CG1111 Engineering Principles and Practice I

Principles of DC Motors

(Week 8, Studio 1)

Time	Duration (mins)	Activity
0:00	20	Briefing on activities
0:20	20	Activity #1: Understanding the relationship between rotational speed and motor's supply voltage for the case of a constant load
0:40	30	Activity #2: Characterizing a DC motor
1:10	80	Activity #3: Speed control of DC motor using PWM
2:30	5	Final discussions and wrap-up

Introduction:

- Motors come in different sizes depending on the mechanical power required - ranging from small motors such as the ones used in electric toothbrushes, to huge motors used in heavy industry. Motors can also be classified according to the type of electrical supply used (AC/ DC) or according to their construction (e.g., induction motor, synchronous motor, stepper motor, etc.). In this module, we will focus on permanent magnet DC (PMDC) motors to learn about the basic principles of motors.

- In the e-lecture, you have learnt that $\omega = \frac{V_m}{K_e} - \frac{R_m I_m}{K_e}$. In Activity #1, you will try to understand what this equation tells us about the relationship between a PMDC motor's rotational speed and its supply voltage when the load is kept constant.

- In Activity #2, you will try to characterize a PMDC motor (i.e., estimating its parameters) by taking measurements of its rotational speed and input current as the load is varied.

- From the knowledge gained in Activity #1, you would have realized that a motor's speed can be varied by changing the motor's terminal voltage. But what about in many practical systems where the source voltages are fixed (e.g., those that run on batteries)? In such cases, the ***average*** terminal voltage applied to the motor can be varied using a technique known as ***Pulse Width Modulation (PWM)***. In this approach, the motor's terminals are connected to the voltage source via a set of electronic switches that can be turned ON and OFF. If the switch is alternately turned ON for T_{on} seconds and turned OFF for T_{off} seconds, then the following ratio is called the **duty cycle** of the PWM:

$$\text{Duty Cycle} = \frac{T_{\text{on}}}{T_{\text{on}} + T_{\text{off}}} = \frac{T_{\text{on}}}{T_{\text{p}}}$$

The average DC voltage applied to the motor can be varied by changing the duty cycle of the PWM as follows:

$$\text{Average Terminal Voltage} = \text{Duty Cyle} \times \text{Supply Voltage}$$

In Activity #3, you will be experimenting with PWM speed control of a DC motor.

Objectives:

- To understand the behaviour of PMDC motors under different operating conditions.
- To understand the relevant parameters of a PMDC motor, and how to estimate them through experiments (i.e., characterizing the motor).
- To understand how to perform speed control of a PMDC motor using PWM.

Materials:

- Breadboard and connecting wires
- USB breakout cable
- BitScope
- Motor + wheel set
- Arduino Uno + USB cable
- Dual H-Bridge motor driver IC L293D chip

Note:

- For Activity #1 and #2, we will be using an online motor simulator software to obtain the experimental data. The hardware provided will only be used in Activity #3.

Activity #1: Understanding the relationship between rotational speed and motor's supply voltage for the case of a constant load (20 mins)

In your childhood days, you might have noticed that your battery-operated toy cars ran faster when fresh new batteries were used. Why is that so? How does the battery's voltage affect the motor's speed if everything else were kept the same? You will find that out in this studio activity.

In the experiment, we shall keep a constant load on the motor, while varying the motor's supply voltage as we observe its effect on the motor's speed. For the case of a toy car, keeping the load constant would mean running the toy car on a flat ground, or on a slope with a constant gradient.

You will be using an online motor simulator software that we have developed for CG1111 for this activity: https://khenus.github.io/DCMotorSim/

Procedure:

1. The motor simulator has 50 different sets of motor parameters. You need to select the parameter set number according to the following algorithm. Note down your parameter set number in your Learning Journal.

$$\text{Parameter set no.} = (\text{“Last two numerical digits of your student number”} \bmod 50) + 1$$

2. Next, set the motor's terminal voltage V_m to 7.0 V. Then adjust the Torque Load on the slider bar to roughly about 1/3 of the slider's range. You will **keep the torque load fixed** at this setting for the rest of Activity #1.

3. Click the "Submit" button to obtain the rotational speed of the motor. Record the reading using the template given in Table I below.

Table I:

Motor Voltage V_m (V)	Rotational Speed N (RPM)
7	
8	
9	
10	
11	
12	

4. Repeat Steps 2 and 3 for the remaining voltage values of V_m as stated in Table I.

5. Using Microsoft Excel, plot the graph of Rotational Speed N (RPM) vs. Motor Voltage V_m (V), using N as the vertical axis. Which type of Trendline should you use? Why?
(Hint: Refer to equation $\omega = \frac{V_m}{K_e} - \frac{R_m I_m}{K_e}$)

6. Choose the option to display the equation on chart. Copy the graph into your Learning Journal, and note down the equation as well. Should the graph pass through the origin? Explain your answer.

Activity #2: Characterizing a DC motor (30 mins)

Recall from Slide 9 of the preparatory materials that the force exerted on a current-carrying conductor in the presence of a magnetic field is $F = B \times I \times l \times \sin\theta$. For a PMDC motor, because B, l and θ are all constant for a given motor as a result of its construction, the motor's torque will be proportional to the amount of current I_m passing through the motor's coils. This is also why the motor's torque is given by $T_{\text{shaft}} = K_t I_m$.

When a motor is spinning at a constant speed, the amount of torque produced by the motor is equal to the load torque (+ any frictional torque in a practical motor). Hence, we know that as the load torque increases, the motor's current will also increase.

In Slide 18 of the preparatory materials, we saw the circuit representation of a PMDC motor, and also an expression for the motor current I_m. In this part of the studio activity, we are interested in obtaining the relationship between I_m and the angular speed ω, for a fixed motor voltage V_m. (Note that this relationship is also representative of the torque vs. angular speed relationship since $T_{\text{shaft}} = K_t I_m$.) From the relationship obtained, we will derive several other motor parameters that characterize the given motor.

Procedure:

1. Set the motor's terminal voltage V_m in the online motor simulator to 12.0 V.

2. To study the relationship between the motor current I_m and its angular speed ω for the given terminal voltage of 12 V, we will obtain several sets of readings by adjusting the torque load only. To start with, adjust the Torque Load on the slider bar to roughly about 2/3 of the slider's range. Note down its motor current, rotational speed *N* (in RPM), and calculate its angular speed ω (in rad/s), using the template given in Table II below. You should use the formula feature in Microsoft Excel to help you speed up the calculations of ω.

Table II:

Torque Load setting	Motor Current I_m (A)	Rotational Speed *N* (RPM)	Angular Speed ω (rad/s)
~2/3 of slider's range			
~1/2 of slider's range			
~1/3 of slider's range			
~1/6 of slider's range			
~1/10 of slider's range			

3. Repeat this for four more torque load settings, using the suggested settings in the first column of Table II.

4. Using Microsoft Excel, plot the motor current I_m vs. angular speed ω graph (using I_m as the vertical axis). Choose the option to display the equation on chart. Right-click on the equation and pick "Format Trendline Label", so as to increase the number of decimal places to be displayed (say, 5). Copy the graph into your Learning Journal, and note down the equation as well.

5. Calculate the value of the stall current with the help of your graph's equation.

6. Calculate the no-load speed (in RPM), ignoring any frictional torque (i.e., assuming no current is needed to maintain the motor's speed).

   (Note: This is the theoretical maximum speed if we choose to ignore any frictional torque. In a practical motor's datasheet, the no-load speed accounts for the frictional torque that is present. A small amount of current is still needed to maintain the motor's speed even when there is no external mechanical load.)

7. Calculate the following motor parameters:
   (a) Motor resistance R_m
   (b) Back emf constant K_e (the unit is V-s/rad)
   (Note: Since we have plotted motor current I_m vs. angular speed ω, you should rely on the equation on Slide 18 of the preparatory materials.)

8. What is the value of the torque constant K_t?
   (Note: The unit is N-m/A; you can use mN-m/A if the value is small.)

Activity #3: Speed control of DC motor using PWM (80 mins)

As mentioned in the introduction, the ***average*** terminal voltage applied to a motor can be varied using ***PWM***. This average DC voltage can be varied by changing the duty cycle of the PWM as follows:

$$\text{Average Terminal Voltage} = \text{Duty Cyle} \times \text{Supply Voltage}$$

The following shows an example of how the PWM signal may look like for different duty cycles:

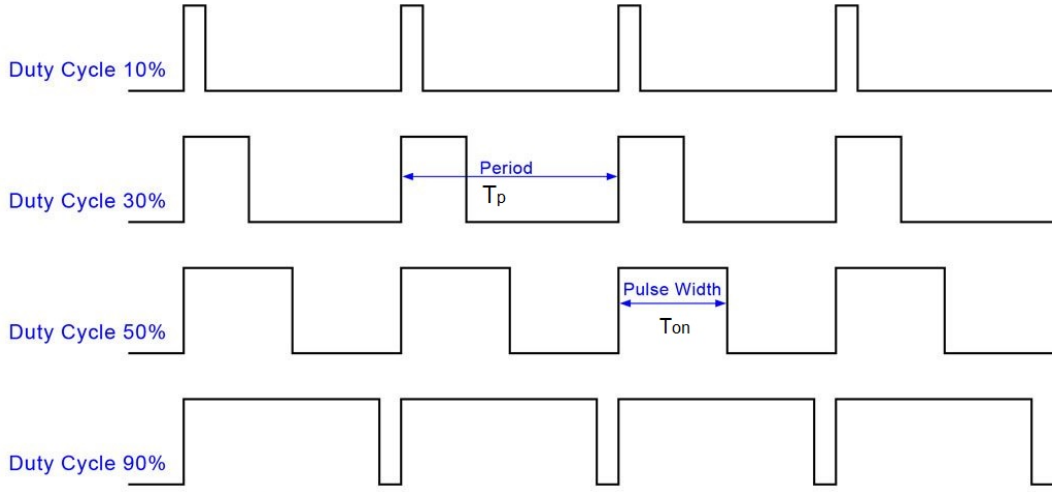

Figure 1: PWM signal for different duty cycles.

For a practical motor that needs to be operated bi-directionally (e.g., the wheels of a robotic vehicle such as the mBot), we must be able to reverse the direction of the current through the motor. The easiest way to do this is to use an H-Bridge circuit, as shown in Figure 2 below:

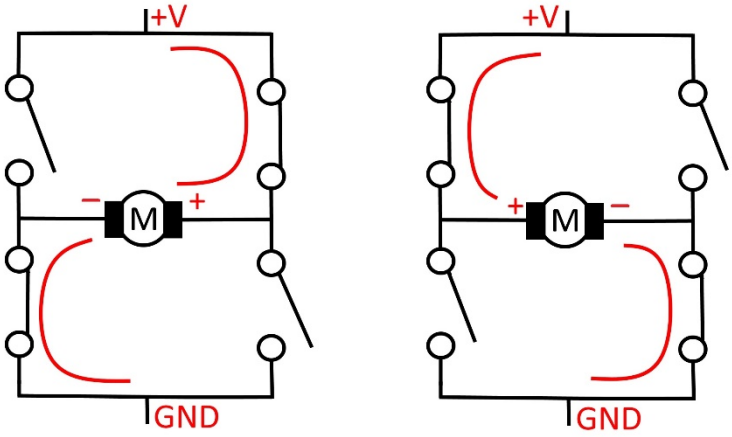

Figure 2: Bi-directional control of DC motor using H-Bridge.

In this activity, you will use the integrated circuit (IC) L293D (see Figure 3) chip, which is a dual H-Bridge motor controller. It includes **two H-bridges** and can control two motors. Although we will be using the Arduino Uno to generate the PWM signals needed for controlling the motor's speed, it is important to realize that the Arduino Uno only has an output current limit of around **20 mA**. This current is insufficient for driving a robotic vehicle's wheel motors directly. Hence, another important function of the motor controller chip is to provide the high current required by the motor using an external power supply, so that the current does not have to come directly from the microcontroller such as the Arduino Uno in our case.

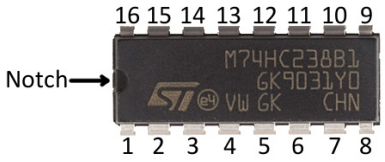

Figure 3: L293D IC - dual H-bridge motor controller, with pin numbering (take note of the "notch").

The functional descriptions for the different pins of L293D are illustrated in Figure 4. Note that the Enable pins, Control pins, and Motor connection pins are duplicated as the IC provides two H-bridge motor controllers. You will control only one motor in this studio and hence **only one set of the pins will be used** – those labelled in **blue**, and **black** (the common connections).

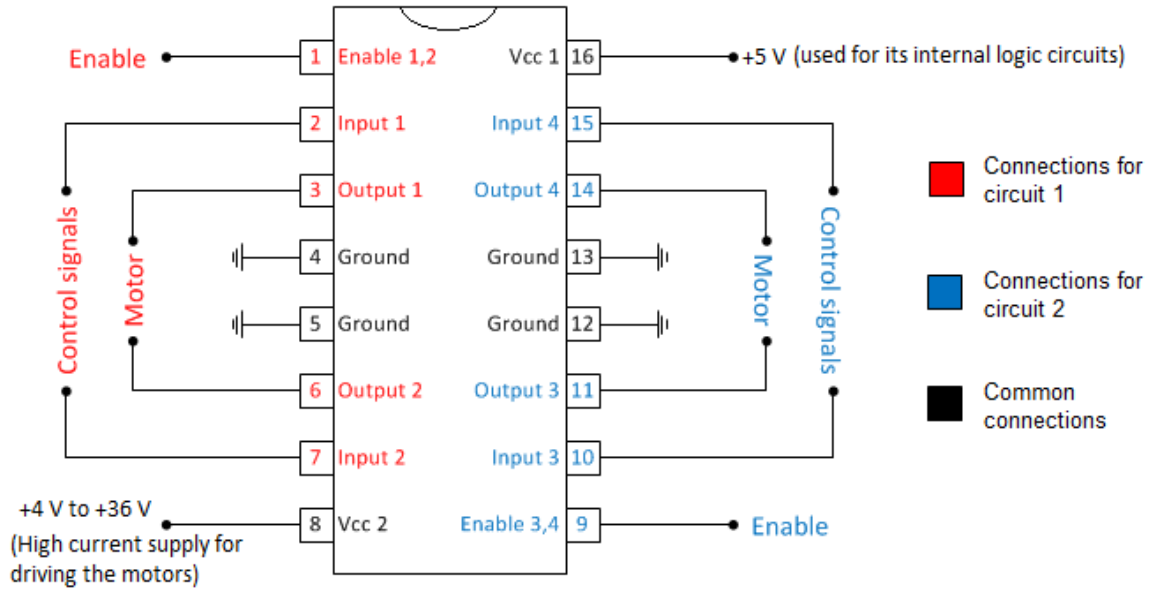

Figure 4: Functional descriptions of L293D chip.

Assuming that we are using circuit #2 of the L293D chip (i.e., the right side with pins labelled in blue), we will use pins **9**, **10**, **11**, **14** and **15** to connect to the motor and the microcontroller. Signals applied to the pins labelled "Enable" and "Control Signals" are either logic LOW or logic HIGH which can be controlled by a microcontroller, e.g., Arduino Uno.

The following table summarizes how the L293D chip operates for bi-directional motor control:

Pin 9 (Enable 3,4)	Pin 10 (Input 3)	Pin 15 (Input 4)	Function
HIGH	LOW	HIGH	Spin in one direction
HIGH	HIGH	LOW	Spin in opposite direction
HIGH	LOW	LOW	Stop
HIGH	HIGH	HIGH	Stop
LOW	-	-	Not enabled

This IC chip can be used in several ways to implement speed control of motor. One possibility is to set Pin 10 and Pin 15 just once for the desired direction of spin. Then, control the logic level of Pin 9 (Enable) according to the required duty cycle.

As an alternative, we can set Pin 9 to logic HIGH just once to always enable the motor driver, and then control the logic levels of Pin 10 and Pin 15 according to the desired duty cycle. This scheme is shown in the sample Arduino code given below (you can download it from LumiNUS).

```
// Give names to the pins of Arduino to be used and
// define their values as integer
int Ena = 7;      // IO port 7 will be connected to Pin 9 (Enable) of L293D
int Mot1 = 5;     // IO port 5 will be connected to Pin 15 of L293D
int Mot2 = 6;     // IO port 6 will be connected to Pin 10 of L293D
int tp;           // Define a variable for period of PWM
int tON;          // Define a variable for ON time
int tOFF;         // Define a variable for OFF time

// Following section will be run once at the beginning.
void setup()
{
      pinMode(Ena, OUTPUT);   // OUTPUT from Arduino
      pinMode(Mot1, OUTPUT);  // OUTPUT from Arduino
      pinMode(Mot2, OUTPUT);  // OUTPUT from Arduino
      digitalWrite(Ena, HIGH);      //Enable pin is set to logic HIGH
      digitalWrite(Mot1, LOW);      // Both control pins are initialized
      digitalWrite(Mot2, LOW);      // to LOW so motor won't spin for now
      tp = 500;   // Period is 500 microseconds; must be >= tON
      tON = 300;  // ON for 300 microseconds
      tOFF = tp-tON;    // OFF for remaining time of the period
}

// Following section toggles pin 15 of L293D between ON and OFF states
// according to the duty cycle.  It will continue looping indefinitely.
void loop()
{
      digitalWrite(Mot1, HIGH); //port 5 (pin 15 of L293) is set to HIGH
      delayMicroseconds(tON);   //wait for tON microseconds
      digitalWrite(Mot1, LOW);  //port 5 is set to LOW
      delayMicroseconds(tOFF);  //wait for tOFF microseconds
}
```

Procedure:

1. Connect the Arduino Uno to the USB port of your laptop, and open the Arduino IDE. Select the correct COM port in the IDE, and ensure that the board is set as "Arduino/Genuino Uno". Copy and paste the sample code given in LumiNUS into your IDE. Save the code and upload to the Arduino Uno.

2. Use the BitScope to observe the signals at I/O pins 5, 6 and 7 of your Arduino Uno. Set the timescale to **100 us/Div**. You should see:
    a. Constant HIGH at Digital IO port #7. (This port will be connected to the "Enable" pin – pin 9 – of the L293D chip later.)
    b. Constant LOW at port #6. (This port will be connected to one of the control pins – pin 10 – of the L293D chip later.)
    c. The Digital IO port #5 toggles between HIGH and LOW. It remains HIGH for approximately t_{on} = 300 μs, and LOW for approximately t_{off} = 200 μs. (This port will be connected to the second control pin – pin 15 – of the L293D chip later.) Note that the timings are not exact as each instruction code has some overhead. (Note: In CG1112, you will be learning about implementing PWM through bare-metal programming of the on-board timers, which has much more precise timing.)

3. You can modify the PWM duty cycle by changing the value of t_{on} in the setup() section of the Arduino code. Change t_{on} to 280 μs, save the code, and upload to Arduino. Observe the changes in the signals at Port #5.

4. You can change the PWM frequency by changing the value of t_p in the code. The value of 500 (as shown in the code) results in a period of 500 μs, or equivalently, a PWM frequency of 2 kHz. Changing this value to 1000 will result in a frequency of 1 kHz. Try changing and uploading to the Arduino. Then, observe the changes in the signals at Port #5.

5. Connect Arduino's 5 V and GND to the breadboard as shown in Figure 5 below. Also, make the remaining necessary connections between the Arduino board, L293D chip, and the motor according to the schematic. Connect all the wires, **except for the red wire of your USB breakout cable.** (Note that for this experiment, it does not matter which of the two motor wires you connect to L293D's pins 11 and 14.) As explained earlier, the motor's current requirement is large, and it must come from another power supply (in our case it is from the USB breakout cable) and ***not*** from your Arduino Uno. The Arduino Uno's +5V is merely used as the power supply for the internal digital logic circuits inside the L293D chip.

    Important:
    - Although the USB breakout cable's nominal voltage is also 5 V, do not connect it to Arduino's 5 V pin as they are regulated by different voltage regulators. **Connecting them together can cause damage to your equipment.** It is just coincidental that our motor's supply voltage is also 5 V because we are using the USB breakout cable. The L293D chip's Pin 8 can take a range of 4.5 V to 36 V as the motor's supply voltage, as seen from its datasheet. Hence, in many other applications, it will not be 5 V.

    - **All the Grounds in the circuit must be connected to each other.** This is important for any circuit you build; otherwise, it may lead to malfunction/equipment damage.

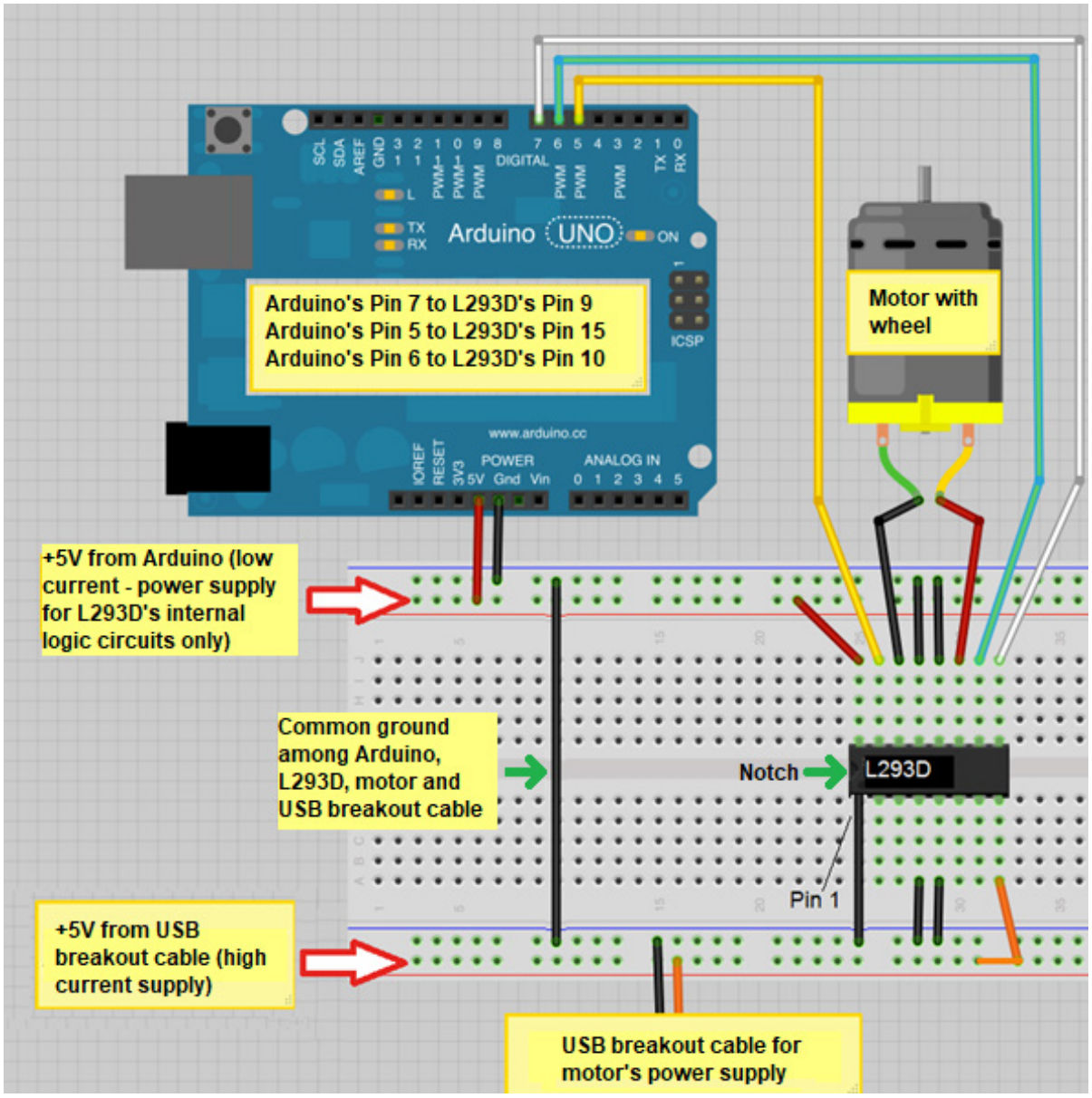

Figure 5: Diagram illustrating connections of Arduino, L293D chip, USB breakout cable, and motor.

6. In the following experiment, you will try using different PWM frequencies according to the values given in Table III by changing the values of t_p. Set the value of t_{on} to maintain a duty cycle of **50%** for each frequency. Hold the motor using one hand and ensure that the wheel will be free to spin in the air. You can now connect the red wire of the USB breakout cable to your breadboard. For each PWM frequency, listen to any humming noise coming from the motor (caused by torque ripples), and note down in Table III whether you can hear it. It may be easier to hear the PWM noise if you stop the motor's spinning by grabbing to the wheel to make it stop. From your observations, which of the following frequencies in Table III would be more appropriate if there are humans working in the vicinity of a PWM-controlled motor?

Table III:

PWM Frequency (KHz)	t_p (in μs)	t_{on} (in μs)	PWM noise audible? (Yes/No)
1	1000	500	Yes
2			
4			
10			
20			

7. Now, using a PWM frequency of 20 KHz, try observing the wheel's speed for the following duty cycles shown in Table IV. If the wheel does not spin, you may try to help it a bit by giving it a twist. Can you observe the increase in speed as the duty cycle increases?

Table IV:

Duty Cycle (%)	t_{on} (in μs)
50	25
60	
70	
80	
90	
100	

8. Finally, modify the Arduino code, so that the wheel will spin in one direction for 10 seconds, and then reverse for 10 seconds. Repeat this indefinitely. You may choose any duty cycle you want.

END OF STUDIO SESSION