

SYSC 4805 B

Lab Section – L3

Lab 3

µC Communications, Motor Driver Board, and
Wheel Encoder

Group – 1

Group Members:

Raneem Abouseeta (101180464)

Aryan Huq Khan (101205385)

Rozba Hakam (101190098)

Ibrahim Faisal (101209598)

Ali Zaid (101223823)

Date: 27-09-2024

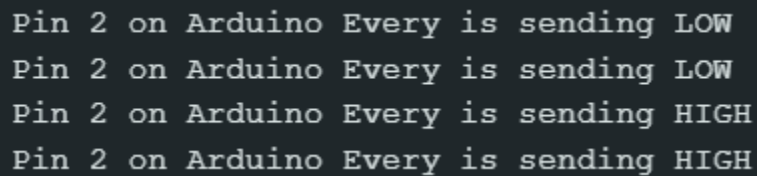
Due Date: 04-10-2024

Task 1: Communication Between Arduino Due and Arduino Nano Every

This section covers how the Arduino Due and Nano Every can communicate, which is super important when building more complex systems. For example, one board might collect sensor data while the other processes it or controls other devices. A key thing to watch out for is their different operating voltages—Due works at 3.3V, and Nano Every at 5V—so you need a logic level shifter to avoid damaging them. There are different ways they can talk to each other: simple Digital I/O for basic signals, UART for easy serial communication, SPI for faster data transfer, and I2C for connecting multiple devices with just two wires. The right choice depends on how fast and complex the communication needs to be. [3]

Experiment 1.1: Digital pins communication

1.1.4)



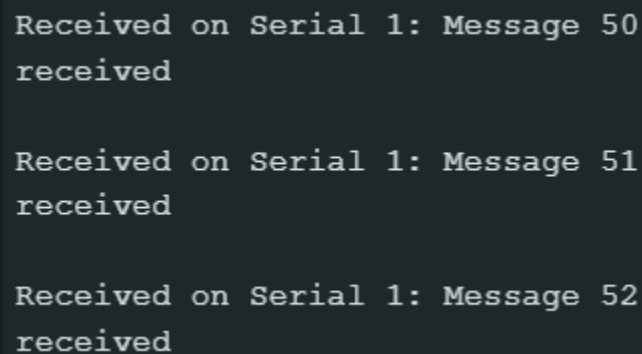
```
Pin 2 on Arduino Every is sending LOW
Pin 2 on Arduino Every is sending LOW
Pin 2 on Arduino Every is sending HIGH
Pin 2 on Arduino Every is sending HIGH
```

Figure 1: Serial Monitor Showing Digital I/O pins communicating [1]

Experiment 1.2: Bidirectional communication using UART

1.2.4)

Due:



```
Received on Serial 1: Message 50
received

Received on Serial 1: Message 51
received

Received on Serial 1: Message 52
received
```

Figure 2: Serial Monitor showing bidirectional communication using UART on a Due [1]

Nano:

```
Received on Serial 1: Hi from Arduino Nano Every #50  
Received on Serial 1: Hi from Arduino Nano Every #51  
Received on Serial 1: Hi from Arduino Nano Every #52
```

Figure 3: Serial Monitor showing bidirectional communication using UART on a Nano [1]

Experiment 1.3: Sending commands using I2C communication

1.3.5)

Due:

```
PWM value 49 applied on Arduino Nano Every.  
PWM value 41 applied on Arduino Nano Every.  
PWM value 48 applied on Arduino Nano Every.
```

Figure 4: Figure 4: Serial Monitor showing communication using I2C on a Due [1]

Nano:

```
Receivd PWM value: 49  
Receivd PWM value: 41  
Receivd PWM value: 48
```

Figure 5: Serial Monitor showing communication using I2C on a Nano [1]

1.3.6)

Waveforms

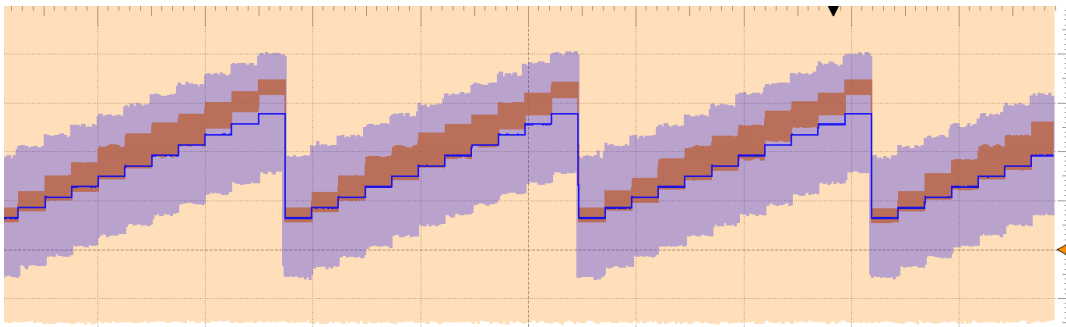


Figure 6: Waveforms showing Sawtooth like wave [2]

Low:

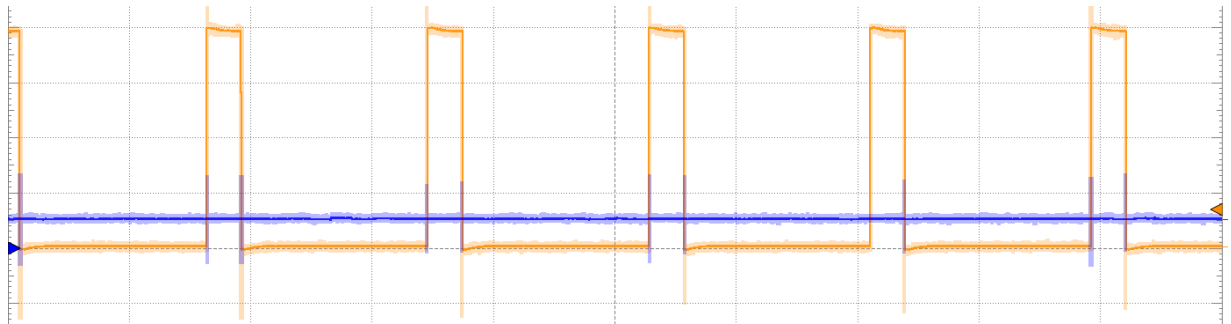


Figure 7: Waveforms showing low [2]

High:

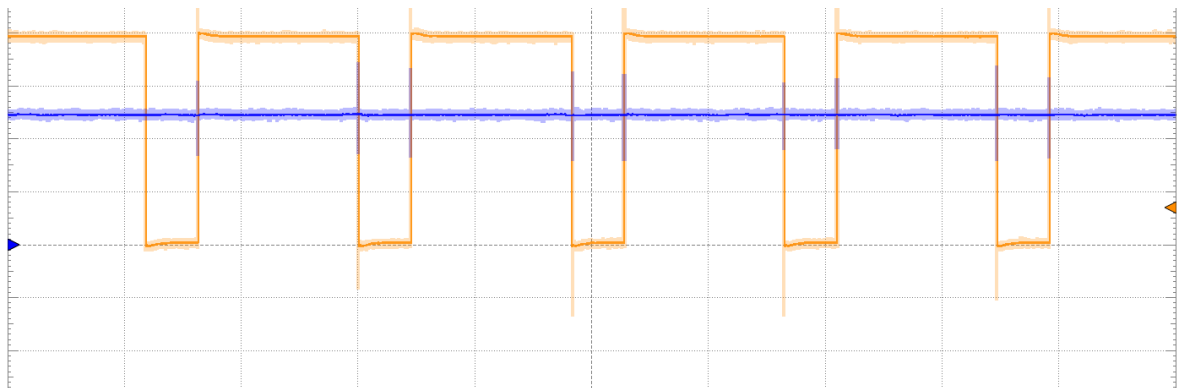


Figure 8: Waveforms showing high [2]

Discussion

1. What is the importance of the logic level shifter?

A logic level shifter is important for safely interfacing devices with different voltage levels.

2. Can a voltage divider be used instead? Why?

A voltage divider can be used, but it's less reliable for high-speed signals like I2C due to inconsistent voltage levels.

3. In Experiment 1.2, provide a sample code to use interrupts instead of using the “available” method.

Sample code for interrupts could use `attachInterrupt()` to handle events like data reception.

4. Does the I2C script for the Nano Every use interrupts? If yes, where exactly?

Yes, the I2C script for the Nano Every uses interrupts in the Wire library when handling data reception and transmission.

Task 2: Testing the Motor Driver Board

The Cytron motor driver board helps control up to four DC motors and operates between 7-25V with a current limit of 1.5A per channel. To avoid short circuits, it's important not to place it directly on metal, and you should use a logic level shifter to keep the motor power supply (5V) separate from the control signals (3.3V). The board uses H-Bridge circuits to control motor direction by adjusting the current flow. Motor speed is managed using PWM, where the duty cycle controls how much power the motor gets. Using a high PWM frequency keeps the motor running smoothly, taking advantage of its inertia to avoid jerky movements while adjusting speed and torque. [3]

Experiment 2.1: Testing the Motor Driver Board using the AD2

2.1.9) (a)

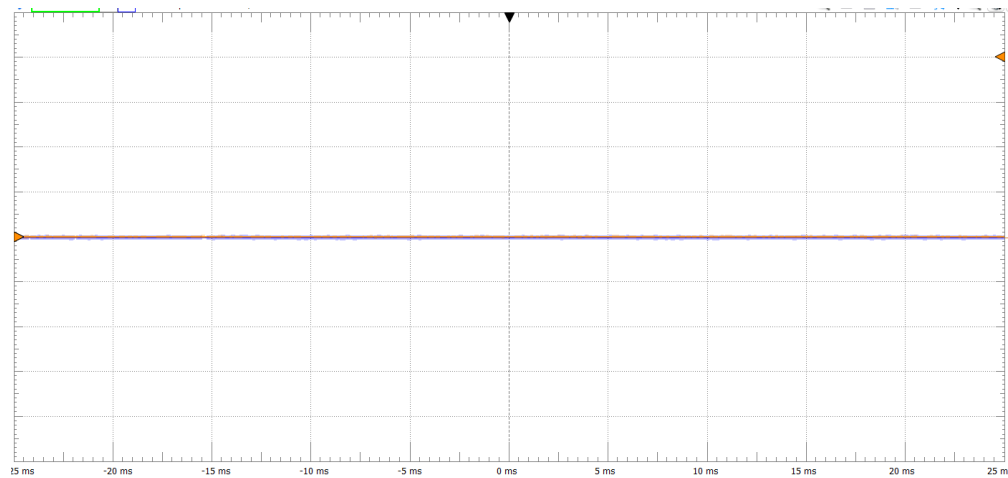


Figure 9: Symmetry of 0% with frequency of 100Hz [2]

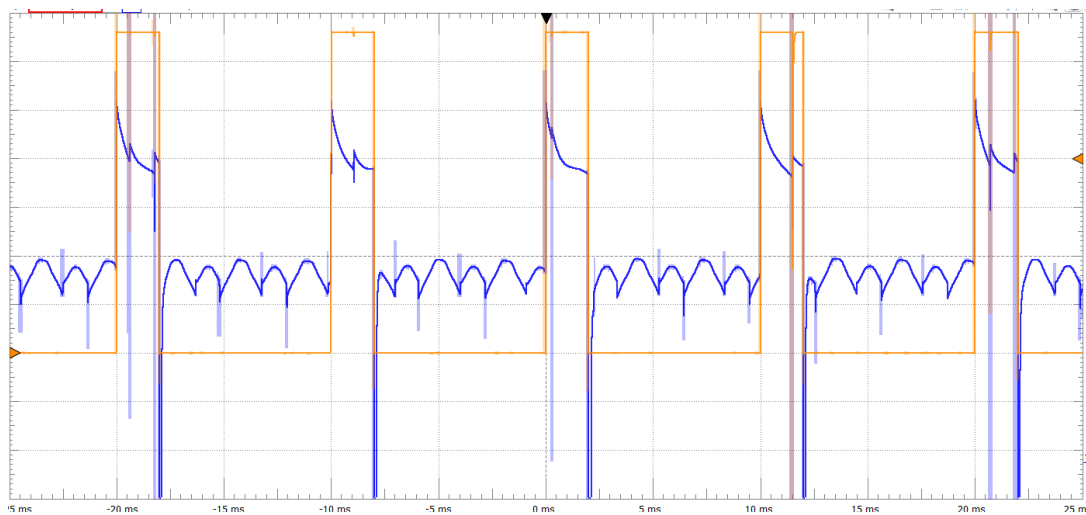


Figure 10: Symmetry of 20% with frequency of 100Hz [2]

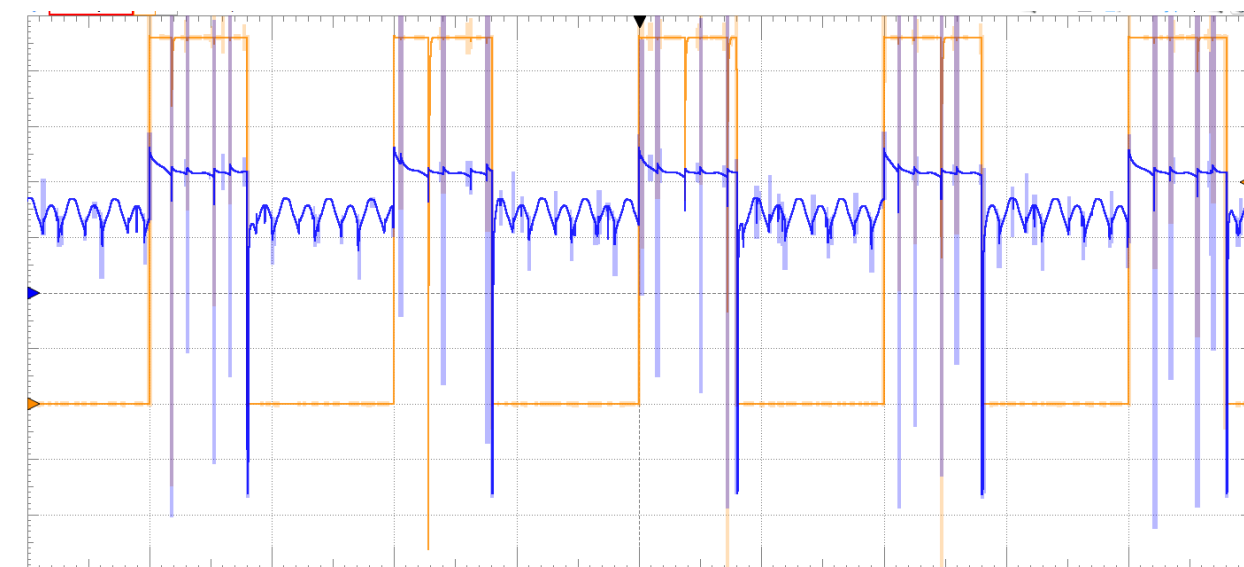


Figure 11: Symmetry of 40% with frequency of 100Hz [2]

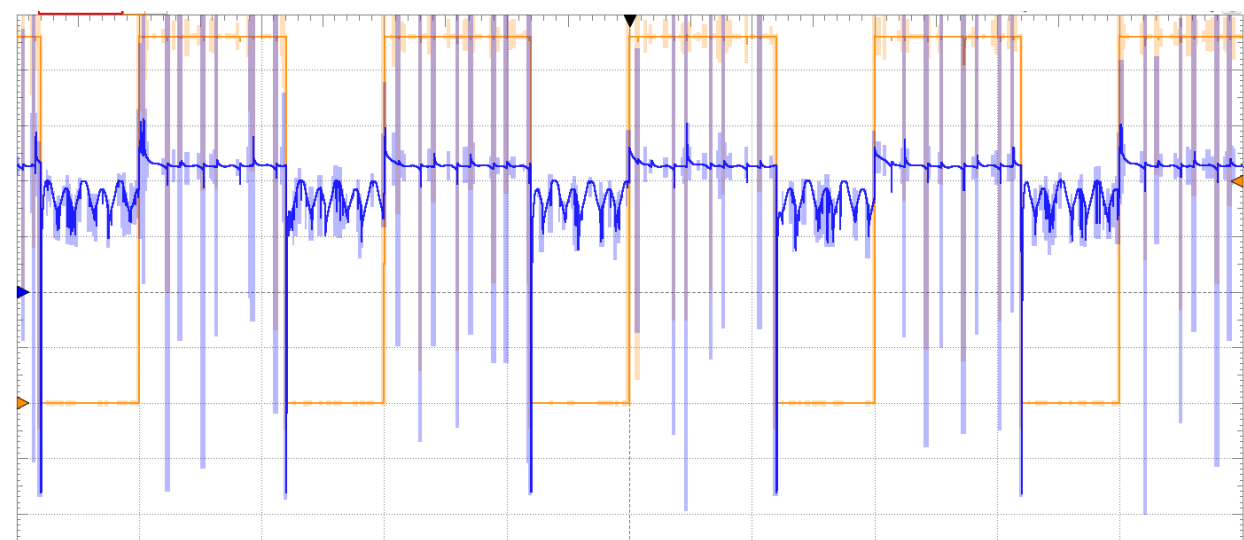


Figure 12: Symmetry of 60% with frequency of 100Hz [2]

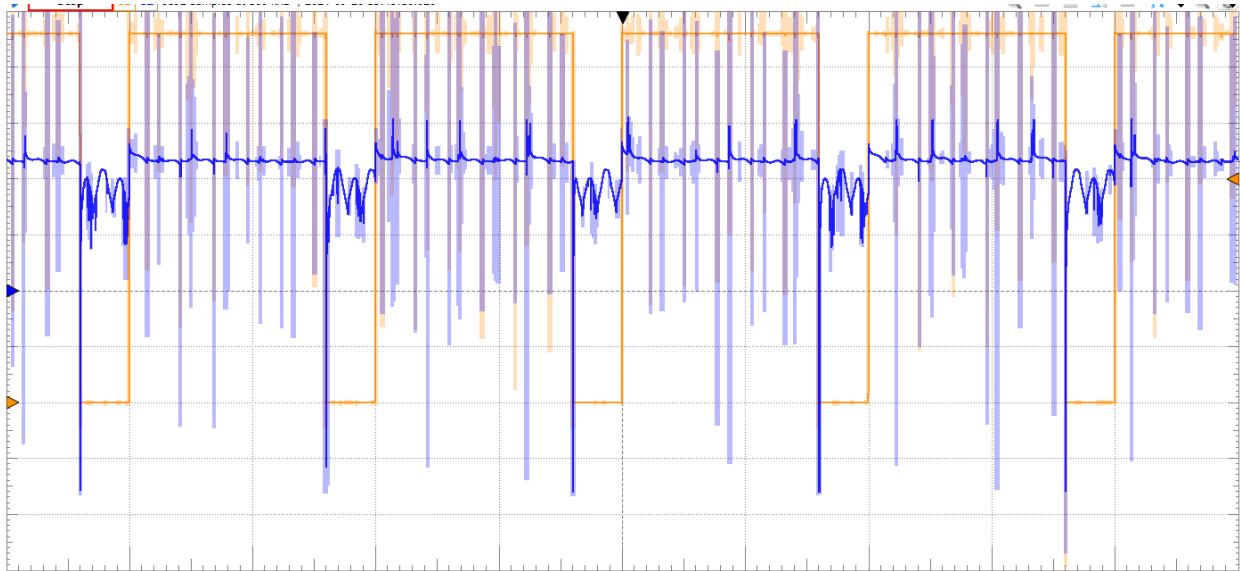


Figure 13: Symmetry of 80% with frequency of 100Hz [2]

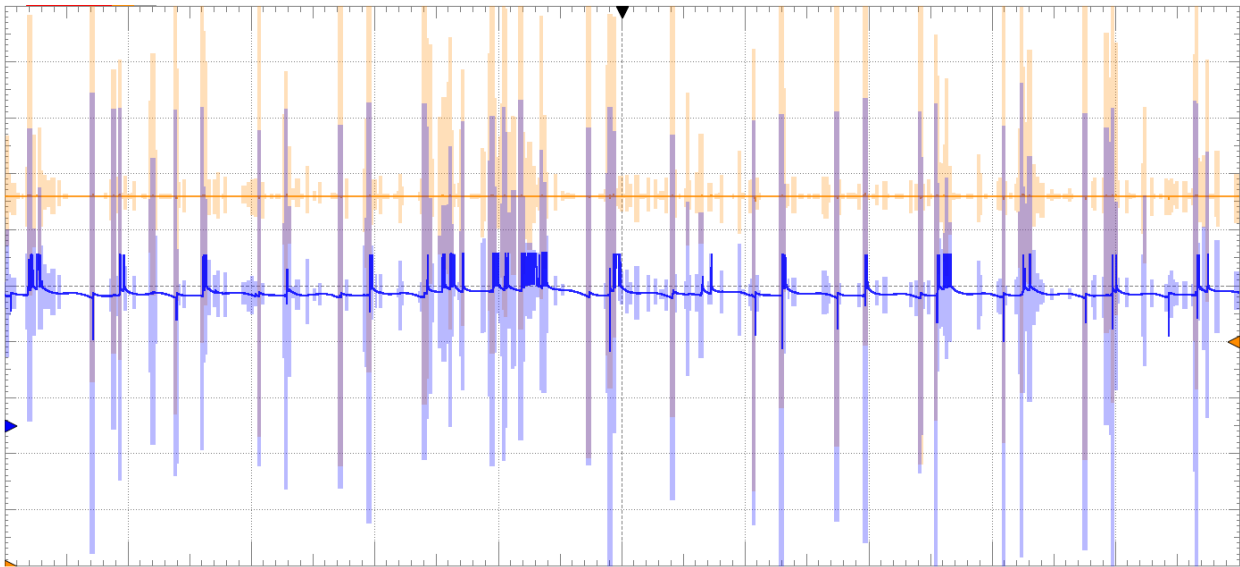


Figure 14: Symmetry of 100% with frequency of 100Hz [2]

What do you notice?

We notice that as the duty cycle increases there is increase in disruption in the channel 2 pulse.

(b)

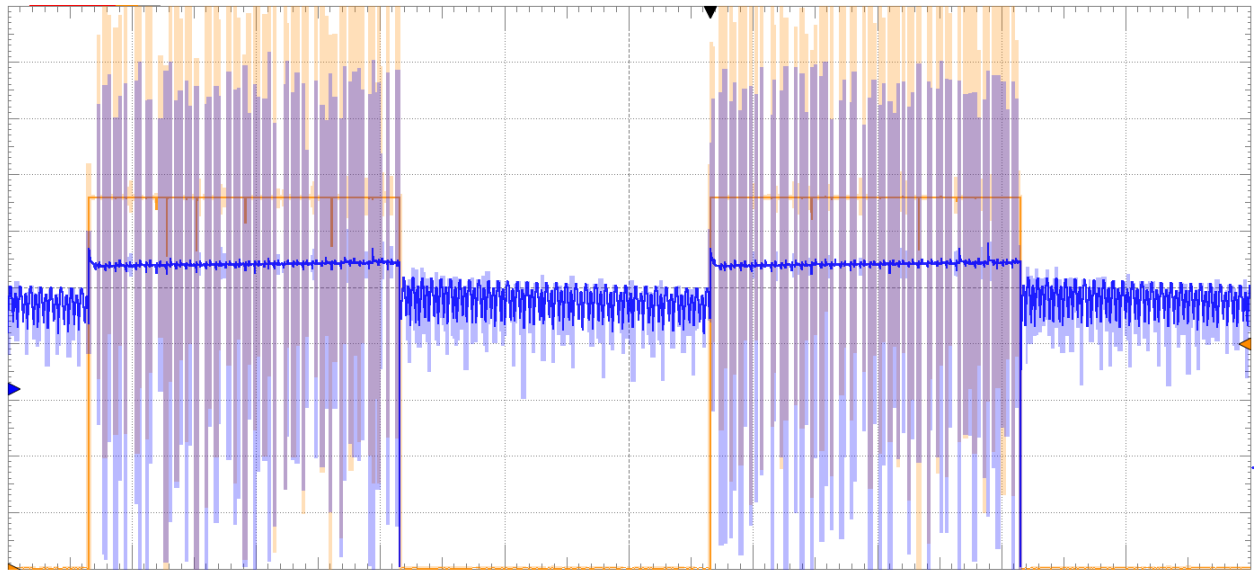


Figure 15: Symmetry of 50% with frequency of 10 Hz [2]

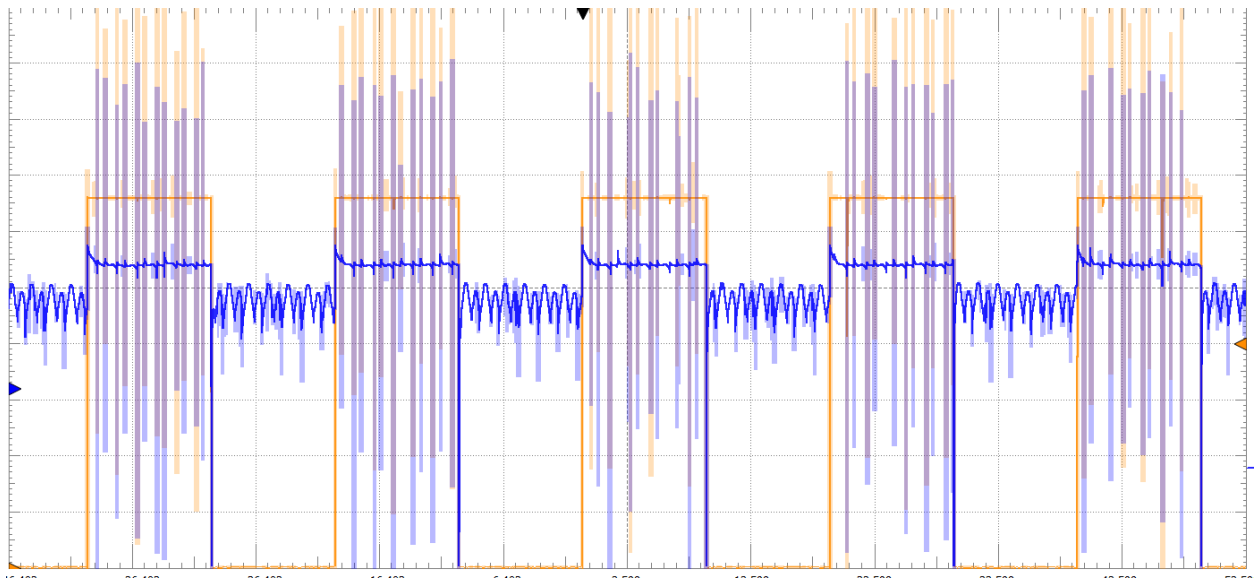


Figure 16: Symmetry of 50% with frequency of 50 Hz [2]

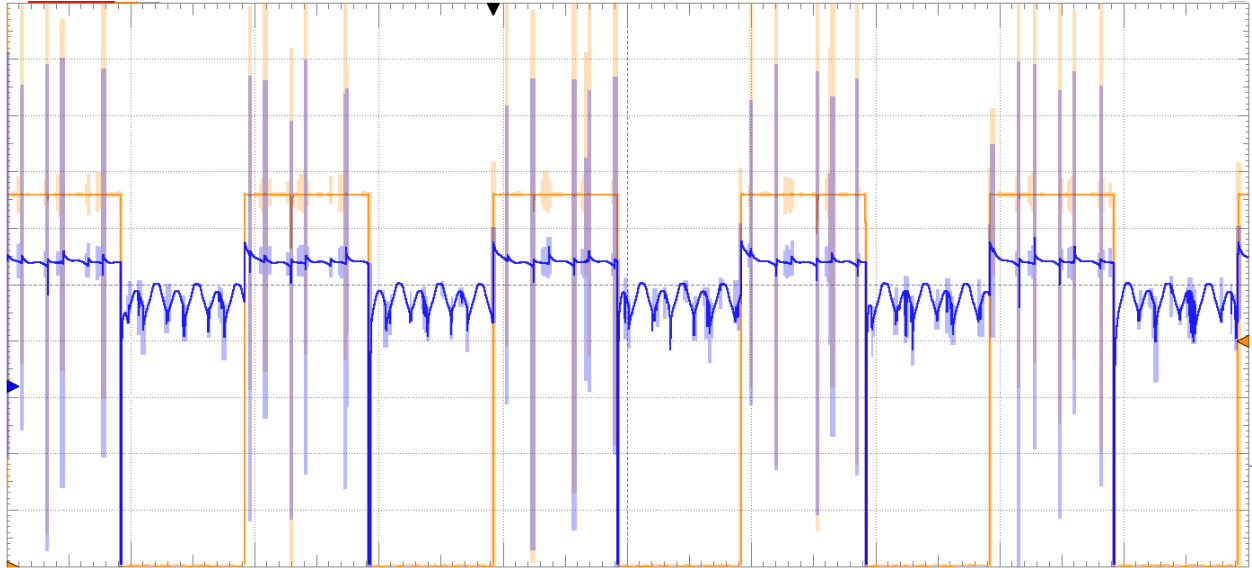


Figure 17: Symmetry of 50% with frequency of 100 Hz [2]

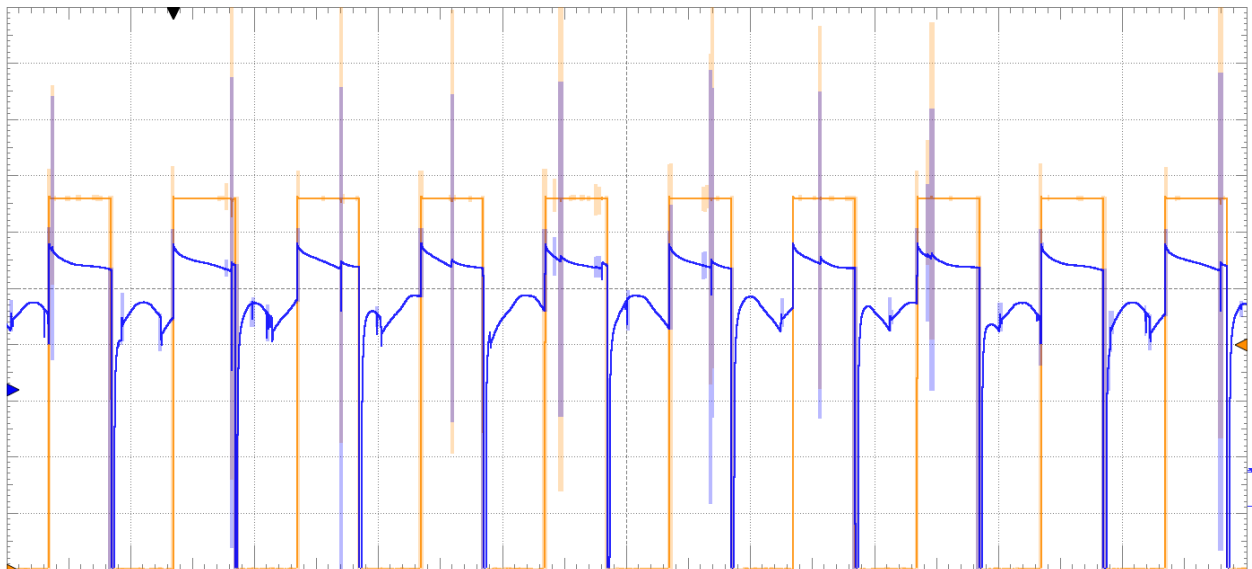


Figure 18: Symmetry of 50% with frequency of 500 Hz [2]

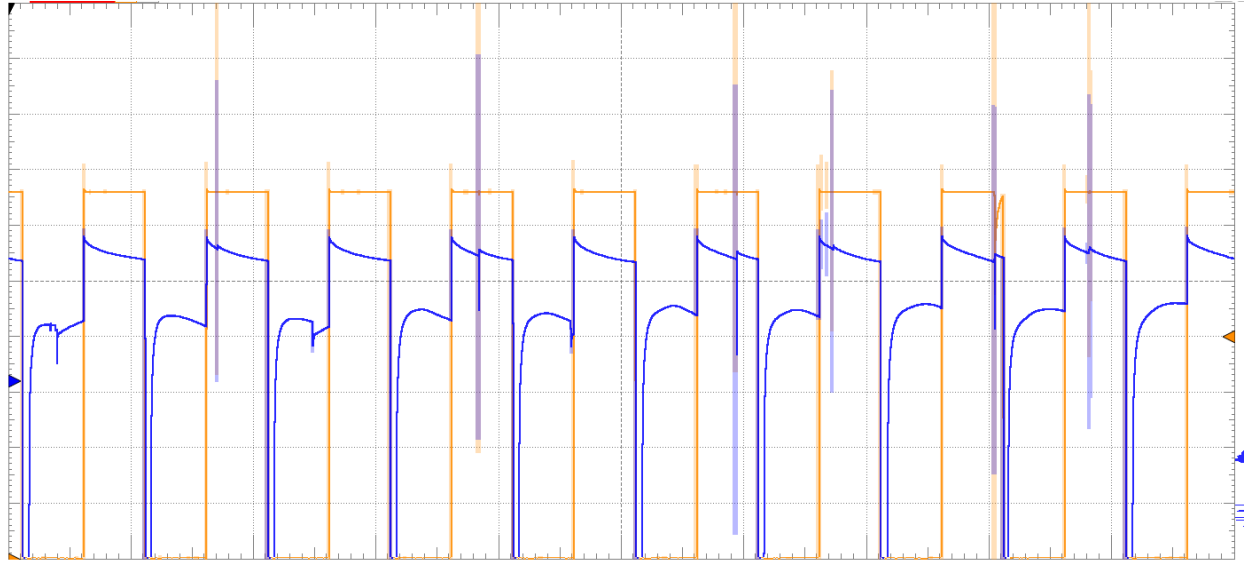


Figure 19: Symmetry of 50% with frequency of 1 kHz [2]

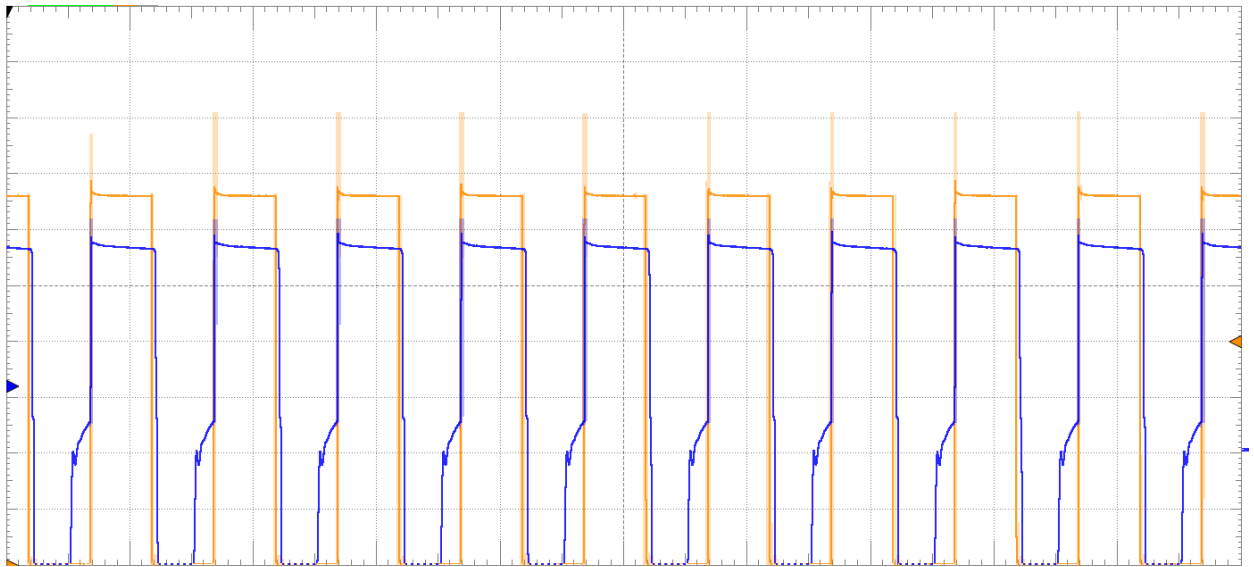


Figure 20: Symmetry of 50% with frequency of 20 kHz [2]

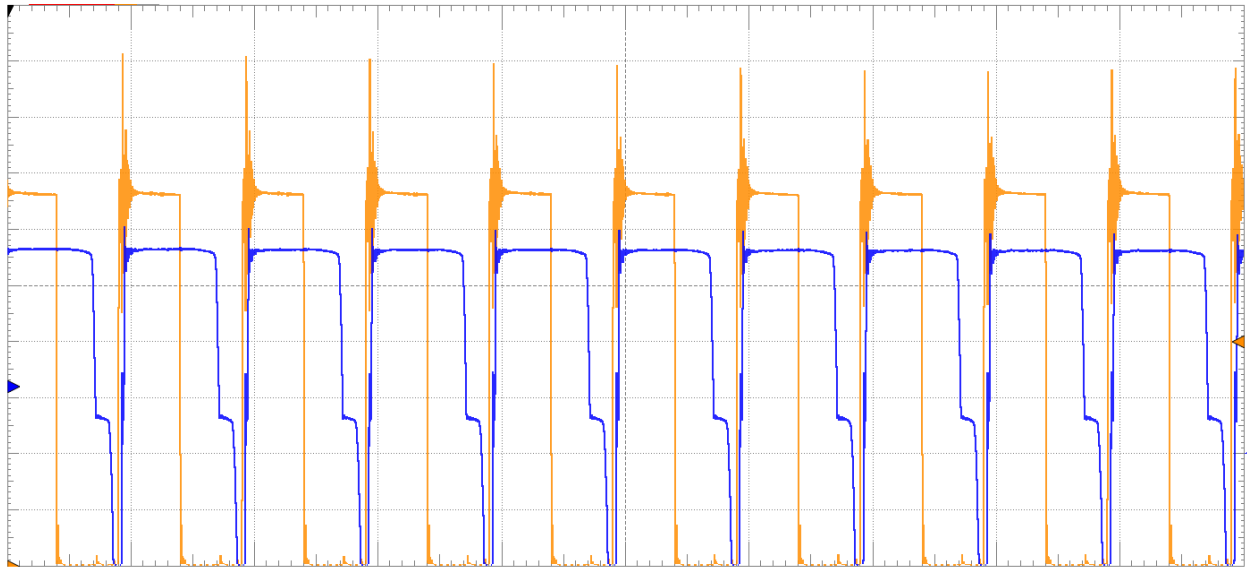


Figure 21: Symmetry of 50% with frequency of 200 kHz [2]

What do you notice?

We notice that at 20kHz and 200kHz there was no noise from the motor and as we increased the frequency, the noise and pulses also increased.

(c)

Write down in your report the frequency that leads to the highest speed under each select of the duty cycle

The frequency that leads to the highest speed under each select of the duty cycle is 50 Hz.

2.1.10)

Discussion Questions:

(a) At 20KHz, the Cytron motor driver board should still be providing the correct signal. You can check the motor driver board output on Ch 2 by reversing the M1 direction pin between 3.3V and Gnd which will flip the Ch 2 signal of the oscilloscope with respect to Ch 1. However, the motor may not be moving at all. Why?

At 20kHz, we have a high PWM frequency which shows a stable correct in the Waveform scope. Humans are not able to hear frequencies at 20kHz or higher and this frequency is only showing vibrations in the air which cannot be heard.

(b) You should notice that at 50% duty cycle, the 1Khz default frequency of the AnalogWrite() function does not lead to the fastest motor rotation, as the case with frequencies in the range 50 to 200 Hz. 10 Hz and below the motor is skipping, and above 300, the motor is not receiving the appropriate power. Which frequency gives the highest speed? Is it the same regardless of the duty cycle?

The motor usually runs fastest at frequencies between 50 and 200 Hz, but the exact frequency for the highest speed can depend on the duty cycle you're using. So, the best frequency might change if you adjust the duty cycle.

Task 3: Testing the Wheel Encoders

A motor encoder helps track how many times a motor shaft turns. It works by using a gear attached to the motor that blocks a light as it spins. A sensor detects these light pulses and converts them into an electrical signal, which matches the motor's speed. Some motors have built-in encoders, and more advanced ones, called quadrature encoders, can also tell the direction of rotation. By counting the rotations and factoring in time and wheel size, you can figure out the speed of a robot. However, if the wheels slip, the encoder might give accurate rotation counts but incorrect speed readings since the robot isn't moving forward. [3]

Experiment 3.1: Testing the wheel encoder using AD2

3.1.4) (a)

(i) Changing Duty Cycles

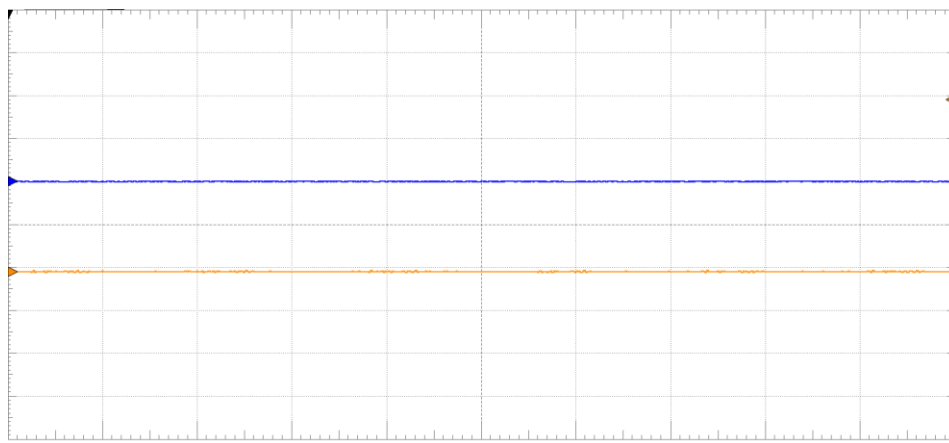


Figure 22: Symmetry of 0% with frequency of 100Hz [2]

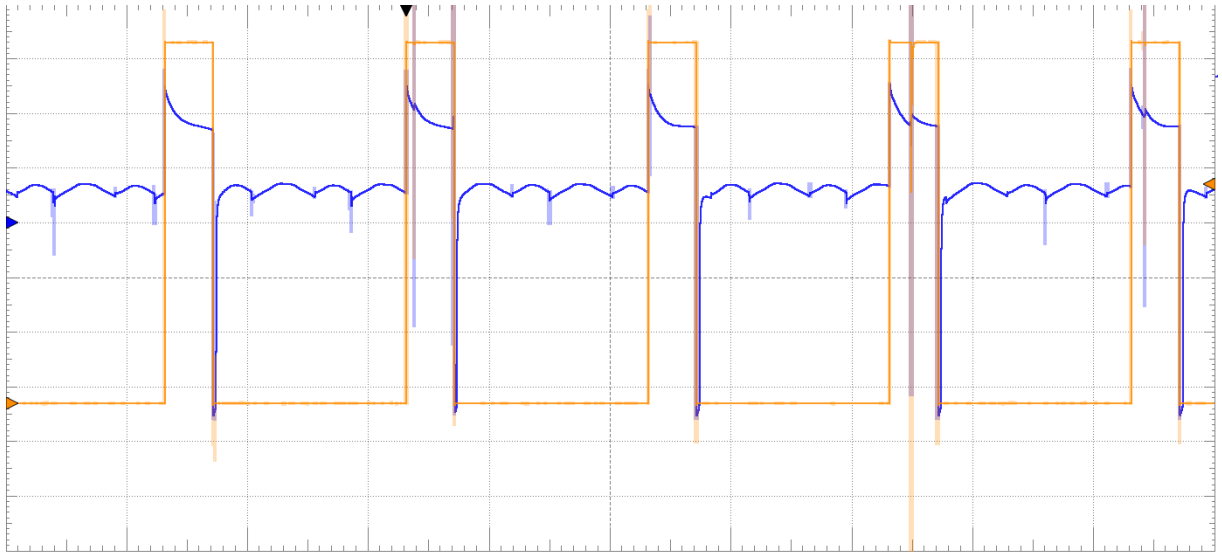


Figure 23: Symmetry of 20% with frequency of 100Hz [2]

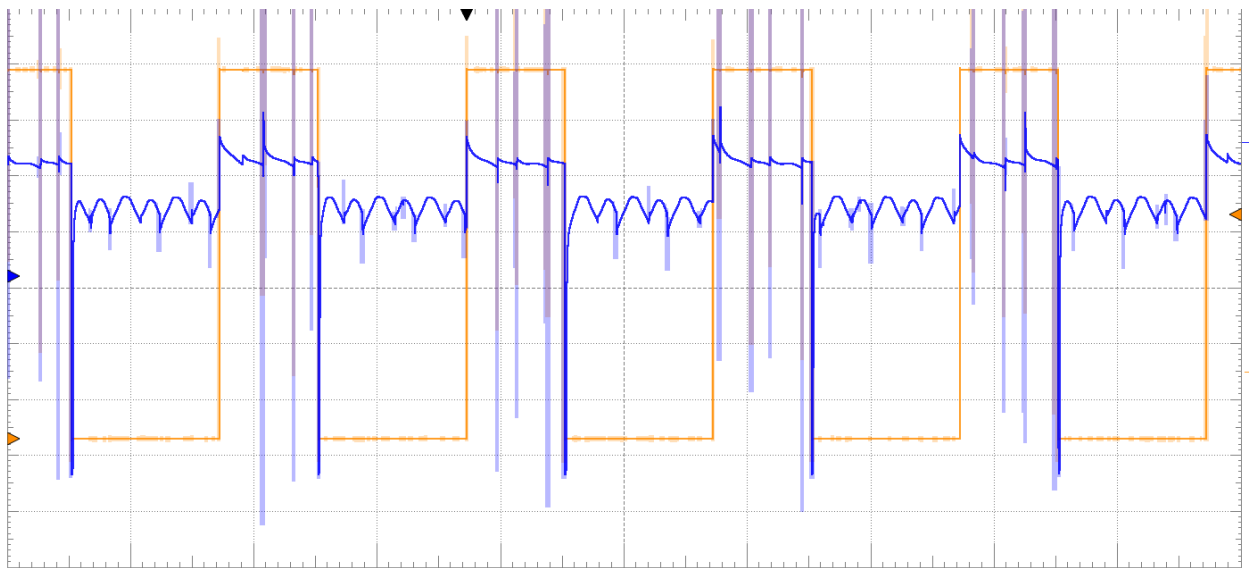


Figure 24: Symmetry of 40% with frequency of 100Hz [2]

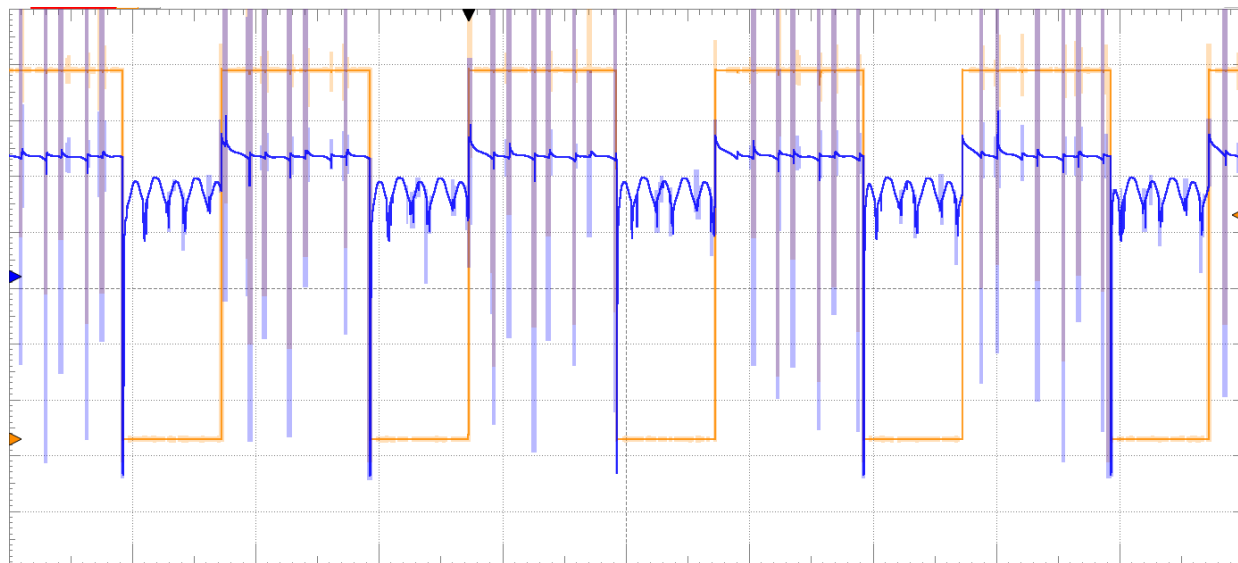


Figure 25: Symmetry of 60% with frequency of 100Hz [2]

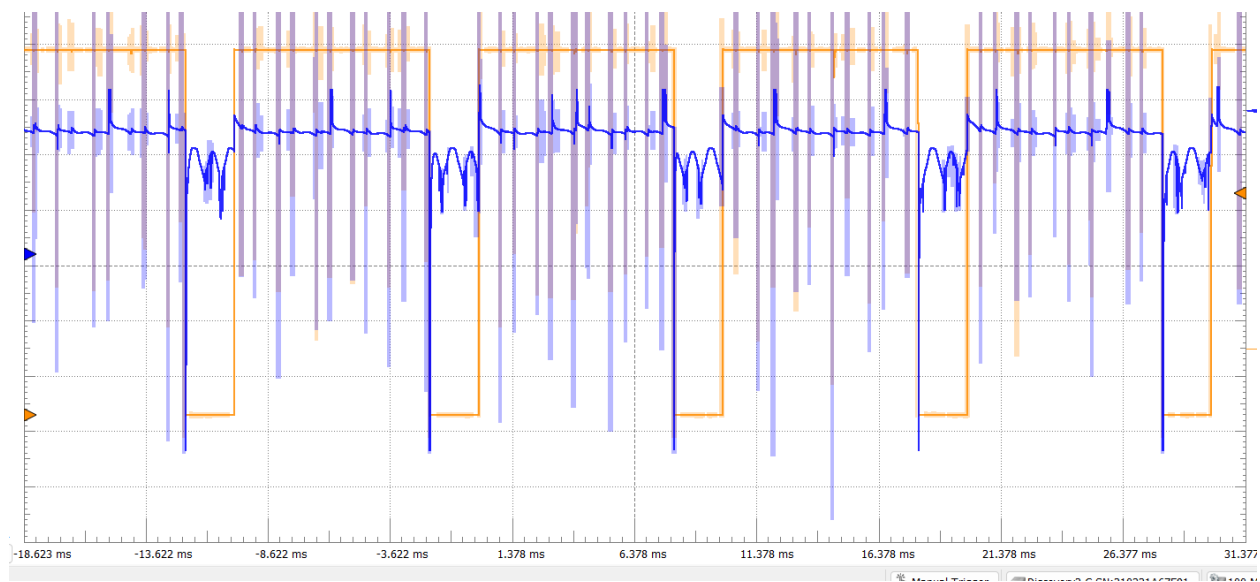


Figure 26: Symmetry of 80% with frequency of 100Hz [2]

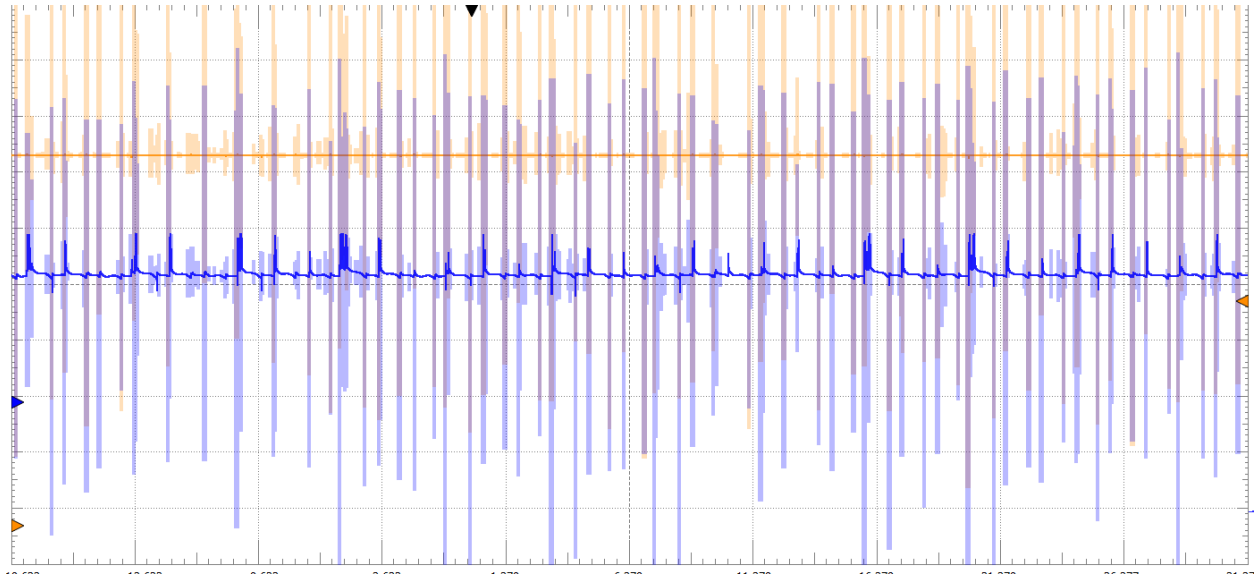


Figure 27: Symmetry of 80% with frequency of 100Hz [2]

(ii) Changing Frequency

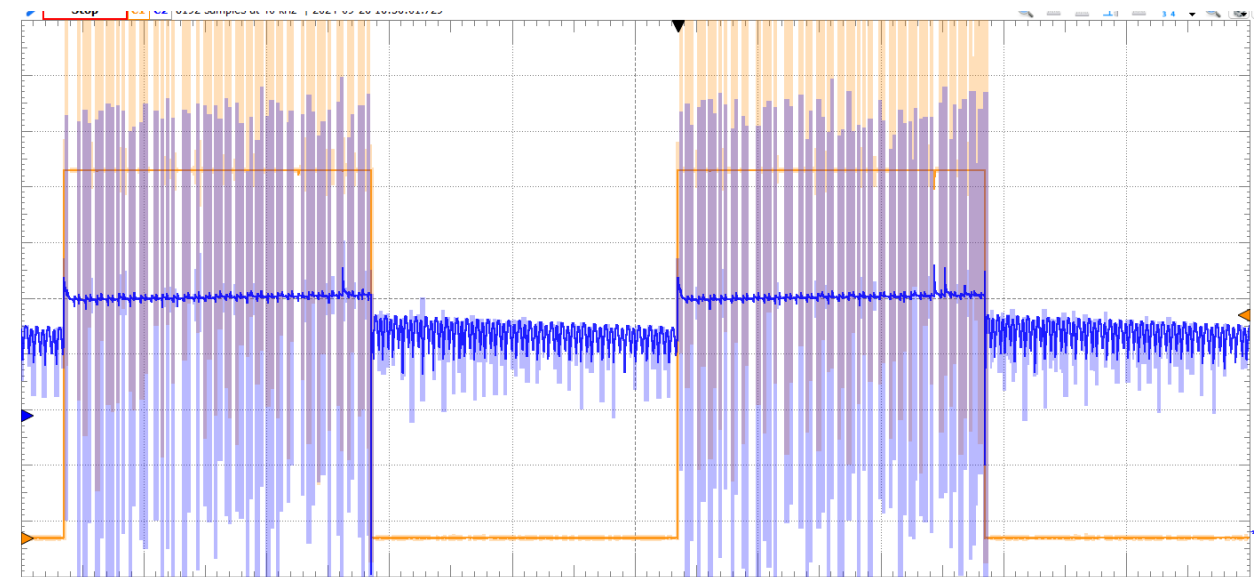


Figure 28: Symmetry of 50% with frequency of 10Hz [2]

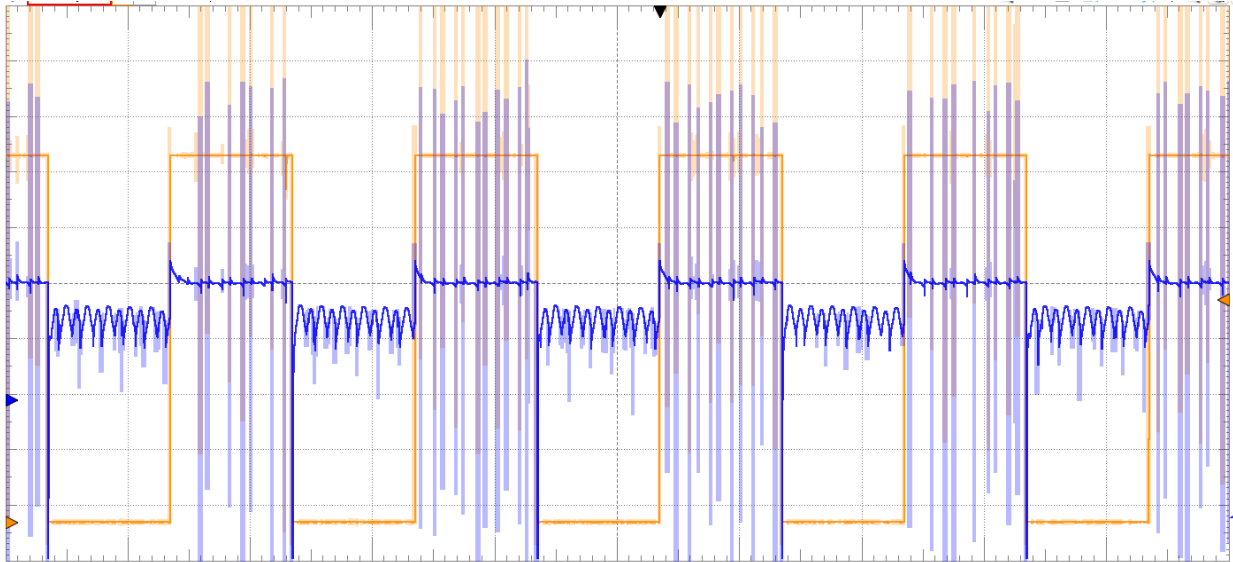


Figure 29: Symmetry of 50% with frequency of 50Hz [2]



Figure 30: Symmetry of 50% with frequency of 100Hz [2]

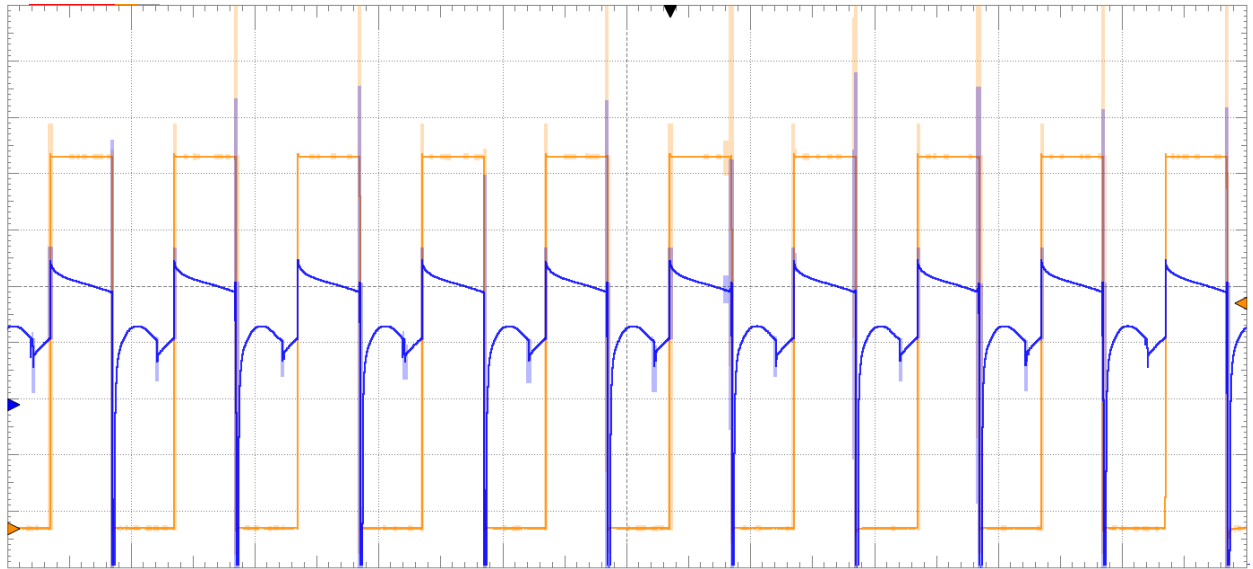


Figure 31: Symmetry of 50% with frequency of 500Hz [2]

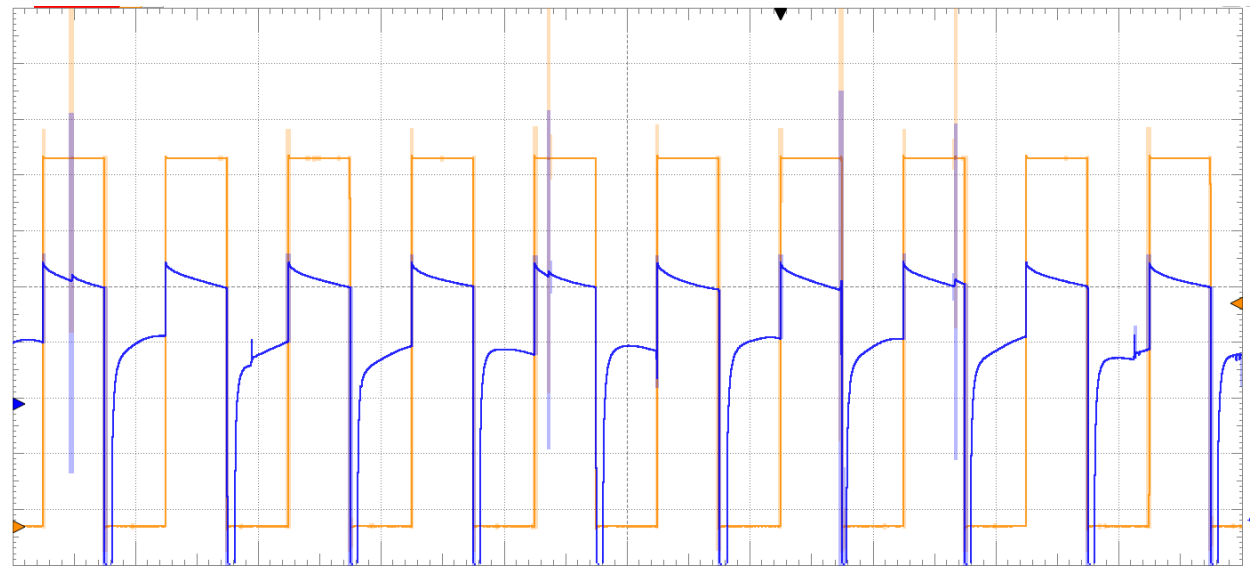


Figure 32: Symmetry of 50% with frequency of 1kHz [2]

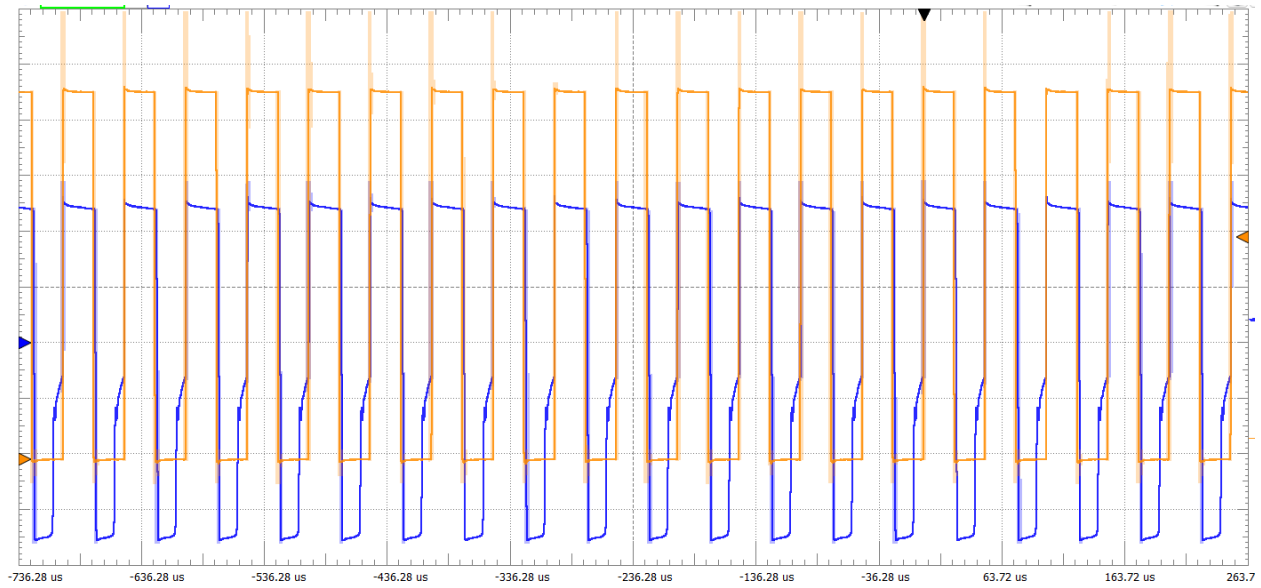


Figure 33: Symmetry of 50% with frequency of 20kHz [2]

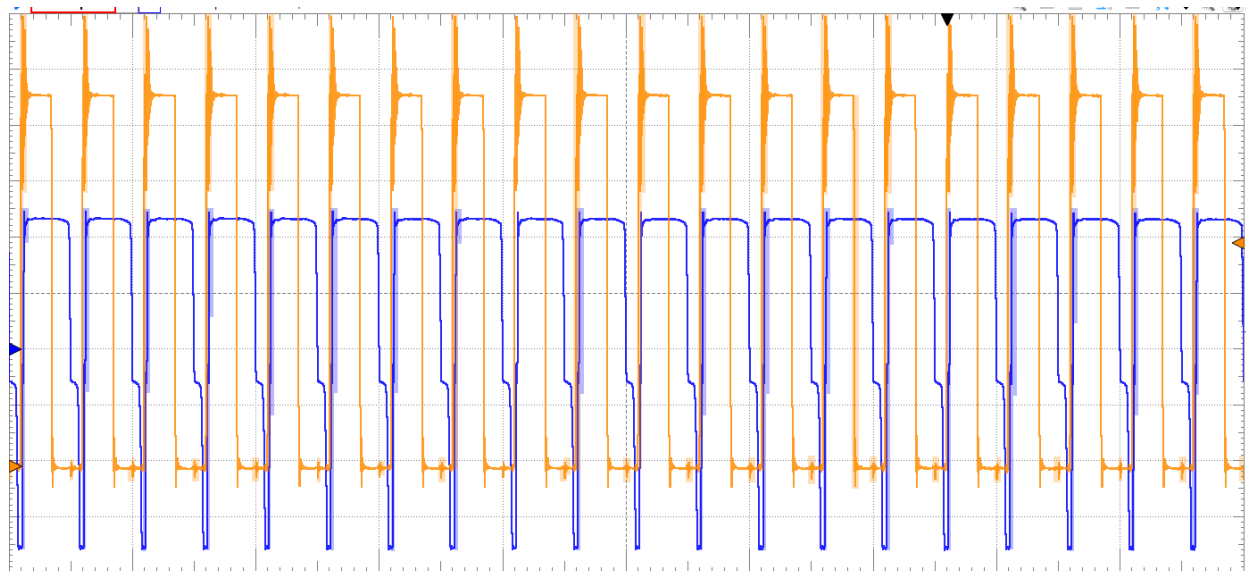


Figure 34: Symmetry of 50% with frequency of 200kHz [2]

(b) Radius of the Wheel = $r = 3.25\text{cm}$

Circumference = $C = 2\pi r$

$C = 2 \times 3.14 \times 3.25 = 20.42\text{cm}$

Smallest Period = $50.29 \times 10 = 0.503\text{ s}$

(c) Maximum Speed = $20.42/0.503 = 40.60$ cm/s

Experiment 3.2: Reading time using the Arduino Due Board

Experiment 3.2.2: Measuring time by using Interrupts

```
2
1
1
2
0
2
0
0
1
1
1
2
```

Figure 35: Measuring time using Interrupts without debouncing [1]

```
26
26
26
26
27
26
26
26
26
```

Figure 36: Measuring time using Interrupts with debouncing [1]

Experiment 3.2.3: Measuring time by the Timer Counter unit.

```
Section L3, Group 1: 0.000119 msec  
Section L3, Group 1: 0.000571 msec  
Section L3, Group 1: 0.806857 msec  
Section L3, Group 1: 0.000167 msec  
Section L3, Group 1: 0.000214 msec  
Section L3, Group 1: 0.000095 msec  
Section L3, Group 1: 0.934881 msec  
Section L3, Group 1: 1.351595 msec  
Section L3, Group 1: 0.000071 msec  
Section L3, Group 1: 0.483024 msec  
Section L3, Group 1: 0.000095 msec
```

Figure 37: Measuring time by Timer Counter Unit [1]

```
Section L3, Group 1: 1.790500 msec  
Section L3, Group 1: 0.002476 msec  
Section L3, Group 1: 0.002357 msec  
Section L3, Group 1: 0.000619 msec  
Section L3, Group 1: 0.000071 msec  
Section L3, Group 1: 0.000095 msec  
Section L3, Group 1: 0.000262 msec  
Section L3, Group 1: 0.003429 msec  
Section L3, Group 1: 1.406429 msec  
Section L3, Group 1: 0.000333 msec  
Section L3, Group 1: 0.002238 msec
```

Figure 38: Measuring time by Timer Counter Unit with Interrupts [1]

Experiment 3.3: Testing the wheel encoder using the Arduino Due Board

3.1.3)

Table 3.1: PWM parameters effect on unloaded motors speed

| Duty Cycle | F = 50Hz | F = 100Hz | F = 500Hz | F = 1kHz |
|------------|----------|-----------|-----------|----------|
| 0% | 0 | 0 | 0 | 0 |
| 40% | | | | |
| 80% | | | | |
| 100% | | | | |

```
Raw Count: 6.000000 ticks, Rotation Time: 0.000143 sec, Speed: 142940.000000 cm/s
Raw Count: 7.000000 ticks, Rotation Time: 0.000167 sec, Speed: 122520.000000 cm/s
Raw Count: 3.000000 ticks, Rotation Time: 0.000071 sec, Speed: 285880.000000 cm/s
Raw Count: 7.000000 ticks, Rotation Time: 0.000167 sec, Speed: 122520.000000 cm/s
Raw Count: 2.000000 ticks, Rotation Time: 0.000048 sec, Speed: 428820.000000 cm/s
Raw Count: 4.000000 ticks, Rotation Time: 0.000095 sec, Speed: 214410.000000 cm/s
Raw Count: 12.000000 ticks, Rotation Time: 0.000286 sec, Speed: 71470.000000 cm/s
Raw Count: 8.000000 ticks, Rotation Time: 0.000190 sec, Speed: 107205.000000 cm/s
Raw Count: 3.000000 ticks, Rotation Time: 0.000071 sec, Speed: 285880.000000 cm/s
Raw Count: 5.000000 ticks, Rotation Time: 0.000119 sec, Speed: 171527.984375 cm/s
Raw Count: 2.000000 ticks, Rotation Time: 0.000048 sec, Speed: 428820.000000 cm/s
```

Figure 39: Serial Monitor showing the high-speed readings due to noise [1]

As we took the readings, we noticed that as the duty cycle increased past 30%, our period decreased greatly. While at 20%, we obtained values which were acceptable. As we approached 40%, our values increased drastically as seen in the image. We assume that the noise from the encoder could have caused this issue.

We attempted a few options:

- We tried adding the debouncing code used in measuring time by using interrupts on the Arduino due board.
- We tried replacing the wires and ensured they were not blocking the wheel encoder
- We used the other wheel encoder and tested the output signal using the oscilloscope

Unfortunately, these options did not solve the issue we experienced with the measuring error which caused us not to record incorrect values.

Discussion

1. What are the causes of the bouncing behavior?

Bouncing happens because mechanical switches vibrate when they make or break contact.

2. Suggest a hardware circuit (provide a schematic and description) to overcome the bouncing issue.

In the hardware debouncing technique we use a SR Flip Flop to prevent the circuit from switch bounces. This is the best debouncing method among all. [4]

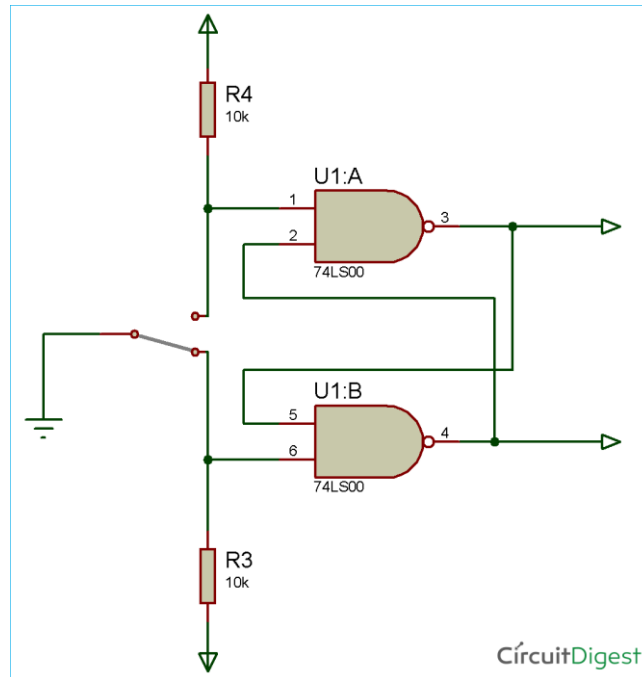


Figure 40: Schematic showing debouncing in hardware [4]

3. Can the encoder used in this experiment detect the rotation direction? If yes, explain how. If no, please provide an example of an encoder that can be used to detect the rotation direction.

Yes, the encoder can detect direction using two signals (A and B channels) to see which one comes first.

Task 4: Completing the Robot setup and taking a first drive.

Experiment 4.1: Testing the loaded motors

Table 3.1: PWM parameters effect on unloaded motors speed

| Duty Cycle | F = 50Hz | F = 100Hz | F = 500Hz | F = 1kHz |
|------------|----------|-----------|-----------|----------|
| 0% | 0 | 0 | 0 | 0 |
| 40% | | | | |
| 80% | | | | |
| 100% | | | | |

Experiment 4.2: A 0-radius 90° turn

4.2.1)

- Distance between left and right wheels $B = \dots$
- For χ of 1.48, the value of $X_0 = \dots$
- The arc length of quarter a circle with radius $X_0 = \dots$
- The wheel diameter $R = \dots$
- The wheel circumference =
- The number of revolutions per the quarter circle arc distance (per a 90° turn) =
- The number of pulses per revolution (encoder #teeth) =
- The number of encoder pulses per a 90° turn =

References

[1] – Docs.arduino.cc, <https://docs.arduino.cc/software/ide/#ide-v2>.

[2] – “Digilent waveforms,” Digilent, <https://digilent.com/shop/software/digilent-waveforms/>.

[3] - <https://brightspace.carleton.ca/d2l/le/content/292069/viewContent/3957421/View> (Lab Manual)

[4] - <https://circuitdigest.com/electronic-circuits/what-is-switch-bouncing-and-how-to-prevent-it-using-debounce-circuit#:~:text=1.,best%20debouncing%20method%20among%20all>.