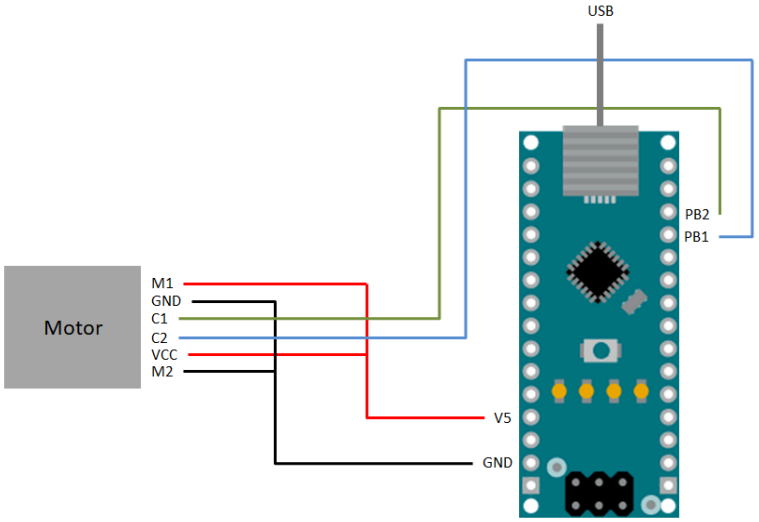
BARNOUIN François
TRAN Émilie

Report Rotary encoder driver

Here is the GitHub link to the repository containing the video and the source code associated to this report: https://github.com/Franz1234567/Rotary_motor_proj1_embedd.git

Part 1

Here is our setup:



Let us compute the maximum pulse rate given the documentation
Which gives us:

PPS = (15000/60)*7
= 1.75 kHz

By using the Shannon Theorem, we can find the maximum sampling rate period so that we do not miss any pulses:

Shannon Theorem : 2*Ts <= T ⇔ fs >= 2f

Maximum: Ts = 1/(2*PPS) = 285 µs (the minimum sampling rate is thus 3.5kHz)

If we want to focus on only one encoder, we can divide Ts by 2.

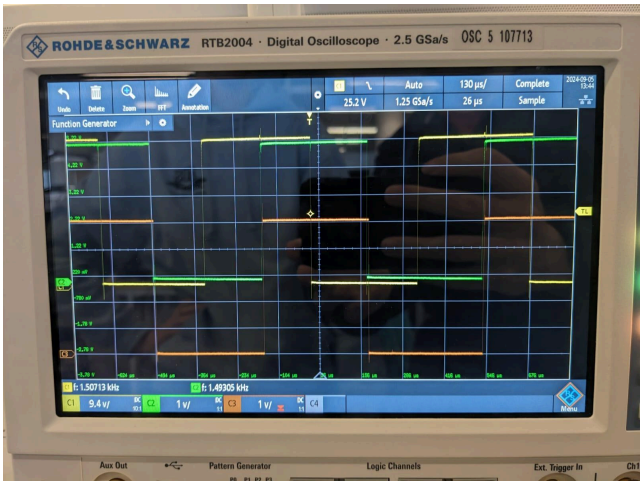
We monitor state changes of two encoder pins, encA and encB, and determine rotation direction based on which pin changes first. If encA changes before encB and their previous states were equal, the counter is incremented, while if encB changes first, it is decremented, indicating opposite rotation. When the previous states differ, the same logic applies with reversed adjustments. An LED toggles with each significant state change to signal an action. Here is a little sketch/picture explaining this logic:

	Same value	Diff value
C1 changes next	count++	count--
C2 changes next	count–	count++

If we just want to look at the C1 encoder, we just need to check the state change of C1 and then look if C2 is equal or opposite to C1. But in our code we chose to search for all the pin changes so the program can be adapted if we want to focus on one or two encoders.

While loop to wait changes

First, our logic is to put another while in the infinite loop that is waiting for a change in the encoder states. Depending on which pin changes, we can know the direction of the motor. When a pin changes, the LED toggles. As we can see in the picture, the LED is perfectly synchronized with the encoder C1 as expected.



green: encoder C1
yellow: encoder C2
orange: LED (connected to D13)

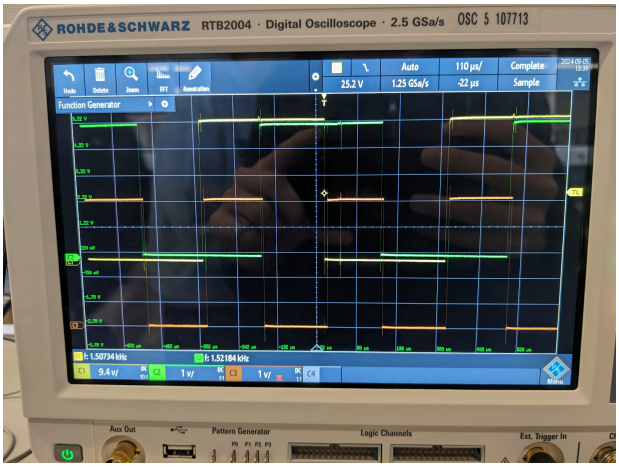
However, this is not efficient, it's expensive to continuously check the values of the encoder pins, especially with a small Arduino Nano.

Use of sampling rate

Now, we are going to test if the sampling rate that we computed earlier is correct. So we commented on the while loop and added a delay of 280µs (a little less than the maximum period we found earlier). We can see the results on the image below. As expected, the LED is not perfectly synchronized with an encoder signal anymore. However it still toggles when the encoder C1 changes, it seems that we are not missing any encoder pulse of C1.

green: encoder C1
yellow: encoder C2
orange: LED

Note: If we wanted to have the LED that toggles for every changes of both encoder, we would have had the following results on the oscilloscope: (with the while method here)



green: encoder C1
yellow: encoder C2
orange: LED

Part 2

To print the current count, we used the serial monitor. However, using SerialPrintln() repeatedly for each change to print the count value introduces a small delay. This delay interferes with the timing-sensitive nature of encoder reading that we are implementing. That is why in the video the count is completely lost (if it is the mott that is running, if we turn the motor manually it can be slow enough to count as shown in the video). We checked if the LED was also not working in the oscilloscope and it was the case.

Here is a small screenshot showing the prints doing nonsense:

```
-9
-9
-10
-9
-9
-10
-9
-9
-10
-9
-9
-10
-9
-9
-10
```

Thus, we tried to print less frequently (every time the count was a multiple of 100, we printed the value of count for the encoder C1). This approach worked but we lost a degree of precision. You can see more in the video associated with this report.

Another effect of adding a print that we noticed is the following:

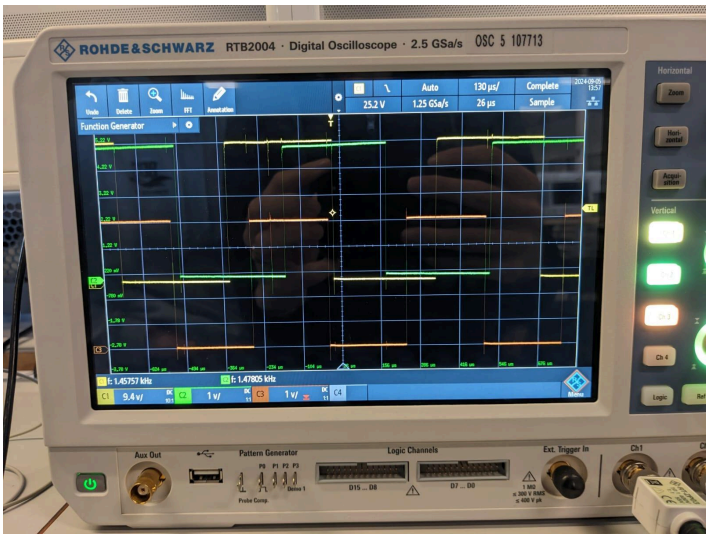
Here is the RAM and Flash used after compilation when we don't print
RAM: [ ] 0.0% (used 0 bytes from 2048 bytes)
Flash: [ ] 1.3% (used 392 bytes from 30720 bytes)

Here is the RAM and Flash used after compilation when we print:
RAM: [=] 8.7% (used 179 bytes from 2048 bytes)
Flash: [=] 5.4% (used 1644 bytes from 30720 bytes)

It increases the RAM and Flash memory consumption, we guess it's related to the use of the UART communication protocol (to manage, store, convert and transmit the data).

Part 3

This time, we use hardware interrupts like in the Assignment 2_1. They trigger the encoder reading at a set period. This is more accurate and timely compared to adding the delay manually or using the while like we did at first. Indeed, in this context we are not continuously checking the encoder pins but only precisely when we need to so this is more efficient and responsive. We can see in the picture below the results that show the consistency and reliability of using such interrupts, the LED always blinks at the same interval.



green: encoder C1
yellow: encoder C2
orange: LED