Question 1 - Encoders

Nitin G EE22B041

March 2023

1 Incremental Quadrature Encoder

1.1 Part A

Devise a way to input the phase A and B signals and count the number of ticks in a fixed time interval (T) of your choice. Publish this count through the Serial Monitor after every T units of time. Explain your implementation in brief.

The implementation of this can be seen in the *wokwi* file: `https://wokwi.com/projects/359837630782072833`.

Here we have attached the channel A and channel B from the encoder to the pins 2 and 3 on the arduino uno. This is because Interrupts work only on pins 2 and 3 in the arduino uno.

In the `setup()` function, we initialise the pins to INPUT mode. We also initialise the interrupt using the `attachInterrupt()` function. What this does is, essentially sets up an interrupt which activates every time the signal from the pin changes (or rises or falls, in this case it is set to falling. So the function is called once every cycle).Everytime this happens, it calls the function passed to it. Here we pass Channel A to the interrupter. In this case, it calls a function which adds +1 to the tics counter in the case of clockwise rotation and -1 in the case of anti-clockwise rotation. To determine if it is clockwise or anticlockwise, we use the function `getdirection()`. It compares the value of the Channel B when the interrupt is called, i.e when the Channel A signal is in the falling edge.By design of the encoder, Channel A has a phase difference of +90 degrees with the Channel B when the rotation is clockwise. So, at the falling edge, Channel B has a HIGH value. And similarly, if Channel B has a LOW value, then it is rotating anticlockwise direction. Using this, we can read the value at the Channel B and determine the direction. If it is clockwise then we add +1, else -1. That is, we are taking the convention to be that clockwise is positive.

Then, in the loop we print the number of tics and then reset it to 0. Then we have a delay of T milliseconds. Therefore this will measure the number of tics in the fixed time interval T. And it will be printed every T milliseconds. This works only because the interrupt works regardless of whether there is a delay() or not. That is, even when a delay() is called, Interrupt will still trigger the function on a falling edge.

We also note the reason behind our choice of T here: We need a small enough value so that our average is almost real time. But we also need a fairly large time interval, i.e it cant be one millisecond so that our system will be stable. For this we use T to be around 200 ms.

1.2 Part B

Derive an expression that uses this tick count and time interval T to give the Rotations per Minute (rpm) of the motor shaft the encoder is attached to. Mention any other variables required and/or suitable assumptions that you make in the process.

$$v_{avg} = \frac{tickcount}{T} \times \frac{60}{n} \tag{1}$$

We need to assume here a new variable,n, that is the number of pulses that the encoder will create in one revolution. We shall assume this to be 1024. And now we can use this formula to calculate the instantaneous RPM of the device.

2 Rotary Encoder

Assume you have a similar encoder with 10 output bits (2^0 to 2^9), which gives an output 0 or 1 compatible with Arduino Uno logic levels. Describe how you would use the Arduino UNO to find the value of the absolute position (Between 0 and 1023).

We get 10 bits of input data from the absolute encoder. The encoder used in Bolt and Vikram can output data in various binary codes, as we infer from the datasheet. It can output both BCD, Binary (the normal one where it just counts up to 1023 from 0 as we flip bits in the order where its binary value increases by one each time. Basically just normal binary), and gray codes. Gray codes are the most efficient here as in the real world, when the contacts in the encoder are shifting, they dont all shift at the same time, in case of binary this will make it look like there is a sudden shift from one position to another totally far away position suddenly before it comes back to the correct position. This could break our system. This is the problem that the gray codes solve by smartly defining the coding such that this wont happen. This works because, there is only change in one bit from one encoding to the next encoding.

We here assume that the bits are outputed in the binary coding only. Now we can simply read these bits and multiply each one with its corresponding power of 2 and add it all up. Simple stuff.

For bonus points, you could simulate the code on some simulation platform like `https://tinkercad.com` or `https://wokwi.com/` by using

toggle switches to represent a 0 or 1 data level and input it to an Arduino UNO and calculate the position using your above implementation.

The Simulation can be found at: `https://wokwi.com/projects/359845294587368449`

The rest of the questions are answered in ipynb files and python scripts which are attached in the github repo mentioned below.

2.1 Part D

In [ ]:
```
#We encode the faulty datasets from decimal to binary to analyse the data
num_pins = 10
def encode_binary(num, n):
    return bin(num)[2:].rjust(n, '0')


f1 = open("partd_dataset_fault1.txt", 'r')
f2 = open("partd_dataset_fault2.txt", 'r')

op1 = open("partd_encoded_fault1.txt", 'w')
op2 = open("partd_encoded_fault2.txt", 'w')

lines1 = f1.readlines()
num_lines1 = len(lines1)
lines2 = f2.readlines()
num_lines2 = len(lines2)

for i in range(num_lines1):
    encoded_num = encode_binary(int(lines1[i]), num_pins)
    op1.write(encoded_num + '\n')

for i in range(num_lines2):
    encoded_num = encode_binary(int(lines2[i]), num_pins)
    op2.write(encoded_num + '\n')
```

In [ ]:
```
#Checking if one of the bits is always stuck at a constant value, indicat
def checkStaticBit(f, num_pins):
    data = f.read().split()
    bits_info = []
    for _ in range(10): #because there are 10 bits, we want to store valu
        bits_info.append([])
    for i in range(len(bits_info)):
        for d in data:
            bits_info[i].append(int(d[i]))


    for i in range(len(bits_info)):
        #print(bits_info[i])
        if all(bits_info[i]):
            print(f'pin {num_pins - i} has a faulty connection. It only s

        elif not(any(bits_info[i])):
            print(f'pin {num_pins - i} has a faulty connection. It only s
```

In [ ]:
```
f1e = open("partd_encoded_fault1.txt", 'r')
checkStaticBit(f1e, 10)
```

```
pin 9 has a faulty connection. It only shows value of 1.
```

Trying something with f2e. trying to see the gray equivalent too see if the problem is because one contact pin touches the next before the other contact pin does, which is the exact problem that graycodes solve.

In [ ]:
```
def bintogray(bin_num): #bin num is a str
    graynum = bin_num[0]
    for i in range(1, len(bin_num)):
        graynum += str(int(bin_num[i-1]) ^ int(bin_num[i]))
    return graynum


def checkSomething(f, num_pins):
    data = f.read().split()
    graydata = list(map(bintogray, data))
    return graydata
```

In [ ]:
```
f2e = open("partd_encoded_fault2.txt", 'r')
filey = open('partd_fault2_graycode.txt', 'w')

newdata = checkSomething(f2e, 10)
for line in newdata:
    filey.write(line + '\n')

fileyread = open('partd_fault2_graycode.txt', 'r')
checkStaticBit(fileyread, 10)
```

This does not seem to be correct. I dont think it has anything to do with that. We will try another approach. I think the error is due to loose contact. So the bit switched from 1 to 0 or vice versa rather quickly. To find this in the higher order bits, we can assume that the max change from one reading to the next is 100. Now we can search for all cases where this is not the case, and we can check which of the higher order bits has changed. We can flip this bit and draw the graphs again.

3 NOTE

To access all the source files, screenshots and ipynb files, etc., please visit this github repo:
`https://github.com/MachanHoid/AbhiyaanElecApp.git`