing timer
      _lastDebounceTime = currentTime;
    }

    // whatever the reading is at, it's been there for longer than the debounce
    // delay, so take it as the actual current state:
    if ((currentTime - _lastDebounceTime) > _debounceDelay)
    {
      // if the button state has changed:
      if (currentState != _buttonState)
      {
        // Add the current reading from the input pin
        _buttonState = currentState;

        // only toggle the LED if the new button state is HIGH
        if (_buttonState == HIGH)
        {
          _ledState = !_ledState;
        }
      }
    }

    // set the LED:
    digitalWrite(_ledPin, _ledState);

    // save the reading. Next time through the loop, it'll be the lastButtonState:
    _lastButtonState = currentState;
  }

  void Button::_debugConstructor(/* Debug Parameters */)
  {
#ifdef _dB_
    // Used simply whether or not
    // Constructor Instantiated Button CLASS
    Serial.println("Button");
    // Write Serial Prints Here...
#endif
  }

  void Button::_allDebug(/* Debug Parameters */)
  {
#ifdef _dB_
    // Write Serial Prints Here...
#endif
  }
}
#endif
```

```
//
// Carpenter Software
// File: Class Timer.h (debug version)
// Folder: joystick_L298N_debug
// Purpose: Github Depository (MageMCU)
//
// Teacher's and Parent's copy...
// Dedicated to Parents and Teachers of the USA.
// Teachers Note: The _BEGIN_DEBUG is defined (see
//                DEBUG.h file), also MAIN _ANALOG_
//                is defined for debugging the
//                analog readings. For safety
//                reasons while debugging, the
//                motors are conditioned not to
//                operate. Be warned. the Student's
//                copy has no safety feature.
//
// Hardware
// MCU: Atmega328P (Microchip)
// * Board: Arduino Uno (Italy)
// Thumb Joystick: (China)
// Motor Driver: L298N (ST)
// * Motor Module: (China)
// 12V Geared Motors (ratio 1:75) (China)
// Note: motors: current peaks on average 1.14 amps
//       for static friction, and it levels down to
//       about 0.29 amps for dynamic friction. When
//       joystick with hands-off (center position),
//       is about 0.10 amps. The robot was sitting
//       on stand with no weight on wheels, otherwise,
//       the electrical currents would have increased.
// Battery 12V - APX1250 (12V 5.0Ah) Sealed Lead-Acid
// Note: battery: with five years design life (using
//       now for 7 years.) Requires special battery
//       charger from APEX.
//
// By Jesse Carpenter (Github as MageMCU)
//
// CHANGELOG
// August 30, 2021 - debug comments
// August 24, 2021 - editing
// June 19, 2021 - programming debug version
//
// Atmega328P Total Usgae
// RAM : [==] 15.3 % (used 314 bytes from 2048 bytes)
// Flash : [==] 21.2 % (used 6846 bytes from 32256 bytes)
//
// MIT LICENSE
//

#ifndef __UNO_TIMER_H__
#define __UNO_TIMER_H__

#include <Arduino.h>
#include "DEBUG.h"

// Using Adruino Uno
namespace uno
{
  class Timer
  {
  private:
    long _elapsedTime;
    long _lastMillisecond;

    void _debugConstructor(/* Debug Parameters */);
    void _allDebug(/* Debug Parameters */);
```

```
  public:
    Timer();
    void resetTimer();
    boolean isTimer(int incrementedTime);
  };

  Timer::Timer()
  {
    _elapsedTime = -1;
    _lastMillisecond = 0;

    _debugConstructor(/* Debug Arguments */);
  }

  void Timer::resetTimer()
  {
    _elapsedTime = -1;
  }

  boolean Timer::isTimer(int incrementedTime)
  {
    long currentTime = millis();
    if (_elapsedTime == -1)
    {
      _elapsedTime = currentTime + (long)incrementedTime;
    }
    else if (currentTime > _elapsedTime)
    {
      _elapsedTime = currentTime + (long)incrementedTime;
      return true;
    }
    return false;
  }

  void Timer::_debugConstructor(/* Debug Parameters */)
  {
#ifdef _dT_
    // Used simply whether or not
    // Constructor Instantiated Timer CLASS
    Serial.println("Timer");
    // Write Serial Prints Here...
#endif
  }

  void Timer::_allDebug(/* Debug Parameters */)
  {
#ifdef _dT_
    // Write Serial Prints Here...
#endif
  }
}
#endif
```

```
//
// Carpenter Software
// File: DEBUG DEBUG.h (debug version)
// Folder: joystick_L298N_debug
// Purpose: Github Depository (MageMCU)
//
// Teacher's and Parent's copy...
// Dedicated to Parents and Teachers of the USA.
// Teachers Note: The _BEGIN_DEBUG is defined (see
//                DEBUG.h file), also MAIN _ANALOG_
//                is defined for debugging the
//                analog readings. For safety
//                reasons while debugging, the
//                motors are conditioned not to
//                operate. Be warned. the Student's
//                copy has no safety feature.
//
// Hardware
// MCU: Atmega328P (Microchip)
// * Board: Arduino Uno (Italy)
// Thumb Joystick: (China)
// Motor Driver: L298N (ST)
// * Motor Module: (China)
// 12V Geared Motors (ratio 1:75) (China)
// Note: motors: current peaks on average 1.14 amps
//       for static friction, and it levels down to
//       about 0.29 amps for dynamic friction. When
//       joystick with hands-off (center position),
//       is about 0.10 amps. The robot was sitting
//       on stand with no weight on wheels, otherwise,
//       the electrical currents would have increased.
// Battery 12V - APX1250 (12V 5.0Ah) Sealed Lead-Acid
// Note: battery: with five years design life (using
//       now for 7 years.) Requires special battery
//       charger from APEX.
//
// By Jesse Carpenter (Github as MageMCU)
//
// CHANGELOG
// August 30, 2021 - debug comments
// August 24, 2021 - editing
// June 19, 2021 - programming debug version
//
// Atmega328P Total Usgae
// RAM : [==] 15.3 % (used 314 bytes from 2048 bytes)
// Flash : [==] 21.2 % (used 6846 bytes from 32256 bytes)
//
// MIT LICENSE
//
#ifndef __UNO_DEBUG_H__
#define __UNO_DEBUG_H__

// MAIN AND ALL DEPENDENCIES
// FOR DEBUGGING...
////////////////////////////////
// BEGIN DEBUG HEADERS AND MAIN //
//          UN-COMMENT          //
////////////////////////////////
#define _BEGIN_DEBUG
////////////////////////////////
//          DO NOT TOUCH         //
////////////////////////////////
#ifdef _BEGIN_DEBUG
// DEBUG MAIN
#define _dM_
#ifdef _dM_

////////////////////////////////
```

```
//             MAIN             //
//////////////////////////////////

// (1) MAIN ANALOG DEBUG
#define _ANALOG_

// (4) MAIN JOYSTICK DEBUG
// #define _JOYSTICK_

// MAIN L298N DEBUG
// #define _L298N_

//////////////////////////////////
//            HEADERS           //
//////////////////////////////////

// (2) DEBUG JOYSTICK
// #define _dJ_

// (3) DEBUG L298N
// #define _dL_

// DEBUG TIMER
// #define _dT_

// DEBUG BUTTON
// #define _dB_

//////////////////////////////////
//           JOYSTICK           //
//////////////////////////////////
#ifdef _dJ_

// _alignRawData()
#define _ALIGN_

// _shiftFromZeroToMidpoint()
#define _SHIFT_

// _adjustToDynamicFriction()
#define _DYNAMIC_

// _integerToDouble()
#define _DOUBLE_

// _joystAlgorithm()
#define _NORMAL_

//IF-DEF-HEADER-dJ//////////////////////
#endif

//////////////////////////////////
//           L298N              //
//////////////////////////////////
#ifdef _dL_

// Write definitions here
// The code is small...
// #define _SOMETHING_ you want to DEBUG

//IF-DEF-HEADER-dL//////////////////////
#endif
//IF-DEF-MAIN//////////////////////
#endif
// IF-DEF-ALL //////////////////////
#endif
// DEFINE-ALL    ///////////////////
#endif
```