School of Engineering, Discipline of Mechanics and Mechatronics, pp. 1–18

Lab 2: Motor Control with the Arduino Platform

ENGG1500

Semester 1 2020

Nicholas O'Dell, Brenton Schulz, Dylan Cuskelly.

---

1. Introduction

In this lab you will connect your motors and Arduino to the motor driver and write basic motor control code. As with Lab 1 the Arduino Sensor Shield, Figure 1, will continue to be used so that any assembly performed in this lab can be kept intact throughout the project.

This lab assumes that your motors, Arduino and motor shield are mounted to your vehicle's chassis (as shown in the assembly video).

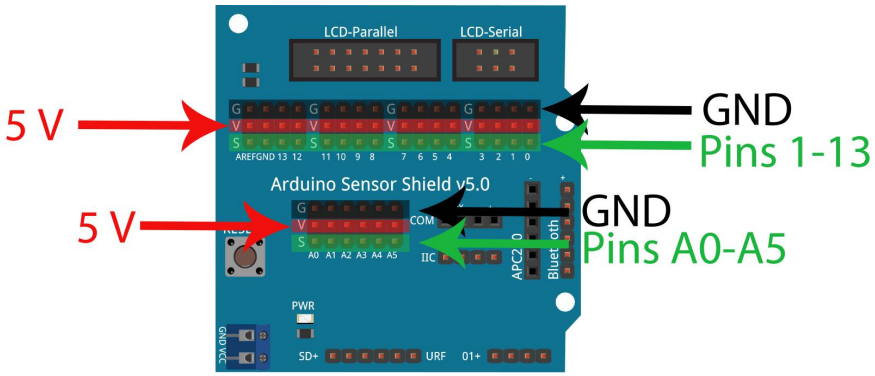

Figure 1: Arduino sensor shield showing the **GND, 5V** and **I/O pins**.

2. Motor Wiring

In order to allow the Arduino to control higher voltages and large electric currents required by the motor, an interfacing circuit known as a motor driver is required. In your kit an L298N H-bridge motor driver, shown in Figure 2, is provided for this purpose.

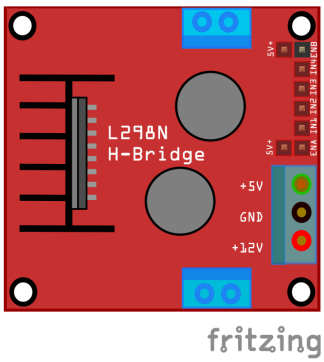

Figure 2: L298N dual motor driver.

The motor driver contains screw terminals to which high current wiring is connected and a collection of pin headers which forms the digital control interface to the Arduino. The battery will be connected to the three-pin screw terminal block and the motors connect to the two-pin screw terminal blocks. The L298N is the large component bolted to the black heat sink.

Task: Perform the following steps to connect the motor driver;

1) Retrieve the small screwdriver and the DC power cable cable from the kit box.
2) Connect the 2 and 3 pin connectors to the power distribution board, these should only connect one way, as shown in Figure 3.
3) As seen in Figure 3, connect the **yellow** wire from the 3 pin wire to the **12 V** screw terminal on the motor driver. Start by loosening the screw terminal, then insert the wire, and re-tighten the screw terminal. Do the same for the: **black** wire and the **GND** screw terminal, and the **red** wire and the **+5 V** screw terminal.

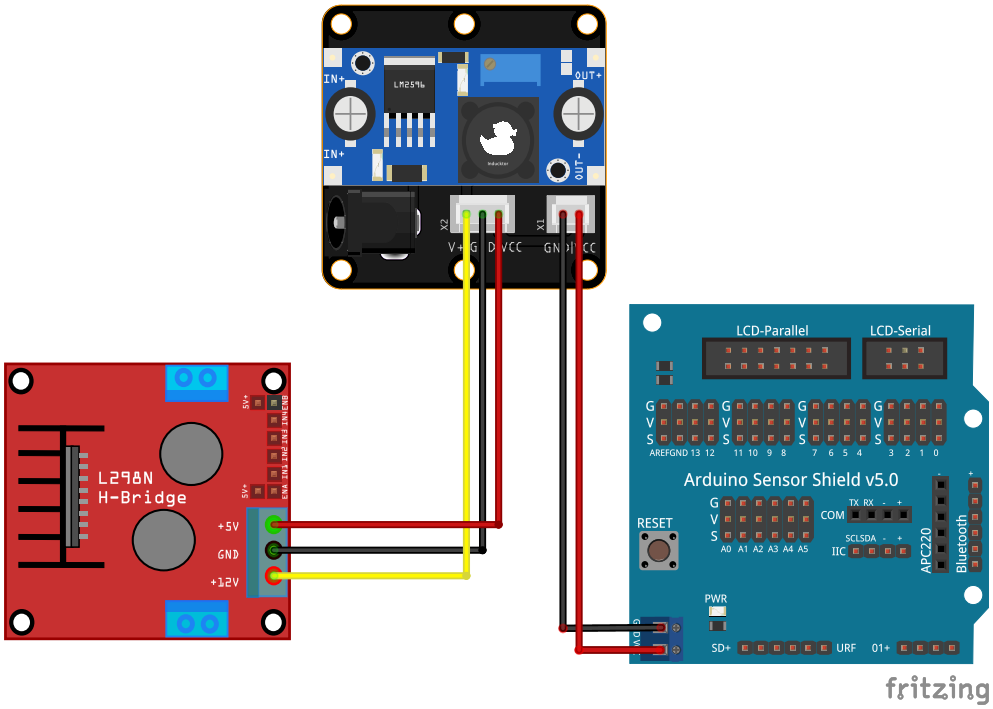

Figure 3: How to connect power to the motor driver

4) As seen in Figure 3, connect the **black** wire from the 2 pin connector with the **GND** screw terminal on the sensor shield. Also connect the **red** wire from the same connector with the **VCC** screw terminal on the sensor shield.
5) Your power connections should resemble Figure 4. Ask a demonstrator to check the wiring before plugging in the battery or Arduino USB cable.
6) The digital logic states of the **IN1-IN4** pins control the motor directions as per Table 1. The pins **ENA** and **ENB** enable each output. If the enable pin is logic 1, a voltage, of polarity defined by the **IN** pins, becomes present on its corresponding output screw terminal.
   a) Connect **IN1** on the motor driver to pin **8** of the Arduino (green wire, Figure 5).
   b) Connect **IN2** on the motor driver to pin **9** of the Arduino (green wire, Figure 5).
   c) Connect **IN3** on the motor driver to pin **10** of the Arduino (yellow wire, Figure 5).

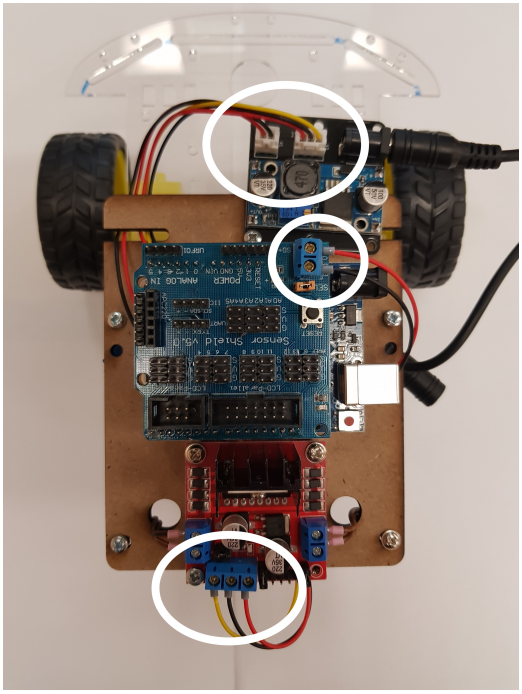

Figure 4: Power connection.

d) Connect **IN4** on the motor driver to pin **11** of the Arduino (yellow wire, Figure 5).

e) Disconnect the pin-jumpers[1] connected to **ENA**, **ENB**, these are not needed. Ensure the one sitting behind the power connector is left in place, this enables the 5 V regulator, which we are using to power the Arduino.

f) Connect **ENA** on the motor driver to pin **5** of the Arduino.

g) Connect **ENB** on the motor driver to pin **6** of the Arduino.

h) Connect your left motor to the **OUT1** and **OUT2** screw terminals and your right motor to the **OUT3** and **OUT4** screw terminals. These connections will provide power to the motors. Wiring the motors 'around the wrong way' will result in the motors spinning the opposite direction. The direction they spin depends on the orientation they were mounted to the chassis, we will correct this later in the lab.

Note that all the wires you have just inserted into screw terminals have been terminated with bootlace crimps[2] to prevent fraying and to thicken small wires, this is good practice to achieve a good connection in a screw terminal when drawing high current.

You should now have a circuit that looks like Figure 5.

[1] These are small black connectors on ENA and ENB

[2] This could also be achieved by applying a small amount of solder to the ends of the wire.

Table 1: L298N Logic table

IN1	IN2	The State of DC Motor A
0	0	OFF
0	1	Rotate Clockwise
1	0	Rotate Counterclockwise
1	1	OFF
IN3	IN4	The State of DC Motor B
0	0	OFF
0	1	Rotate Clockwise
1	0	Rotate Counterclockwise
1	1	OFF

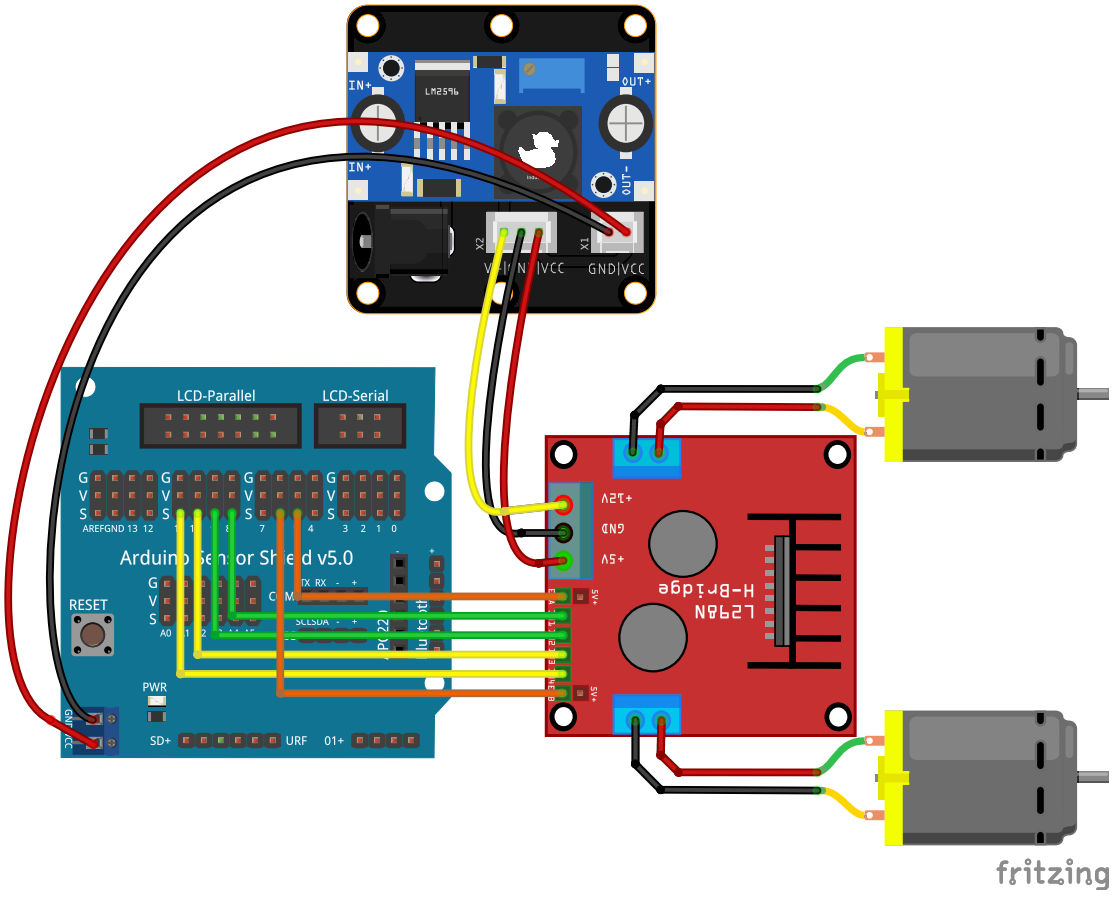

Figure 5: How to connect the motor driver to the Arduino

Warning

Use the cable ties in your kit to tie the motor wires to the chassis. This will prevent breaking the brittle tabs on the motors if the wires get snagged.

3. Motor Control

We are now capable of controlling our motors with the Arduino using a combination of digital output and *pulse width modulation* (PWM). Using the `digitalWrite()` on digital pins 8-ll we can control each motor's direction independently, and by using `analogWrite()` on digital pins 5 and 6 we can control their speed.

3.1. Arduino PWM Output

Pulse width modulation is a method for generating an arbitrary DC voltage from a digital output. It works by generating pulses at a fixed frequency (≈1000 Hz on the Arduino) while varying the pulse width. When this signal is used to drive a DC motor the result is similar to powering the DC motor with the average voltage of the PWM signal, as shown in Figure 6.

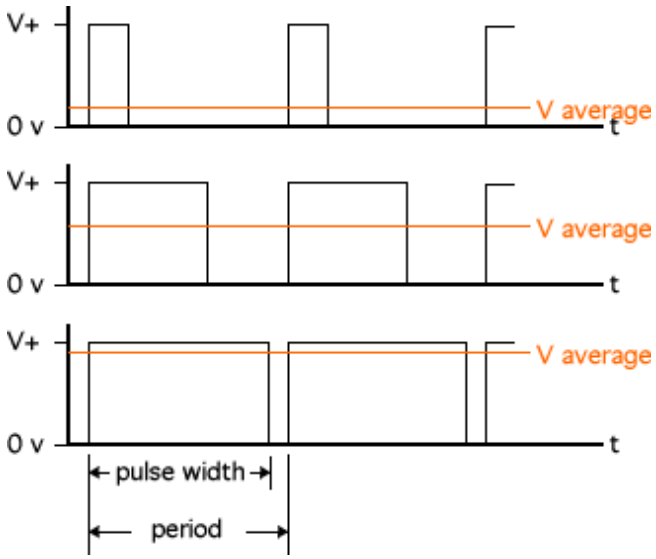

Figure 6: PWM graphic, showing average voltage as a function of duty cycle.

The Arduino function `analogWrite()` controls the PWM duty cylce on a given pin, this function takes two arguments;

- **Pin**: On the Arduino UNO only 6 pins are able to be controlled with `analogWrite`, they are marked with a '∼' (pins 3, 5, 6, 9, 10, and 11). You connected **ENA** and **ENB** to pins 5 and 6 for this reason.
- **Value**: An integer in the range: $[0, 255]$[3] is used to control the output between always off and always on. I.e., to control the output at 50 percent you would use 127, since it is half of 255. Due to static friction in the motor's gearbox the motor will not rotate unless a PWM value above a certain threshold is used, ≈ 50 in our case. This value will change based on the quality and wear of the motors.

Example: To control the output of pin 5 at half of full voltage the following line of code would be used:

[3] This is a default, it is assosiated with an 8 bit timer, i.e., $2^8 = 256$

```
analogWrite(5,127);
```

For further information on this function go to the Arduino reference page, https://www.arduino.cc/en/Reference/analogWrite.

4. Motor Control in Arduino

Task: Achieve a moving vehicle.

1) Ensure the battery is switched off.
2) Begin with the code in Listing 1, available on Blackboard.
3) Upload the sketch to your Arduino, make sure that the battery is switched off. Unplug your Arduino from the computer.
4) After the code has been loaded onto the Arduino place the Arduino on the ground and turn the battery on.
5) If both of your motors do not travel forwards, swap the wires in the screw terminals of the motor that is going backwards. (The castor wheel is attached to the rear of the vehicle).
6) Complete the **TO DO** statements in the code:
   i) Set both motor directions to backwards.
   ii) Set the speed of both motors to full speed.
   iii) Insert a 2.5 second (2500 ms) delay.

   The code below has four functions that have been declared within the file, until now we have only used functions that are built into the Arduino. A function performs a predefined task and if something is performed multiple times it is good practice and much more efficient to put it in a function.

   In order to set the left motor direction to forward, **IN1** needs to be set `HIGH` and **IN2** needs to be set `LOW`. Therefore the function `leftForwards()` sets these pins appropriately and the other functions perform similar tasks. To see how to declare your own functions for your project go to https://www.arduino.cc/en/Reference/FunctionDeclaration.

Listing 1: Motor Control

```
//START
void setup() {
  // put your setup code here, to run once:
  pinMode(5,OUTPUT); //set ENA as an output
  pinMode(6,OUTPUT); //set ENB as an output
  pinMode(8,OUTPUT); //set IN1 as an output
  pinMode(9,OUTPUT); //set IN2 as an output
  pinMode(10,OUTPUT); //set IN3 as an output
  pinMode(11,OUTPUT); //set IN4 as an output
  delay(5000);//delay before starting loop
}
```

```
void loop() {
  // put your main code here, to run repeatedly:
  leftForwards();       //set the left motor to run forwards by calling the
       //...function leftForwards, listed below
  rightForwards();      //set the right motor to run forwards by calling the
       //...function rightForwards,listed below
  analogWrite(5,127);   //set the left motor to half speed (ish)
  analogWrite(6,127);   //set the right motor to half speed (ish)
  delay(5000);  //allow the motors to run forwards for 5000 ms (5 seconds)
  analogWrite(5,0);     //stop the left motor
  analogWrite(6,0);     //stop the right motor
  delay(5000);  //stop the motors for 5 seconds

  //TODO: set both motor directions to backwards
  //TODO: set the speed of both motors to full speed
  //TODO: insert a delay to allow the motors to run backwards for 2.5 seconds
       //...before this loop repeats
}

void leftForwards(void) //This function sets IN1 = LOW and IN2 = HIGH in order to
     //... set the direction to forwards for motor 1
{
  digitalWrite(8,LOW); //IN1
  digitalWrite(9,HIGH); //IN2
}

void leftBackwards(void) //This function sets IN1 = HIGH and IN2 = LOW in order
     //...to set the direction to backwards for motor 1
{
  digitalWrite(8,HIGH); //IN1
  digitalWrite(9,LOW); //IN2
}

void rightForwards(void)  //This function sets IN3 = LOW and IN4 = HIGH in order
     //...to set the direction to forwards for motor 2
{
  digitalWrite(10,LOW); //IN3
  digitalWrite(11,HIGH); //IN4
}
void rightBackwards(void) //This function sets IN3 = HIGH and IN4 = LOW in order
     //...to set the direction to forwards for motor 2
{
  digitalWrite(10,HIGH); //IN3
  digitalWrite(11,LOW); //IN4
}
//END
```

7) Your vehicle should now be moving forward at half pace for 5 seconds stopping for 5 seconds and reversing for 2.5 seconds at full speed. Take note that it does not end up where it started, this is the result of controlling the motors without any sensor feedback.

5. React to obstacle

Task: React to an obstacle by stopping when one appears.

To achieve this task we need the code to perform a decision. The following *UML*[4] diagram and *pseudo code* documents the algorithm which must be evaluated by the Arduino code.

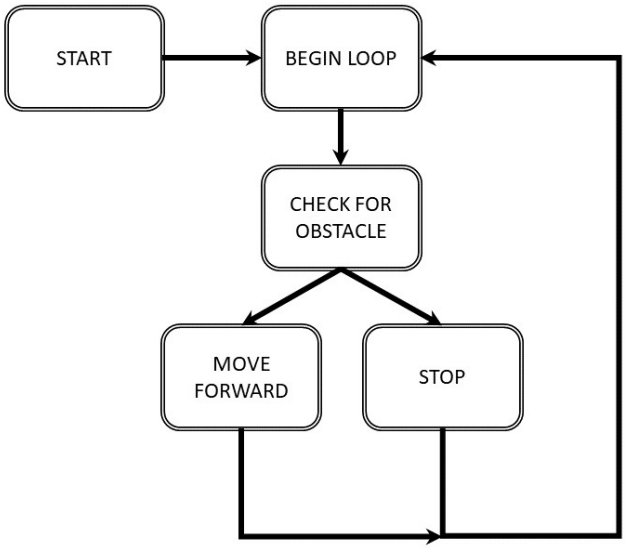

Figure 7: Obstacle avoidance UML graphic.

Listing 2: Obstacle avoidance algorithm.

```
BEGIN
  BEGIN LOOP
    IF an object is detected
      Stop
    Else
      Move forward
    END LOOP
END
```

The following Arduino code shows one implementation of the algorithm above. It uses `if-else` statements which are covered formally in Lab 3.

Listing 3: Obstacle avoidance using flow control.

```
int motorPin = 5;
int motorSpeed=0;
int sensorVal = digitalRead(A0); //1 if no obstacle is present otherwise 0
if(sensorVal == 0) //if an obstacle is present
{
  motorSpeed = 0; //This will be executed only if an obstacle is present
}else
{
  motorSpeed = 200; //Otherwise this statement will be executed instead
}
```

[4]Unified Modelling Lanuage, https://en.wikipedia.org/wiki/Unified_Modeling_Language

```
analogWrite(motorPin,motorSpeed);
```

An alternative implementation can use arithmetic instead of `if-else` *flow control*. Remember that the `digitalRead()` function returns either 0 or 1, which is just an integer. This integer value can be used as part of arithmetic expressions. In the following code extract the variable `sensorVal` becomes either 0 or 1, and is multiplied by a PWM constant to either switch on or off the motor depending on the logic state of pin A0.

Listing 4: Obstacle avoidance using arithmetic.

```
//Let's say the sensor is on pin 12, and the PWM pin we are using is pin 5

int sensorVal = digitalRead(12);    //1 if no obstacle is present otherwise 0

//let's say our desired speed is 200.
//multiply the previous by the desired speed (* is multiply)
int motorSpeed = sensorVal * 200;
analogWrite(5,motorSpeed);      //write signal to motor
```

Perform the following steps to complete this task;

1) Attach a line sensor as shown in Figure 8, to the front of your vehicle with a 25 mm standoff.

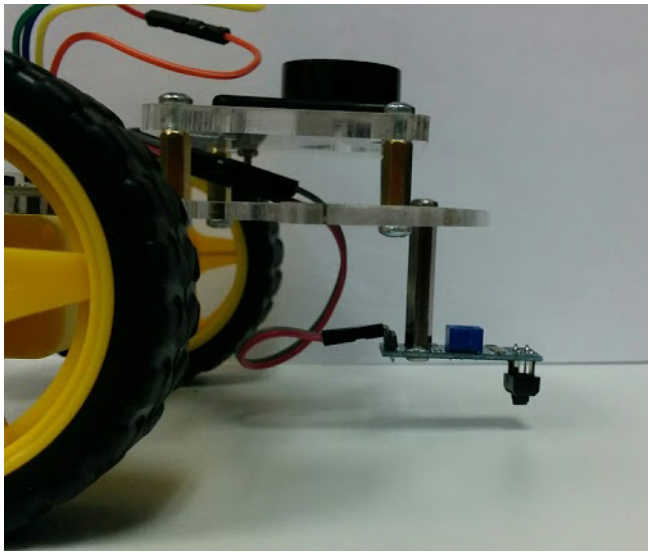

Figure 8: Line sensor mounted on the front of the robot.

2) Upload the following code, available in the lab template on Blackboard.

```
//START
void setup() {
  // put your setup code here, to run once:
  pinMode(5,OUTPUT); //set ENA as an output
  pinMode(6,OUTPUT); //set ENB as an output
  pinMode(8,OUTPUT); //set IN1 as an output
  pinMode(9,OUTPUT); //set IN2 as an output
  pinMode(10,OUTPUT); //set IN3 as an output
```

```
  pinMode(11,OUTPUT); //set IN4 as an output
  delay(5000);//delay before starting loop
}

void loop() {
  // put your main code here, to run repeatedly:
  leftForwards();  //set the left motor to run forwards
  rightForwards();  //set the right motor to run forwards
  int sensorVal = digitalRead(12);    //1 if no obstacle is present otherwise 0

  //let's say our desired speeds are 255 and 180
  //multiply the previous by the desired speed (* is multiply)
  analogWrite(5,sensorVal*255); //circle
  analogWrite(6,sensorVal*180); //circle
}

void leftForwards(void)
{
  digitalWrite(8,LOW);
  digitalWrite(9,HIGH);
}

void leftBackwards(void)
{
  digitalWrite(8,HIGH);
  digitalWrite(9,LOW);
}

void rightForwards(void)
{
  digitalWrite(10,LOW);
  digitalWrite(11,HIGH);
}
void rightBackwards(void)
{
  digitalWrite(10,HIGH);
  digitalWrite(11,LOW);
}
//END
```

3) The vehicle should be going in a circle and stopping when it runs over a reflective surface, E.g., paper, adjust the program so that the robot spins on the spot instead of going in a circle.

   i) Set one motor direction to forwards and one to backwards, by calling either `rightBackwards`, `leftBackwards` in the pace of its forward counterpart.

   ii) Set both the PWM levels to be equal.

You now have a robot that can sense and respond to its environment, you have completed the major building block to a functioning project.

6. Using the ENGG1500 Encoder library

Task: Use the ENGG1500 *encoder* library, which is available to download from Blackboard, to make your robot drive forwards and then backwards to the same location.

It is difficult to estimate position based on the motor run time as we did earlier. As such rotary encoder hardware is provided with the kit and is interfaced with the *encoder* library. The encoder library provides the functions `enc_getLeft()` and `enc_getRight()` which return the number of encoder counts since they were last reset with the encoder reset function (see Appendix A or the Project Reference Manual for the full documentation).

The photo-interrupter printed circuit board (PCB) supplied with your kit has been designed and made by the university and bolts to the chassis with M3 bolt/s and sits over the slotted disks on the motor shafts. The PCB contains two photo-interrupters which protrude through the chassis over the encoder wheels. On one side of the slotted disks is an IR transmitter and on the other side is an IR receiver[5].

The *encoder* library assumes that the sensor on the left wheel (`ENC_L`) is connected to **pin 2** and the sensor on the right wheel (`ENC_R`) is connected to **pin 3** of the Arduino.

The encoder PCB contains 3 LEDs attached to: power, `ENC_L`, and `ENC_R` to assist in hardware and software debugging[6].

Create a new sketch and complete the following:

1) Attach the encoder module to your robot as shown in Figures 9 and 10. Be careful that the encoder wheels are not rubbing on the sensors, there should be a small clearance on either side.

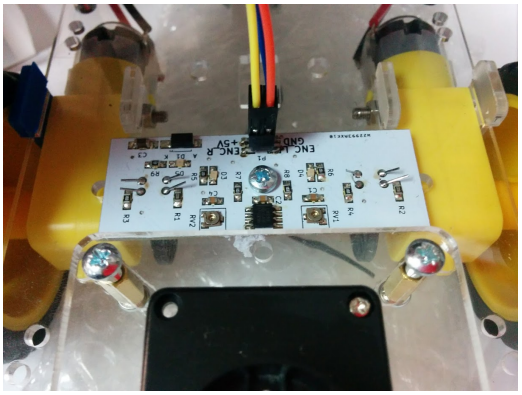

Figure 9: Encoder mounting top view.

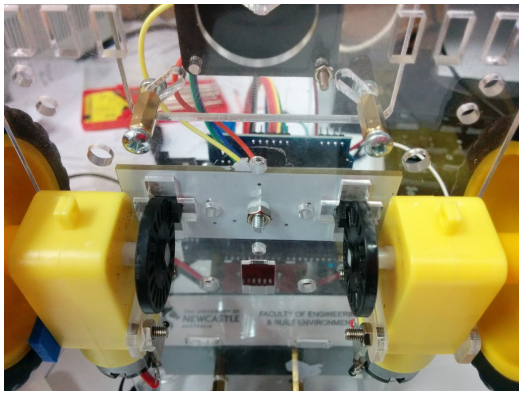

Figure 10: Encoder mounting bottom view.

2) Download the software library, `ENGG1500Lib_master.zip` file located with Lab 2 on Blackboard.

3) Include the *ENGG1500* library: Select **(Sketch → Include Library → Add .ZIP Library)** and select the zip file you just downloaded.

4) Upload the following code which drives the robot forwards a particular number of "clicks" and then reverses the same number. Note that this code can be loaded by selecting, **File →**

[5] See the Project Reference Manual for more information on encoders.

[6] https://image.slidesharecdn.com/introtoarduino-120912225238-phpapp01/95/hello-arduino-nerd-nite-austin-24-728.jpg?cb=1357799819

Examples → ENGG1500Lib → Encoder Test[7].

Listing 5: Encoder Test

```
//START
#include <ENGG1500Lib.h>

void setup() {
  // put your setup code here, to run once:
  pinMode(5,OUTPUT); //set ENA as an output
  pinMode(6,OUTPUT); //set ENB as an output
  pinMode(8,OUTPUT); //set IN1 as an output
  pinMode(9,OUTPUT); //set IN2 as an output
  pinMode(10,OUTPUT); //set IN3 as an output
  pinMode(11,OUTPUT); //set IN4 as an output
  enc_init();
  delay(2000);
}

void loop() {
  // put your main code here, to run repeatedly:
  enc_clear();//set both encoder counts to 0
  leftForwards();//set motor directions to forwards
  rightForwards();

  while((enc_getLeft()+enc_getRight())/2 < 50){
    analogWrite(5,255);
    analogWrite(6,255);
  }

  analogWrite(5,0);
  analogWrite(6,0);
  delay(200);
  enc_clear();
  leftBackwards();
  rightBackwards();

  while((enc_getLeft()+enc_getRight())/2 < 50){
    analogWrite(5,200);
    analogWrite(6,200);
  }
  delay(200);

}

void leftForwards(void)
{
  digitalWrite(8,LOW);
  digitalWrite(9,HIGH);
}

void leftBackwards(void)
{
```

[7] All Arduino libraries come with examples you can access in this way.

```
  digitalWrite(8,HIGH);
  digitalWrite(9,LOW);
}

void rightForwards(void)
{
  digitalWrite(10,LOW);
  digitalWrite(11,HIGH);
}
void rightBackwards(void)
{
  digitalWrite(10,HIGH);
  digitalWrite(11,LOW);
}
//END
```

5) Change the 50 in the code so the distance the vehicle drives forwards/backwards is exactly 300 mm, given the following information.

   - Each encoder count increases by 20 every revolution.
   - The diameter of the wheels is 65 mm.

   Use a `#define` to define a variable instead of hard-coding the number directly, i.e., `#define ENCCLICKS 50`.

Info

Note that the forwards and backwards operations are not precise an error accumulates after each shuffle operation. Try to work out what the source of this error is and think of ways to avoid it. **Hint:** The error is hardware based.

7. Extension Task

There may be a source of error on the encoders due to the stringing on the 3D printed encoder wheels, causing spurious readings. If this is an issue you simply need to clean up the 3D prints with a file or scraper.

Task: Determine if there is a source of error in the encoders due to stringing from the 3D print.

1. Ensure your motors and encoders are connected correctly.
2. Upload the following code to your Arduino.

```
enum {
  ENA = 5,
  ENB = 6,
  IN1 = 8,
  IN2 = 9,
  IN3 = 10,
  IN4 = 11,
```

```
  ENCL = 2,
  ENCR = 3,
};

int wheelPWM = 80;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);

  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);

  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);

  pinMode(ENCL, INPUT);
  pinMode(ENCR, INPUT);

  delay(2000);

  digitalWrite(IN1, 0);
  digitalWrite(IN2, 1);
  digitalWrite(IN3, 0);
  digitalWrite(IN4, 1);

  analogWrite(ENA, wheelPWM);
  analogWrite(ENB, wheelPWM);
}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.print(digitalRead(ENCL));
  Serial.print("\t");
  Serial.println(digitalRead(ENCR) + 2);
  delay(1);
}
```

3. Change the variable `wheelPWM` to the lowest possible speed at which your wheels still reliably turn.
4. Use the Arduino *serial plotter* to inspect the output from the encoders (**Tools→Serial Plotter**). The output should look very uniform, if there are abnormalities such as short spikes then you should consider cleaning up the 3D print.

8. Required Tasks for Next Week

- Read the Arduino reference pages on `if()` and `if-else` statements:
  - https://www.Arduino.cc/en/reference/if

  - https://www.Arduino.cc/en/reference/else

- Ensure the ultrasonic sensor sub-assembly is mounted to the vehicle chassis for next week's lab.

9. Recommendations

Here are some suggestions on what you should work on next:

- Ensure the ultrasonic sensor sub-assembly is mounted to the vehicle chassis for next week's lab.
- Consider making a function that takes direction and speed arguments for each motor and performs control allocation.

Hint

Putting tasks like this in a well designed function, this one in particular, will save you a lot of time in the coming labs and the project.

```
/*This function is used for motor control allocation it has the following
     //...interface. (You should write a little blurb like this when you
     //...write a function to help your group members know how to use it)
Input Arguments:
- int dirL/dirR: 0 or 1 indicating direction of each motor.
- int pwmL/pwmR: 0-255 speed to allocate using analogWrite()
Return Value:
none
Example:
motorCtrlAlloc(1,1,255,255); //This sets both motor directions to forward,
     //...and full speed.
*/
void motorCtrlAlloc(int dirL,int dirR, int pwmL, int pwmR)
{
//Check that dirL and dirR are valid direction values according the the
     //...function specification above.

//Check that pwmL and pwmR are valid PWM values in the range [0,255]

//For debugging you may want to print an error message if any of the
     //...conditions above fail, you could consider changing the function
     //...type from void to int and returning an error code instead.

//Change the direction of left motor
//Change the direction of the right motor

//Set the PWM value for the left motor
//Set the PWM value for the right motor
}
```

- Pick some of the track segments and go through them with your team and brainstorm how you might overcome it conceptually using UML diagrams, consult the Project Reference Manual for advice on programming in a team. Start with the simpler track segments that contain only a line. When thinking of potential solutions think about how which sensors you will use and how you will use them.

  You don't need to know how you will eventually code the solution in order to come up with potential solutions at a conceptual level, start this ASAP!

A. Software Reference

A.1. enc_init

Purpose

Initialises the interrupt pins 2 and 3 for the encoder.

Prototype

```
void enc_init(void);
```

Arguments

None.

Return

None.

Example

This function must be run within your setup code in order to use any of the subsequent functions.

```
void setup()
{
    enc_init();
}
```

A.2. enc_getLeft

Purpose

Obtains the encoder position of the left wheel.

Prototype

```
int enc_getLeft(void);
```

Arguments

None.

Return Value

Returns the position of the encoder where a positive value is forward.

Example

To store the current encoder position into variable **leftPos**:

```
int leftPos = 0;
leftPos = enc_getLeft();
```

A.3. enc_getRight

This function has the same use and function calls as enc_getLeft().

A.4. enc_clear

Purpose

Set the encoder count to zero.

Prototype

```
void enc_clear(void);
```

Arguments

None

Return Value

None

Example

```
enc_clear();
```