Peristaltic Pump System

Mario Martinez, Computer Engineering

Texas Tech University

May 2019

Abstract

This paper describes a peristaltic pump system for microfluidic device use. All components are described including operation, schematic, code and results.

Table of Contents

|  |  |  |
| --- | --- | --- |
| Introduction | …………………………………………….... | 5 |
| Overall System | …………………………………………….... | 5 |
| Power (12V DC Power Supply) | …………………………………………….... | 6 |
| Motor controller (L298N) | …………………………………………….... | 6 |
| Peristaltic Pump | …………………………………………….... | 8 |
| User interface (LCD & controls) | …………………………………………….... | 9 |
| Microcontroller (ATMEGA328P) | …………………………………………….... | 12 |
| Code | …………………………………………….... | 12 |
| Design | …………………………………………….... | 16 |
| Data and results | …………………………………………….... | 16 |
| Conclusion | …………………………………………….... | 16 |
| References | …………………………………………….... | 17 |
| Appendices | …………………………………………….... | 19 |

List of Figures

|  |  |  |  |
| --- | --- | --- | --- |
| Figure 1 | Block Diagram of Overall System | ………………………………. | 5 |
| Figure 2 | L298N Motor Driver and Circuit | ……………………………… | 6 |
| Figure 3 | Flowchart of power provided | ……………………………… | 7 |
| Figure 4 | Adafruit peristaltic Pump | ……………………………… | 8 |
| Figure 5 | Multi Pump Block Diagram | ……………………………… | 8 |
| Figure 6 | Multi Pump Circuit | ……………………………… | 9 |
| Figure 7 | LCD with I2C module | ……………………………… | 9 |
| Figure 8 | 10K Potentiometer | ……………………………… | 9 |
| Figure 9 | Toggle Switch | ……………………………… | 9 |
| Figure 10 | Flowchart of User Interface | ……………………………… | 10 |
| Figure 11 | ATMEGA238P Pinout | ……………………………… | 11 |
| Figure 12 | Schematic for ATMEGA independent of Arduino Board | ……………………………… | 11 |
| Figure 13 | Code Snippet of Speed Control Function | ……………………………… | 12 |
| Figure 14 | Code Snippet of Motor States | ……………………………… | 13 |
| Figure 15 | Code Snippet of Timer Function | ……………………………… | 13 |
| Figure 16 | 3D CAD model of system | ……………………………… | 15 |
| Figure 17 | 3D CAD model of Pump Mount and Reservoir mount | ……………………………… | 15 |

List of Tables

|  |  |  |
| --- | --- | --- |
| Table 1: | Motor States | 12 |
| Tabel 2: | Table of Measured Flow rates for 4 pumps | 16 |

**Introduction**

The main objective for the system is to draw liquid from multiple reservoirs into multiple microfluidic devices at a variable speed that the user can control. The pumps will stay on for 90 seconds after which, the motors will turn off. The flow and voltage of the pumps are also displayed for the user.

The system is powered on by a DC Power supply to receive a constant voltage. The motor controller is used to power and control the speed of the pumps (motors) as well as provide power to the PCB. an Arduino microcontroller is used to operate the system as it is easily programed and supported using the arduino IDE.

**Overall System**

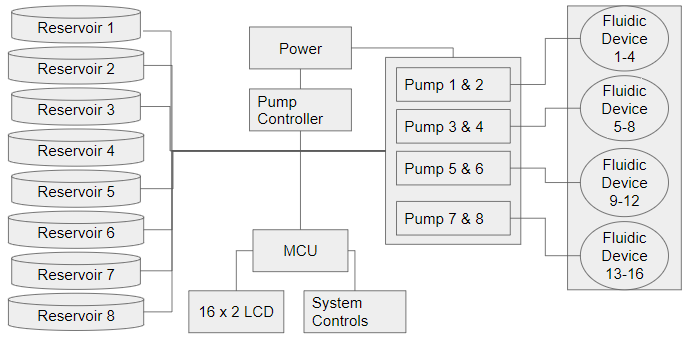


Figure 1: Block Diagram of Overall System [9]

The system works in two states, on or off, controlled by the user with a toggle switch. When on, the system draws liquid from the reservoirs for 90 seconds before coming to a stop. The system does not enter its off state unless the user flips the toggle switch to its off state. In this state, the system timer resets and the process can be repeated. In both states, the system will display both the voltage, controlled by the user via potentiometer, and flow rate. Regardless of the state, the pumps all flow in the same direction and direction cannot be changed.

**Power (12V DC Power Supply)**

Power is supplied to the system with a 12V, 5A, 60W power supply. Power is routed onto the PCB and onto the motor controller. The controller comes with an internal 5V output and steps down the voltage for the remaining components. The pump motors are connected in parallel to one another and connected to OUT1/OUT2 (Output A) OUT3/OUT4 (Output B) of the motor controller which provide a maximum of 10V for the pump motors.

**Motor controller (L298N)**

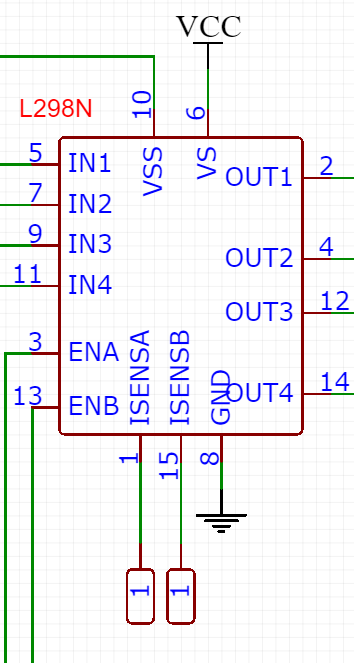
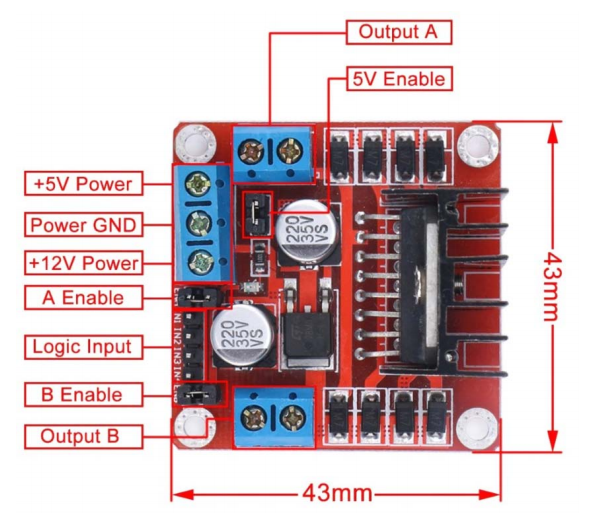


Figure 2: L298N Motor Driver and Circuit [11] [10]

The L298N both provides power to the PCB/motors and controls the speed of the motor pumps. The L298N can take between 3.2V ~ 40V as input with a peak current of 2A.

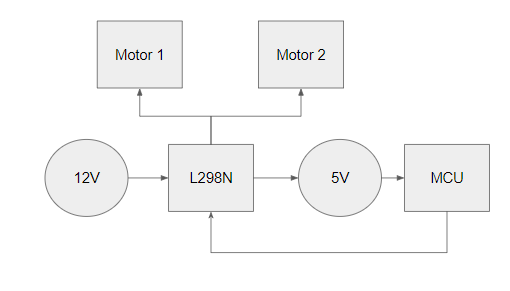


Figure 3: Flowchart of power provided [9]

To provide power, the motor takes no more than 12V input, which provides power to the motors, and output 5V for use (VCC). To control the speed, the L298N provides a PWM signal for the motors which can be controlled with a potentiometer by varying the output for both enable pins (ENA, ENB). The digital logic pins (IN1, IN2, IN3, IN4) control the direction of the motors and are always set to:

IN1/IN3 = HIGH

IN2/IN4 = LOW

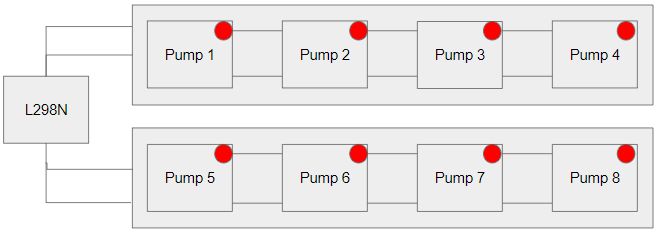
unless turning off the motors in which case all pins are set to LOW.

**Peristaltic Pump**



Figure 4: Adafruit peristaltic Pump [2]

The peristaltic pumps work by rotating a roller that compresses and relaxes a flexible tube, pushing liquid or gas from one end of the tube to the other. To rotate the roller, a 12V, 200~300 mA, 5000 RPM DC motor is used which can output a max flow rate of 100 mL/min.

Figure 5: Multi Pump Block Diagram [9]

For this system, two sets of four pumps are connected in parallel to one another with each set acting as one individual pump. Underneath each pump, a marker indicated the motorpin. All marked motor pins should be connected in series to one another.

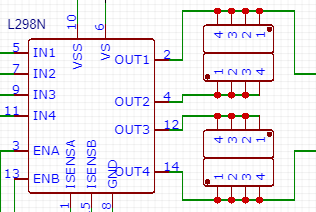


Figure 6: Multi Pump Circuit [10]

The pumps are wired into OUT1/OUT2 (Output A) OUT3/OUT4 (Output B) of the motor controller. Because the maximum voltage that the controller can provide for the pumps is 10V, the pumps will have a maximum operating of 10V in this system.

**User interface (LCD & controls)**

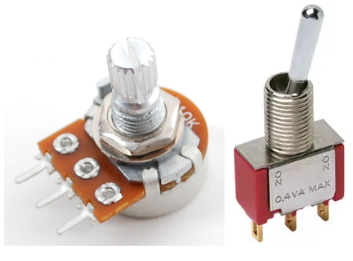
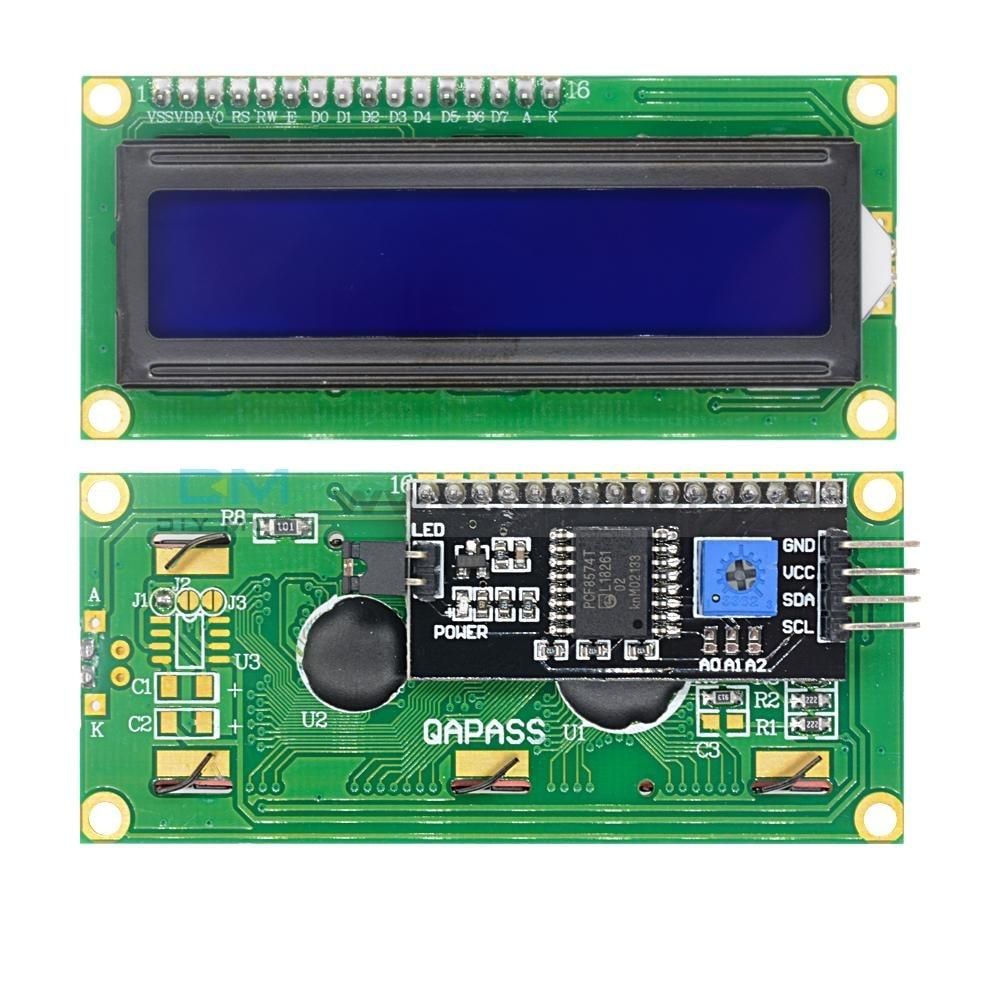


Figure 7, 8 & 9: LCD with I2C module (Left) & 10K potentiometer and Toggle Switch (Right)

The user interface consists of a 16 pin, 16x2 LCD for displaying the voltage and flow rate as well as a 10K potentiometer [4] to control the flow rate and a 5V, 2A toggle switch [5] to control the state of the system. The LCD incorporates an I2C module [3] to reduce the pin count from 16 pins to 4 pins.

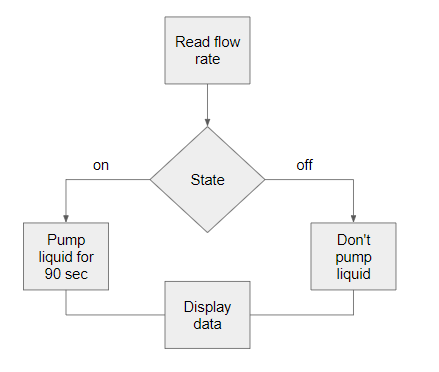


Figure 10: Flowchart of User Interface [9]

The system has two states, on and off. When on, the system operate at the indicated flow rate for 90 seconds, after which the system comes to a stop. When off, the system will no longer pump any liquid. In both states the LCD will output the flowrate and voltage of the pump and motors. Even when the system is off, the LCD will continue to display the flow rate and voltage controlled by the potentiometer.

This system has the Signal and Clock pins of the I2C/LCD connected to the Analog 5 and Analog 4 pins of the microcontroller respectively, the POT signal connected to Analog 0 of the microcontroller, and the toggle switch connected to Digital 2 pin.

**Microcontroller (ATMEGA328P)**

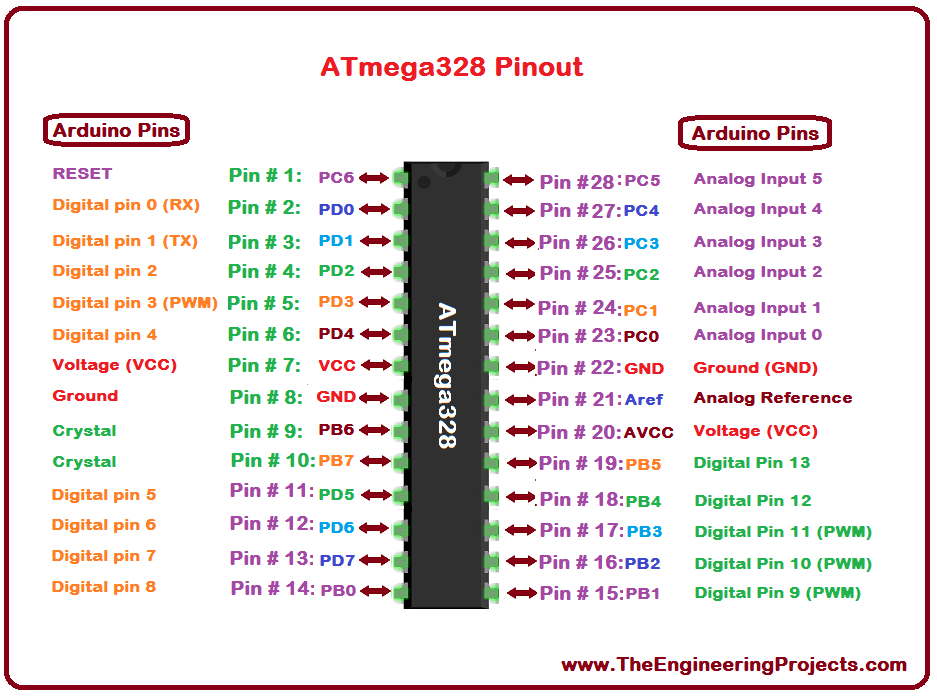


Figure 11: ATMEGA238P Pinout [12]

The entire system is controlled with the ATMEGA328P microcontroller which is powered by a 5V voltage source. To operate the microcontroller, two grounded 22pF capacitors are connected in series with an external 16 MHz crystal which is connected to pins 9 and 10. AREF, Pin 21, must also be connected to the 5V source, however due to PCB constraints, AREF is externally connected via a wire to the 5V source.

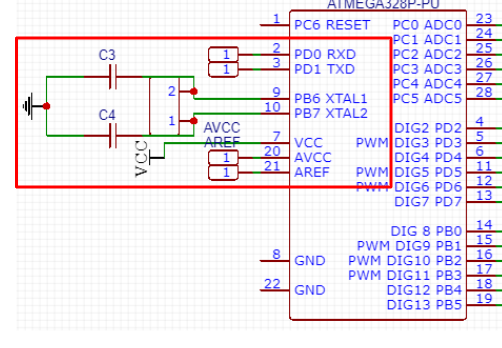


Figure 12: Schematic for ATMEGA independent of Arduino Board [10]

**Code**

The code is split into several sections. Speed control, System States, Timing, Calculations, and Display. Similar to the user interface, the code works by first reading the potentiometer and then checking the state of the system. When the system is off, all motors are off and the state of the system is reset. However when the system is on, a timer checks if 90 seconds have passed. After 90 seconds, the motors are turned off but the system is considered to be on.

To control the speed, the input of the potentiometer is read and set to a variable (*potValue*). This variable is mapped from 0 to 255 and set to another variable (*pwmOutput*). This variable is then written to both enable pins (enA, enB).

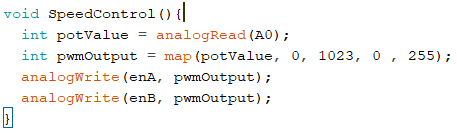


Figure 13: Code Snippet of Speed Control Function [10]

To control the state of the motors, the following table corresponds to which L298N pins are set to HIGH/LOW to turn to motors on or off.

|  |  |  |
| --- | --- | --- |
|  | on | off |
| in1 | HIGH | LOW |
| in2 | LOW | LOW |
| in3 | HIGH | LOW |
| in4 | LOW | LOW |

Table 1: Motor States

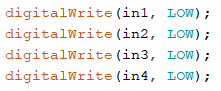
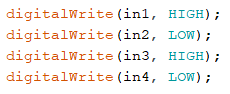


Figure 14: Code Snippet of Motor States: On - left / Off - Right [10]

The system state is determined by reading the digital signal provided by the toggle switch pin and placed in the variable *state*. If a digital HIGH is read, a timer function runs while a check is made to see if the variable *counter* is equal to 90 or not. If true, all motors are turned off, otherwise all motors are turned on. If a digital LOW is read, all the motors are turned off and the timer is reset by setting the variable *counter* to 0.

The timer function works by keeping track of the amount of milliseconds the program has run, checking to see if 90 seconds have passed, and resetting the timer. To keep track of the amount of milliseconds that have passed, the function millis is used and set to a variable called *currentMillis*. This variable is subtracted by another variable, *state1*, set to 0 by default, and checked if the result is less than 90 milliseconds. If true, state1 is set to the currentMillis which resets the timer, and the variable *counter* is set to 90. This timer reset is also done when the system enters its off state.

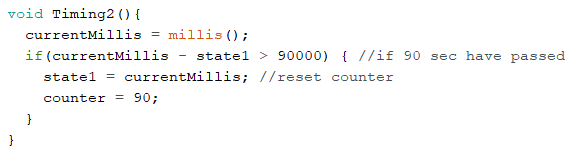


Figure 15: Code Snippet of Timer Function [10]

The program utilises the Liquid Crystal I2C library made by [fdebrabander](https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library/blob/master/LiquidCrystal_I2C.h) [8]. To display, the program sets the text “Flow: ” and the variable *flow\_rate* to the top of the LCD and the text “Volt: ” and the variable *input\_voltage* to the bottom of the LCD. the LCD will display a flow rate of 0 if the voltage detected is less than 5V.

The voltage is calculated by measuring the maximum voltage the motor controller can output for the motors (maxV), while all motors run at the highest setting provided by the potentiometer, multiplied by the potentiometer, and divided by 255 since that is the maximum the POT will give.

*input\_voltage = (maxV \* pwmOutput) / 255*

Formula 1: Formula for mapping POT values to Motor Voltage

This formula maps the POT value to the voltage that the motor driver provides for the pumps. This value is rounded to the nearest tenth for stability.

The flow rate is calculated by manually measuring how much liquid the pumps output at a specified voltage. This is done four times at different voltages. Using a polynomial regression tool (4th degree) [7], a formula is generated to fit the curve of the data taken.

Formula 2: Formula for Calculating Flow rate

The result of this value is placed in the variable *flow\_rate* and rounded to the nearest tenth for stability.

**Design**

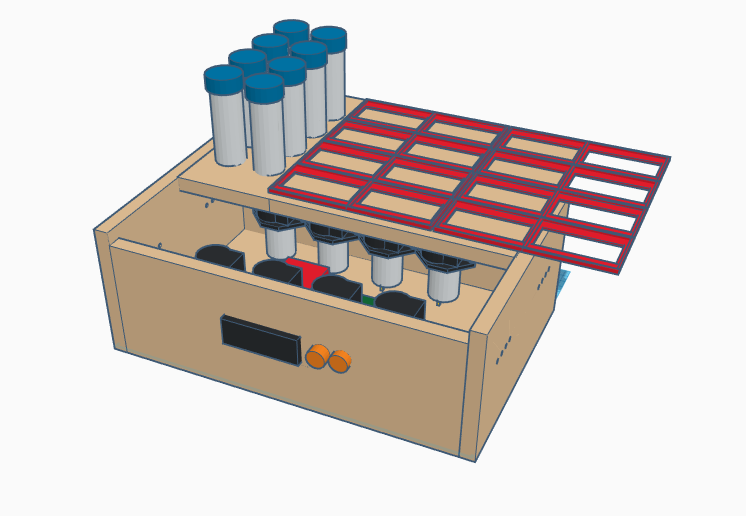


Figure 16: 3D CAD model of system

The design of the system consists of an 8x10x5 (LxWxH) enclosure with a removable top. Both the test tubes and microfluidic devices sit on top. The pumps, PCB are contained inside while the LCD and controls are on the face of the system container. The container material consists of Hardwood with multiple holes drilled for the pump tubes, LCD and controls, and reservoirs.

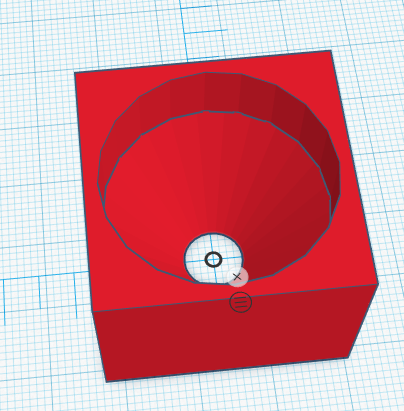
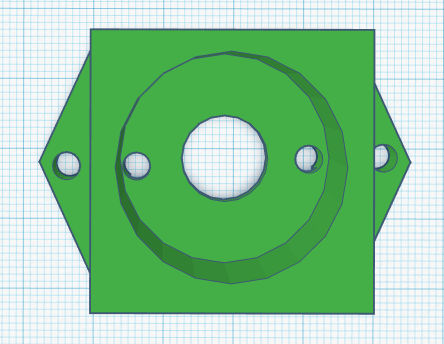


Figure 17: 3D CAD model of Pump Mount and Reservoir mount [9]

3D printed mounts are installed to mount each of the pumps, the PCB, microfluidic devices, and reservoirs. All mounts are super glued onto the wood.

**Data and results**

As stated previously, flow rate is calculated by manually measuring how much liquid the pumps can output per minute. However not all pumps will flow at the same flow rate. Some will flow at only half the expected flow rate while other will preform between +/- 2.5 ml/min.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pump 1   |  |  | | --- | --- | | Voltage | Flow | | 9V | 27.5 | | 8V | 22.5 | | 7V | 21.25 | | 6V | 16.25 | | Pump 2   |  |  | | --- | --- | | Voltage | Flow | | 9V | 43.75 | | 8V | 20.75 | | 7V | 17 | | 6V | 12.5 | | Pump 3   |  |  | | --- | --- | | Voltage | Flow | | 9V | 40 | | 8V | 22.5 | | 7V | 17 | | 6V | 12.5 | | Pump 4   |  |  | | --- | --- | | Voltage | Flow | | 9V | 25 | | 8V | 20 | | 7V | 21.25 | | 6V | 7.5 | |

Table 2: Table of Measured Flow rates for 4 pumps

**Conclusion**

Although the system is able to pump liquid at a variable speed, the flow rate displayed may be incorrect. The L298N motor controller cannot consistently control more than two motors at a time. Either multiple L298N drivers should be used or a single motor driver that specifies multiple motor use should be used.

## References

[1]*Sparkfun.com*, 2019. [Online]. Available: https://www.sparkfun.com/datasheets/Robotics/L298\_H\_Bridge.pdf. [Accessed: 10- May- 2019]

[2]A. Industries, "Peristaltic Liquid Pump with Silicone Tubing - 12V DC Power", *Adafruit.com*, 2019. [Online]. Available: https://www.adafruit.com/product/1150. [Accessed: 10- May- 2019]

[3]*Opencircuit.nl*, 2019. [Online]. Available: https://opencircuit.nl/ProductInfo/1000061/I2C-LCD-interface.pdf. [Accessed: 10- May- 2019]

[4]"10K Potentiometer Pinout, Working & Datasheet Explained", *Components101.com*, 2019. [Online]. Available: https://components101.com/potentiometer. [Accessed: 10- May- 2019]

[5]*Spec\_sheets.e-switch.com*, 2019. [Online]. Available: http://spec\_sheets.e-switch.com/specs/ST110007.pdf. [Accessed: 10- May- 2019]

[6]*Sparkfun.com*, 2019. [Online]. Available: https://www.sparkfun.com/datasheets/Components/SMD/ATMega328.pdf. [Accessed: 10- May- 2019]

[7]"Regression Tools - Online Polynomial Regression", *Xuru.org*, 2019. [Online]. Available: http://www.xuru.org/rt/PR.asp. [Accessed: 10- May- 2019]

[8]"fdebrabander/Arduino-LiquidCrystal-I2C-library", *GitHub*, 2019. [Online]. Available: https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library. [Accessed: 10- May- 2019]

[9]M. Martinez, "Final Presentation", *Google Docs*, 2019. [Online]. Available: https://docs.google.com/presentation/d/e/2PACX-1vTdenOasp1ehTpx3x7591r-3ISicg3iCcS\_-SnG85Digey4IZQUAb9HP8wSzczyFjpC5AgiG7OrVUxs/pub?start=true&loop=true&delayms=3000. [Accessed: 10- May- 2019]

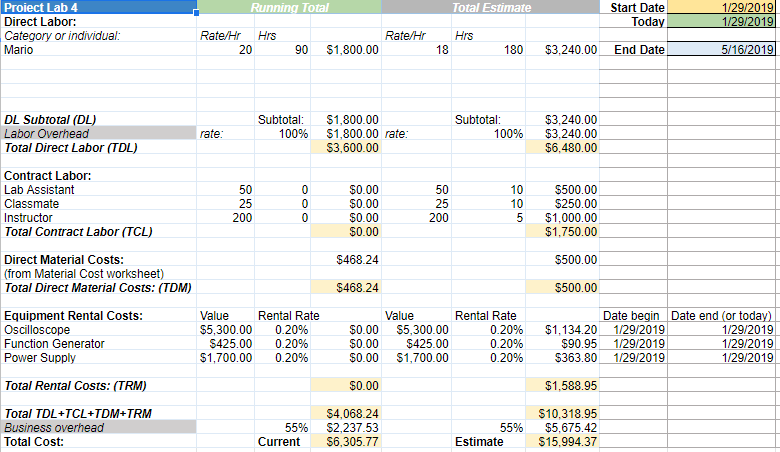
[10]M. Martinez, "mar87649/Peristaltic-Pump", *GitHub*, 2019. [Online]. Available: https://github.com/mar87649/Peristaltic-Pump. [Accessed: 10- May- 2019]

[11]*Handsontec.com*, 2019. [Online]. Available: http://www.handsontec.com/dataspecs/L298N%20Motor%20Driver.pdf. [Accessed: 10- May- 2019]

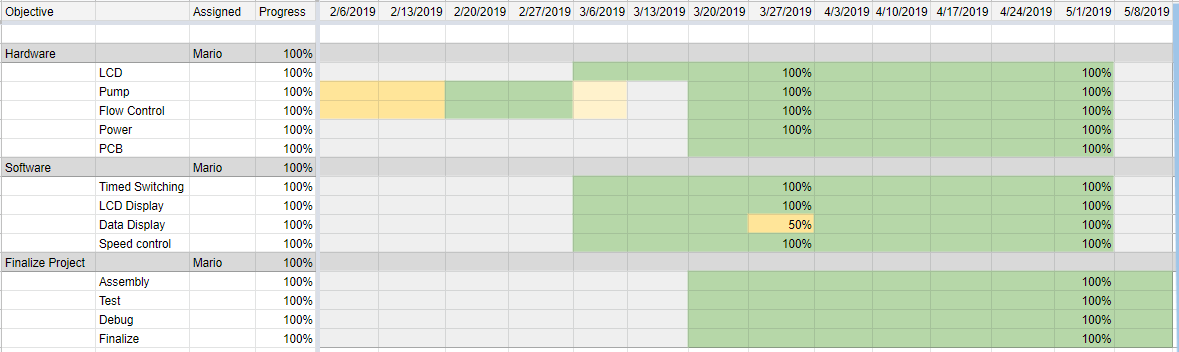
[12]S. Nasir, "Introduction to ATmega328 - The Engineering Projects", *The Engineering Projects*, 2019. [Online]. Available: https://www.theengineeringprojects.com/2017/08/introduction-to-atmega328.html. [Accessed: 10- May- 2019]

Appendices

Appendix A: Budget



# Appendix B: Gantt Chart



# Appendix C: Code

#include <LiquidCrystal.h>

#include <MegunoLink.h>

#include <CommandHandler.h>

#include <TCPCommandHandler.h>

#include <ArduinoTimer.h>

#include <CircularBuffer.h>

#include <EEPROMStore.h>

#include <Filter.h>

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

LiquidCrystal\_I2C lcd(0x27,20,4);

ExponentialFilter<float> ADCFilter(100, 0);

#define enA 11

#define in1 12

#define in2 13

#define enB 10

#define in3 9

#define in4 8

//Polynomial Regression variables

double a = - 98.28571429;

double b = 28.28571429;

double c = - 1.428571429;

//Display variables

double input\_voltage = 0.0;

double flow\_rate = 0.0;

//motor control variables

int state=0;

const int controlPin = 2;

//Timing Variables

unsigned long startMillis;

unsigned long currentMillis = 0;

unsigned long previousMillis = 0;

long int1 = 5\*1000;

long int2 = 7\*1000;

long interval = 95\*1000;

int motorstate = 1;

int motorState1;

int motorState2;

int counter = 0;

long state1 = 0;

long state2 = 0;

//Conversion & other variables

float temp=0.0;

float r1=10000.0;

float r2=1100.0;

int analog\_value;

float analogM1=0;

float analogM2=0;

float smooth\_input;

const int numReadings = 10;

int readings[numReadings]; // the readings from the analog input

int readIndex = 0; // the index of the current reading

int total = 0; // the running total

int average = 0; // the average

/////////////////////////////////////////////////////////////////////////

void setup()

{

//Serial Comm setup

Serial.begin(9600);

//Pin setup

pinMode(controlPin, INPUT); //toggle switch

pinMode(enA, OUTPUT); //motor pins

pinMode(in1, OUTPUT);

pinMode(in2, OUTPUT);

pinMode(enB, OUTPUT); //motor pins

pinMode(in3, OUTPUT);

pinMode(in4, OUTPUT);

// Set initial motor rotation direction

int motorState1 = LOW;

int motorState2 = LOW;

digitalWrite(in1, HIGH);

digitalWrite(in2, LOW);

digitalWrite(in3, HIGH);

digitalWrite(in4, LOW);

//Filter setup

for (int thisReading = 0; thisReading < numReadings; thisReading++) {readings[thisReading] = 0;}

lcd.init(); // initialize the lcd

lcd.init();

lcd.backlight();

}

//////////////////////////////////////////////////////////////////////////////////

void loop()

{

//Speed Control

SpeedControl();

//conversion for filter

int potValue = analogRead(A0); // Read potentiometer value

int pwmOutput = map(potValue, 0, 1023, 0 , 255); // Map the potentiometer value from 0 to 255

analogM1 = analogRead(A1);

analogM2 = analogRead(A2);

analog\_value = LargestOf(analogM1,analogM2);

smooth\_input = smooth(analog\_value);

ADCFilter.Filter(smooth\_input);

temp = (ADCFilter.Current() \* 5.0) / 1024.0;

//Serial monitor info

Serial.print("v= ");

Serial.println(input\_voltage);

Serial.print("f= ");

Serial.println(flow\_rate);

TimePlot Plot;

Plot.SendData("Raw", smooth\_input);

Plot.SendData("Filtered", ADCFilter.Current());

//motor states

state = digitalRead(controlPin);

if(state==1){

Timing2(); //turn off motors every 90 sec

if (counter == 90){

digitalWrite(in1, LOW);

digitalWrite(in2, LOW);

digitalWrite(in3, LOW);

digitalWrite(in4, LOW);

} else{

digitalWrite(in1, HIGH);

digitalWrite(in2, LOW);

digitalWrite(in3, HIGH);

digitalWrite(in4, LOW);

}

}

else{

digitalWrite(in1, LOW);

digitalWrite(in2, LOW);

digitalWrite(in3, LOW);

digitalWrite(in4, LOW);

counter = 0;

currentMillis = millis();

state1 = currentMillis;

}

//Timing

//Timing();

//Display

input\_voltage = calcVoltage\_pot(pwmOutput);

flow\_rate = calcFlow\_pot(input\_voltage);

Display2();

delay(50);

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

float calcVoltage\_pot(float pwmOutput){

input\_voltage = RoundTenth(pwmOutput\*0.03529);

return input\_voltage;

}

float calcFlow\_pot(float input\_voltage){

if (input\_voltage < 5){

flow\_rate = 0;

}else{

flow\_rate = -.1875000009\*pow(input\_voltage, 4)

+ 5.995370399\*pow(input\_voltage, 3)

- 71.77083365\*pow(input\_voltage, 2)

+ 386.236774\*input\_voltage

- 763.1507962;

}

flow\_rate = RoundTenth(flow\_rate);

return flow\_rate;

}

float RoundTenth(float a){

a = (a\*10) + .5;

a = float(int(a));

a = a/10;

return a;

}

float LargestOf(int a, int b){

if (a>b){

return a;

}else{

return b;

}

}

void SpeedControl(){

int potValue = analogRead(A0); // Read potentiometer value

int pwmOutput = map(potValue, 0, 1023, 0 , 255); // Map the potentiometer value from 0 to 255

analogWrite(enA, pwmOutput); // Send PWM signal to L298N Enable pin

analogWrite(enB, pwmOutput); // Send PWM signal to L298N Enable pin

}

void Timing(){

currentMillis = millis();

if(currentMillis - state1 > int1) { //if 90 sec have passed

state1 = currentMillis;

State1(); //turn motor A off and motor B on

}

if(currentMillis - state2 > int2) { //if 2 sec after state

state1 = state1 + 2000;

state2 = currentMillis;

State2(); //turn motor B off and motor A on

}

}

void Timing2(){

currentMillis = millis();

if(currentMillis - state1 > 90000) { //if 90 sec have passed

state1 = currentMillis; //reset counter

counter = 90;

}

}

void Display(){

lcd.setCursor(0, 0);

lcd.print("Flow Rate: ");

lcd.print(flow\_rate);

//lcd.print("ml/min ");

lcd.setCursor(0, 1);

lcd.print("Voltage: ");

lcd.print(input\_voltage);

}

void Display2(){

lcd.setCursor(0,0);

lcd.print("Flow: ");

lcd.print(flow\_rate);

lcd.print(" ml/s");

lcd.setCursor(0,1);

lcd.print("Volt: ");

lcd.print(input\_voltage);

lcd.print(" V");

}

void State1(){

motorState1 = LOW; // Turn off motor 1

digitalWrite(in1, LOW);

digitalWrite(in2, LOW);

motorState2 = HIGH; // Turn on motor 2

digitalWrite(in3, HIGH);

digitalWrite(in4, LOW);

}

void State2(){

motorState1 = HIGH; // Turn on motor 1

digitalWrite(in1, HIGH);

digitalWrite(in2, LOW);

motorState2 = LOW; // Turn off motor 2

digitalWrite(in3, LOW);

digitalWrite(in4, LOW);

}

float smooth(float data) {

// subtract the last reading:

total = total - readings[readIndex];

// read from the sensor:

readings[readIndex] = data;

// add the reading to the total:

total = total + readings[readIndex];

// advance to the next position in the array:

readIndex = readIndex + 1;

// if we're at the end of the array...

if (readIndex >= numReadings) {

// ...wrap around to the beginning:

readIndex = 0;

}

// calculate the average:

average = total / numReadings;

// send it to the computer as ASCII digits

return average;

}

# 