University of Colorado Boulder
ECEE Department

ECEN 2270 - Electronics Design Lab - Spring 2024
Location: Engineering Center, ECEE 281, T,TH, 3:30 - 5:20 PM
Instructor: Steven Dunbar
Lab Title: Lab 4: Speed and Position Control
Date of Experiment: April 21, 2024
Name: Connor Sorrell

Introduction and Objectives

Introduction:

In this lab, we focused on advancing the practical application of our control system by finishing off the implementation of various integrating circuits and components on the breadboard. This involved finishing the speed feedback control loop on the other side of the breadboard, adding decoupling capacitors throughout the build, and tinkering with component values until a practical robot was complete. We then powered and tested an Ardunio, confirming its operational capabilities to control our motor direction and speed via its digital pins. This integration allowed the robot to execute pre-coded movements accurately, which adhered to the specific parameters of speed and position control. The arduino was coded with the help of an ISR, in which we counted encoder pulses which enabled us to accurately correlate a reference number into precise movements of our robot, including specific distances and turning angle. With an already fully implemented breadboard, the addition of the arduino coding allowed us to have a fully functional robot which passed all control tests and demonstrated its ability to move straight for any distance and turn either direction at any degree. The lab was easily our favorite of the semester, as we could see all our hard work coming to fruition. It also reinforced our skills in programming, as well as bridged the gap between the hardware implementation and the code behind the car as well. Overall, the lab went smooth and has inspired us to work hard on a unique final project for the car.

Objectives:

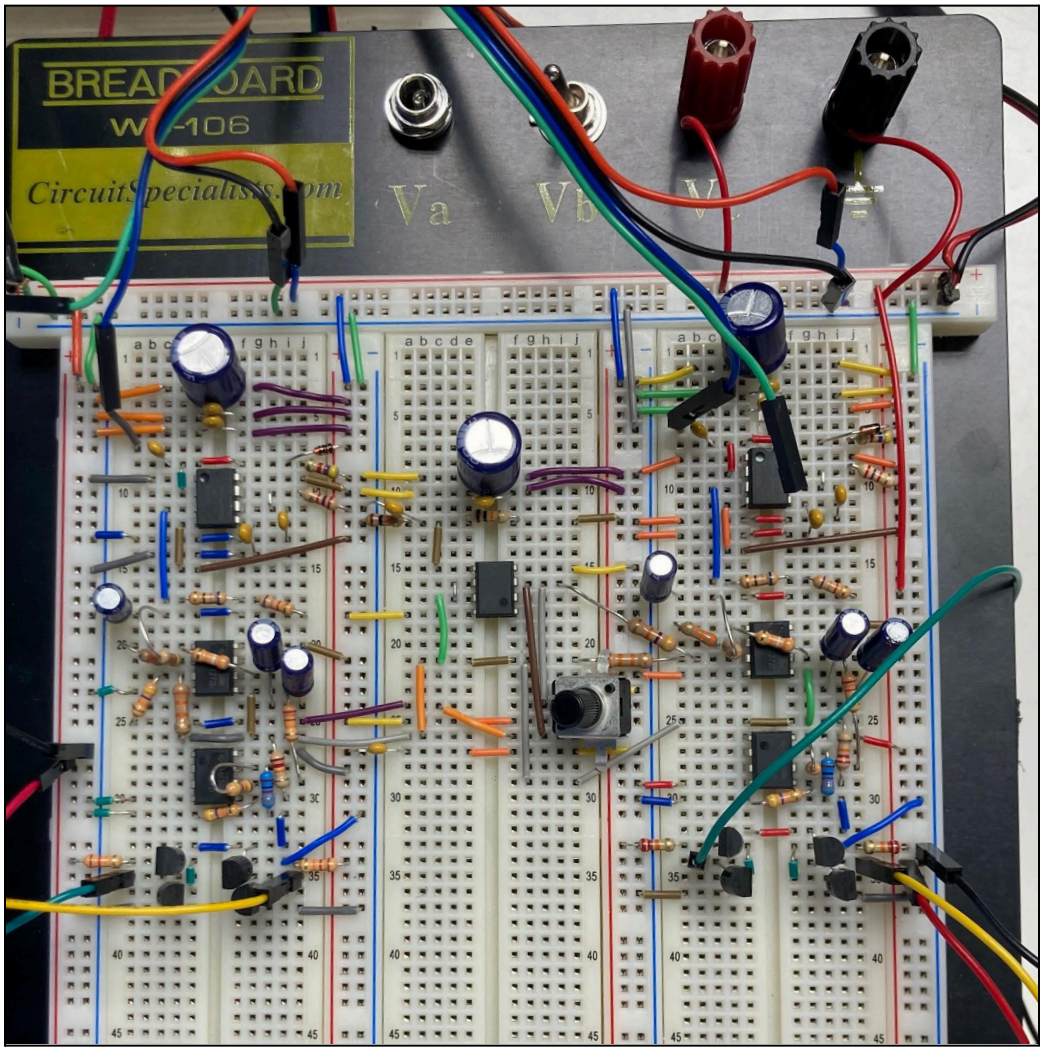
Throughout the lab, there are some key objectives that are accomplished:

- The speed feedback control loop is implemented on the other side of the breadboard; all circuit implementations are built and function properly.
- The arduino is powered and tested and confirmed to work as expected.
- The motor direction control and speed control are connected to the digital pins of the arduino and all work together seamlessly.
- The robot moves as intended with the arduino and it works together with the speed control circuit as planned.
- The interrupt overhead is understood, estimated and confirmed.
- Comparators are used as level shifters to cap the PWM output at 5V in order to adhere to the properties of the arduino.
- Number of CPU cycles available for programming in each ISR (at motors full speed) is determined.
- Encoder pulses are counted and translated to the cars position control; a specific number of pulses translates to a certain distance, turn radius, etc.
- The encoder count is confirmed by repeated comprehensive position control tests.

- Code is written that effectively controls the arduino, can move forward, backward and turn both ways.
- The robot passes a position control test. It can move straight, do a counterclockwise 180 degree turn, move straight again, and do a clockwise 180 degree turn.
- The robot is fully functional and the breadboard adheres to circuit building guidelines.

Experiment 4A: Speed and Position Control

Experiment 4.A.2: Speed Controller for Second Side of Robot

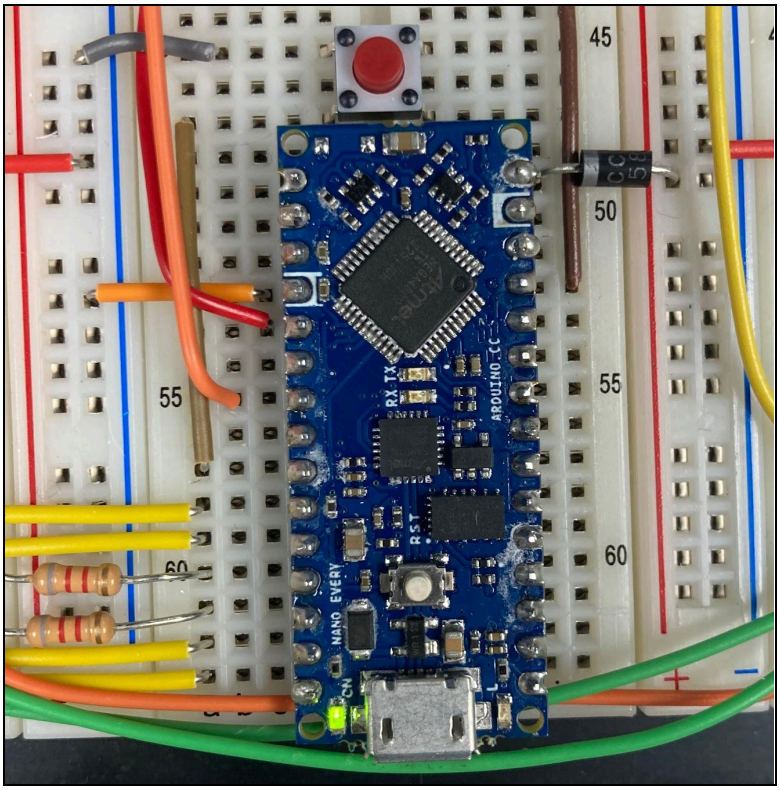


The above image shows the complete schematic of the speed controller on one robot. The left schematic was made in a near-perfect copy of the right schematic. One difference was the wiring of the Vref and Vg, as those references were on the other side of the middle column opamp. At this stage, we also added several decoupling capacitors across the breadboard to the integrated circuits.

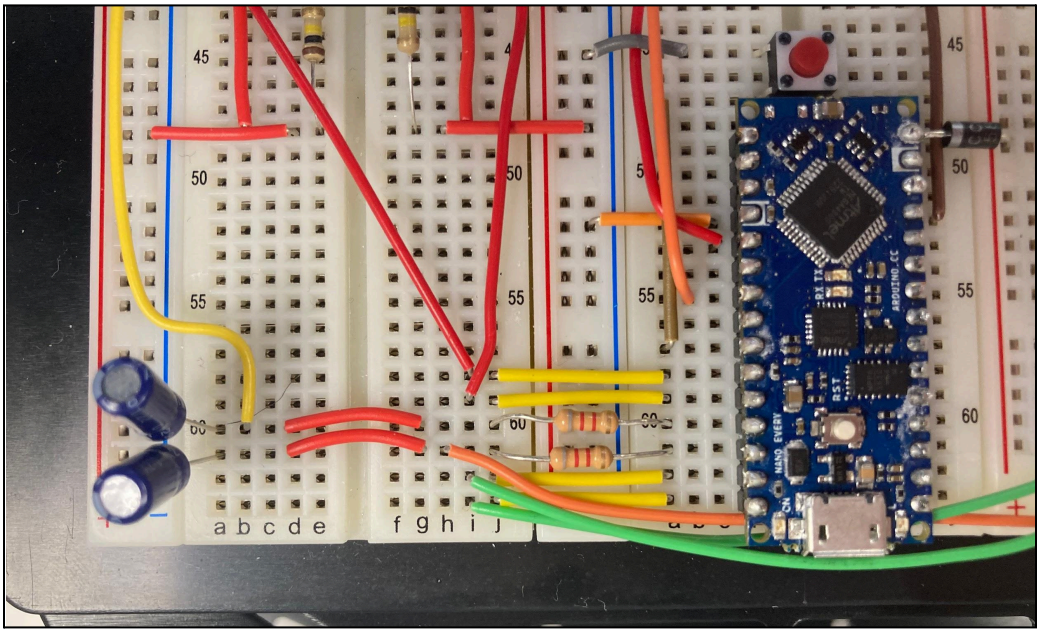
Experiment 4.A.3: Test Arduino

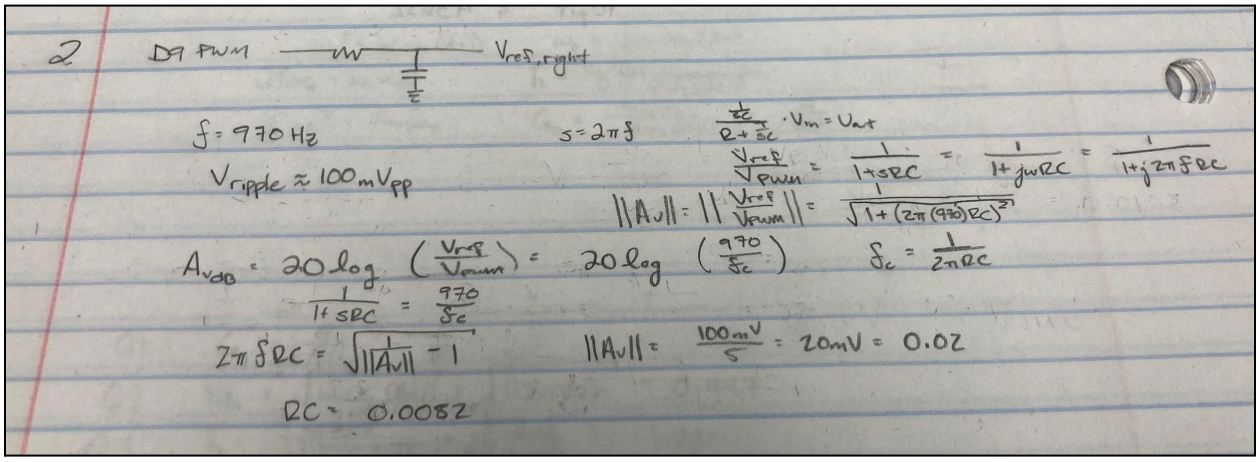
```
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// 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(2000);                      // wait for 2 seconds
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
  delay(200);                      // wait for 0.2 seconds (200ms)
}
```

Experiment 4.A.4: Power Arduino



Experiment 4.A.5: Connect Vref Outputs





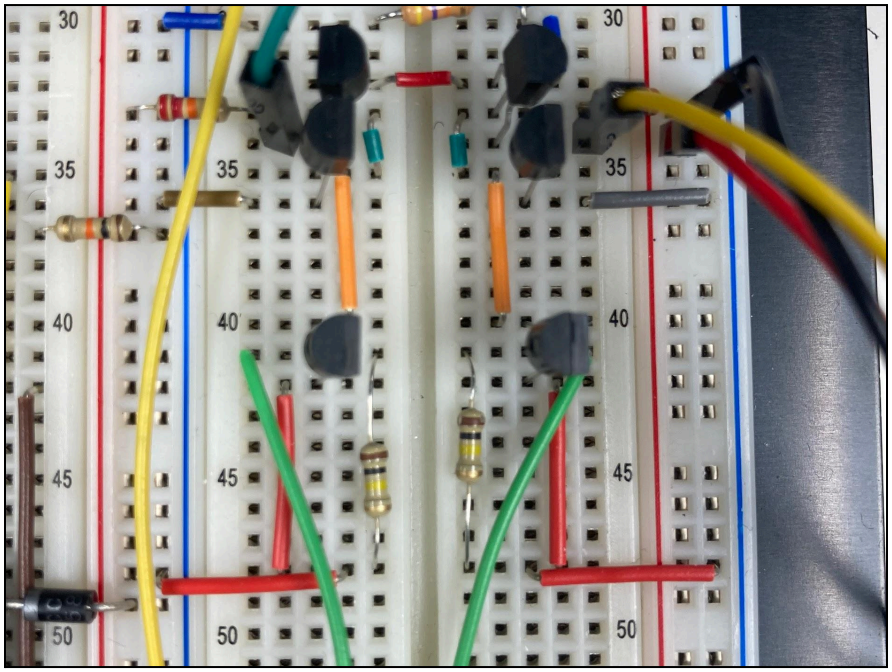
The simple low-pass filter, seen above, was created to convert our PWM signal to an analog Vref with a magnitude of 0-5V. The computation for this filter is seen above, and the time constant necessary to achieve the correct attenuation is 0.0082. To achieve this value, a capacitor of one microfarad matched with a resistor of 8.2 kiloohms was chosen.

Experiment 4.A.6: Adjust the Speed Control Feedback Circuits

```
// define pins
const int pinRightPWM = 9;
const int pinLeftPWM = 10;

void setup() {
  pinMode(pinRightPWM, OUTPUT);
  pinMode(pinLeftPWM, OUTPUT);
  analogWrite(pinRightPWM, 4*51);  //Vref = 4V
  analogWrite(pinLeftPWM, 4*0.99*51);  //Vref = 4V; left runs fast so make it 99%
of Right
}

void loop() {
}
```

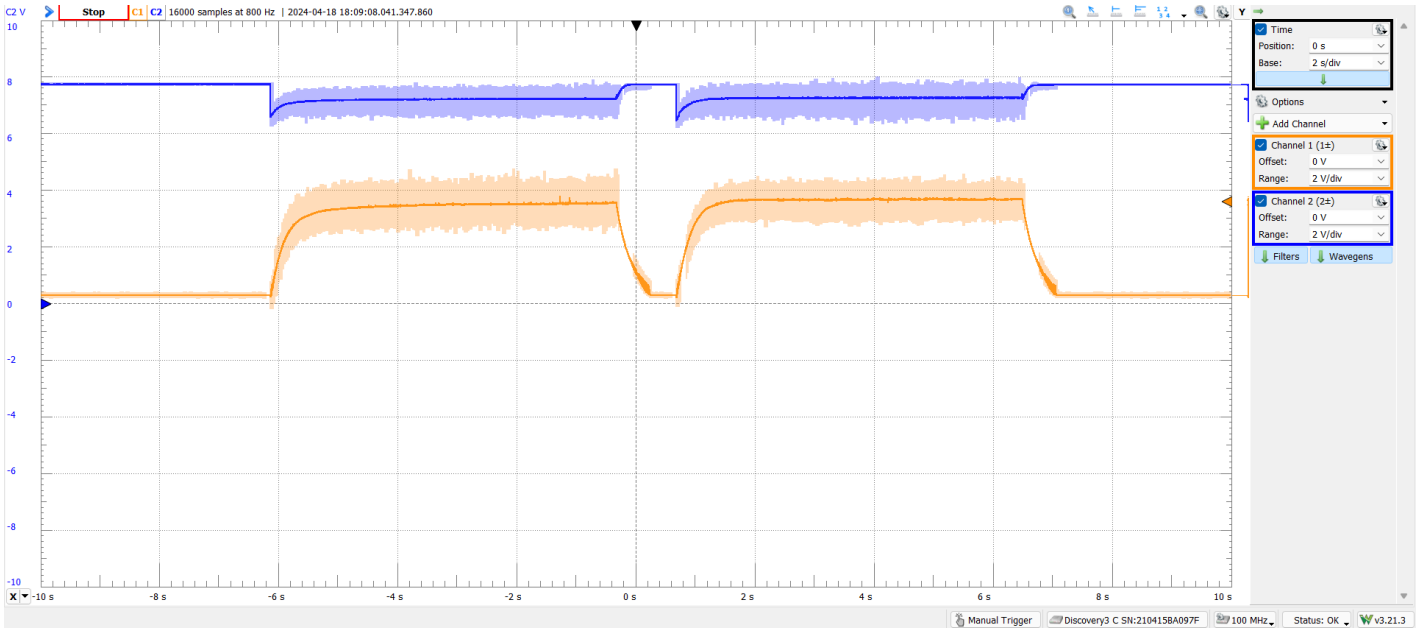
The Arduino code to generate the PWM signals for Vref is shown above. When probed, each of our max speeds was around 5V, so the t_{on} value was not modified. To verify each motor ran at the same speed at incremental Vref values, we ran our robots for five seconds on the floor. Analyzing the robots' direction path, we were able to notice if one motor ran fast. In the case of the code above, the left motor ran barely faster, so the left Vref was lowered by 1% to account for the difference.

Experiment 4.A.7: Connect Motor Direction Control to Arduino



Experiment 4.A.8: Robot Speed Control with Arduino

```
// define pins
const int pinON = 6;
const int pinRightForward = 8;
const int pinRightBackward = 7;
const int pinRightPWM = 9;
const int pinLeftPWM = 10;
const int pinLeftForward = 12;
const int pinLeftBackward = 11;

void setup() {
  pinMode(pinON, INPUT_PULLUP);
  pinMode(pinLeftForward, OUTPUT);
  pinMode(pinLeftBackward, OUTPUT);
  pinMode(pinLeftPWM, OUTPUT);
  pinMode(pinRightForward, OUTPUT);
  pinMode(pinRightBackward, OUTPUT);
  pinMode(pinRightPWM, OUTPUT);
  pinMode(13, OUTPUT);
  digitalWrite(pinLeftForward, LOW);
  digitalWrite(pinLeftBackward, LOW);
  digitalWrite(pinRightForward, LOW);
  digitalWrite(pinRightBackward, LOW);
  analogWrite(pinLeftPWM, 200);  //Vref, duty cycle 200/255
  analogWrite(pinRightPWM, 200);  //Vref, duty cycle 200/255
}

void loop() {
  digitalWrite(13, LOW);
  do {} while (digitalRead(pinON) == HIGH);
  digitalWrite(13, HIGH);
  delay(1000);

  digitalWrite(pinLeftForward, HIGH);
  digitalWrite(pinRightBackward, HIGH);

  delay(5800);
  digitalWrite(pinLeftForward, LOW);
  digitalWrite(pinRightBackward, LOW);

  delay(1000);
  digitalWrite(pinRightForward, HIGH);
  digitalWrite(pinLeftBackward, HIGH);

  delay(5800);
  digitalWrite(pinRightForward, LOW);
  digitalWrite(pinLeftBackward, LOW);
}
```



This simulation shows the speed Vs (orange) and the controller output Vo (blue) during one execution of the loop with the robot wheels off the ground.

Experiment 4.A.9: Robot Movement Repeatability Using Speed Control

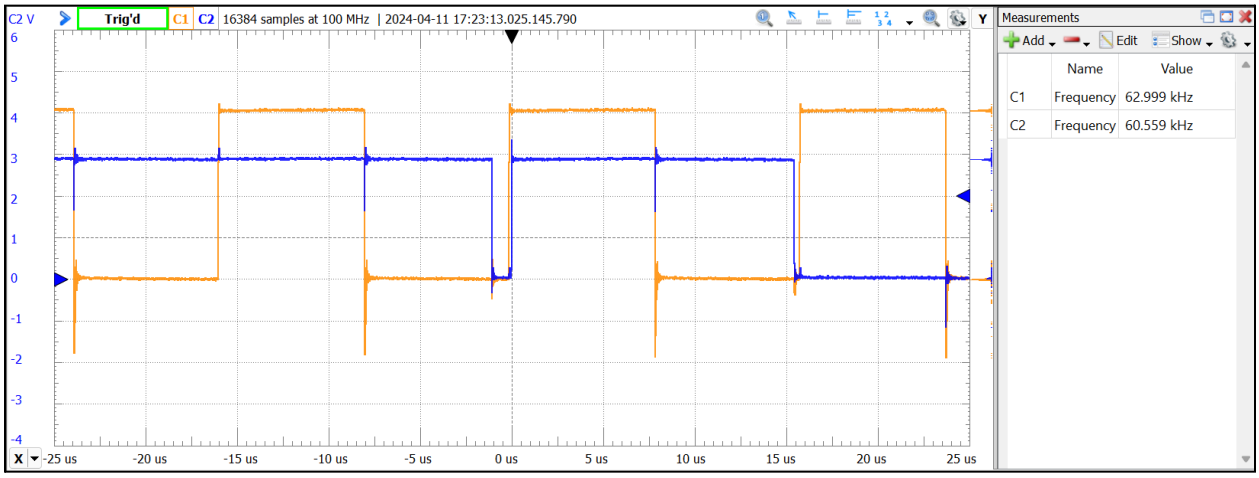
```
// define pins
const int pinON = 6;
const int pinRightForward = 8;
const int pinRightBackward = 7;
const int pinRightPWM = 9;
const int pinLeftPWM = 10;
const int pinLeftForward = 12;
const int pinLeftBackward = 11;

void setup() {
  pinMode(pinON, INPUT_PULLUP);
  pinMode(pinLeftForward, OUTPUT);
  pinMode(pinLeftBackward, OUTPUT);
  pinMode(pinLeftPWM, OUTPUT);
  pinMode(pinRightForward, OUTPUT);
  pinMode(pinRightBackward, OUTPUT);
  pinMode(pinRightPWM, OUTPUT);
  pinMode(13, OUTPUT);
  digitalWrite(pinLeftForward, LOW);
  digitalWrite(pinLeftBackward, LOW);
  digitalWrite(pinRightForward, LOW);
  digitalWrite(pinRightBackward, LOW);
  analogWrite(pinLeftPWM, 200);  //Vref, duty cycle 200/255
  analogWrite(pinRightPWM, 200);  //Vref, duty cycle 200/255
}

void loop() {
  digitalWrite(13, LOW);
  do {} while (digitalRead(pinON) == HIGH);
  digitalWrite(13, HIGH);

  delay(1000);        // wait one second

  digitalWrite(pinLeftForward, HIGH);
  digitalWrite(pinRightForward, HIGH);    //drive 2 feet
  delay(2000);

  digitalWrite(pinRightForward, LOW);
  digitalWrite(pinRightBackward, HIGH);  //do 180 clockwise
  delay(1500);

  digitalWrite(pinRightBackward, LOW);
  digitalWrite(pinRightForward, HIGH);    //drive 2 feet
  delay(2000);

  digitalWrite(pinLeftForward, LOW);
  digitalWrite(pinLeftBackward, HIGH);  //do 180 counter clockwise
  delay(1500);
```

```
  digitalWrite(pinRightForward, LOW);
  digitalWrite(pinLeftBackward, LOW);
}
```

During most tests, the robot performed the desired actions, then came back within ± 2 inches. Although the robot ran the correct route in most tests, ~30% of the time, the robot ran off course unexpectedly. Mainly, the turning executions were quite inconsistent, not making perfect 180 turns. The timing method used here proved to be not as accurate as the pulse counting method used in Experiment 4B.

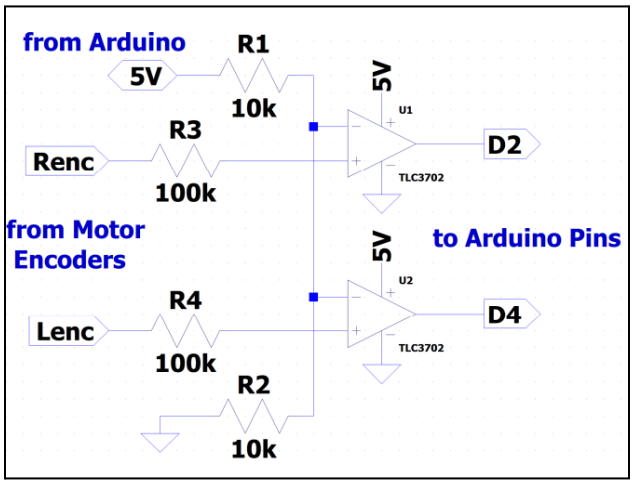
Experiment 4B: Position Control

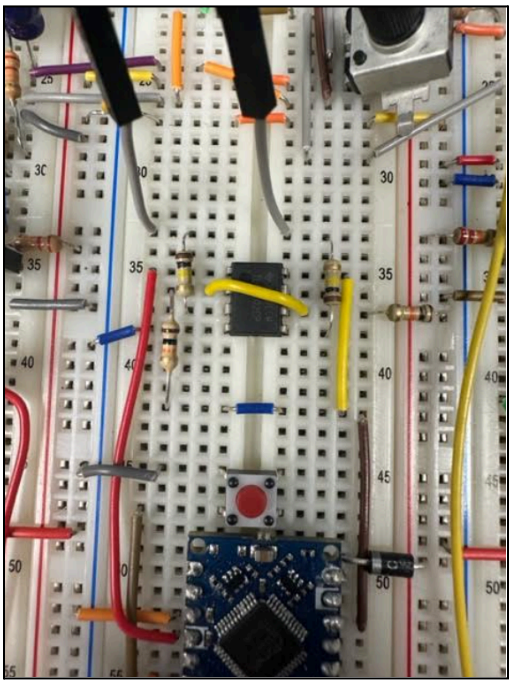
Experiment 4.B.2: Robot Movement Repeatability Using Speed Control



As seen from the picture, the ISR starts skipping pulses at around 63 kHz. This implies that the overhead of the ISR is 1/63 kHz = ~15 microseconds.

Experiment 4.B.3: Use Comparators as Level Shifters





Experiment 4.B.4: Read Encoder with Arduino

Our strategy to check the correct counting of the pulses was by counting the number of encoder pulses and comparing that to the distance traveled by the robot. We then scaled this ratio of pulses to distance up, concluding that 2500 pulses for each side of the car drive the car two feet.

Our test procedure involved mostly guess-and-checking until the distance was perfectly two feet.

To write Arduino ISR's to count the the number of encoder pulses, we used the following code highlighted in yellow:

```
volatile int enc_count = 0;
//const int pinON = 6;
//const int pinRightForward = 8;
//const int pinRightBackward = 7;
//const int pinRightPWM = 9;
//const int pinLeftPWM = 10;
//const int pinLeftForward = 12;
//const int pinLeftBackward = 11;
volatile int target = 0;

void setup() {
//  pinMode(pinON, INPUT_PULLUP);
//  pinMode(pinLeftForward, OUTPUT);
//  pinMode(pinLeftBackward, OUTPUT);
//  pinMode(pinLeftPWM, OUTPUT);
//  pinMode(pinRightForward, OUTPUT);
//  pinMode(pinRightBackward, OUTPUT);
//  pinMode(pinRightPWM, OUTPUT);
//  pinMode(13, OUTPUT);
//  digitalWrite(pinLeftForward, LOW);
//  digitalWrite(pinLeftBackward, LOW);
//  digitalWrite(pinRightForward, LOW);
//  digitalWrite(pinRightBackward, LOW);
//  analogWrite(pinLeftPWM, 100);
//  analogWrite(pinRightPWM, 240);
 attachInterrupt(2, ISR_count, RISING);
}
```

```
void loop() {
  enc_count = 0;
  delay(1000);
  target = 2500; // 2500 pulses equates to 2 feet of distance

      //*do stuff* (in this case, drive forward)
  digitalWrite(pinLeftForward, HIGH);
  digitalWrite(pinRightForward, HIGH);
  do {} while(enc_count <= target);
  digitalWrite(pinLeftForward, LOW);
  digitalWrite(pinRightForward, LOW);
  delay(3000);
  enc_count = 0;
  target = 2000;
  /*do more more stuff until 2000 pulses*/
 }

void ISR_count()
{
  enc_count++;
}
```

In the lines highlighted in yellow, the ISR can be seen. It starts by initializing a variable to zero, which will count our number of pulses. Everytime the ISR is triggered, the count increments. Using this logic, we can equate a specific number of pulses to an action in which the robot performs.

Experiment 4.B.5: Position Control by Counting Encoder Pulses

Arduino program that allows the robot to move precisely by a specified number of pulses:

```
void twoftforward()
{
  enc_count = 0;
  /*do stuff*/
  target = 2550; // 2500 gets me 2 ft exact, 1250 pulses per foot
  digitalWrite(pinLeftForward, HIGH);
  digitalWrite(pinRightForward, HIGH);
  do {} while(enc_count <= target);
  digitalWrite(pinLeftForward, LOW);
  digitalWrite(pinRightForward, LOW);
  return;
}
void ISR_count(){
  enc_count++;
}
```

Using these together, we can power a wheel until the desired number of pulses is reached.

180 degrees clockwise:

```
void oneeightyclockwise()
{
  target = 1300; //do 180 clockwise
  enc_count = 0;
  digitalWrite(pinRightBackward, HIGH);
  digitalWrite(pinLeftForward, HIGH);
  do {} while(enc_count <= target);
  digitalWrite(pinRightBackward, LOW);
  digitalWrite(pinLeftForward, LOW);
}
```

180 degrees counterclockwise:

```
void oneeightycounter()
{
  target = 1380; //do 180 counterclockwise
  enc_count = 0;
  digitalWrite(pinLeftBackward, HIGH);
  digitalWrite(pinRightForward, HIGH);
  do {} while(enc_count <= target);
  digitalWrite(pinLeftBackward, LOW);
  digitalWrite(pinRightForward, LOW);
```

Experiment 4.B.6: Position Control Tests

By putting all the code together from 4.B.5, we can call our functions as so:

```
void loop() {
  digitalWrite(13, LOW);
  do {} while (digitalRead(pinON) == HIGH);
  digitalWrite(13, HIGH);
  delay(1000);
  twoftforward();
  delay(1000);
  /*more stuff!*/
  oneeightyclockwise();
  delay(1000);
  twoftforward();
  delay(1000);
  oneeightycounter();
}
```

While performing this accuracy test, the robot came within $\pm$ 2in consistently, and much more accurately than the test performed in 4.A.9. The turning and linear movement was much more consistent using the pulse counter method.

Experiment 4.B.7: Advanced Position Control Test (extra credit)

```
volatile int enc_count = 0;
const int pinON = 6;
const int pinRightForward = 8;
const int pinRightBackward = 7;
const int pinRightPWM = 9;
const int pinLeftPWM = 10;
const int pinLeftForward = 12;
const int pinLeftBackward = 11;
volatile int target = 0;

void setup() {
  pinMode(pinON, INPUT_PULLUP);
  pinMode(pinLeftForward, OUTPUT);
  pinMode(pinLeftBackward, OUTPUT);
  pinMode(pinLeftPWM, OUTPUT);
  pinMode(pinRightForward, OUTPUT);
  pinMode(pinRightBackward, OUTPUT);
  pinMode(pinRightPWM, OUTPUT);
  pinMode(13, OUTPUT);
  digitalWrite(pinLeftForward, LOW);
  digitalWrite(pinLeftBackward, LOW);
  digitalWrite(pinRightForward, LOW);
  digitalWrite(pinRightBackward, LOW);
  analogWrite(pinLeftPWM, 200);  //Vref, duty cycle 200/255
  analogWrite(pinRightPWM, 200);  //Vref, duty cycle 200/255
 attachInterrupt(2, ISR_count, RISING);
 Serial.begin(9600);
}
void loop() {
  do {} while (digitalRead(pinON) == HIGH);

   /*RUN TWO FEET NORTH*/
  target = 2500; // 2500 gets me 2 ft exact front to front
  enc_count = 0;
  delay(1000);
  digitalWrite(pinLeftForward, HIGH);
  digitalWrite(pinRightForward, HIGH);
  do {} while(enc_count <= target);
  digitalWrite(pinLeftForward, LOW);
  digitalWrite(pinRightForward, LOW);
  delay(1000);

  /*Turn 90 degrees CCW*/
  target = 700;
  enc_count = 0;
  digitalWrite(pinLeftBackward, HIGH);  //do 90 deg counter clockwise
  digitalWrite(pinRightForward, HIGH);
  do {} while(enc_count <= target);
```

```
 digitalWrite(pinLeftBackward, LOW);
 digitalWrite(pinRightForward, LOW);
 delay(1000);

/*RUN FOUR FEET WEST*/
 target = 5250;
 enc_count = 0;
 digitalWrite(pinLeftForward, HIGH);
 digitalWrite(pinRightForward, HIGH);
 do {} while(enc_count <= target);
 digitalWrite(pinLeftForward, LOW);
 digitalWrite(pinRightForward, LOW);
 delay(1000);

 /*Turn 90 degrees CW*/
 target = 600;
 enc_count = 0;
 digitalWrite(pinRightBackward, HIGH);  //do 180 clockwise  750 pulses per 90 deg
 digitalWrite(pinLeftForward, HIGH);
 do {} while(enc_count <= target);
 digitalWrite(pinRightBackward, LOW);
 digitalWrite(pinLeftForward, LOW);
 delay(1000);

 /*RUN TWO FEET NORTH*/
 target = 2500; // 2500 gets me 2 ft exact front to front
 enc_count = 0;
 digitalWrite(pinLeftForward, HIGH);
 digitalWrite(pinRightForward, HIGH);
 do {} while(enc_count <= target);
 digitalWrite(pinLeftForward, LOW);
 digitalWrite(pinRightForward, LOW);
 delay(1000);

 /*Turn 90 degrees CW*/
 target = 700;
 enc_count = 0;
 digitalWrite(pinRightBackward, HIGH);  //do 180 clockwise  750 pulses per 90 deg
 digitalWrite(pinLeftForward, HIGH);
 do {} while(enc_count <= target);
 digitalWrite(pinRightBackward, LOW);
 digitalWrite(pinLeftForward, LOW);
 delay(1000);

 /*RUN TWO FEET EAST*/
 target = 2200; // 2500 gets me 2 ft exact front to front
 enc_count = 0;
 digitalWrite(pinLeftForward, HIGH);
 digitalWrite(pinRightForward, HIGH);
```

```
do {} while(enc_count <= target);
digitalWrite(pinLeftForward, LOW);
digitalWrite(pinRightForward, LOW);
delay(1000);

/*Turn 90 degrees CW*/
target = 650;
enc_count = 0;
digitalWrite(pinRightBackward, HIGH);  //do 180 clockwise  750 pulses per 90 deg
digitalWrite(pinLeftForward, HIGH);
do {} while(enc_count <= target);
digitalWrite(pinRightBackward, LOW);
digitalWrite(pinLeftForward, LOW);
delay(1000);


/*RUN FOUR FEET SOUTH*/
target = 5000;
enc_count = 0;
digitalWrite(pinLeftForward, HIGH);
digitalWrite(pinRightForward, HIGH);
do {} while(enc_count <= target);
digitalWrite(pinLeftForward, LOW);
digitalWrite(pinRightForward, LOW);
delay(1000);

/*Turn 90 degrees CCW*/
target = 725;
enc_count = 0;
digitalWrite(pinLeftBackward, HIGH);  //do 90 deg counter clockwise
digitalWrite(pinRightForward, HIGH);
do {} while(enc_count <= target);
digitalWrite(pinLeftBackward, LOW);
digitalWrite(pinRightForward, LOW);
delay(1000);

/*RUN TWO FEET EAST*/
target = 2700; // 2500 gets me 2 ft exact front to front
enc_count = 0;
delay(1000);
digitalWrite(pinLeftForward, HIGH);
digitalWrite(pinRightForward, HIGH);
do {} while(enc_count <= target);
digitalWrite(pinLeftForward, LOW);
digitalWrite(pinRightForward, LOW);
delay(1000);

/*Turn 90 degrees CCW*/
target = 725;
```

```
  enc_count = 0;
  digitalWrite(pinLeftBackward, HIGH);  //do 90 deg counter clockwise
  digitalWrite(pinRightForward, HIGH);
  do {} while(enc_count <= target);
  digitalWrite(pinLeftBackward, LOW);
  digitalWrite(pinRightForward, LOW);
  delay(1000);
}
void ISR_count(){
  enc_count++;
}
```

Conclusion:

In the end, our lab work ended in the successful completion of a fully functional electronic car, demonstrating seamless integration between a precisely tuned and implemented speed feedback control loop, and careful Arduino programming. The combination of the two allowed our robot to perform accurate movements and turns with more than decent accuracy. The addition of the encoder pulse counting enabled our robot to perform complex movements extremely efficiently, and the integrated circuits allowed those precise maneuvers to happen by providing an efficient speed and motor direction control system. The car is now fully functional and showcases hard work and persistence, boosting our confidence and enthusiasm for future projects. This hands-on experience not only taught us countless practical engineering skills, but it also provided us with many learning lessons and strengthened our intangible skills. Moving forward, the techniques mastered in these series of labs will lead to more ambitious applications and designs not only in the final project, but in our careers as well.

Lab Exploration 4A:

1&3). Hardware interrupts are generated by external devices that are connected to the computer system, including but not limited to keyboards, mice, timers, or network interfaces. These interrupts are generally triggered by signals that are sent from hardware components to the CPU to request attention or notify the CPU about a specified event. During a hardware interrupt, the CPU effectively distributes its current execution flow and transfers control to a specified interrupt handler routine, which is responsible for servicing the interrupt. There are different types of hardware interrupts based on their priority, such as Non-maskable Interrupts (NMI) and Maskable Interrupts. Software interrupts however are generated internally by the CPU in response to specific conditions or instructions encountered during the program execution. They are triggered through software instructions, such as system calls, illegal instructions, divide-by-zero errors, or even page defaults. Software interrupts are implemented to request services from the operating system to handle exceptional conditions that are encountered during program execution. During a software interrupt, the CPU switches from the user model to a kernel mode and transfers control to the corresponding interrupt handler routine provided by the operating system. These interrupts are utilized for tasks such as process scheduling, memory management, or input/put operations.

2). Polling is a technique that is commonly used in computer science and electronics to continuously evaluate the state of a device or a system by repeatedly sending queries or requests for information. This is done to determine if the computer system or electronics needs attention or if certain conditions are met.

4). Whenever an interrupt occurs in a microcontroller program, the normal flow of execution is temporarily disabled, while the microcontroller jumps to a predefined location within its memory to execute a specific interrupt service routine also known as an ISR that is associated with the interrupt that occurred.

5). The Arduino Nano Every, like other Arduino boards, utilizes interrupts to handle external events effectively. The interrupt structure is based on the microcontroller embedded with it. This microcontroller is the ATmega4809, which includes several types of interrupts. On the digital pins of the Arduino nano, the microcontroller supports external interrupts. Furthermore, it has several external interrupt pins usually labeled INT0, INT1, etc. These can be triggered through changes on the pins, such as rising edges, falling edges, or logic level changes. There are also Pin change interrupts which can be triggered by changes on any of the digital pins. Rather than just specific pins like external interrupts, which allows for more flexible interrupt handling in projects where the specified pin causing the interrupt is not known in advance. The Arduino Nano also has Timer interrupts, which are built-in hardware timers that can generate interrupts at specified intervals. These can be utilized for tasks such as timekeeping, generating PWM signals, or triggering Periodic Events. Finally, the Arduino Nano contains Serial communication

interrupts. These can be triggered by the serial communication hardware when data is received or transmitted over UART, SPI, or I2C interfaces.

6).

```
const int ledPin = 13;
// Define button pin
const int buttonPin = 2;
// Define variable to hold the button state
volatile int buttonState = LOW;

void setup() {
  //Set LED pin as an output
  pinMode(ledPin, OUTPUT);
  // Set button pin as an input with pull-up resistor
  pinMode(buttonPin, INPUT_PULLUP);
  // Assign interrupt to the button pin
  attachInterrupt(digitalPinToInterrupt(buttonPin), buttonInterrupt,
CHANGE);
}

void loop() {
}

// Interrupt service routine for the button
void buttonInterrupt() {
  // Show the state of the button
  buttonState = digitalRead(buttonPin);
  // If button is pressed turn off
  if (buttonState == LOW) {
    // Toggle the state of the LED
    digitalWrite(ledPin, !digitalRead(ledPin));
  }
}
```

We utilized the attachInterrupt() function to connect an interrupt service routine (ISR) to a specific pin. When the external event (in this case, a change in the state of the button connected to the interrupt pin) occurs, the microcontroller halts its current execution and immediately jumps to execute the ISR.