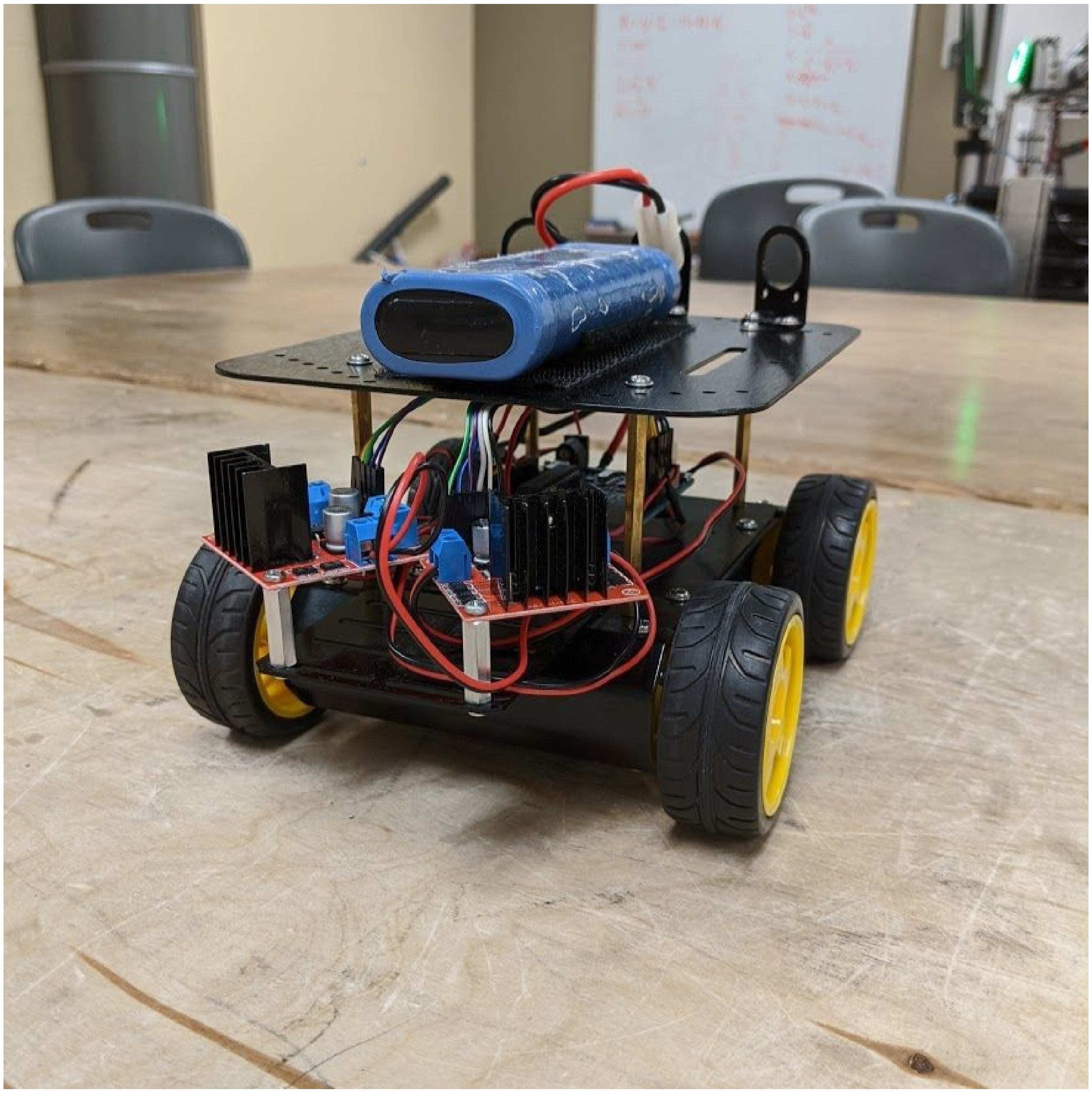
Arduino Lab



Coby Asselin
Thomas Buckley
Emma Gilroy
Dylan Huff
Aidan O'Leary

3/2/20-3/6/20

Objective: To configure and program a robot to drive in a circle and figure 8 using an Arduino microcontroller.

Materials:
- An Arduino Microcontroller
- 2 - L298N Motor Control Modules
- 1 - VEX Battery
- 1 - Transfer Cable
- 1 - Simple Robot Chassis
- Computer with Arduino Software
- Google Docs

Procedure:
1. First, we familiarized ourselves with the coding and programming of the Arduino Microcontroller by using Project Lead the Way as a guide. We also researched a guide on the Speed Controllers we would be using, which were L298N Motor Control Modules.

2. We then took the top panel of the robot off and figured out which pins of the Arduino were mapped to the pins of the motor controllers. We used this information to map a certain function to each of the pins of the Arduino.

3. After figuring out the function of each of the pins, we assigned each pin a certain name that would correspond to its function. For example, we assigned to pin 0 the name "frontLeftForward", meaning that a high on this pin would activate the forward function of the front left wheel. The outcome of this process is shown in **Section 1** and **Table 1**.

4. Then, using the new pin assignments, we created certain functions to accomplish basic goals for certain commands. For example, one set of instructions would tell the speed controllers to activate the forward settings on all four motors, at the highest motor speeds. This command, called "allForward", would make the robot drive forward in a straight line. All of these commands can be found in **Section 2**.

5. After this, it was time to create our setup code. Because we used no input sensors, our robot would rely on internal code. This means that we have no inputs, so we set all of our pins to outputs as shown in **Section 3**.

6. After all of this programming was complete, it was time to create full programs using our specific commands as building blocks. This was where our two objectives diverged.

7. For our circle objective, we used the “leftCircle” command. This command was created to drive one side of the robot’s wheels much faster than the other side, which would result in a gradual turn to the left. We also ran this with an “allStop” command before and after, to give us time to prepare for the program to repeat. This program can be found in **Section 4**.

8. For the figure-eight objective, we initially tried to use a program that created a wide loop, drove to the opposite side of our driving area, and took another wide loop. This proved to be somewhat unpredictable, so we ended up using a much more blocky approach, where we created the shape of an 8 using 90-degree square turns. This approach proved much more predictable. This program can be found in **Section 5**.

Discussion:

The only problem we may have come across was the timing of the motors (i.e. how far they would go and how much they would run) and the amount of power each motor would need to make a full circle. We also had problems with the robot because the motors were somewhat unpredictable, and we often saw the same executed program have different results. We also had problems with the Arduino, due to the number of times of testing, the connections loosened and we had problems finding where the issues resided.

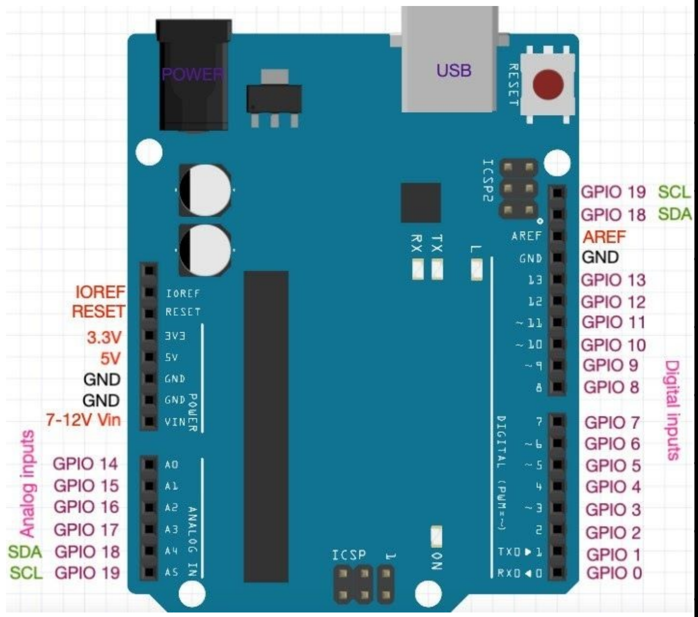
Conclusion:

The Arduino Microcontroller is a versatile microcontroller package. The device takes encoded data from a computer, which is verified and then compiled by the Arduino Software, which is a free application available on most devices. Essentially, when supplied power, the Arduino will place digital highs or lows on the output pins as directed by the program that it’s running. Additionally, some Arduino pins have an analog capability, which allows a user to output different voltage levels for increased control.

For our purposes, the Arduino would be used to communicate with two separate L298N motor speed controllers. In our program, we defined every pin with a name which corresponded to its function. For example, the pin designated as “frontLeftForward” will activate the forward function of the front left motor. Or, a pin designated “pwmFrontLeft” would use the pwm function to control the speed of the Front Left motor.

In programming, the Arduino requires that each pin used is declared as an input or an output, which we accomplished in our setup code. We also wrote a set of commands, such as “allForward” or “turnLeft”, which contained detailed instructions that tell every motor which direction to move and at what power level. We combined these commands in our main program to accomplish our various goals, like driving in a circle or completing a figure-8.

Table 1: Arduino Uno Pinout Diagram

Arduino Uno Physical Pinouts	GPIO #	GPOI function	GPIO #	GPOI function
	GPIO 0	frontLeftForward	GPIO 10	pwmFrontLeft
	GPIO 1	frontLeftReverse	GPIO 11	N/A
	GPIO 2	backLeftForward	GPIO 12	frontRightForward
	GPIO 3	N/A	GPIO 13	backLeftReverse
	GPIO 4	frontRightReverse	GPIO 14	N/A
	GPIO 5	pwmBackLeft	GPIO 15	N/A
	GPIO 6	pwmBackRight	GPIO 16	N/A
	GPIO 7	backRightForward	GPIO 17	N/A
	GPIO 8	backRightReverse	GPIO 18	N/A
	GPIO 9	pwmFrontRight	GPIO 19	N/A

The Program:

Section 1 [Designations for Pins]:

```
//Front Left Motor
const int pwmFrontLeft = 10;
const int frontLeftForward = 0;
const int frontLeftReverse = 1;
//Back Left Motor
const int pwmBackLeft = 5;
const int backLeftForward = 2;
const int backLeftReverse = 13;
//Front Right Motor
const int pwmFrontRight = 9;
const int frontRightForward = 12;
const int frontRightReverse = 4;
//Back Right Motor
const int pwmBackRight = 6;
const int backRightForward = 7;
const int backRightReverse = 8;
```

Section 2 [Instructions for Individual Functions]:

```
 //Stop Command
 void allStop() {
   digitalWrite(frontLeftForward, LOW);
   digitalWrite(frontLeftReverse, LOW);
   digitalWrite(backLeftForward, LOW);
   digitalWrite(backLeftReverse, LOW);
   digitalWrite(frontRightForward, LOW);
   digitalWrite(frontRightReverse, LOW);
   digitalWrite(backRightForward, LOW);
   digitalWrite(backRightReverse, LOW);
   analogWrite(pwmFrontLeft, 0);
   analogWrite(pwmBackLeft, 0);
   analogWrite(pwmFrontRight, 0);
   analogWrite(pwmBackRight, 0);
  }
```

```
// The above command turns all motors completely off.
```

```
 //Go Forward
 void allForward() {
   digitalWrite(frontLeftForward, HIGH);
   digitalWrite(frontLeftReverse, LOW);
   digitalWrite(backLeftForward, HIGH);
   digitalWrite(backLeftReverse, LOW);
   digitalWrite(frontRightForward, HIGH);
   digitalWrite(frontRightReverse, LOW);
   digitalWrite(backRightForward, HIGH);
   digitalWrite(backRightReverse, LOW);
   analogWrite(pwmFrontLeft, 255);
   analogWrite(pwmBackLeft, 255);
   analogWrite(pwmFrontRight, 255);
   analogWrite(pwmBackRight, 255);
  }
```

```
// The allForward command activates the forward functions of each motor, and sets
them to their maximum power levels via the PWM pins.
```

```
 //Go Backwards
 void allReverse() {
   digitalWrite(frontLeftForward, LOW);
   digitalWrite(frontLeftReverse, HIGH);
   digitalWrite(backLeftForward, LOW);
   digitalWrite(backLeftReverse, HIGH);
   digitalWrite(frontRightForward, LOW);
   digitalWrite(frontRightReverse, HIGH);
```

```
  digitalWrite(backRightForward, LOW);
  digitalWrite(backRightReverse, HIGH);
  analogWrite(pwmFrontLeft, 255);
  analogWrite(pwmBackLeft, 255);
  analogWrite(pwmFrontRight, 255);
  analogWrite(pwmBackRight, 255);
 }
```

// The allReverse command is identical to the allForward command, except it activates all of the reverse functions instead of the forward functions.

```
//Circle to the Left
void leftCircle() {
  digitalWrite(frontLeftForward, HIGH);
  digitalWrite(frontLeftReverse, LOW);
  digitalWrite(backLeftForward, HIGH);
  digitalWrite(backLeftReverse, LOW);
  digitalWrite(frontRightForward, HIGH);
  digitalWrite(frontRightReverse, LOW);
  digitalWrite(backRightForward, HIGH);
  digitalWrite(backRightReverse, LOW);
  analogWrite(pwmFrontLeft, 63);
  analogWrite(pwmBackLeft, 63);
  analogWrite(pwmFrontRight, 255);
  analogWrite(pwmBackRight, 255);
}
```

//The leftCircle command drives all wheels forward, but has the left side drive at a much slower power level. This causes the robot to move forward, while also steadily turning left, creating a circle shape if allowed to continue.

```
//Circle to the Right
void rightCircle() {
  digitalWrite(frontLeftForward, HIGH);
  digitalWrite(frontLeftReverse, LOW);
  digitalWrite(backLeftForward, HIGH);
  digitalWrite(backLeftReverse, LOW);
  digitalWrite(frontRightForward, HIGH);
  digitalWrite(frontRightReverse, LOW);
  digitalWrite(backRightForward, HIGH);
  digitalWrite(backRightReverse, LOW);
  analogWrite(pwmFrontLeft, 255);
  analogWrite(pwmBackLeft, 255);
  analogWrite(pwmFrontRight, 63);
  analogWrite(pwmBackRight, 63);
}
```

//Identical to the leftCircle command, this makes the robot circle to the right instead of to the left due to more power being given to the left side.

```
//Turn to the Right
void turnRight() {
  digitalWrite(frontLeftForward, HIGH);
  digitalWrite(frontLeftReverse, LOW);
  digitalWrite(backLeftForward, HIGH);
  digitalWrite(backLeftReverse, LOW);
  digitalWrite(frontRightForward, LOW);
  digitalWrite(frontRightReverse, HIGH);
  digitalWrite(backRightForward, LOW);
  digitalWrite(backRightReverse, HIGH);
  analogWrite(pwmFrontLeft, 255);
  analogWrite(pwmBackLeft, 255);
  analogWrite(pwmFrontRight, 255);
  analogWrite(pwmBackRight, 255);
//The turnRight command drives the left side of the wheels forward, and drives the
right side backward, at full power. This allows the robot to turn to the right
accurately, without changing position.

//Turn to the Left
void turnLeft() {
   digitalWrite(frontLeftForward, LOW);
  digitalWrite(frontLeftReverse, HIGH);
  digitalWrite(backLeftForward, LOW);
  digitalWrite(backLeftReverse, HIGH);
  digitalWrite(frontRightForward, HIGH);
  digitalWrite(frontRightReverse, LOW);
  digitalWrite(backRightForward, HIGH);
  digitalWrite(backRightReverse, LOW);
  analogWrite(pwmFrontLeft, 255);
  analogWrite(pwmBackLeft, 255);
  analogWrite(pwmFrontRight, 255);
  analogWrite(pwmBackRight, 255);
}
//The turnLeft command is identical to the turnRight command. The only difference
is that the right wheels, instead of the left wheels, are driving forward.
```

Section 3 [Setup Code]:

```
//Setup Code
void setup() {
  pinMode(pwmFrontLeft, OUTPUT);
  pinMode(frontLeftForward, OUTPUT);
  pinMode(frontLeftReverse, OUTPUT);
  pinMode(pwmBackLeft, OUTPUT);
  pinMode(backLeftForward, OUTPUT);
  pinMode(backLeftReverse, OUTPUT);
  pinMode(pwmFrontRight, OUTPUT);
  pinMode(frontRightForward, OUTPUT);
  pinMode(frontRightReverse, OUTPUT);
  pinMode(pwmBackRight, OUTPUT);
  pinMode(backRightForward, OUTPUT);
  pinMode(backRightReverse, OUTPUT);
}
```

Section 4 [Circle Program]:

```
//Main Code
void loop() {
  allStop();
  delay(1000);
  leftCircle();
  delay(10000);
  allStop();
  delay(10000);
}
```

Section 5 [Figure 8 Program]:

```
//Below is the Initial Attempt described in Step 8.
//Main Code
void loop() {
  allStop();
  delay(1000);
  leftCircle();
  delay(5000);
  rightCircle();
  delay(5500);
}
```

```
//Following is the final attempt, using 90-degree turns.
 //Main Code
 void loop() {
        allStop();
        delay(1000);
        allForward();
        delay(1000);
        turnLeft();
        delay(460);
        allForward();
        delay(1000);
        turnLeft();
        delay(460);
        allForward();
        delay(1000);
        turnRight();
        delay(460);
        allForward();
        delay(1000);
        turnRight();
         delay(460);
         allForward();
        delay(1000);
        turnRight();
        delay(460);
        allForward();
        delay(1000);
        turnRight();
        delay(460);
        allForward();
        delay(1000);
        turnLeft();
        delay(460);
        allForward();
        delay(1000);
        turnLeft();
        delay(460);
 }
```