Zachary Bang

Aleida Diaz-Roque

Elkaim

ECE 118: Intro to Mechatronics

Lab 3 Motors

Table of Contents

In this lab we explored the control and operation of different types of motors including DC motors, servos, and stepper motors. The lab was structured to enhance our understanding of motor mechanics, control techniques, and the management of inductive kickback. Utilizing the Uno32 stack, which integrates I/O boards and a power distribution board, we ensured the safe and effective functioning of our projects.

Part 1 - Driving an RC Servo

In this part of the lab we explored how to control an RC Servo using a potentiometer that would determine the pulse time for the RC Servo. Initially we obtained a reading from the potentiometer and after realizing how noisy the input was we applied an exponential moving average filter as seen below.

```
// EMA filtering
currPot = AD_ReadADPin(AD_PORTV3);
reading = ((ALPHA) * currPot) + ((1 - ALPHA) * prevPot);
prevPot = reading;
```

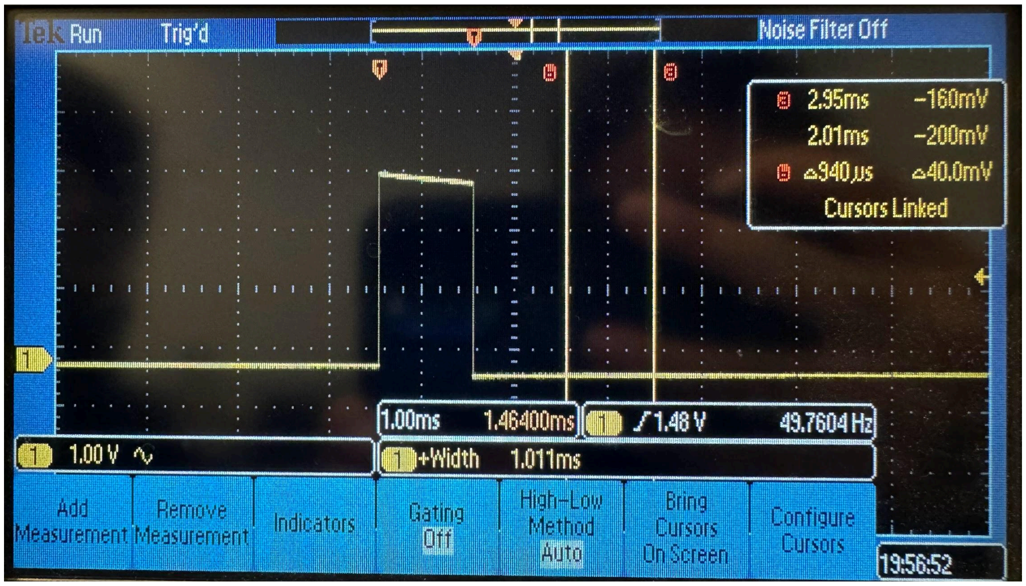
The filter reduced the noise down to a more steady 3 or 4 values, rather than the previously observed range of approximately 50. After obtaining a more consistent reading, we started with implementing the LEDs tracking the potentiometer which in turn would also display the pulse of the RC Servo. We scaled the filtered reading from the potentiometer into 12 for the 12 LEDs in the three banks and then used bit masking to connect all three banks. This was done as below.

```
// Scaling POT reading to 12 LEDs
LED = (reading / 1024.0) * 13.0;

// Scaling LED to 4 values
pattern = (LED-1)%4 +1;

// If a new value is detected make the value 1-4
if (currPot != prevPot) {
     pattern = (1<<pattern)-1;
}
```

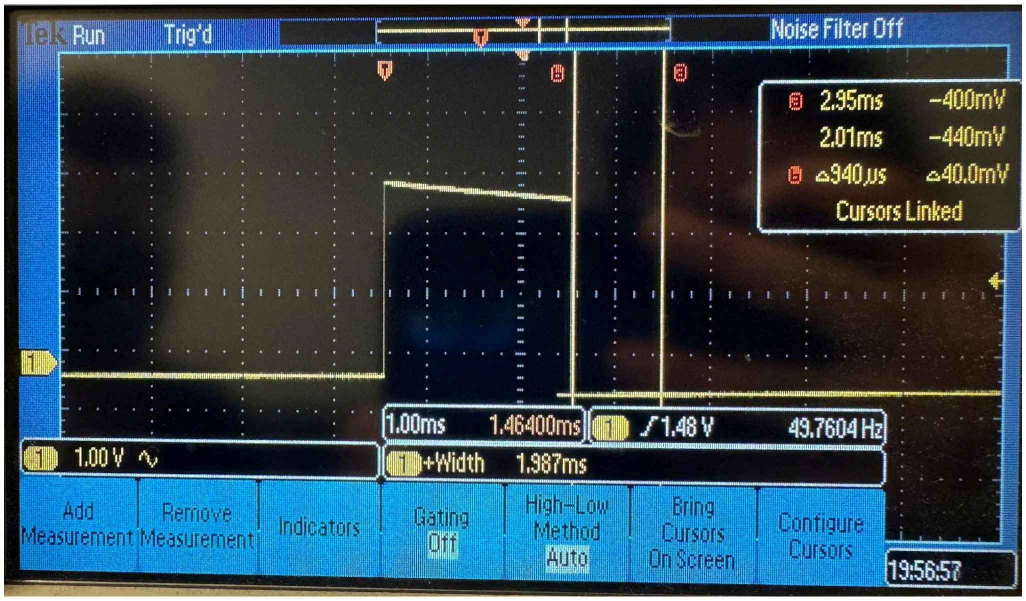
Once the LED pattern was connected to potentiometer changes, we focused on controlling the pulse on the RC Servo as below by using the potentiometer reading

```
pulse = ((reading/1023.0) * 1000.0) + 1000.0;
RC_SetPulseTime(RC_PORTV04, pulse);
```

The range of the servo was less than 360 degrees with about a 180 degree change as observed, although we did not have a protractor to measure. At the minimum of the potentiometer we observed that the width of the pulse signal was about 1.011 ms as observed below.



By passing in the highest potentiometer value we observed a pulse width of 1.987 ms.

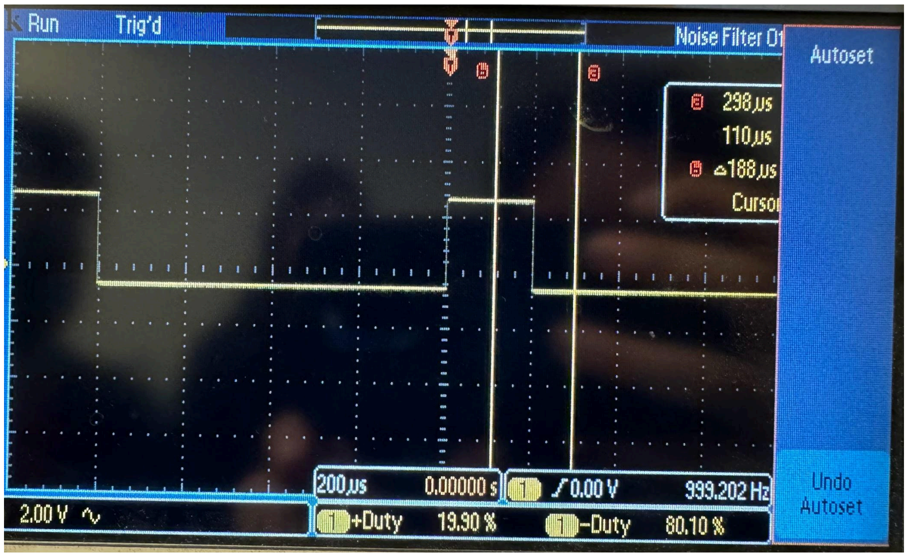


Part 2 - Unidirectional Drive of a DC Motor

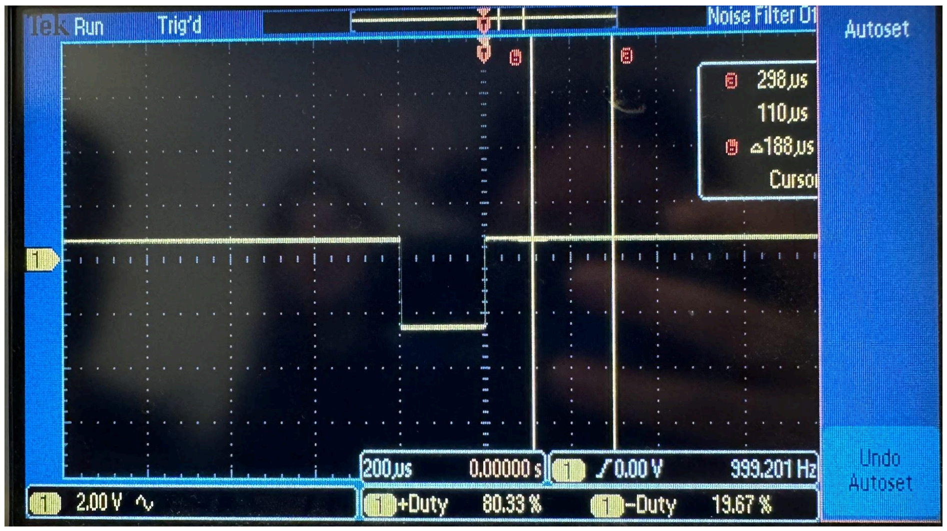
In this part of the lab we moved on to explore how to control a DC Motor in one direction as well as using LEDs to track the speed of the motor using a potentiometer. The same EMA filtering was applied as before to the potentiometer readings and the same bit masking to control the LEDs was used as well. This time instead of setting the pulse time to control the motor we set the duty cycle by scaling the potentiometer reading as below, to hold values between 1000-2000 as those were the parameters for the function. The range also represents the width of the pulse, 1us to 2us, that the DC motor would receive.

```
Duty = (reading/1023.0) * 1000.0;
PWM_SetDutyCycle(PWM_PORTZ06, Duty);
```

When the duty cycle was set to 20% we observed the response below.



When the duty cycle was set to 80% we observed the response below.



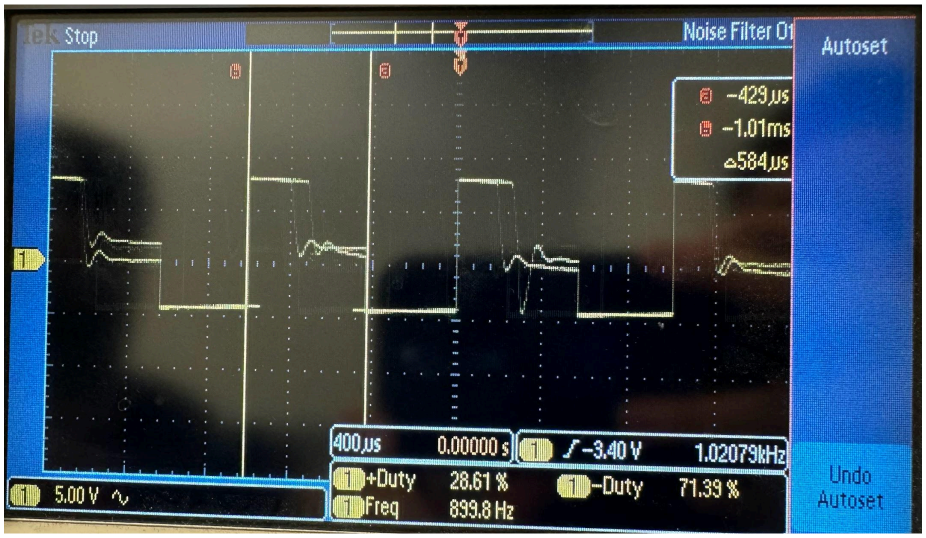
Changing the duty cycle of the PWM signal used to control the DC motor effectively changed the motor speed. Since average voltage delivered to the motor is proportional to the percentage of the time that the power supply is turned on during each PWM cycle, which corresponds to the duty cycle, an increase resulted in a faster rotation.

Part 3 - Snubbing The Inductive Kickback

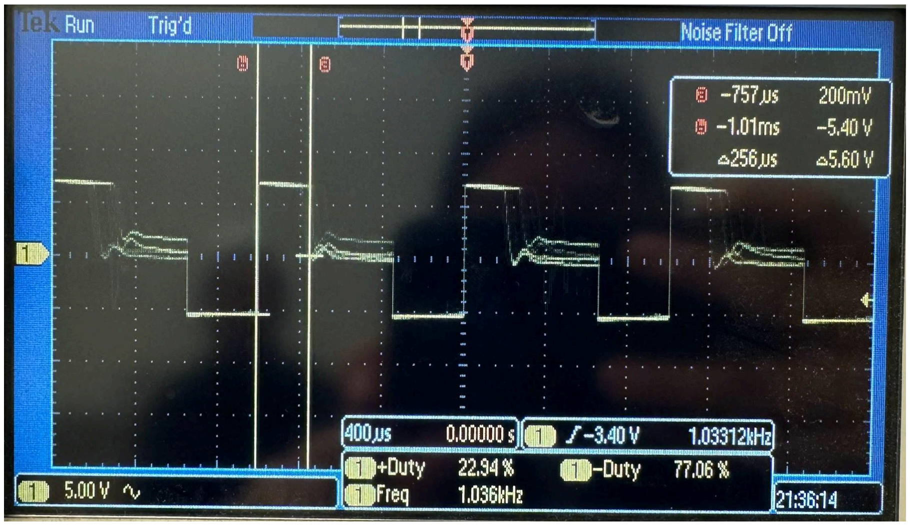
The majority of the setup for this part of the lab carried over from the previous part, leaving only the implementation of the two diode collector circuits. Both were assembled as instructed in the lab documentation, with the normal diode first and the zener diode second.

Upon applying resistive torque to the motor, an immediate effect occurs in the waveform of the circuit. The unstable, half discharge of the waveform stabilizes in proportion to the torque applied. As well, the maximum voltage across the measured component drops. This occurs in both diode circuits due to their application as collector circuits. The charge of the power supply builds up in the circuit as the motor is forced to slow down, which cuts down the amount of power the motor is able to discharge in comparison to the power supplied. As the motor approaches a stop with the resistive torque, the waveform levels out completely and more closely resembles a step function with a small amount of inductive kickback that is snubbed by the diode circuit.

Below is the trace of the zener diode implementation.



Below is a trace of the normal diode implementation.



Part 4 - Bidirectional Control of a DC Motor

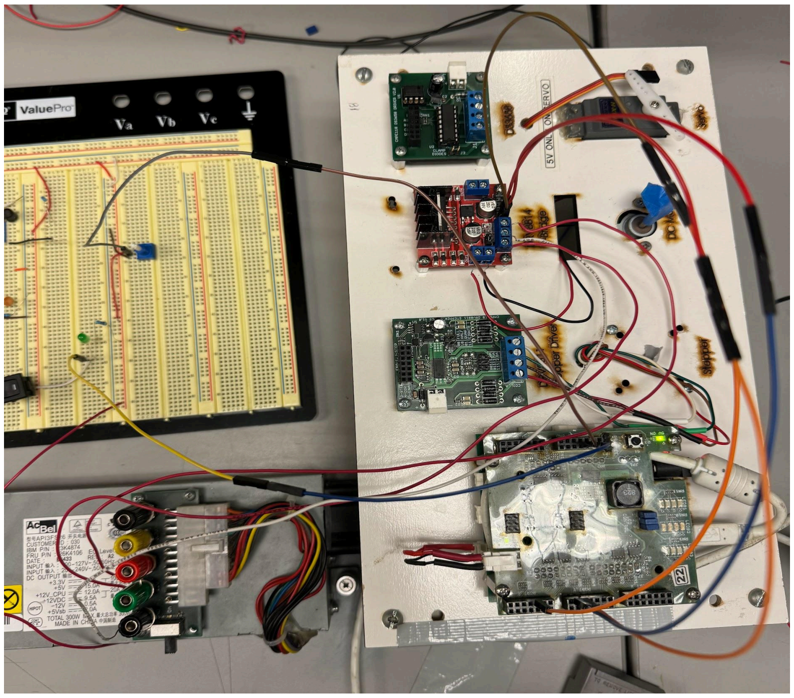
We continued to work on DC motors only this time by rotating the motor in both directions using a switch to control the direction. First we began by taking the input from the potentiometer and filtering the reading using the EMA filter from the previous parts. For the LEDs we used two banks, one for each direction and scaled the potentiometer reading by 4 values as there were only four LEDs in each bank to display speed.

After connecting the switch output to the UNO32 and verifying that it could control which LED bank was on, we moved on to use that direction marker to control the DC motor. If there was a connection determined, so the switch was on, then one of the pins controlling direction (IN1) would receive a high signal and the other pin (IN2) would receive a low signal therefore rotating to the right. If the connection was not made, so the switch was off, then a high signal would be sent to the opposite pin as before. IN2 would receive the high signal and IN1 would receive a low signal. Turning on and off the signals of the motors is what controls the direction of rotation of the shaft in the motor.

The control of the LEDs and the motos direction was done based on the switch output as seen below.

```
// Setting speed of motor
Duty = (reading / SCALEDIVIDE) * 1000.0;
PWM_SetDutyCycle(PWM_PORTZ06, Duty);

// External LED switch
switchval = AD_ReadADPin(AD_PORTV3);
if (switchval > 0) {
     connection = CLOSE;
} else {
     connection = OPEN;
}

if (connection) {
     LED_OnBank(LED_BANK1, speedPattern);
     LED_OffBank(LED_BANK1, invspeedPattern);
     LED_OffBank(LED_BANK2, 0x0F);
     LED_OffBank(LED_BANK3, 0x0F);

     IO_PortsSetPortBits(PORTY, PIN10);
     IO_PortsClearPortBits(PORTY, PIN12);

} else {
     LED_OnBank(LED_BANK3, speedPattern);
     LED_OffBank(LED_BANK3, invspeedPattern);
     LED_OffBank((LED_BANK1 | LED_BANK2), 0x0F);

     IO_PortsSetPortBits(PORTY, PIN12);
     IO_PortsClearPortBits(PORTY, PIN10);
}
```

Our setup was as below.



Part 5 - Control of a Stepper Motor

For the last two parts of the lab we explored controlling a stepper motor first with an H-Bridge as in this section and afterwards with a dedicated driver board. We started by connecting the driver motor using the predetermined pins as in the header file, Stepper.h. Once the pins were connected correctly we ran the test harness many times to observe how the motor would behave. After changing direction and “setRate” we understood how the motor would start behaving.

We moved on to implement the Half-Step drive using a state machine similar to the one already created for Full Step Drive. This was done based on the state machine depiction for the way the coils are activated or Half-Step drive using eight states as shown below.

```
void HalfStepDrive(void) {
    switch (coilState) {
        case step_one:
            // coil drive both forward
            COIL_A_DIRECTION = 1;
            COIL_A_DIRECTION_INV = 0;
            COIL_B_DIRECTION = 0;
            COIL_B_DIRECTION_INV = 0;
            if (stepDir == FORWARD) {
                coilState = step_two;
            } else {
                coilState = step_eight; ////
            }
            break;

        case step_two:
            // coil drive A forward, B reverse
            COIL_A_DIRECTION = 1;
            COIL_A_DIRECTION_INV = 0;
            COIL_B_DIRECTION = 1;
            COIL_B_DIRECTION_INV = 0;
            if (stepDir == FORWARD) {
                coilState = step_three;
            } else {
                coilState = step_one;
            }
            break;

        case step_three:
            // coil drive both reverse
            COIL_A_DIRECTION = 0;
            COIL_A_DIRECTION_INV = 0;
            COIL_B_DIRECTION = 1;
            COIL_B_DIRECTION_INV = 0;
            if (stepDir == FORWARD) {
                coilState = step_four;
            } else {
                coilState = step_two;
            }
            break;

        case step_four:
            // coil drive A reverse, B forward
            COIL_A_DIRECTION = 0;
            COIL_A_DIRECTION_INV = 1;
            COIL_B_DIRECTION = 1;
            COIL_B_DIRECTION_INV = 0;
            if (stepDir == FORWARD) {
                coilState = step_five;
            } else {
                coilState = step_three;
            }

            break;
        case step_five:
            // coil drive A reverse, B forward
            COIL_A_DIRECTION = 0;
            COIL_A_DIRECTION_INV = 1;
            COIL_B_DIRECTION = 0;
            COIL_B_DIRECTION_INV = 0;
            if (stepDir == FORWARD) {
                coilState = step_six;
            } else {
                coilState = step_four;
            }

            break;
        case step_six:
            // coil drive A reverse, B forward
            COIL_A_DIRECTION = 0;
            COIL_A_DIRECTION_INV = 1;
            COIL_B_DIRECTION = 0;
            COIL_B_DIRECTION_INV = 1;
            if (stepDir == FORWARD) {
                coilState = step_seven;
            } else {
                coilState = step_five;
            }

            break;
        case step_seven:
            // coil drive A reverse, B forward
            COIL_A_DIRECTION = 0;
            COIL_A_DIRECTION_INV = 0;
            COIL_B_DIRECTION = 0;
            COIL_B_DIRECTION_INV = 1;
            if (stepDir == FORWARD) {
                coilState = step_eight;
            } else {
                coilState = step_six;
            }

            break;
        case step_eight:
            // coil drive A reverse, B forward
            COIL_A_DIRECTION = 1;
            COIL_A_DIRECTION_INV = 0;
            COIL_B_DIRECTION = 0;
            COIL_B_DIRECTION_INV = 1;
            if (stepDir == FORWARD) {
                coilState = step_one;
            } else {
                coilState = step_seven;
            }

            break;
    }
}
```

The implementation for Wave drive is also based on the previously defined function for Full Step Drive using four states as shown below.

```
void WaveStepDrive(void) {
    switch (coilState) {
        case step_one:
            // coil drive both forward
            COIL_A_DIRECTION = 1;
            COIL_A_DIRECTION_INV = 0;
            COIL_B_DIRECTION = 0;
            COIL_B_DIRECTION_INV = 0;
            if (stepDir == FORWARD) {
                coilState = step_two;
            } else {
                coilState = step_four;
            }
            break;

        case step_two:
            // coil drive A forward, B reverse
            COIL_A_DIRECTION = 0;
            COIL_A_DIRECTION_INV = 0;
            COIL_B_DIRECTION = 1;
            COIL_B_DIRECTION_INV = 0;
            if (stepDir == FORWARD) {
                coilState = step_three;
            } else {
                coilState = step_one;
            }
            break;

        case step_three:
            // coil drive both reverse
            COIL_A_DIRECTION = 0;
            COIL_A_DIRECTION_INV = 1;
            COIL_B_DIRECTION = 0;
            COIL_B_DIRECTION_INV = 0;
            if (stepDir == FORWARD) {
                coilState = step_four;
            } else {
                coilState = step_two;
            }
            break;

        case step_four:
            // coil drive A reverse, B forward
            COIL_A_DIRECTION = 0;
            COIL_A_DIRECTION_INV = 0;
            COIL_B_DIRECTION = 0;
            COIL_B_DIRECTION_INV = 1;
            if (stepDir == FORWARD) {
                coilState = step_one;
            } else {
                coilState = step_three;
            }
            break;
    }
}
```

Once we implemented the remaining two drive modes we tested them iteratively to determine maximum rate at which we could step without losing any steps using a marker to determine its rotation past or before its original starting point. For Full Step drive we determined the maximum rate to be about 705. For Half Step drive, the maximum was 1370 and for Wave drive the maximum was much closer to Full Step at 680.

Part 6 - Stepper Motor Using Dedicated Board

For this last portion we continued working with stepper motors only this time with a dedicated board. This was connected following the DRV8811 Stepper Board documentation. Since the board will control the coils for us, we need to pass in a step wave signal. We achieved this with a state machine with two states that would drive the step low and then high continuously as shown below.

```
void DDrive(void) {
    switch(coilState) {
        case step_one:
            COIL_A_DIRECTION_INV = 1; // step
            COIL_A_DIRECTION = stepDir;
            coilState = step_two;
            break;
        case step_two:
            COIL_A_DIRECTION_INV = 0;
            COIL_A_DIRECTION = stepDir;
            coilState = step_one;
            break;
    }
}
```

After determining that the state machine outputs a step signal on the oscilloscope, we connected the signals from the UNO32 to the DRV8811 Stepper Board. To determine the maximum rate at which we could step without losing any steps, we started by testing various values. Below is a table of our findings for the max value based on different drive modes and various current limits on the coils

Drive Mode	Coil Current			
	0.5 Amp	1.0 Amp	1.5 Amp	2.0 Amp
Full Step	1729	1737	1745	1750
Half Step	3763	3765	3766	3768

The limits when using a dedicated board are much higher for both full and half step drives as compared to using an H-Bridge to drive the motor.