Autonomous Vehicle and Robot Sensors

Jinki Kim, PhD

William Hulse

Christian Walker

Department of Mechanical Engineering

Georgia Southern University

The lab manual is intended for students' use in the course MENG 5090 at Georgia Southern University.

TABLE OF CONTENTS

LAB 1: INTRODUCTION TO ARDUINO

Objective

In this lab, students will use the Arduino platform to better understand programming fundamentals like syntax, control structures, and functions.

Introduction

1. Arduino

Arduino is an open-source platform that utilizes microcontrollers in conjunction with compatible software or an IDE (Integrated Development Environment). Arduino is often a beginner friendly option for learning about general programming, embedded systems, and interfacing with various sensors and actuators. As an open-source platform, hardware and software tools are widely accessible, making it ideal for prototyping and experimentation in fields like mechatronics or robotics. In this lab, students will explore the fundamentals of Arduino programming while also gaining practical experience in integrating Arduino with common electronic components like sensors and motors. In becoming familiar with the platform, students can develop the skills necessary to design and implement innovative mechatronic systems, empowering them to tackle real-world engineering challenges with creativity and confidence. Let's embark on this exciting journey of discovery and innovation with Arduino as our guide.

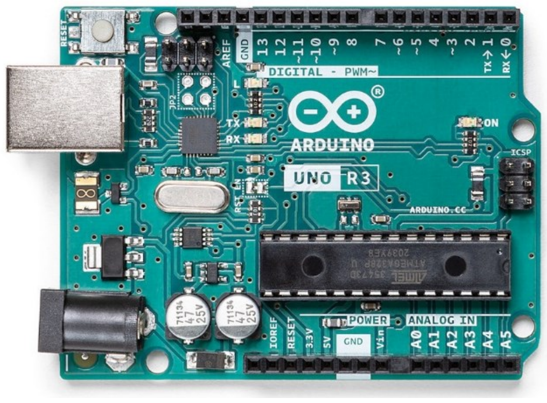


Figure 1. Arduino Uno Rev3 platform

2. Tinkercad

Tinkercad is a web-based computer-aided design (CAD) software Autodesk product. The website includes a community gallery to view popular builds along with plenty of resources such as tutorials, lesson plans, and challenges. In addition to 3D modeling of mechanical systems, Tinkercad is capable of circuit simulation and interfacing with an Arduino microcontroller. Throughout this course, Tinkercad will prove useful for designing and generating circuits and programs for various applications.

Once you have opened Tinkercad and signed into your account (detail info provided in Prelab section), click on the blue “+ Create” button to the right of your profile information on the left side of the page, then click on the circuits option. You will see the circuit tinkering page. Next, drag and drop a breadboard (small), Arduino Uno R3, a LED, and a resistor from the toolbar on the right into the workspace. If any items need to be rotated or mirrored (if possible), then select the appropriate button that are included in the top banner. Wires can be added by clicking where either end should go. You can edit the wire by double clicking anywhere on it to add a bend point on it to move as well. Most ports that you can start or end wires upon give a brief, helpful description of it on the item, such as cathode or anode, positive or negative, and more.

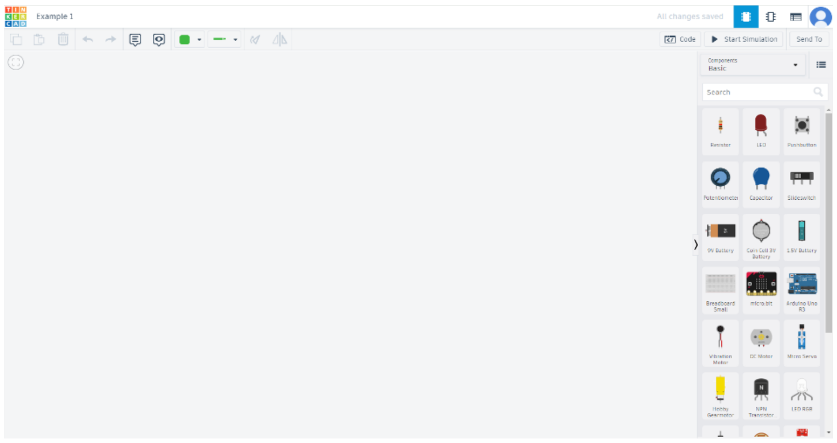


Figure 2. Circuit design page in Tinkercad

A light-emitting diode (LED) is a semiconductor device that emits light when supplied with current. The first official LED was created in 1962 by Nick Holonyak and has since become an extremely common component used in a wide variety of applications. Within Tinkercad, every time an LED is added to a circuit, a resistor must be connected to it or otherwise the LED may be blown with high current. As shown below (fig. 3), a simple circuit to turn on the LED can be created by wiring the ground and 5V ports from the Arduino microcontroller board to the breadboard, and then connecting the power to the resistor and the ground to the end of the LED like shown. To start the simulation and test it, click the button that says "Start Simulation" on the right side of the top banner.

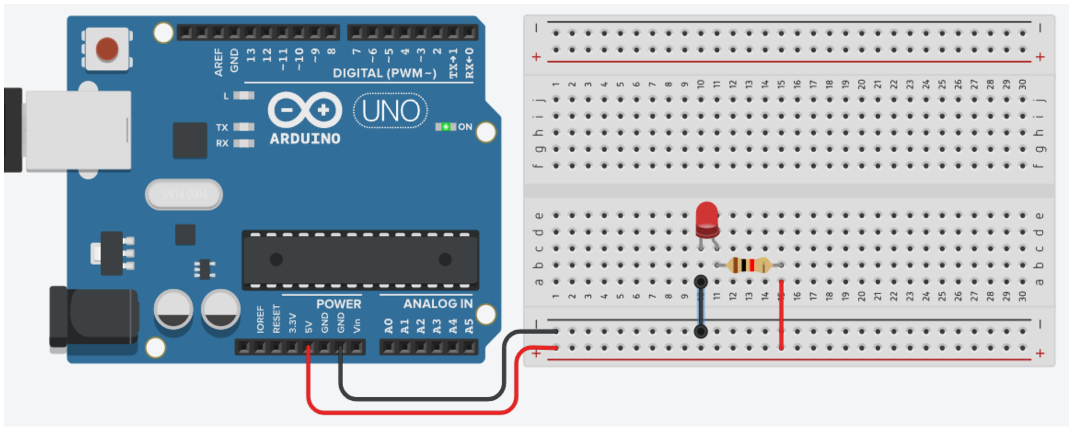


Figure 3. Simple circuit configuration for lighting up an LED.

Liquid crystal displays, or LCD's, are flat-panel electric displays that uses a varied electric voltage applied to a layer of liquid crystal. LCD's have been used in portable electronic games, viewfinders for digital cameras, video projection systems, electronic billboards, computer monitors, and flat-panel televisions. Within Tinkercad, LCD displays can be dragged and dropped into the workspace from the right column like everything else. The display must be wired up to the Arduino microcontroller at 6 ports, along with two ports to connect the power supply (that could be from the Arduino as well) and the ground.

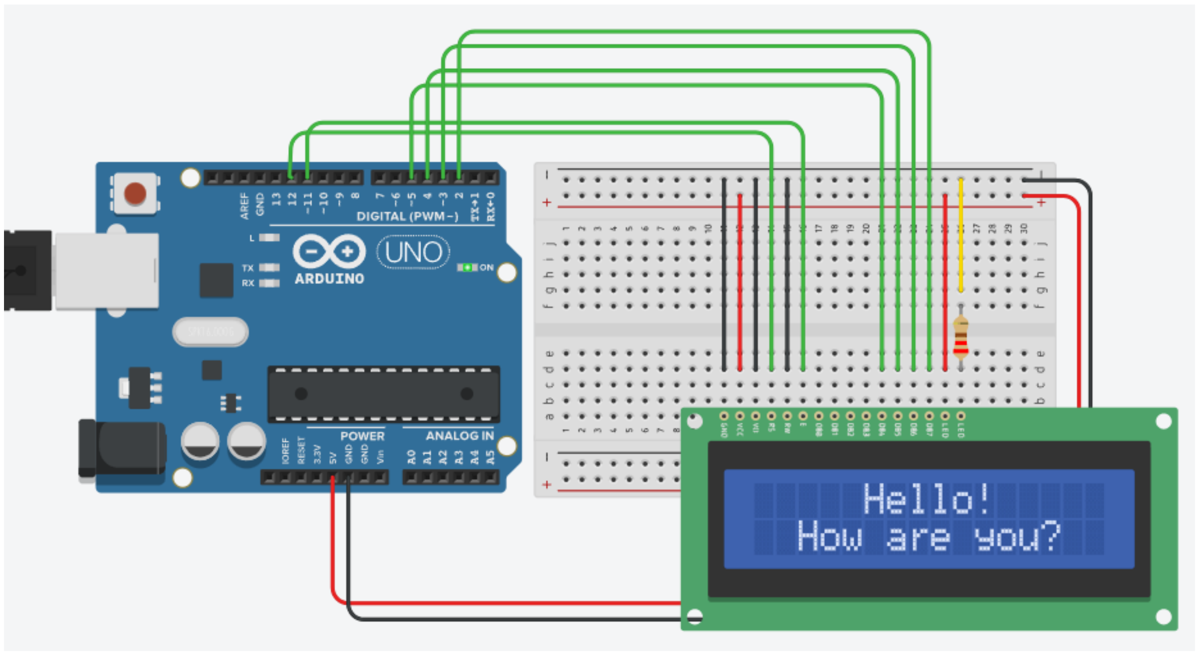


Figure 4. Circuit configuration for LCD with Arduino.

To code within Tinkercad, click on the “Code” button next to “Start Simulation” on the right side of the top banner. The code can then be created within Tinkercad by either blocks, blocks + text, or text. The blocks option lets you drag and drop colored shapes for each part of the code and control the sequence, and the blocks + text option lets the user write code by dragging and dropping the blocks as well but shows equivalent text code side-by-side. The text option, which is the one that will be focused on, lets users write code in C++.

Each code should contain the setup() function where all variables and pin modes are set up, and the loop() function where all of the code that controls the board is looped. Before writing each function though, “void” must be typed to tell the program that the function will not be returning a value. An important method to make the code easier to follow is to add comments by typing a double forward slash at the beginning of the desired line. These comments can be added anywhere after the double forward slash, whether they are in their own line of the code or in the same line as other code.

Before anything else in the code, a preprocessor directive must be written to allow communication with the Arduino Uno to the LCD display through a set library of code and this is indicated by the hashtag (#) at the start of line 1. Line 3 of the code tells the Arduino which pins the required connections are set to between the two components. The void setup() function specifies the dimensions of the LCD display screen, and the void loop() function contains the message. Two ways to write the text within the display are shown within the example code. First, by setting the cursor at the start of the text input at the top left coordinate being (0,0) and adding a couple spaces before the display text in the print command. The second option is by setting the cursor closer to the center of the display and adding less spaces in the display text

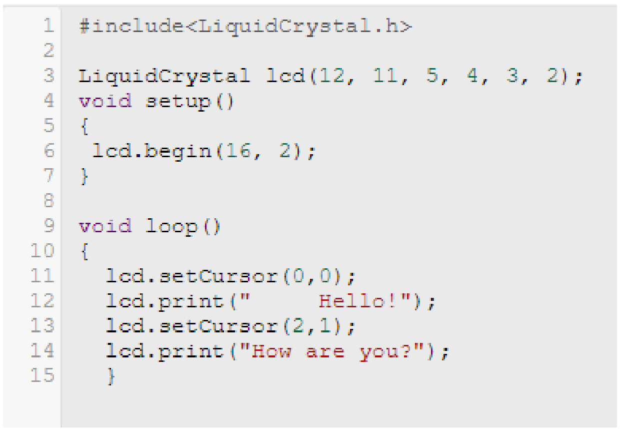
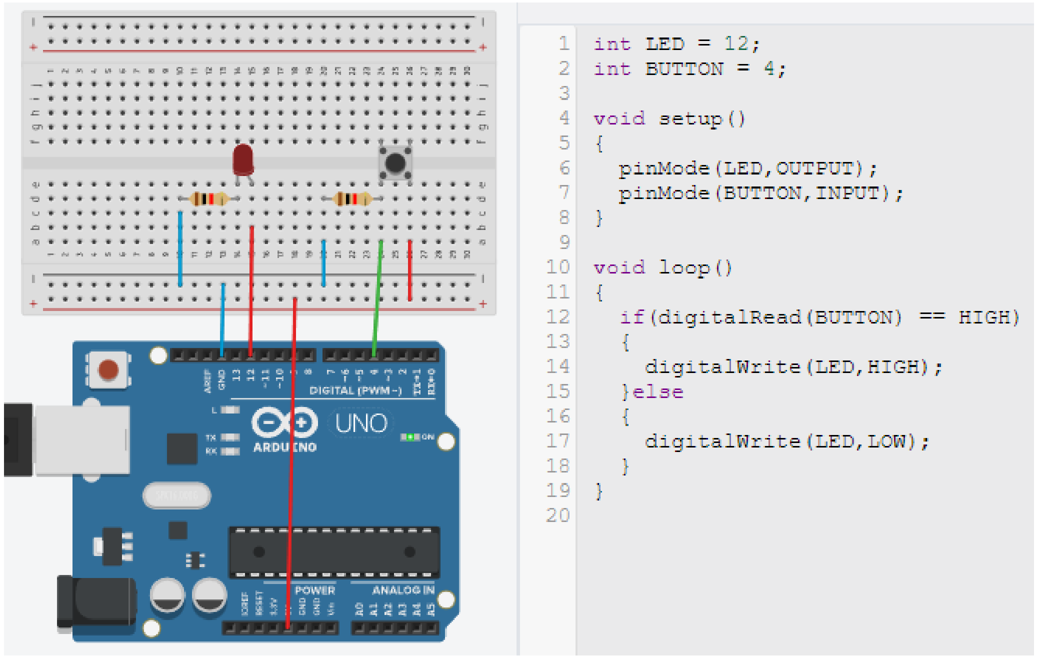

```
#include<LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup()
{
 lcd.begin(16, 2);
}

void loop()
{
  lcd.setCursor(0,0);
  lcd.print("     Hello!");
  lcd.setCursor(2,1);
  lcd.print("How are you?");
  }
```

Figure 5. Code example for the LCD circuit.

Now incorporating a push button from the item options in the right column, here is another example circuit coded to turn on a LED at the push of the button when the simulation is running. The LED and button are both set as integers at the start of the code with the "int" function while telling the Arduino which pin they are connected to on the board. Then the LED and button are set to being an output and input respectively by setting their pinmodes in the void setup() function. Within the void loop() function, the digitalRead command reads the given input (HIGH for on or LOW for off in the button's case) and the digitalWrite gives an input command to make the if/else statement follow the desired requirements of the button to turn the LED on when pressed.



```
int LED = 12;
int BUTTON = 4;

void setup()
{
  pinMode(LED,OUTPUT);
  pinMode(BUTTON,INPUT);
}

void loop()
{
  if(digitalRead(BUTTON) == HIGH)
  {
    digitalWrite(LED,HIGH);
  }else
  {
    digitalWrite(LED,LOW);
  }
}
```

Figure 6. Example LED with push button circuit with code

For additional help, watch the attached videos, which are posted on Folio as well.

- Introduction to Tinkercad
  https://youtu.be/gXG85FOJ2XE
- https://youtu.be/YWY_Is0L7fE will be helpful since it is very similar to how you should create the circuits (LEDs and a button) and write the code for obtaining the deliverables.
- Using an LCD display with Arduino.
  Watch the video up to 13:00, https://youtu.be/wEbGhYjn4QI

Pre-lab assignment

1. Tinkercad Classroom

Tinkercad provides a virtual classroom environment, so that I could directly jump in to your circuits/codes and help you out for debugging. Follow the instructions below (posted as announcement in Folio as well) to get into our virtual classroom in TINKERCAD. To log in to our classroom,

1) Go to https://www.Tinkercad.com/**joinclass/WIHS4HIM3FE3**.
2) Enter your **Nickname that is assigned to you**

   Your Nicknames are assigned as the first part of your Georgia Southern email address without the "@georgiasouthern.edu". For ex., ab12345.

   If you do not type in your "ab12345" but pick any other Nickname, Tinkercad will not let you join the classroom.

2. Deliverable

- Create a circuit that has one push button, two LEDs, and a 16x2 LCD display.
- You need to create your code and circuit so that …
  - each LED turns on based on whether the button is pushed or not.
  - the LCD display indicates the state of the button.
    For example,
    If button is pushed, then LED 1 is on, but LED 2 is off + LCD displays “Button ON”;
    if button is not pushed, then LED 1 is off, but LED 2 is on + LCD displays “Button OFF”;
- Prepare a Word document having all your prelab results attached that will also include your results from the experimental validation. Submission on Folio Dropbox is **due before your next lab starts.** You **MUST** have your results presented in our **classroom created in TINKERCAD**, so that I can help you in case you need debugging. If your work is not shown in our TINKERCAD classroom, there will be **no credits** for this virtual lab deliverable even though you submit your WORD document. If you have any questions about TINKERCAD classroom, you may first refer to the instructions of the previous lab where you created your account and also feel free to ask me.

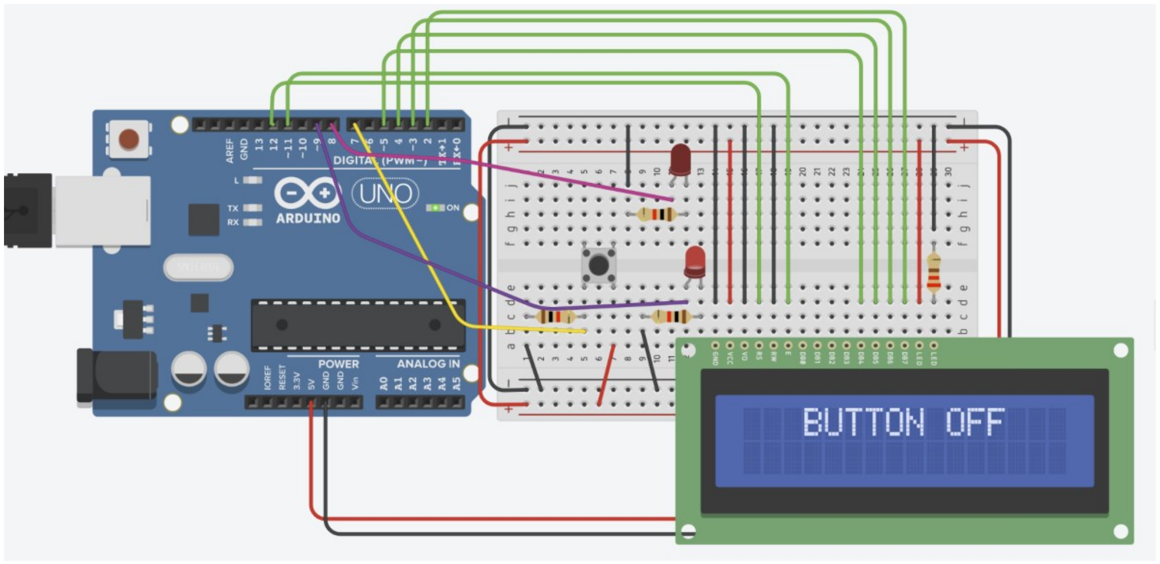
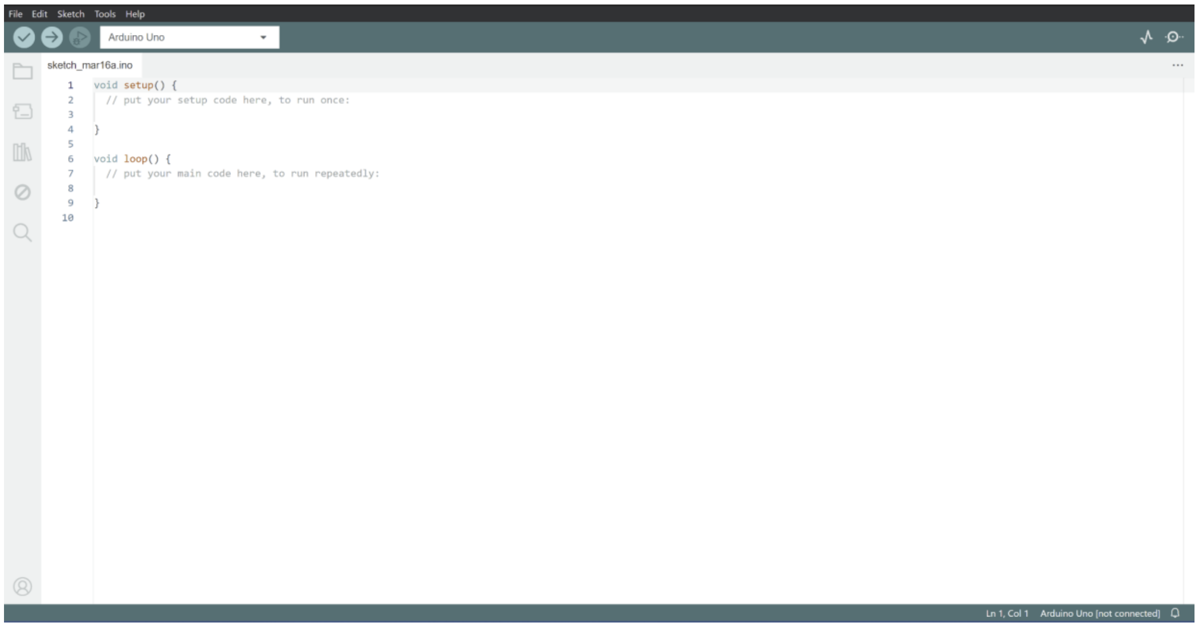


Figure 7. Example circuit diagram for pre-lab deliverable

In-person lab assignment

1. Background

Open the Arduino IDE software on your computer to begin the program and open the window as shown in Figure 8, then connect the Arduino Uno board to the computer to its corresponding USB 2.0 Type A/B connector.


File Edit Sketch Tools Help
Arduino Uno
sketch_mar16a.ino
Ln 1, Col 1 Arduino Uno [not connected]

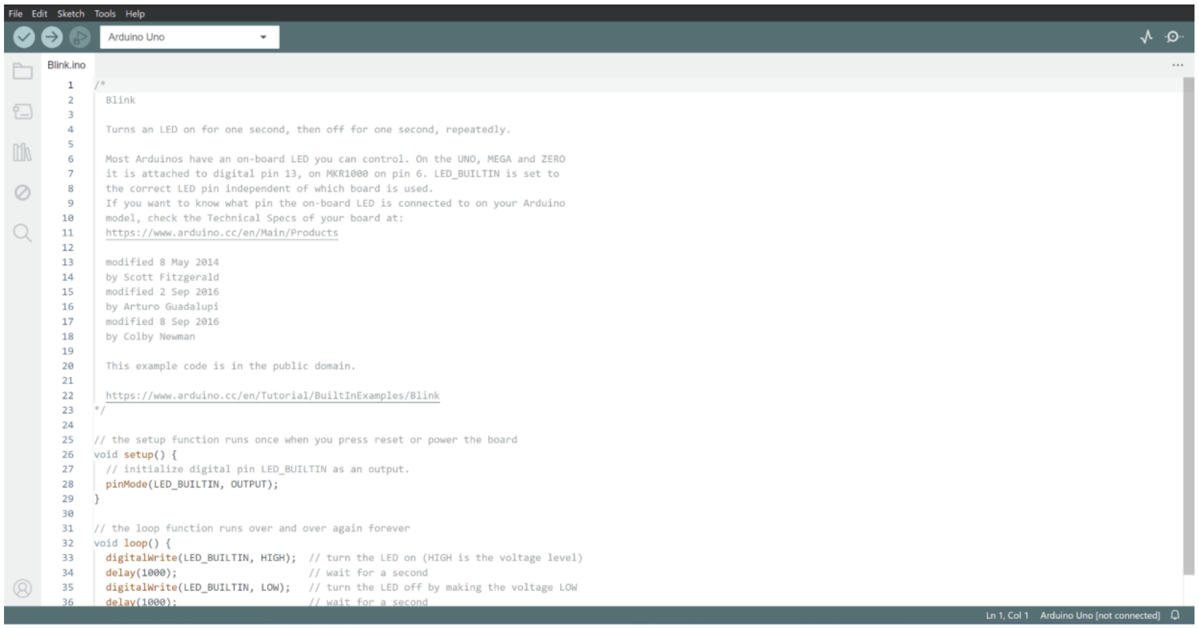
```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Figure 8. Arduino IDE Opening Window

Once Arduino IDE is opened and ready, the board being used must be selected in the top banner in the white dropbox, being the Arduino Uno for all cases in this class. A sketch can then be created by writing code, or copying the code from Tinkercad as Arduino IDE's coding system utilizes C++ in the same way as Tinkercad. The finished sketch can then be verified by clicking on the blue checkmark button on the far left of the top banner or verified and uploaded by clicking on the blue right arrow button to the right of the verification button. The upload button verifies the sketch before uploading, though it is better practice to click the verification button before uploading any sketches.

Keep in mind that only one sketch can be contained in the Arduino board at a time. Uploaded sketches will continually run on the Arduino while it is plugged into the computer and will only stop when either another sketch is uploaded, or the Arduino is unplugged.

Arduino IDE includes basic example sketches that can be found by clicking File and Examples. One simple example that only requires the Arduino Uno board and no circuit being connected to it is the blink sketch. The blink example "blinks" the LED that is part of the Arduino Uno board and can be found by clicking File >> Examples >> 01.Basics >> Blink.



```
/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
```

Figure 9. Example Blink Sketch

For additional information, visit the following links:

- Official getting started with Arduino page
  https://docs.arduino.cc/learn/starting-guide/getting-started-arduino/
- Official beginner's introduction and guide to the Arduino software (IDE)

https://docs.arduino.cc/learn/starting-guide/the-arduino-software-ide/

- YouTube guide to getting started: https://www.youtube.com/watch?v=_-g5sWQyROg

2. Deliverable

- Recreate the previous circuit from Tinkercad that has one push button, two LEDs, and a 16x2 LCD display using the lab equipment using the given Arduino equipment and software in the lab.
- You need to create your code and circuit so that …
  - each LED turns on based on whether the button is pushed or not.
  - the LCD display indicates the state of the button.
- Submit a Word document having all your results attached. Submission on Folio Dropbox is due before your next lab starts.

LAB 2: DC MOTOR CONTROL

Objective

In this lab, students will construct a circuit with a microcontroller to control a DC motor using a PWM and an H-bridge circuit.

Introduction

1. DC Motors

Direct current (DC) motors are widely used electric motors in which the rotating armature and field generating wires are connected in series. One important component of DC motors is the brush assembly, which maintains contact with the armature and provides the necessary charge to the rotating armature (Figure 1). The motor's design provides a high starting torque that is often utilized in industrial applications such as cranes, elevators, and automobile starters.

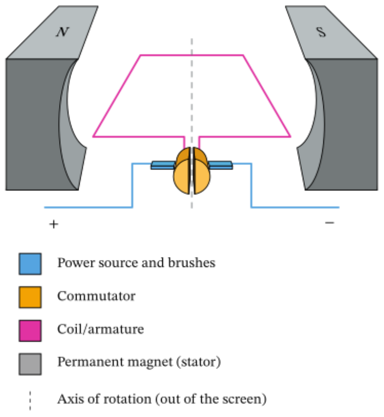

N
S
+
−
Power source and brushes
Commutator
Coil/armature
Permanent magnet (stator)
Axis of rotation (out of the screen)

Figure 1. DC motor diagram [1]

The speed of a DC motor is directly proportional to the induced electro-magnetic field (EMF) of the armature. The revolutions per minute (RPM) of a DC motor can be determined with respect to the supplied voltage using Equations 1 and 2 below:

$$E_b = V - I_a R_a \tag{1}$$

In Eq. 1, E_b is the EMF of the DC motor, V is supply voltage, I_a is armature current, and R_a is armature resistance.

$$N = K\frac{E_b}{\Phi} \tag{2}$$

Using the calculated E_b from Eq. 1, Eq. 2 can be used to calculate the RPM of the motor (N) along with the characteristic constant of the motor (K) and the flux per pole of the DC motor (Φ).

2. Controlling the Speed of a DC Motor Using Pulse Width Modulation (PWM)

Simply supplying the motor with voltage is not suitable for controlling its rotation speed and direction. In all configurations the motor can either be powered or not powered, therefore pulse width modulation is used to control the rotation speed. This technique is used to achieve an analog-like output by pulsing a digital input on and off repeatedly. The achieved "average" voltage is proportional to the duty cycle, or pulse width (Fig. 2) which is the ratio of "on" time to a given time interval and is expressed as a percentage (Eq. 3).

$$duty\ cycle = \frac{\text{"on"}\ time}{period} \times 100 \tag{3}$$

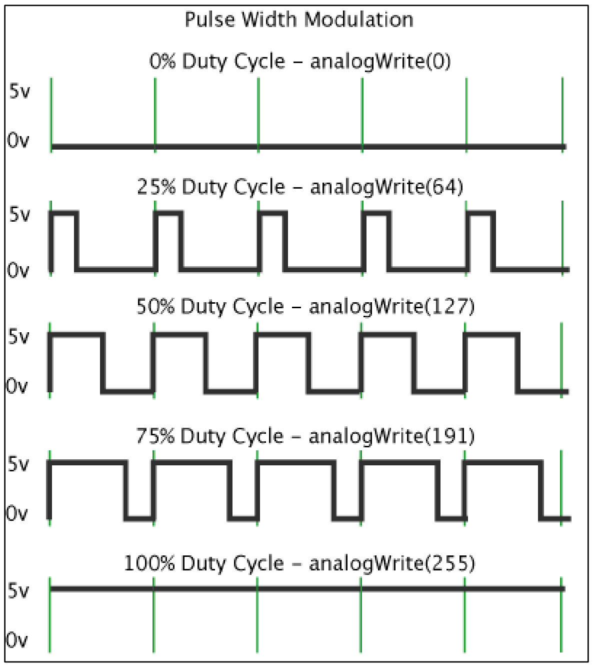


Figure 2. Duty cycles of PWM [2]

3. Controlling the Rotation Direction of a DC Motor Using an H-Bridge Circuit

Having control over the DC motor's speed, the rotation direction is next. Using the simple configuration from before where the motor is provided with current, the motor only spins one direction. If the current polarity is reversed, the motor now spins in the opposite direction. This is the basic principle of an H-bridge circuit, shown in Figure 3. It has four switches at each "corner" with the motor acting as the "bridge". By operating opposing pairs of switches, the motor's rotation direction can be controlled easily. Using a PWM and H-bridge in combination allows for easier configuration, but the ability to program the motor's behavior is still needed.

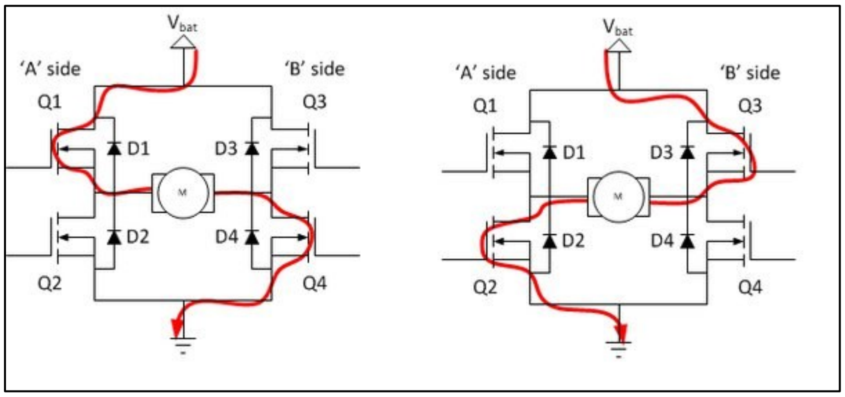


Figure 3. H-Bridge Diagram [5]

4. Programming the Motor Functions Using an L298 Motor Driver

The primary control method uses a microcontroller connected to a L298 DC motor driver to program the movement of the motor, with an Arduino Uno in conjunction with the Arduino computer software acting as the microcontroller in this lab. The L298 driver is a high voltage, high current dual full-bridge driver that is suitable for inductive loads such as relays, solenoids, and both DC and stepper motors. Integrated circuit (IC) chips like the L298 are necessary for these loads as microprocessors generally cannot generate the necessary current. Each bridge is driven by four gates, much like the H-bridge circuit discussed previously. Bridge 1 is driven by *Input 1*, *Input 2*, and *Enable A*. Bridge 2 is driven by *Input 3*, *Input 4*, and *Enable B*. For the purposes of this lab only one bridge will be utilized, but it should be noted that the bridges operate individually and can be used to drive two independent motors simultaneously. When *Enable* is provided with low or no voltage (0), all gates/inputs are inactive. Providing *Enable* with high voltage (1) allows the inputs to set the bridge state. All bridge states are shown (Fig. 4) as well as the pinout (Fig. 5) and circuit diagram (Fig. 6).

Enable	*Input 1*	*Input 2*	Spinning Direction
Low(0)	-	-	Stop
High(1)	Low(0)	Low(0)	Motor OFF (Break)
High(1)	High(1)	Low(0)	Forward
High(1)	Low(0)	High(1)	Backward
High(1)	High(1)	High(1)	Motor OFF (Break)

Figure 4. Bridge State Diagram [4]

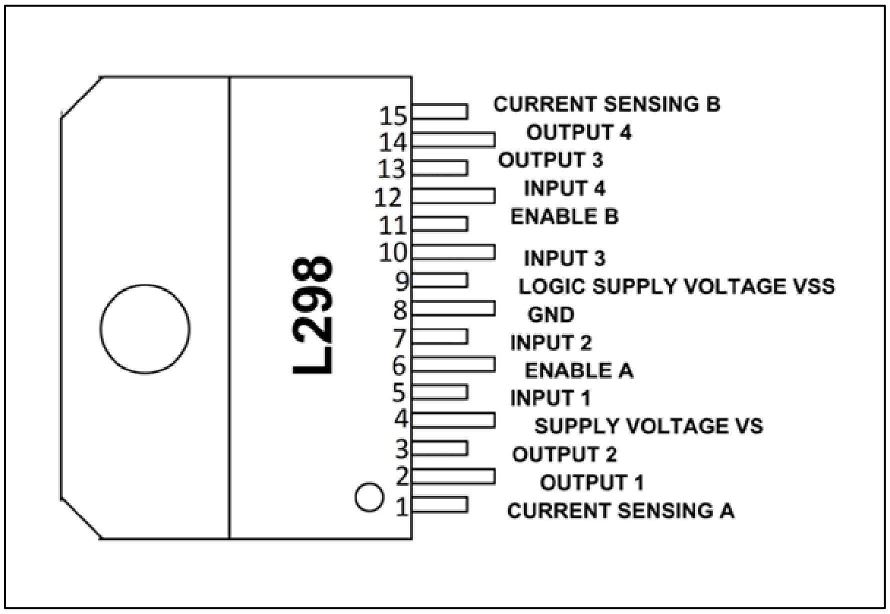


Figure 5. Pinout Diagram of L298 [4]

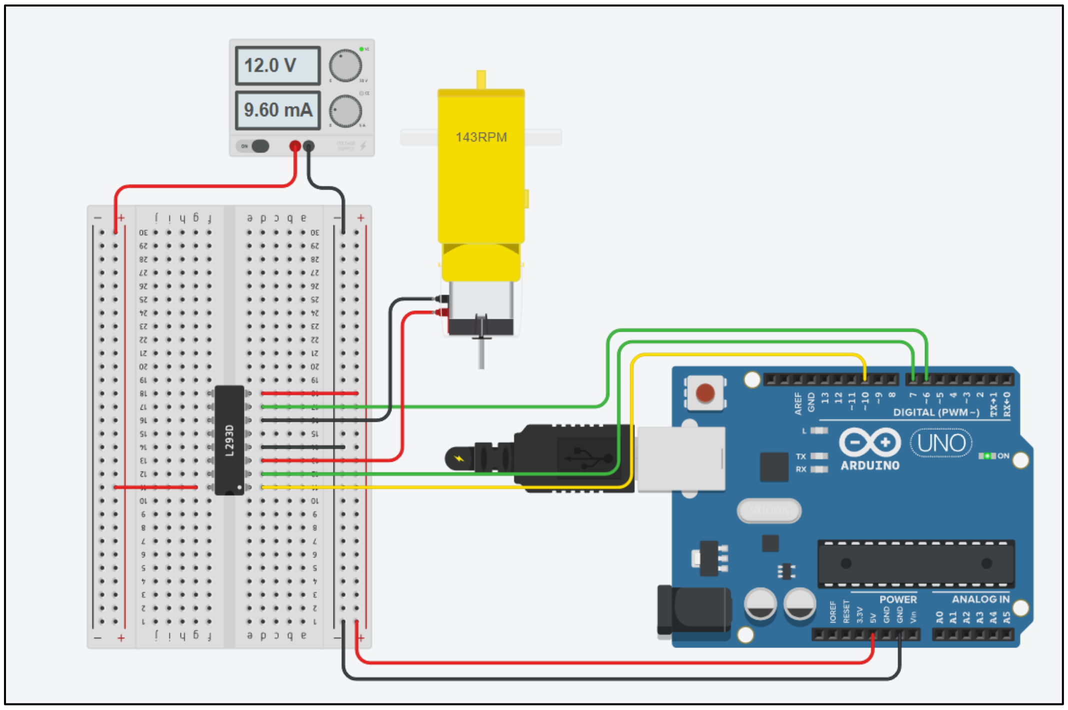


Figure 6. Circuit Diagram

In-person lab assignment

1. Deliverables:

Construct the circuit using an Arduino Uno, L298 chip, power supply, and DC motor, referring to the pinout diagram shown in Figure 5.

a. Generate a program to rotate the DC motor for 5 seconds, stop for 5 seconds, and then rotate in the opposite direction.

b. Modify the above program to rotate the motor clockwise with a constant acceleration for 1 sec, to run for 5 sec at the constant speed, and to decelerate at a constant rate for 1 sec before stopping. Use Arduino's analogWrite function to change the output duty cycle. Emulate the output voltage from 0 V to 5 V by changing the second argument of the analogWrite function, on a scale from 0 (0% duty cycle) to 255 (100% duty cycle). Observe how the rotating speed of the motor changes with the duration of the duty cycle.

c. Modify the above program to rotate the motor clockwise (counterclockwise) with a constant acceleration when a push button is pushed (not pushed).

Sample Code: The following codes may need to be updated for your specific system build.

```
// Motor connections
int enA = 10;
int in1 = 7;
int in2 = 6;
void setup() {
        // Set the motor control pins to outputs
        pinMode(enA, OUTPUT);
        pinMode(in1, OUTPUT);
        pinMode(in2, OUTPUT);
        // Turn off motors
        digitalWrite(in1, LOW);
        digitalWrite(in2, LOW);
}
// Deliverable A (sample)
        void directionControl() {
        // Rotate for 5 seconds
        analogWrite(enA, 255); // PWM range: 0 to 255
        digitalWrite(in1, HIGH);
        digitalWrite(in2, LOW);
        delay(5000);

        // Stop for 5 seconds
        digitalWrite(in1, LOW);
        digitalWrite(in2, LOW);
        delay(5000);

        // Reverse rotation for 5 seconds
        digitalWrite(in1, LOW);
        digitalWrite(in2, HIGH);
        delay(5000);

        // Stop
        digitalWrite(in1, LOW);
        digitalWrite(in2, LOW);

}
```

References:

[1] DC motor information: Link; L298 information

[2] Image courtesy of https://www.nagwa.com/en/explainers/246108560531/

[3] Image courtesy of https://docs.arduino.cc/learn/microcontrollers/analog-output/

[4] L298 Pinout Diagram courtesy of https://components101.com/ics/l298-pin-configuration-features-datasheet

[5] H-Bridge Diagram courtesy of https://quora.com/What-is-the-working-of-a-H-bridge-circuit

LAB 3: IMPLEMENTATION OF HALL SENSOR USING ARDUINO

Objective

In this lab, students will use a microcontroller to measure a hall sensor and display its results on LCDs.

Introduction

1. Hall Effect Sensor

In 1849, U.S. physicist Edwin Herbert Hall discovered a phenomenon when electrons circulating around a semiconductor were deflected in the presence of a magnetic field. This became known as the Hall effect. The Hall sensor has since become one of the most common methods of measuring magnetic fields in the modern era. They are used in the automotive industry in multiple functions to monitor the safety of the vehicle as well as measure the position of the crankshaft or camshaft. Further applications include measuring fluid velocities, metal detection, induction factors, or acting as a switch or proximity sensor. An immunity to both noise and dust makes them reliable and durable sensors, on top of their advantage of remotely measuring without requiring any physical contact.

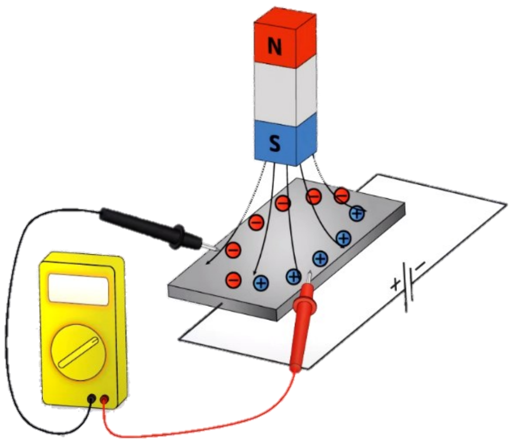


Figure 1. Hall Effect Diagram [3]

In this lab, the SS49E Linear Hall Effect Sensor will be utilized. The output provides an analog voltage representing if a magnetic field is present and has a range of 0.8 – 4.2 V. Amplifiers, voltage regulators, and logic switching circuits are often used with this sensor to overcome noise influences in the output. The SS49E Linear Hall Effect Sensor's pins are numbered 1 – 3 from left to right, when the branded side with text written on it as well as the drafted side edges is facing forward. The first pin is the sensor's power supply, the second its ground connection, and the third its analog output.

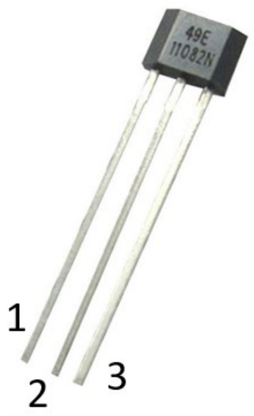


Figure 2. Hall Sensor

- VCC (1): Module power supply, 5V
- GND (2): Ground
- OUT (3): Analog output

2. Sample Setup for Displaying Hall Sensor Measurement

With one Arduino Uno, an SS49E Hall sensor, an I2C LCD display (students may utilize the LCD set up from the previous lab), wires, and a breadboard the following wiring setup can be created as shown in Fig. 3.

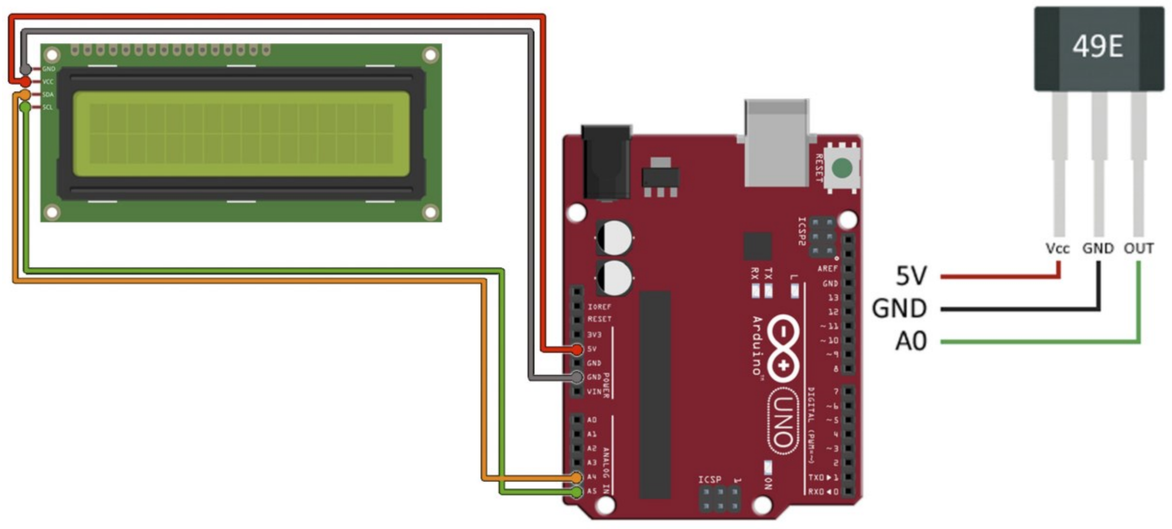


Figure 3. Sample Wiring Setup.

This sample setup utilizes a formula to convert the measured voltage into magnetic flux density and display the results in the serial monitor.

In-person lab assignment

1. Deliverables

Use the Arduino Uno to build a circuit that reads and displays the Hall sensor results on a 16x02 LCD.

- Use Arduino's "analogRead()" function to read the Hall sensor results from an analog input pin. The analogRead() command converts the input voltage range, 0 to 5 volts, to a digital value between 0 and 1023. This is done by a circuit inside the microcontroller called an *analog-to-digital converter* or *ADC*.

- Use an "if-else" statement and set a threshold value depending on your sensor reading, so that you can determine whether the sensor detects the magnetic field or not. For example, if your sensor reading is greater than 500, you call it's detecting the field and thus an LED turns on. If smaller than 500, there's no significant magnetic field around and LED is off.

- Display both the analog readings and the estimated magnetic flux value on a 16x02 LCD. For example, on the first row display the analog readings, and magnetic flux density (G) on the second row. The magnetic flux density could be estimated by using the info provided on the specsheet.

Submit a Word document that has all your results attached, including the circuit diagram for the sensor measurement setup, code, and links to the videos showing your results.

Sample Code: The following codes may need to be updated for your specific system build.

```
const int pinHall = A0;
void setup() {
  pinMode(pinHall, INPUT);
  Serial.begin(9600);
}
void loop() {

  //we measure 10 times adn make the mean
  long measure = 0;
  for(int i = 0; i < 10; i++){
      int value =
      measure += analogRead(pinHall);
  }
  measure /= 10;
  //voltage in mV
  float outputV = measure * 5000.0 / 1023;
  Serial.print("Output Voltaje = ");
  Serial.print(outputV);
  Serial.print(" mV   ");

  //flux density
  float magneticFlux =  outputV * 53.33 - 133.3;
  Serial.print("Magnetic Flux Density = ");
  Serial.print(magneticFlux);
  Serial.print(" mT");
  delay(2000);
}
```

References

[1] Electronic Components Datasheet https://www.alldatasheet.com/datasheetpdf/pdf/1242554/OHHALLSENSOR/OH49E-S.html

[2] Background info & Applications, & Sample 2 https://electronoobs.com/eng_arduino_tut82.php

[3] Image & background info https://howtomechatronics.com/how-it-works/electrical-engineering/hall-effect-hall-effect-sensors-work/

[4] https://electronoobs.com/eng_arduino_tut82.php

LAB 4: STEPPER MOTOR CONTROL

Objective

In this lab, students will control a stepper motor and measure its RPM with a hall-effect sensor using Arduino UNO.

Introduction

1. Stepper Motor

Stepper motors are actuators that rotate in a series of “steps” when supplied with a direct current. While both DC and stepper motors operate via inducing electromagnetic fields, their function and construction are vastly different. DC motors operate using armatures and brushes to achieve continuous rotation under load. A typical stepper motor attracts and repels eccentric toothed rotors, mounted to a permanently magnetized central shaft, along an outer ring of teeth. This outer ring, often referred to as the stator, has a greater number of teeth than the rotors and houses multiple sets of electromagnets (Fig. 1). By actuating these electromagnets in alternating pairs, the toothed rotors are “stepped” along the outer ring due to the inequality in tooth number. This provides increasingly accurate rotation control and circumvents the need for brushes, which contribute to motor inefficiency via parasitic losses like friction. These attributes are highly favored in robotics applications where the ability to program precise movements is essential.

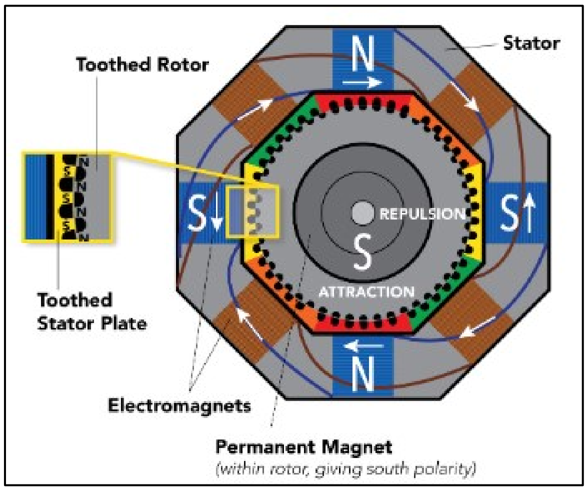


Figure 1 - Basic Diagram of a Stepper Motor

An important characteristic of these motors is the step angle, or angular travel over a single step. This can be calculated in two ways, shown below in equations 1 and 2, where n is the number of steps per revolution, n_R is the number of teeth on the rotor, and p_S is the number of phases in the stator.

Typical stepper motors have a step angle of 1.8 degrees per step, or 200 steps per revolution.

$$\theta = \frac{360°}{n} \tag{1}$$

$$\theta = \frac{360°}{2(n_R)(p_S)} \tag{2}$$

Another important stepper motor characteristic is micro-stepping, or the ability to actuate multiple phases simultaneously. This may contribute to a loss of accuracy but generally reduces the rotors overall travel and increases step resolution. A motor in quarter-step mode multiplies its steps per revolution by a factor of four. Typically, a 1/16-step mode is the maximum step resolution while maintaining accuracy, but some motors can go as high as 1/32-step.

2. Controlling the Motor with an A4988 Driver

The A4988 stepper motor driver has a maximum output capacity of 35 volts and $\pm$ 2 amps and has a rotation resolution down to a sixteenth step. Interfacing with the standard Arduino microcontroller allows the rotation speed and direction to be easily controlled. The pinout diagram for the A4988 driver is provided in Fig. 2-3. The power supply is connected to the VMOT and GND pins of the driver. It is important that the wires for one coil are connected to 1A and 1B and that the wires for the other coil are connected to 2A and 2B. Polarity does not matter in this case. The other GND pin is connected to the ground of the microcontroller and VDD is wired to the 5V supply. MS1, MS2, and MS3 are the resolution selector pins and allow up to 1/16th steps based on the configurations shown in Figure 4. The ENABLE pin disables the driver when powered. The RESET and SLEEP pins are active, low input, meaning the driver is in "sleep/reset" mode when these pins are unpowered. While this may be useful in other applications, they are wired to each other for this lab to ensure the driver functions as desired. Finally, the STEP and DIR pin are connected to the digital pins on the Arduino board. Depending on the polarity of the motor connection, setting the DIR to HIGH will reverse the rotation direction. The number of steps and/or revolutions is controlled within the code by a "for loop." To code for a single revolution of the motor, the loop must be executed 200 times. Rotation speed is entirely dependent on the pulse frequency of the STEP pin, so changing the value of "delayMicroseconds()" to a shorter delay in the code will spin the motor faster.

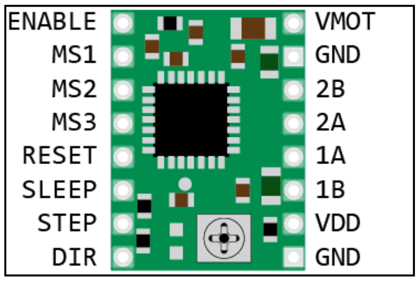


Figure 2. A4988 Motor Driver Pinout Diagram

Before the circuit is operational, it is important to calculate and set the current limit of the driver to protect the stepper motor and driver. Failure to do so may damage the components or result in excess motor noise. The reference voltage (V_{Ref}) will be set by adjusting the potentiometer onboard the A4988 driver. The current limit (C) is set as 1 A, but the maximum value can be found in the A4988 datasheet. The current sense resistance (R_{CS}) is given as 0.068 Ω, which yields a reference voltage of 540 mV.

$$C = \frac{V_{Ref}}{(8 * R_{cs})} \tag{3}$$

To adjust the driver, first disconnect the stepper motor from the circuit and the power supply using the Arduino board. Using a multimeter, probe the GND pin and the potentiometer to measure the reference voltage. Adjust the potentiometer until the desired reference voltage is met.

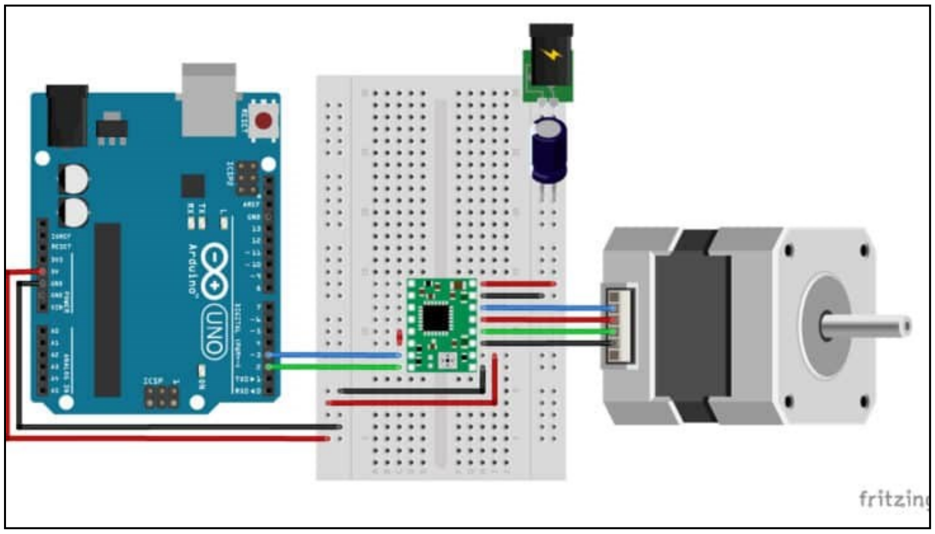


Figure 3. Stepper Motor Controller Circuit Diagram

MS1	MS2	MS3	Micro-step Resolution
Low	Low	Low	Full Step
High	Low	Low	1/2 Step
Low	High	Low	1/4 Step
High	High	Low	1/8 Step
High	High	High	1/16 Step

Figure 4. Micro-step Resolution Table

3. Measuring the RPM of the Hall-effect Sensor

A hall-effect sensor is an integrated circuit that produces electrical signals when induced magnetic fields are detected. In conjunction with the Arduino microcontroller and an LCD display, a rudimentary tachometer can be created to measure and display the RPM of the stepper motor. To determine the RPM, the time between each detection can be calculated via the output of the sensor. A simplified circuit diagram showing the connections between the LCD display and hall-effect sensor can be seen below (Fig. 6).

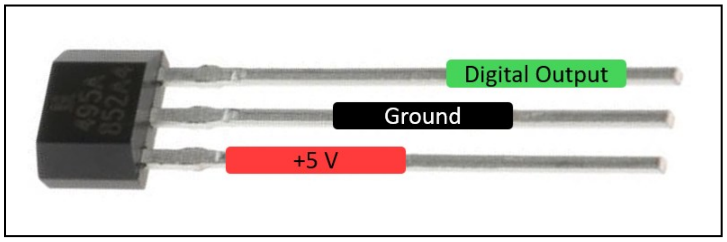


Figure 5. Hall-effect Sensor Pin Diagram

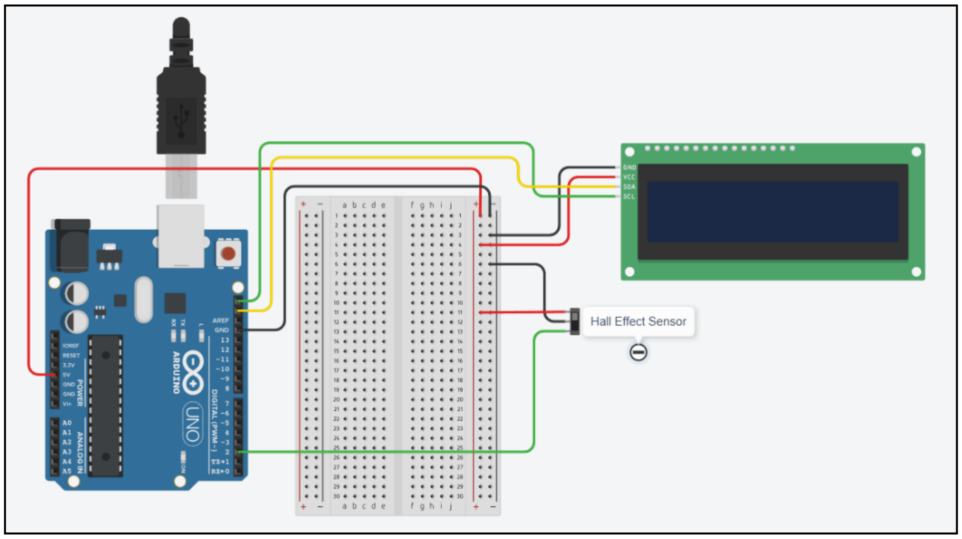


Figure 6. Hall Effect LCD Circuit Diagram

In-person lab assignment

1. Deliverable

1. Construct a circuit using the Arduino Uno, A4988 driver, power supply, and stepper motor.
    a. Calculate and set the current limit of the A4988 motor driver.
    b. Generate a program to rotate the stepper motor for 5 revolutions clockwise, stop for 5 seconds, and then rotate for 5 revolutions in the counterclockwise direction.
2. Use a Hall-effect sensor and display the RPM of the motor on an LCD screen.
3. Run the motor at the maximum available RPM. Compare this value with the stated RPM from the motor's spec sheet.

Sample codes: The following codes may need to be updated for your specific system build.

```
// Controlling a stepper motor with A4988 stepper motor driver
// and Arduino without a library.
// More info: https://www.makerguides.com
// Define stepper motor connections and steps per revolution:
#define dirPin 2
#define stepPin 3
#define stepsPerRevolution 200

void setup() {
  // Declare pins as output:
  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
}

void loop() {
  // Set the spinning direction clockwise:
  digitalWrite(dirPin, HIGH);
  // Spin the stepper motor 1 revolution slowly:
  for (int i = 0; i < stepsPerRevolution; i++) {
    // These four lines result in 1 step:
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(2000);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(2000);
  }

  delay(1000);

  // Set the spinning direction counterclockwise:
  digitalWrite(dirPin, LOW);

  // Spin the stepper motor 1 revolution quickly:
  for (int i = 0; i < stepsPerRevolution; i++) {
    // These four lines result in 1 step:
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(1000);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(1000);
  }

  delay(1000);

  // Set the spinning direction clockwise:
  digitalWrite(dirPin, HIGH);

  // Spin the stepper motor 5 revolutions fast:
  for (int i = 0; i < 5 * stepsPerRevolution; i++) {
    // These four lines result in 1 step:
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(500);
  }

  delay(1000);

  // Set the spinning direction counterclockwise:
  digitalWrite(dirPin, LOW);
  //Spin the stepper motor 5 revolutions fast:
  for (int i = 0; i < 5 * stepsPerRevolution; i++) {
    // These four lines result in 1 step:
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(500);
  }

  delay(1000);
}
```

References:

[1] Stepper Motors with Arduino: https://www.youtube.com/watch?v=0qwrnUeSpYQ

[2] Motor Manual: https://lastminuteengineers.com/a4988-stepper-motor-driver-arduino-tutorial/

[3] Stepper Motor Controller Tutorial Article: https://www.makerguides.com/a4988-stepper-motor-driver-arduino-tutorial/

[4] A4988 Controller Datasheet: https://www.makerguides.com/wp-content/uploads/2019/02/A4988-Datasheet.pdf

[5] Arduino Uno Tachometer RPM using Hall-effect Sensor: https://www.youtube.com/watch?v=pIflB4FQpNE

LAB 5: LIDAR SYSTEM

Objective

In this lab, students will create a basic Light Detection and Ranging (LIDAR) sensor using a microcontroller and signals from a laser-based time-of-flight (TOF) sensor to understand how LIDAR works.

Introduction

1. The Basic Principles of LIDAR:

LIDAR is a method of scanning physical environments and objects using pulses of light. The time between an outgoing pulse and detecting the reflection of that pulse surroundings allows the distance to be calculated using eq. 1, where D is distance, C is the speed of light, and TOF is the elapsed time.

$$D = \frac{C \times TOF}{2} \tag{1}$$

By sending and receiving thousands of light pulses per second, the system generates high-resolution 3D information which proves useful in mapping large complex areas like the topology of the earth. LIDAR also has a foothold in the autonomous driving industry given its ability to accurately measure and map complex and dynamic environments. A typical, scannerless LIDAR sensor consists of a time-of-flight (TOF) sensor, a GPS for positional accuracy, a data processing unit, and certain mechanical components that serve to rotate the sensor like a step motor and slipring.

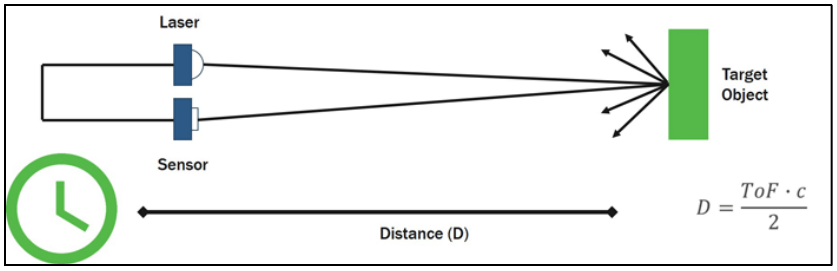


Figure 1. Schematic of LIDAR principle

2. TOF Sensor

A time-of-flight sensor is a camera system used to measure distance between the sensor and a subject. The TOF sensor is what emits and receives pulses of light and calculates the distance. Rotating this sensor continuously allows the system to collect ranges in 360° around the entire robot, not just in a static direction.

3. Microcontroller

To rotate the TOF sensor, a step motor and subsequent motor driver will be connected to the Arduino Uno. The Arduino Uno collects the distance data from the TOF sensor and the angle measurements from the step motor, allowing the system to plot a map of its surroundings.

In-person lab assignment

1. Instructions

A. Download the 3D STL files (for 3D printing the LIDAR cases) from Folio and update the design so that it will fit the bearing, motor, and Arduino Uno. Print the case with 2 perimeters, 0.3mm layer height, PLA material and 20% infill.

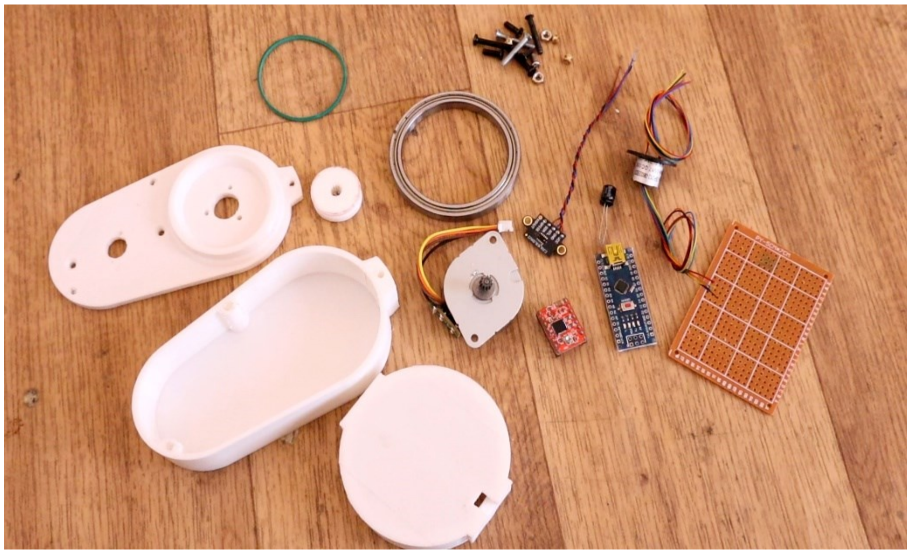


Figure 1. Example of parts utilized for the LIDAR.

B. Connect the system following the schematic provided in Figure 2 as an example. Note that Arduino UNO is utilized in this lab. Connect the I2C pins from the Arduino to the Slip ring. From the slip ring to the VL53L0 distance sensor. Then connect 5V and the D8 pin to the Hall sensor. Connect enable, step and direction to the step motor driver and the motor to the driver. Now connect 5V to the boost converter and set the output to 12V and connect that to the power input of the step motor driver.

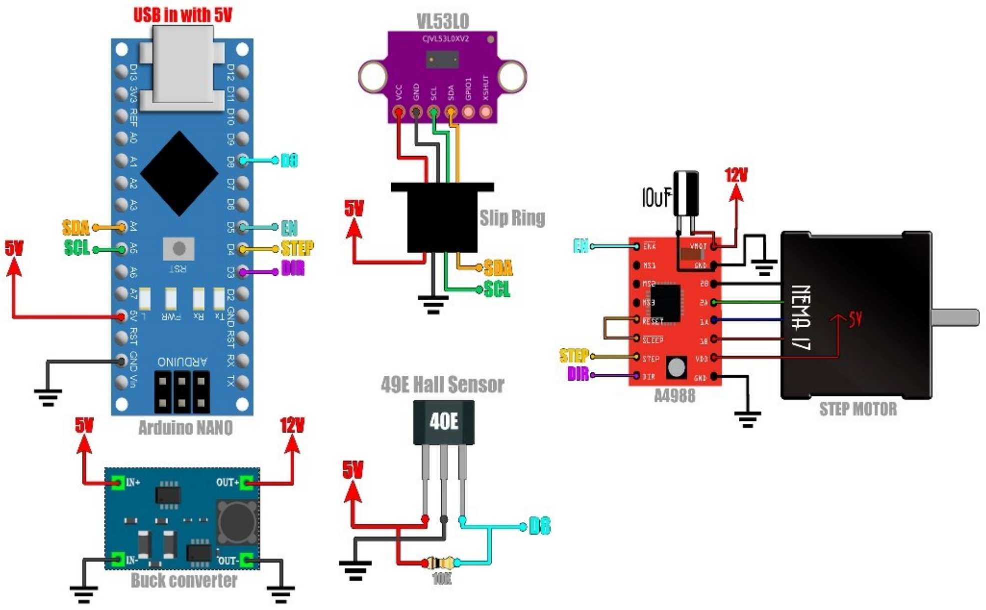


Figure 2. Schematic diagram for connecting sensors.

C. Take the slip ring and place it on the top part of the case. Make sure that the rotating part of the ring is on the upper side of the case so it will spin at the same time as the disc. Add the step motor using two 3M screws and nuts. Get the boost converter and solder wires for 5V. Then supply the converter and set it to 12V and then you can solder wires from the output to the step motor driver. Now the driver is supplied with 12V. Finally, add the hall sensor as in the schematic with a 10 kΩ resistor and connect it to the Arduino. Secure the sensor in place on the side of the rotating disc and on the disc solder the magnet. In this way the rotation of the LIDAR can be detected. Solder the magnet in the opposite side of the distance sensor so 180 degrees of difference can be determined.

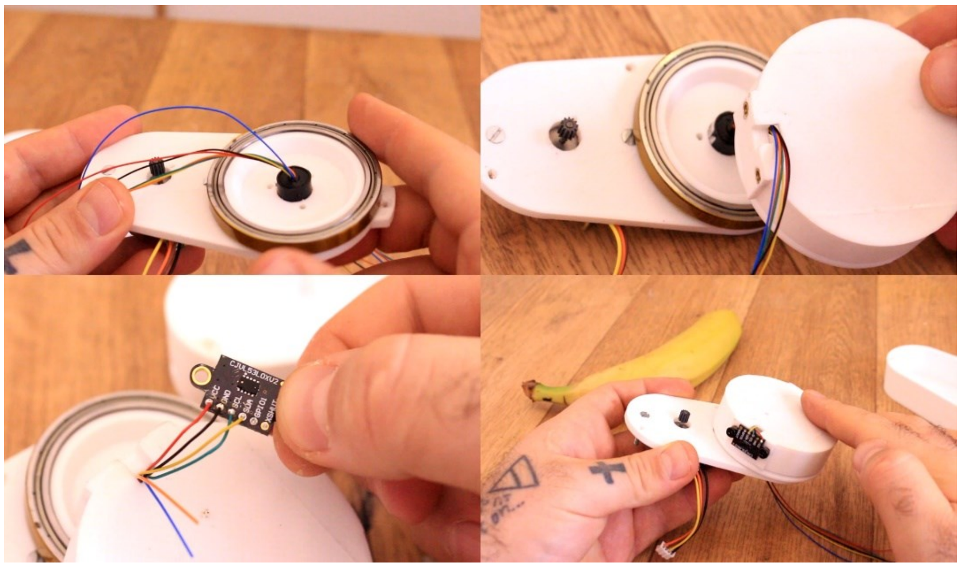


Figure 3. Parts assembly

Sample code: The codes may need to be updated for your specific system build.

```
/* Lidar code by ELECTRONOOBS
 * Get distance and angle and send via serial port RX, TX
 * Tutorial: https://electronoobs.com/eng_arduino_tut110.php
 * Schematic: https://electronoobs.com/eng_arduino_tut110_sch1.php
 * Code: https://electronoobs.com/eng_arduino_tut110_code1.php
 * YouTube channel:
https://www.youtube.com/channel/UCjiVhIvGmRZixSzupD0sS9Q
*/

//Libraries
#include <Wire.h>
#include <VL53L0X.h>        //Downlaod it here:
https://www.electronoobs.com/eng_arduino_Adafruit_VL53L0X.php


VL53L0X sensor;             //Define our sensor
//If you uncomment any of lines below you activate that mode
#define LONG_RANGE
#define HIGH_SPEED
//#define HIGH_ACCURACY



//Outputs/inputs
#define dirPin 3      //Pin for direction of the stepper driver
#define stepPin 4     //Pin for steps of the stepper driver
#define Enable 5      //Pin for enable the stepper driver

//Variables
int Value = 1200;               //Delay value between steps
float angle = 0;                //Start angle

/* ----------------Step angle calculation----------------
 * We need 1.5 rotations for 360°. (pully ratio 1.5 : 1)
 * Each 200 steps the motor will make a rotation.
 * We move 2 steps and the we make a measurement.
 * This equals to 360°/(200steps * 1.5) * 2 = 2.4angle/loop  ->
   ----------------Step angle calculation----------------*/
float angle_step = 2.4;         //So place that value here

float maxdist = 400; //I've set the maximum distance around the sensor to
only 400mm. Change to any other value.
bool loop_starts = false;
byte last_PIN_state;


void setup() {
  // Declare pins as output:
  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
  pinMode(Enable, OUTPUT);
  digitalWrite(Enable,LOW);     //Place enable to low so the driver is
enabeled
  digitalWrite(dirPin, HIGH);   //Place dirPin to HIGH so we spin CW
  Serial.begin(9600);           //Start serial port
  Wire.begin();
  sensor.init();
  sensor.setTimeout(500);
  PCICR |= (1 << PCIE0);    //enable PCMSK0 scan so we can use interrupts
  PCMSK0 |= (1 << PCINT0);  //Set pin "D8" trigger an interrupt on "any"
state change.
  //See interrupt vector below the void loop



  #if defined LONG_RANGE
    // lower the return signal rate limit (default is 0.25 MCPS)
    sensor.setSignalRateLimit(0.1);
    // increase laser pulse periods (defaults are 14 and 10 PCLKs)
    sensor.setVcselPulsePeriod(VL53L0X::VcselPeriodPreRange, 18);
    sensor.setVcselPulsePeriod(VL53L0X::VcselPeriodFinalRange, 14);
  #endif
  #if defined HIGH_SPEED
    // reduce timing budget to 20 ms (default is about 33 ms)
    sensor.setMeasurementTimingBudget(20000);
  #elif defined HIGH_ACCURACY
  //  // increase timing budget to 200 ms
  //  sensor.setMeasurementTimingBudget(200000);
  #endif
}//End setup loop



void loop() {
  if (loop_starts)             //We reset angle when the magnet is
detected on D8
  {
    angle = 0;
    loop_starts = false;
  }

  digitalWrite(stepPin, HIGH);                    //Make one step
  delayMicroseconds(Value);                       //Small delay
  digitalWrite(stepPin, LOW);                     //Make another step
  delayMicroseconds(Value);                       //Add another delay
  int r = sensor.readRangeSingleMillimeters();    //Get distance from
sensor
  if (r > maxdist)                                //Limit the dsitance to
maximum set distance above
  {
    r = maxdist;
  }
  Serial.print(angle);          //Print the values to serial port
  Serial.print(",");
  Serial.print(r);
  Serial.println(",");

  angle = angle + angle_step;   //Increase angle value by the angle/loop
value set above (in this case 2.4° each loop)
```

```
}//end of void loop




//This is the magnet detection interruption routine
//----------------------------------------------

ISR(PCINT0_vect){
  if(PINB & B00000001)          //We make an AND with the pin state
register, We verify if pin 8 is HIGH???
  {
    if(last_PIN_state == 0)
    {
      last_PIN_state = 1;
    }
  }
  else if(last_PIN_state == 1)  //Now verify if pin 8 is LOW??? -> Magnet
was detected
  {
    last_PIN_state = 0;
    loop_starts = true;         //If yes, we set loop_starts to true so
we reset the angle value
  }
}//End of ISR
```

For point cloud display, go to the official page (https://processing.org/download) and download Processing, a simple coding software, and utilize the code below. Connect the Arduino to the PC and see which COM you are using. Make sure you upload the Arduino code from above before running the processing code. The LIDAR will start rotating and plotting the distance to the screen. Note that the codes may need to be updated for your specific system build.

```
  // List all the available serial ports
  printArray(Serial.list());

  // Open the port you are using at the rate you want:
  myPort = new Serial(this, Serial.list()[0], 9600);

  size(820, 820);
  noSmooth();
  background(0);
  translate(410, 410);
  stroke(255);
  strokeWeight(3);  // Default
```

2. Deliverable

1. Use the Arduino Uno to build a circuit that both controls the motor and displays the surroundings.
   a. Display the TOF readings on the terminal.

    b. Display the TOF readings on the screen as a point cloud.
    c. Display the TOF readings on the screen as line graphs.
2. Submit a Word document that has all your results attached, including the circuit diagram for the sensor measurement setup, code, and links to the videos showing your results.

Reference

[1] Homemade LIDAR Sensor with Arduino & Processing by Electronoobs: https://youtu.be/fQ2iB7qkrUg

[2] TOF Sensor Spec Sheet: https://www.st.com/en/imaging-and-photonics-solutions/vl53l0x.html

[3] What is LIDAR: https://www.synopsys.com/glossary/what-is-lidar.html#:~:text=LiDAR%20is%20an%20acronym%20for,the%20objects%20in%20the%20scene.

[4] The Basics of LIDAR: https://www.neonscience.org/resources/learning-hub/tutorials/lidar-basics

LAB 6: SIGNAL PROCESSING USING MATLAB

Objective

In this lab, students will utilize MATLAB to process digital signals.

Introduction

1. Why Signal Processing?

Mechanical engineers rely on sensor data to monitor, evaluate, and control systems. From vibration analysis in machinery to temperature control in engines, understanding and processing sensor signals are critical. Signal processing involves analyzing, modifying, and synthesizing signals such as sound, images, and sensor outputs. Effective signal processing can reveal hidden information, reduce noise, and improve system performance.

2. MATLAB as a Signal Processing Tool

MATLAB is a widely used platform for engineering and scientific computations. It offers a comprehensive suite of functions for signal processing, allowing you to:

- **Visualize Data**: Plot and analyze sensor data in various formats
- **Filter Signals**: Reduce noise and extract meaningful information
- **Transform Signals**: Use techniques like Fourier transforms to examine frequency components
- **Analyze Time-Series Data**: Understand how signals change over time
- **Model Systems**: Create mathematical models for simulation and control

3. Key Topics Covered in This Lab Manual

In this lab, you will learn the following concepts related to signal processing:

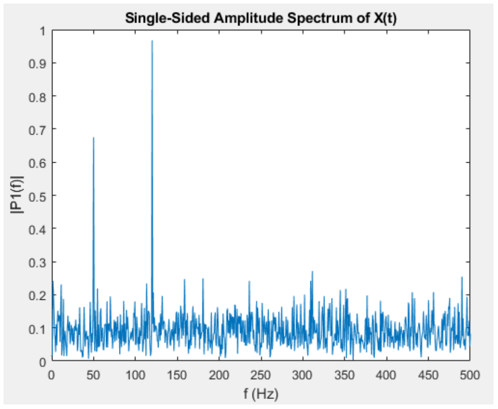
- **Basic Signal Operations:** Understanding signal representations, sampling, and digital signal basics
- **Time-Domain Analysis:** Techniques for analyzing signals in the time domain, including moving averages and simple filters
- **Frequency-Domain Analysis:** Using Fourier transforms to examine signal frequency content
- **Noise Reduction:** Applying filters to reduce noise and enhance signal quality
- **Sensor Data Interpretation:** Applying signal processing techniques to real-world sensor data in a mechanical engineering context

In-person lab assignment

1. Signal Denoising

The following MATLAB code (provided in the previous homework) shows waveform (S) containing a 50 Hz sinusoid of amplitude 0.7 and a 120 Hz sinusoid of amplitude 1, which is sampled at 1000 Hz with a duration of 1.5 seconds. The signal is corrupted by zero-mean white noise with a variance of 4. The spectral response is shown in the figure below:

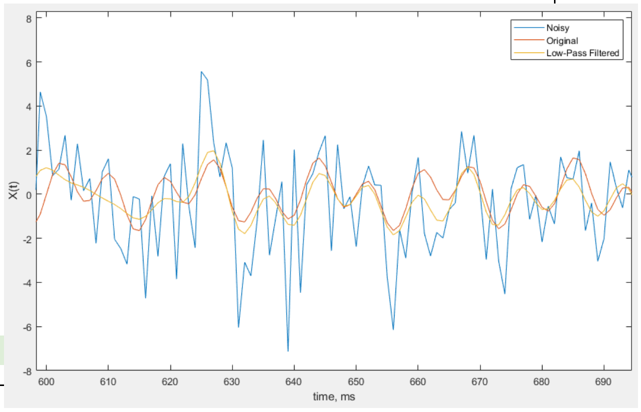
```
clear all;
close all;
clc;

% settings
Fs = 1000;     % Sampling frequency
dt = 1/Fs;     % Sampling period
n = 1500;      % Length of signal
t = (0:n-1)*dt; % Time vector
L = n/2+1;  % Length of the freq to plot (only the first half)

% create a signal
S = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t); % Original signal
X = S + 2*randn(size(t)); % contaminated signal with noise (st. dev. = 2)

% test plot
figure;              % open a blank fig
plot(1000*t,S);      % plot original (all data points)
% test plot
figure;              % open a blank fig
plot(1000*t,X);      % plot the noisy on top of the original
hold on;             % hold
plot(1000*t,S);      % plot original (just the first 50 points)
title("Signal Corrupted with Zero-Mean Random Noise")
xlabel("t, ms")
ylabel("X(t)")

% run the fft
Y = fft(X);
P2 = abs(Y/n);
P1 = P2(1:L);
P1(2:end-1) = 2*P1(2:end-1);

% define the frequency
f = (Fs/n)*(0:(n/2)); % define your freq.
figure;
plot(f,P1)
title("Single-Sided Amplitude Spectrum of X(t)")
xlabel("f (Hz)"); ylabel("|P1(f)|");
```



1.1. Signal Denoising by Low-Pass Filtering

When the original noisy signal is low-pass filtered in a brutal way, by manually cutting off the high frequency content from the original signal in the frequency domain and reconstructing the signal by using *ifft* command, the resulting signal is compared with the original signals in the following figure.

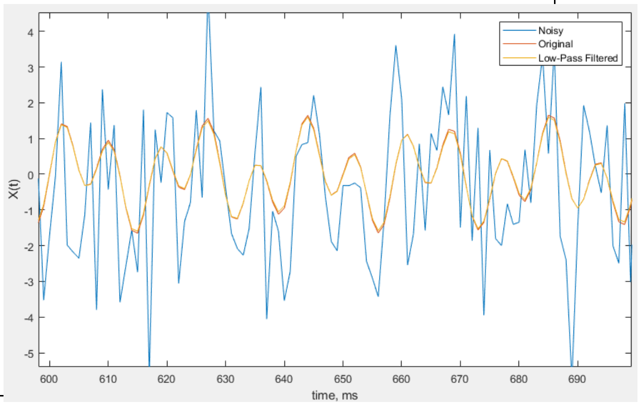
```
%%  low-pass filter : remove everything above f_cut Hz
f_cut = 150;

% find the indices of the frequency vector that corresponds to frequencies
% that are below f_cut
ind = find(f<f_cut);

% set the new freq and spectral arrays.
f_new = f(ind);
Y_new = Y(ind);

figure;
plot(f_new,abs(Y_new));
title("Single-Sided Amplitude Spectrum of X(t)")
xlabel("f (Hz)"); ylabel("|P1(f)|");

% inverse fft to reconstruct the denoised signal
X_filtered = real(ifft(Y_new,n));

figure;
plot(1000*t, X); hold on;
plot(1000*t, S);
plot(1000*t, X_filtered);
xlabel('time, ms'); ylabel("X(t)")
legend('Noisy','Original','Low-Pass Filtered')
```



1.2. Signal Denoising by Thresholding.

The other case is to denoise the signal by manually suppressing the noise floor to zero. The major frequencies of the signals are first identified. The spectral contents of other frequencies that contain spectral amplitude lower than a threshold are set to zero.

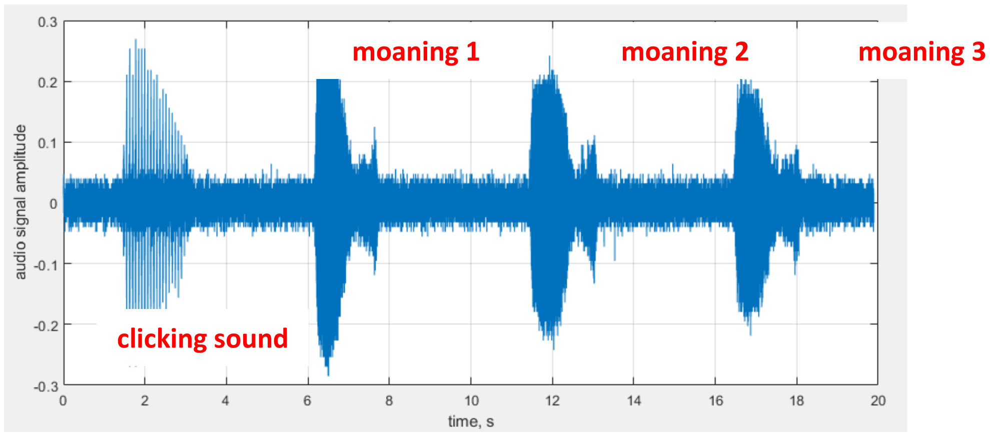
```
%% step 2 : remove small amplitude fft components
trsd = 0.4;

% find the indices of the spectral amplitude vector that are below the
% threshold
ind2 = find(P2 < trsd/2);
Y2 = Y;         % assign Y to Y2
Y2(ind2) = 0;   % set the spectral content zero

figure;
plot(f,abs(Y2(1:L)));
title("Single-Sided Amplitude Spectrum of X(t)")
xlabel("f (Hz)"); ylabel("|P1(f)|");

% inverse fft signal reconstruction
X_filtered2 = ifft(Y2);

figure;
plot(1000*t, X); hold on;
plot(1000*t, S);
plot(1000*t, X_filtered2);
xlabel('time, ms'); ylabel("X(t)")
legend('Noisy','Original','Low-Pass Filtered')
```



2. Signal Processing of Whale Songs

Load the attached audio file that contains audio data from a Pacific blue whale, sampled at 4 kHz. The file is from the library of animal vocalizations maintained by the Cornell University Bioacoustics Research Program. The time scale in the data is compressed by a factor of 10 to raise the pitch and make the calls more audible.

Play the sound using an audio player and listen to the sound. There is one clicking sound and three whale sounds. The following example code shows how to (1) load the audio data, (2) crop the audio data and time of interest (the first moaning sound in this example), and (3) plot the results.



```
clear all
close all
clc

% audio data from a Pacific blue whale, sampled at 4 kHz
% by the Cornell University Bioacoustics Research Program

% read the audio file
whaleFile = 'bluewhale.au';
% x0 is the original time-domain data and fs is the sampling frequency
[x0,Fs] = audioread(whaleFile);

dt = 1/Fs;                         % time interval of the signal
t0 = (0:dt:(length(x0)-1)*dt);  % original time

% plot the overall signal
figure;
plot(t0,x0);
xlabel('time, s'); ylabel('audio signal amplitude')
grid on;

%% crop the signal of interest
whaleMoan1 = x0(2.45e4:3.10e4);     % crops from index no. 2.45e4 to 3.1e4
t1 = (0:1/Fs:(length(whaleMoan1)-1)/Fs); % defines a new time for the cropped data
n = length(whaleMoan1);
L = n/2+1;

figure;
plot(t1,whaleMoan1);
xlabel('Time (seconds)'); ylabel('Amplitude');
xlim([0 t1(end)])
```

3. Deliverable

3.1. From Part 1. Signal Denoising

(a) Run the above code and understand how it works. Show its results.

(b) Low-pass filter the above noisy signal at 75 Hz. Show the denoised time-domain signal and discuss.

(c) Threshold the spectral amplitude at 0.15. Show the denoised time-domain signal and discuss.

3.2. From Part 2. Signal Processing of Whale Songs

(a) Plot the **overall time-domain signal** to observe the waveforms.

(b) Plot the **power spectrum** of the signals for each signal.

(c) For **each signal** (one click and three moaning),

- Find the fundamental frequency.
- Low-pass filter at 200 Hz and reconstruct the waveform, then save as a **sound file** (*.wav file). **Plot the filtered waveform and its spectral response** obtained by FFT. Listen to the sound.

  Use MATLAB low-pass filter command:

  https://www.mathworks.com/help/signal/ref/lowpass.html
- High-pass filter at 500 Hz and reconstruct the waveform, then save as a **sound file** (*.wav file). **Plot the filtered waveform and its spectral response** obtained by FFT. Listen to the sound.

  Use MATLAB low-pass filter command:

  https://www.mathworks.com/help/signal/ref/highpass.html

(d) Discuss your findings.

Deliverable submission: Submit a Word document that has all your results attached, MATLAB code (*.m file), and audio files created.

LAB 7: LIDAR MAPPING ROBOT

Objective

In this lab, students will construct a LIDAR-capable robot that simultaneously maps the surrounding environment while driving.

Introduction

1. The Basic Principles of LIDAR

As covered in the previous lab, Light Detection and Ranging (LIDAR) is a method of scanning physical environments and objects using pulses of light. The time-of-flight (TOF), time between an outgoing pulse and detecting the reflection of that pulse, allows the distance to be calculated. By sending and receiving thousands of light pulses per second, the system generates high-resolution 3D information which proves useful in mapping large complex areas like the topology of the earth. LIDAR also has a foothold in the autonomous driving industry given its ability to accurately measure and map complex and dynamic environments. A typical scanner-less LIDAR sensor consists of a TOF sensor, a GPS for positional accuracy, a data processing unit, and certain mechanical components that serve to rotate the sensor like a step motor and slip-ring. Rotating the LIDAR sensor continuously allows the system to collect ranges in 360* around the entire robot, not just in a static direction.

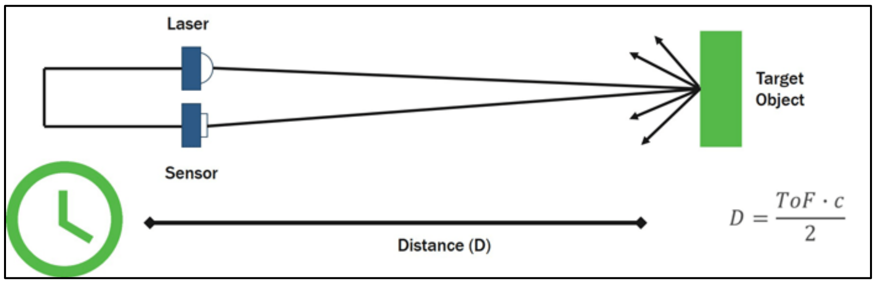


Figure 1. Calculating Distance from Reflected Light

2. An On-board LIDAR Module

The objective of this lab is to create a robot that can simultaneously map its surroundings while navigating. This is achieved through an on-board LIDAR module mounted to the top of the robot. As discussed previously, the module is composed of a TOF sensor spun by a step motor and connected to the Arduino via a slip-ring electrical connector. Students can utilize the LIDAR module and DC motor controller developed from previous labs to build robots that can navigate and scan the surroundings. The chassis of the robot is driven by two DC motors, which interface with the Arduino Uno via the L298 DC motor driver as discussed in Lab 2. The LIDAR module, as previously constructed, is mounted to the top of the robot chassis to ensure the TOF sensor isn't obstructed and has a clear view of the surroundings. This sensor will be able to provide real-time data to the serial monitor which can be used to generate a point-cloud map of the robot environment as it drives. After completing the full robot assembly, the Arduino microcontroller needs to be programmed. Figure 2 shows an example of a completed robot.

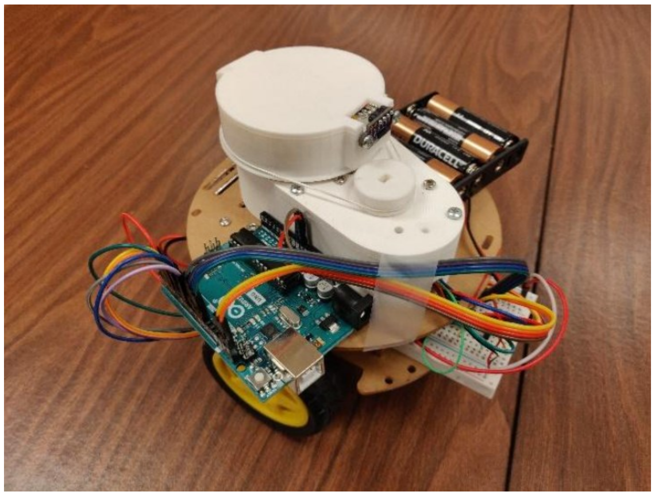


Figure 2 - Example of Completed Robot Assembly

Deliverables:

1. Assemble the robot and code a basic operation program capable of driving, turning, and stopping.
2. Using the data received from the TOF sensor, generate a point-cloud map of the surrounding area.

References:

[1] Stepper Motors with Arduino: https://www.youtube.com/watch?v=0qwrnUeSpYQ

[2] Motor Manual: https://lastminuteengineers.com/a4988-stepper-motor-driver-arduino-tutorial/

[3] Stepper Motor Controller Tutorial Article: https://www.makerguides.com/a4988-stepper-motor-driver-arduino-tutorial/

[4] A4988 Controller Datasheet: https://www.makerguides.com/wp-content/uploads/2019/02/A4988-Datasheet.pdf

[5] TinkerCAD Tutorial Video: https://www.youtube.com/watch?v=yVDuo4e6K0I

[6] Hall-effect Sensor Introduction PDF: https://www.ti.com/lit/po/slyt824a/slyt824a.pdf?ts=1712834125239#:~:text=What%20are%20Hall%2DEffect%20Sensors,accuracy%2C%20consistency%2C%20and%20reliability.

[7] Electronoobs LIDAR Module Tutorial: https://electronoobs.io/tutorial/48#

LAB 8: CONTROLS CASE STUDY – SELF-BALANCING ROBOT

Objective

Understand the fundamentals of PID controls with an Arduino UNO.

Introduction

1. Inertial measurement unit

One integral device for this laboratory experiment is the inertial measurement unit (IMU). An IMU consists of gyroscopes and accelerometers that measure and report the angular rate and specific force/acceleration of an object. Occasionally an IMU can include a magnetometer to measure the magnetic field of the surrounding system.

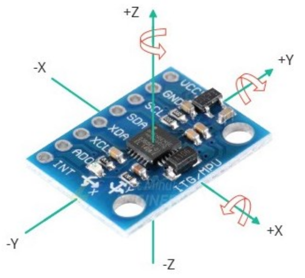


Figure 1. MPU6050 [2]

The MPU6050 accelerometer and gyroscope sensor is the specific IMU used in this lab. It utilizes a micro-electromechanical system (MEMS) gyroscope and in conjunction with the MEMS accelerometer allows the measurement of rotation along all three axes, static acceleration due to gravity, and dynamic acceleration due to motion. The MEMS accelerometer consists of a micro-machined structure on top of a silicon wafer. As deflection occurs, the movement of suspended plates between fixed plates changes the capacitance between them as well, being proportional to the acceleration along the corresponding axis as displayed in Fig. 2.

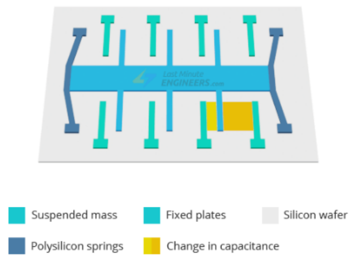


Figure 2. Diagram of MEMS Accelerometer [2]

The MEMS gyroscope utilizes the Coriolis Effect to measure the rotation of the object. The Coriolis Effect states that when a mass moves in a specific direction with a velocity and an external angular rate is applied, the Coriolis Effect generates a force that causes the mass to move perpendicularly, with its displacement directly related to the angular rate applied. To take advantage of this effect, the MEMS gyroscope consists of a proof mass made by four separate parts, shown in Fig. 3, whose movement and rotation allows the detection of the Coriolis effect. When the effect is detected, the constant motion of the driving mass causing a change in capacitance is converted into a voltage signal.

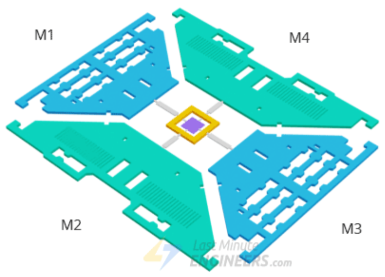


Figure 3. MEMS Gyroscope Proof Mass [2]

The MPU6050 contains 8 pinouts that mainly connect to the power supply, ground, and Inter-Integrated Circuit (I^2C).

1. VCC: Module power supply
2. GND: Ground
3. SCL: Serial Clock Pin for I2C Interface
4. SDA: Serial Data Pin for I2C Interface
5. XDA: External I2C Data Line
6. XCL: External I2C Clock Line
7. AD0: Allows Change of I2C Address
8. INT: Interrupt Output Pin

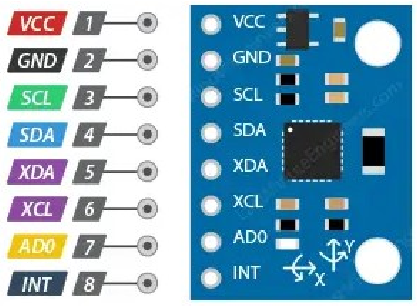


Figure 4. MPU6050 Pinout [2]

2. Proportional-Integral-Derivative (PID) Controllers

A PID (Proportional-Integral-Derivative) controller is used to regulate and process variables in a control loop feedback mechanism to achieve the best desired outcome. PID controllers can find use in nearly every process control application and are commonly used in the heat treatment of metals, drying/evaporating of solvents from painted surfaces, curing of rubber, and baking. Most of these processes require precise temperature control and regulation.

Sample Setup

The self-balancing robot aimed at will essentially be an inverted pendulum where the oscillating motion is above the fixed point. The robot should be able to balance much better if its center of mass is higher relative to the wheel axles.

One way to wire and code a self-balancing robot like the example in Figure 5, these wiring setups and code from the YouTube video [4] can be inspired by.

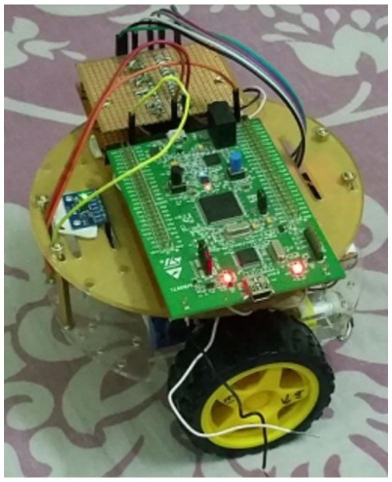


Figure 5. Example Robot

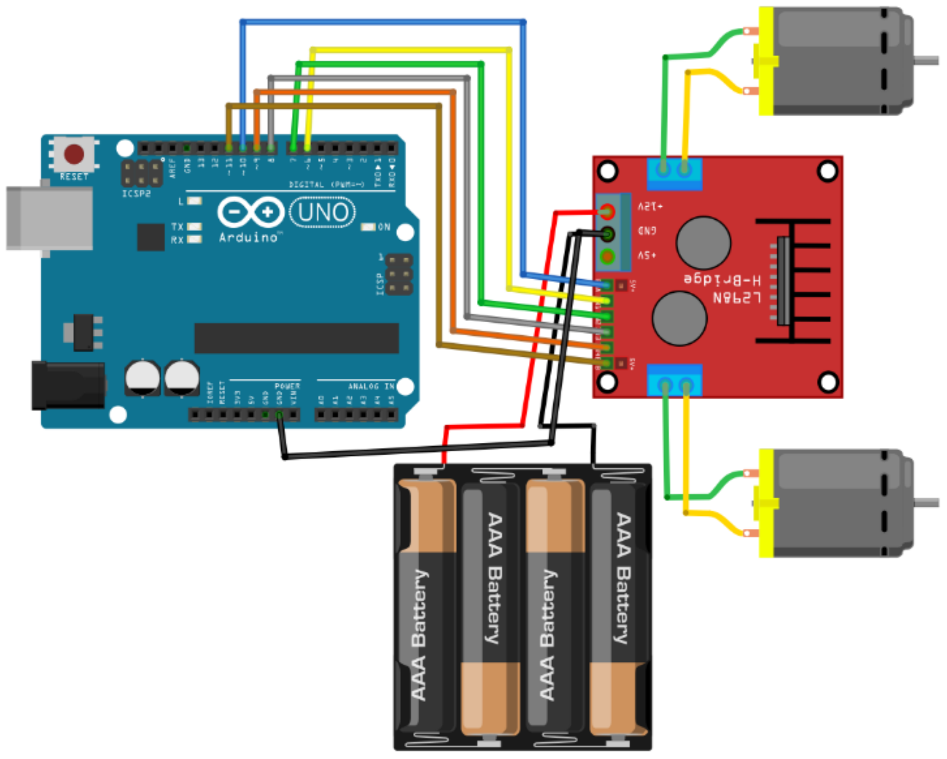


Figure 6. Wiring Setup of Arduino to L298 Motor Driver & 12V Battery Pack

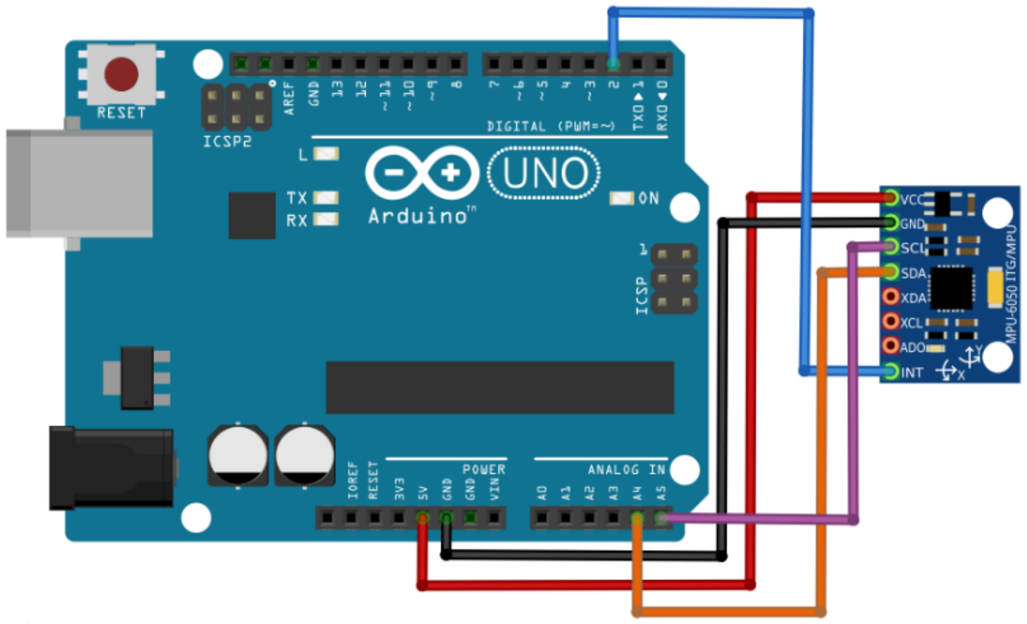


Figure 7. Wiring Setup for Arduino with MPU6050

In-person lab assignment

1. Deliverables

Use the Arduino Uno to build a circuit that reads the input from an IMU and controls the motors to balance the robot.

- Display the IMU readings on the terminal to check
- Record at least three videos that show the robot behavior depending on different PID gains, including the optimal values that show desirable balancing performance.

Submit a Word document that has all your results attached, including the circuit diagram for the robot setup, robot images, code, and links to the videos showing your results.

References

[1] IMU Information - https://www.vectornav.com/resources/inertial-navigation-articles/what-is-an-inertial-measurement-unit-imu

[2] https://lastminuteengineers.com/mpu6050-accel-gyro-arduino-tutorial/

[3] PID Information - https://www.omega.com/en-us/resources/pid-controllers

[4] Tutorial - https://www.youtube.com/watch?v=CON0sWNDUco

[5] Link from YT video with Arduino sketch and brief explanation - https://github.com/makertut/balance-robot