

Commande numérique directe de dispositifs

Les objectifs de ce TP sont les suivants :

- 1) Réaliser un shell pour commander le hacheur, sur la base d'un code fourni
- 2) Réaliser la commande des 4 transistors du hacheur en commande complémentaire décalée
- 3) Faire l'acquisition des différents capteurs
- 4) Réaliser l'asservissement en temps réel

1. Etapes préliminaires

Nous avons commencé par configurer Github afin de mémoriser notre code, et doxygen afin de le commenter automatiquement.

Nous avons ensuite importé sous CubeIDE le code permettant de paramétriser une liaison UART avec plusieurs fonctions de bases telles que help, pinout, start et stop.

2. Commande MCC Basiques

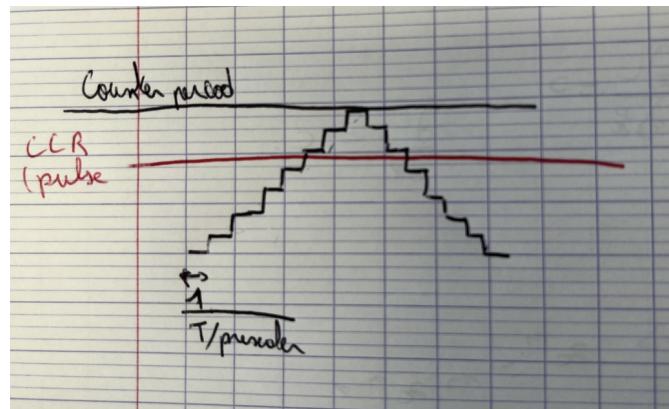
La première partie de cette deuxième étape a été de créer les PWM dont nous aurons besoin pour faire tourner le moteur.

Nous avons créé 4 PWM de fréquence 16kHz, de temps mort 2 μ s et de résolution 10bits.

On choisit un prsc=1 on compte donc une fréquence de 170 MHz.

Afin d'obtenir une fréquence de 16 kHz, on a besoin de compter jusqu'à $170\ 000/16 = 10625$

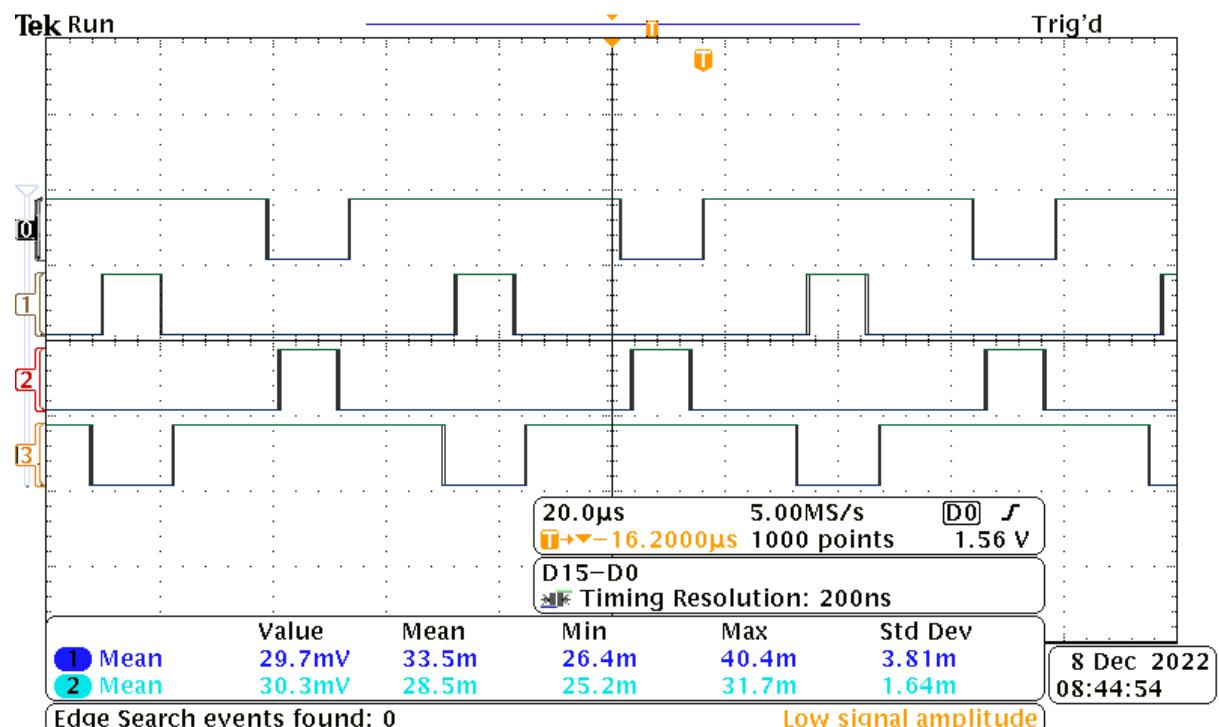
On obtient ainsi 10625 que l'on divise par 2 car nous sommes en mode 'Center aligné' donc notre compteur va compter puis décompter tel que sur la figure suivante :



On choisit donc 5313 pour le Counter Period Register.

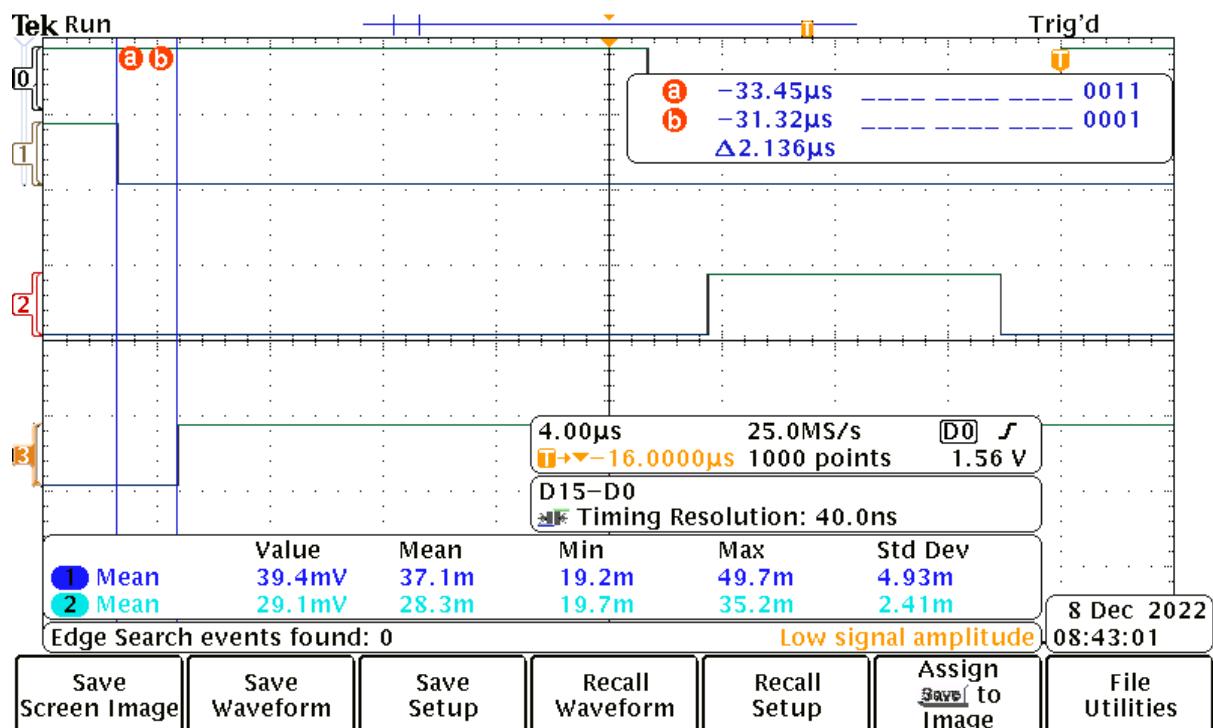
Ensuite le CCR1(Pulse) permet de fixer le rapport cyclique.

Sur la figure suivante on voit bien les 4 PWM en commande décalée :



Afin de fixer le temps mort, on modifie à taton le registre dead-band.

On vérifie sur l'oscilloscope que le temps mort vaut bien $2\mu\text{s}$:



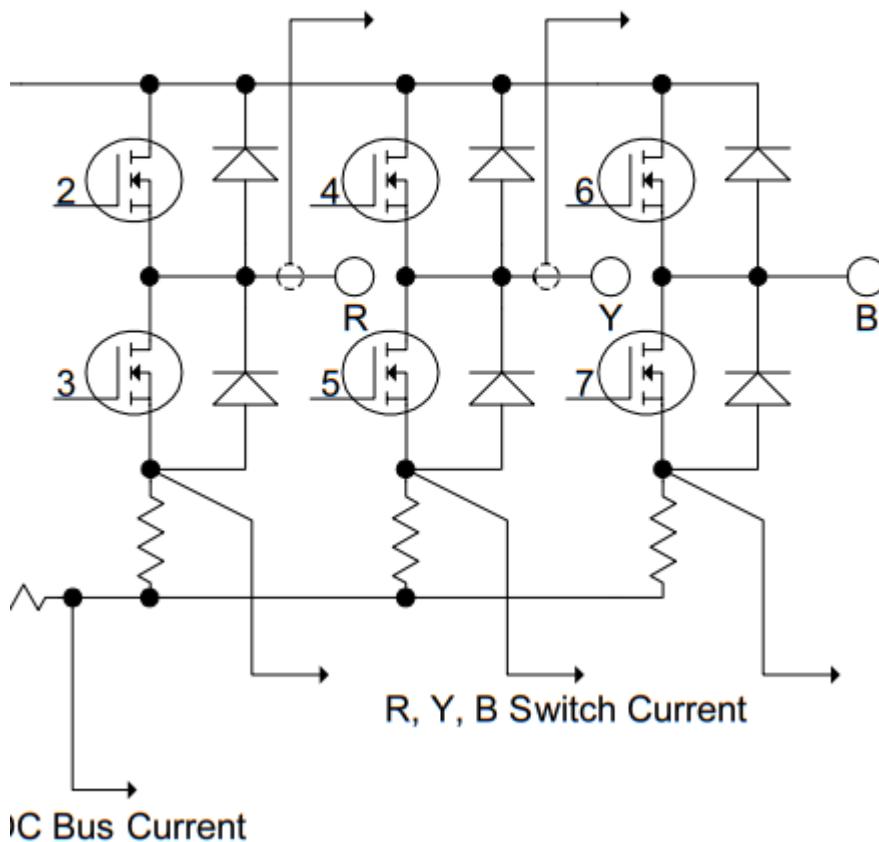
Nous avons ensuite créé une commande afin de modifier la valeur du rapport cyclique en direct sur le shell.

Ainsi en tapant alph suivi du rapport cyclique voulu en pourcentage, on modifie les PWM.

Tout étant prêt, nous pouvons relier la STM au hacheur.

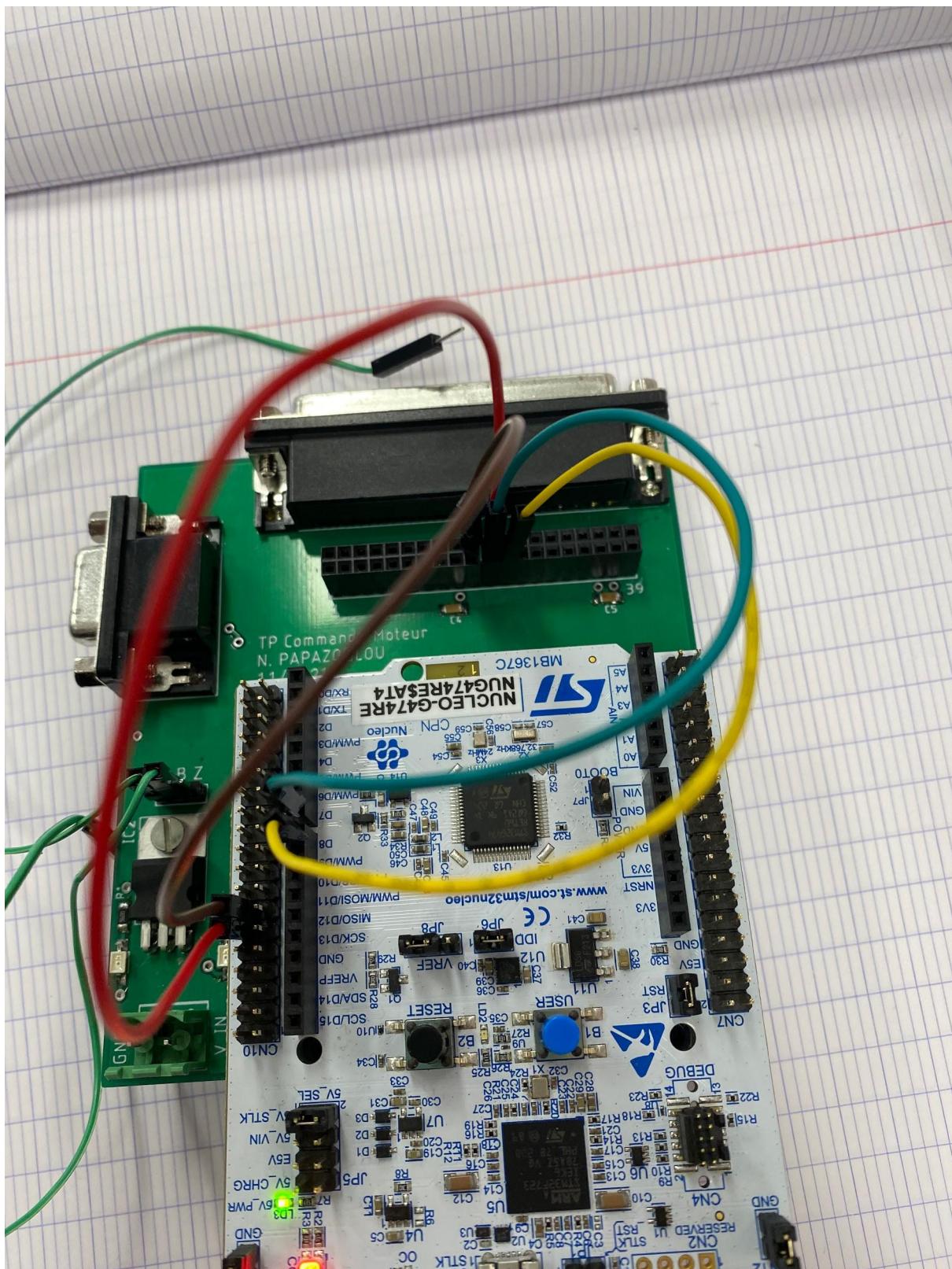
On choisit d'alimenter les phases Jaune et Bleu
Pour cela il faut alimenter les interrupteurs 2-3 et 6-7.

R & Y Isolated Phase Current Feedback



On effectue alors les branchements suivants :

Interrupteur	PWM
Yellow Top (12)	CH2 (PA9)
Yellow Bottom (30)	CH2N (PA12)
Blue Top (11)	CH1 (PA8)
Blue Bottom (29)	CH1N (PA11)

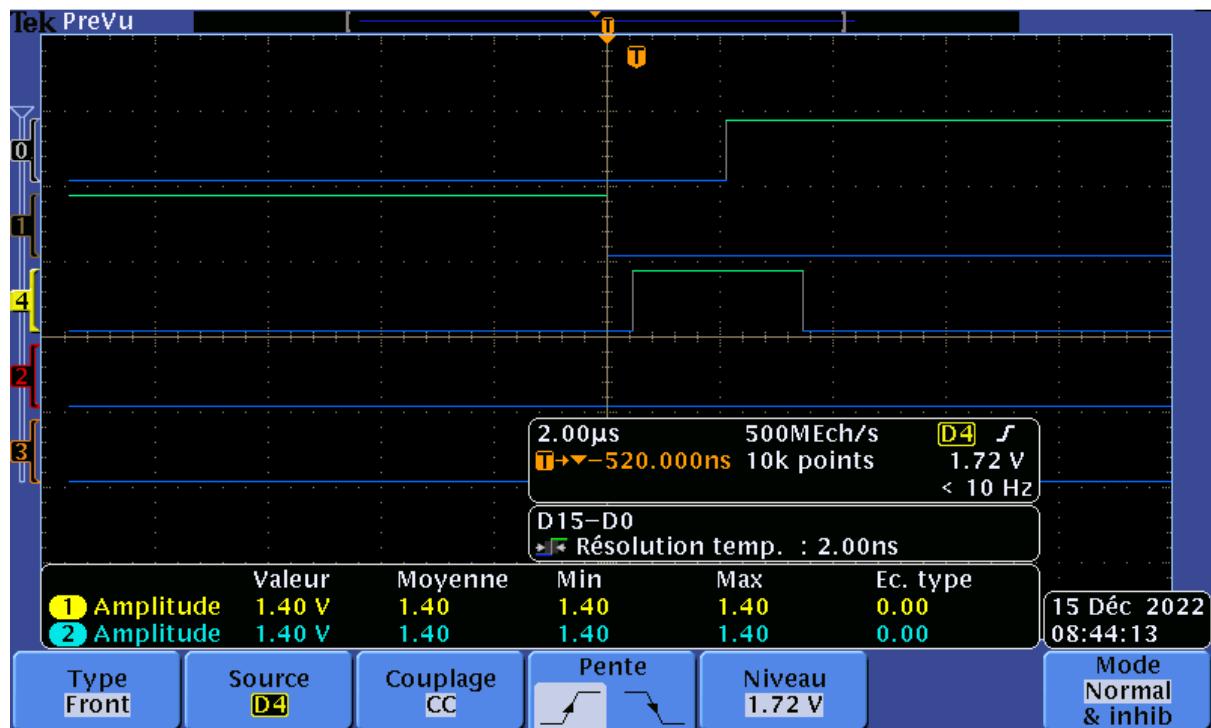


Nous devons désormais commander l'allumage du hacheur. Cette commande se fait en envoyant un pulse de $2\mu\text{s}$ sur la pin ISO_RESET pin 33. Comme nous pouvons le lire le screen suivant

Reset the system by activating the active high ISO_RESET line. The ISO_RESET line is on pin 33 of the 37-pin, D-type (see **Section 1.8 “User Signal Connector Pinout (37-Pin, D-Type)”**). On the dsPICDEM MC1 Motor Control Development Board, this signal is routed to pin 14 of the 30F6010 dsPIC device, which is on Port RE9. The minimum pulse width for the reset is $2\mu\text{s}$. The RESET should be done in coordination with the SPI™ handling routine of the dsPIC device to ensure correct synchronization of the serial interface providing the isolated voltage feedback (see **Section 1.5.7.2 “Isolated Voltage Feedback”**). The system is now ready to use.

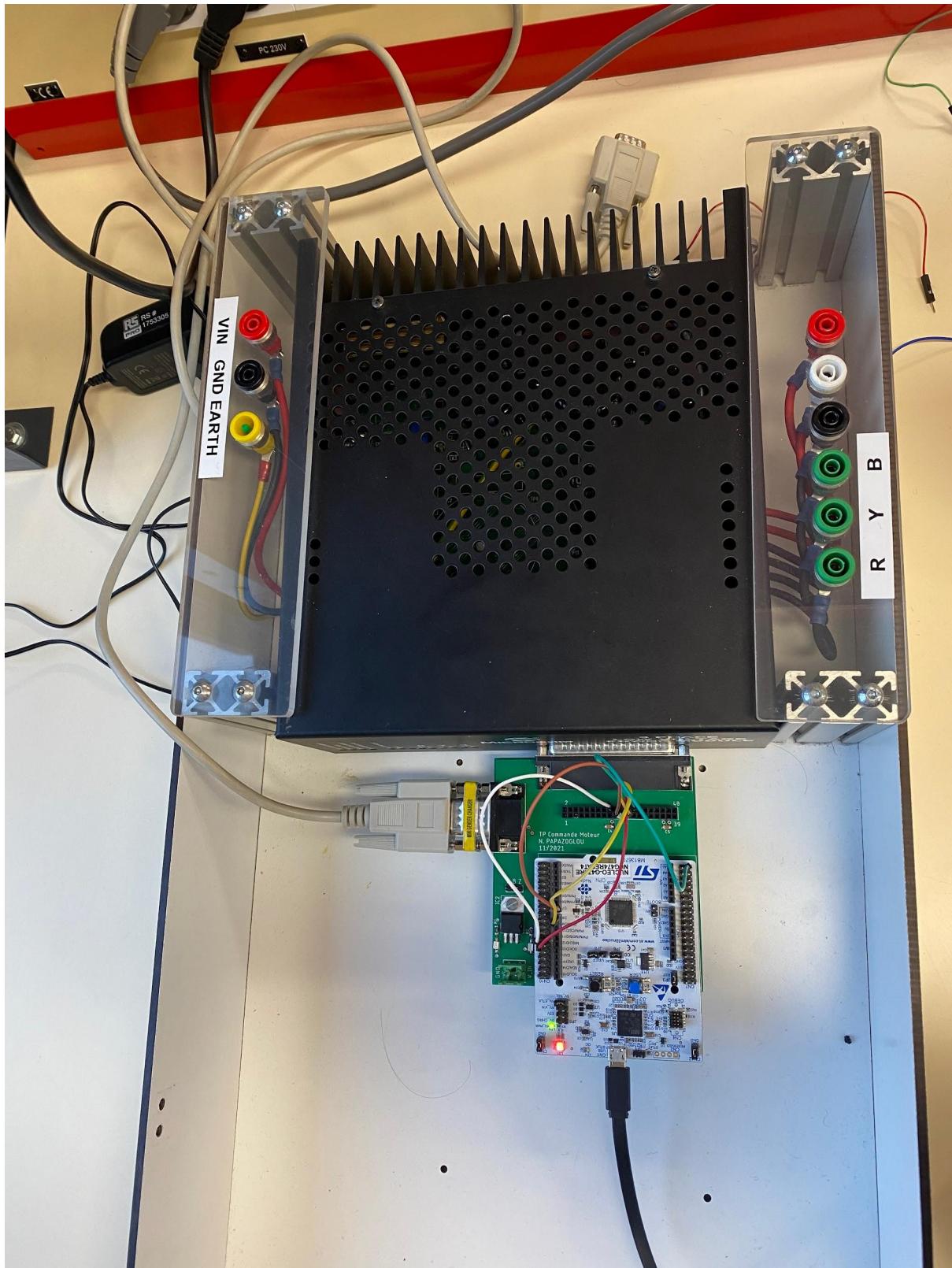
Pour ce faire, on crée une commande start, qui envoie un pulse sur la pin A5. Afin de régler une durée de $2\mu\text{s}$ sans utiliser de timer, on crée une boucle for dont on règle la durée à taton.

Après quelques tests, on fixe une boucle for nous permettant d'avoir un pulse d'environ $2\mu\text{s}$.



MSO3014 - 09:28:57 15/12/2022

Tout est prêt, nous pouvons faire les branchements et connecter le hacheur.



Nous prenons bien garde à initialiser à 50% le rapport cyclique afin d'avoir la MCC immobile, puis on augmente le rapport petit à petit.

On évite les saut de rapport cyclique afin d'éviter les sauts de tension. En effet, un saut de tension implique un pic de courant, le hacheur se met alors en sécurité.

Le moteur tourne bien, et les changements de rapport cyclique permettent bien de varier la vitesse.

Nous pouvons désormais passer à l'étape suivante : créer une commande permettant de choisir la vitesse.

On lit la plaque signalétique du moteur.



A une tension de 48V est associée une vitesse de 3000 tours/min.
On limite la vitesse de sortie à 3000 tours/min.

Ainsi, la vitesse de sortie est la suivante :

$$V = (\alpha - 0.5) * 3000 \text{ tours/min}$$

$$\Rightarrow \alpha = V/3000 + 0.5$$

$$\Rightarrow CCR = 5313 * (V/3000 + 0.5)$$

On crée une nouvelle fonction sur le shell "speed" permettant de modifier la vitesse du moteur. Cependant après plusieurs essais et sur conseil de notre professeur, on décide de passer à la suite sans continuer cette étape.

3. Capteurs de courant et de position

Nous souhaitons mesurer le courant à l'aide de la résistance de shunt.

La première étape est d'utiliser la phase rouge au lieu de bleu.

On avait :

Interrupteur	PWM
Yellow Top (12)	CH2 (PA9)
Yellow Bottom (30)	CH2N (PA12)
Blue Top (11)	CH1 (PA8)
Blue Bottom (29)	CH1N (PA11)

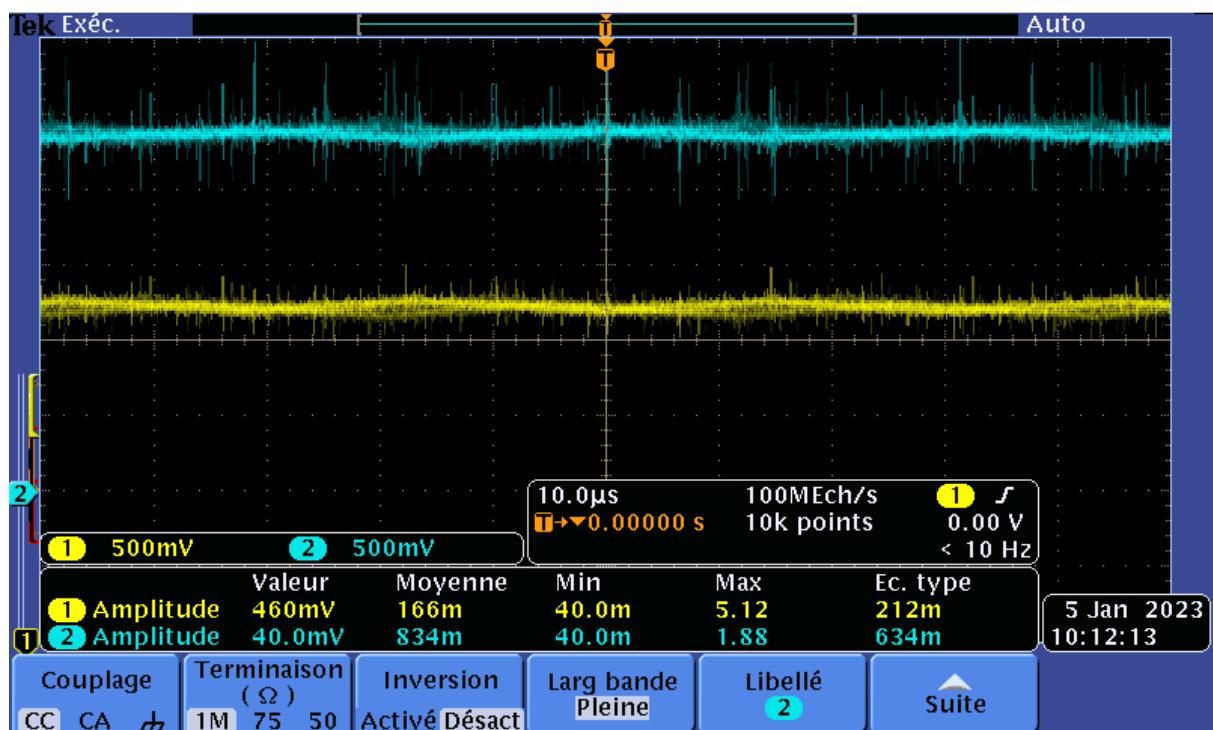
On va donc désormais utiliser les pins suivantes :

Interrupteur	PWM
Yellow Top (12)	CH2 (PA9)
Yellow Bottom (30)	CH2N (PA12)
Red Top (13)	CH1 (PA8)
Red Bottom (31)	CH1N (PA11)

On observe ensuite les PINS suivantes :

16	Yellow Phase Hall Current Sensor Feedback	Y_HALL	Output	Yes
35	Red Phase Hall Current Sensor Feedback	R_HALL	Output	Yes

On observe les signaux suivants :



MSO3014 - 10:57:20 05/01/2023

En jaune le bras Red et en bleu le bras Yellow

On configure ensuite l'ADC du STM afin de récupérer la tension de la résistance de shunt et avoir le courant.

Pour ce faire, on crée un buffer dans lequel le DMA stock les mesures faites par l'ADC. Lorsque le buffer est rempli on génère une interruption qui met un flag à 1, qui va donc lancer le calcul de la moyenne des valeurs contenue dans le buffer puis convertir cette tension image de notre courant

En lisant la doc, nous avons obtenu le rapport entre la tension et le courant.

Inverter Output (R and Y) Hall Current Sensor	12 A / V with 2.5V = 0A
---	-------------------------

On obtient ainsi la mesure du courant :

```
*-----*
| Welcome on Nucleo-STM32G431RB |
*-----*
user@Nucleo-STM32G4RB31>>mesure
courant: 0.13 A
user@Nucleo-STM32G4RB31>>alpha 60
user@Nucleo-STM32G4RB31>>mesure
courant: 1.54 A
user@Nucleo-STM32G4RB31>>mesure
courant: 1.94 A
```

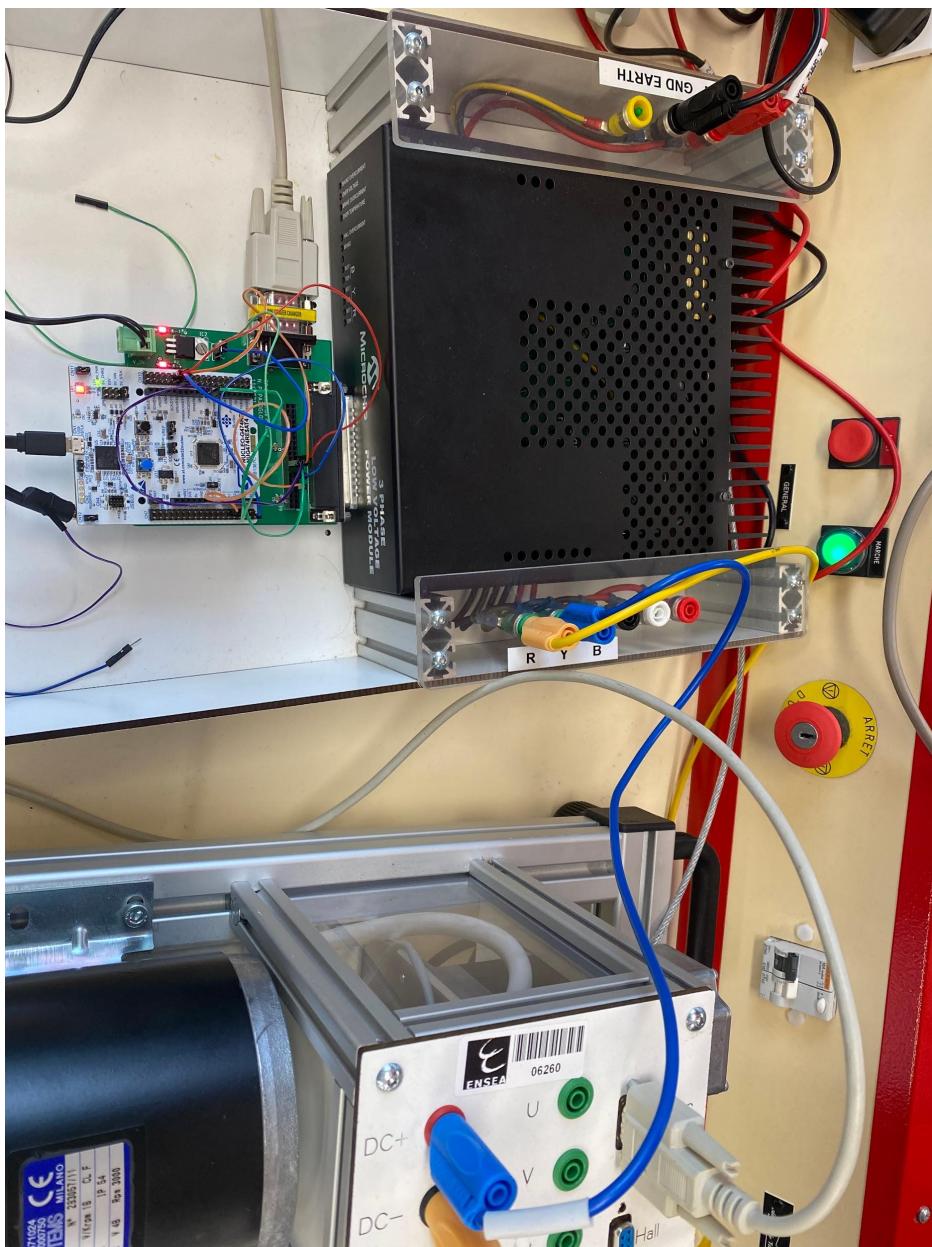
```

else if((strcmp(argv[0], "mesure") == 0))
{
    value= value/10;      //on calcul la moyenne des valeurs enregistré par le DMA dans le buffer
    value= value*3.3/4096; //résolution de 12 bits
    value= (value-2.255)*12; // rapport entre tension courant et un offset de 0A pour 2.5V
    sprintf(uartTxBuffer, "courant: %.2f A \r\n", value);
    HAL_UART_Transmit(&huart2, uartTxBuffer, sizeof(uartTxBuffer), HAL_MAX_DELAY);
    FLAG=0;
}

```

Lecture de la vitesse :

On commence par connecter la roue codeuse située sur la maquette du moteur à notre carte :



On active deux nouveaux timer. TIM4 qui génère une interruption tous les 100ms et TIM3 en mode encodeur qui compte les fronts montants et des descendants.

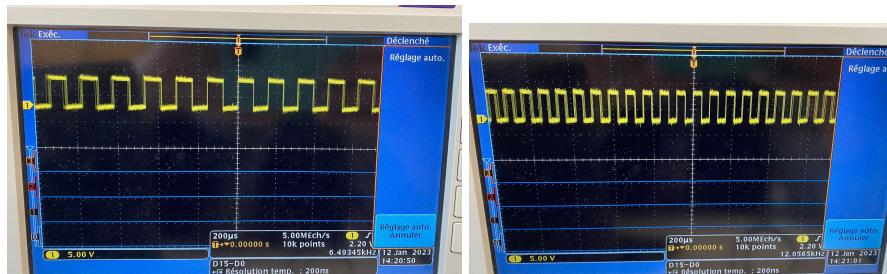
On récupère aussi les deux PWM A et B de la roue codeuse. Qui sont l'image du nombre de tour

Il y a 4096 fronts montants et descendants sur un tour. Ainsi en analysant la fréquence on peut en déduire le nombre de tours par minute.

Toutes les 100ms, on relève le nombre de tours effectués :

$$\text{nombre de tours} = \frac{\text{nombre de fronts montants et descendants des deux signaux}}{4096}$$

On multiplie ensuite cette valeur par 600 pour avoir le nombre de tours par minute.



On vérifie également sur l'oscilloscope les PWM fournis par la roue codeuse, on observe bien une augmentation de la fréquence lorsque l'on augmente la vitesse de rotation de notre moteur

On lit bien la vitesse a l'aide de la commande vitesse sur notre shell :

```
*-----*  
| Welcome on Nucleo-STM32G431RB |  
*-----*  
user@Nucleo-STM32G4RB31>>start  
user@Nucleo-STM32G4RB31>>vitesse  
vitesse: 0.00 tr/min  
user@Nucleo-STM32G4RB31>>alpha 60  
user@Nucleo-STM32G4RB31>>vitesse  
vitesse: 387.74 tr/min  
user@Nucleo-STM32G4RB31>>alpha 65  
user@Nucleo-STM32G4RB31>>vitesse  
vitesse: 705.32 tr/min
```

La MCC a une vitesse maximale de 3000 tr/min. Ainsi pour un alpha de 65% on est censé avoir $0.3 \times 3000 = 900$ tr/min.

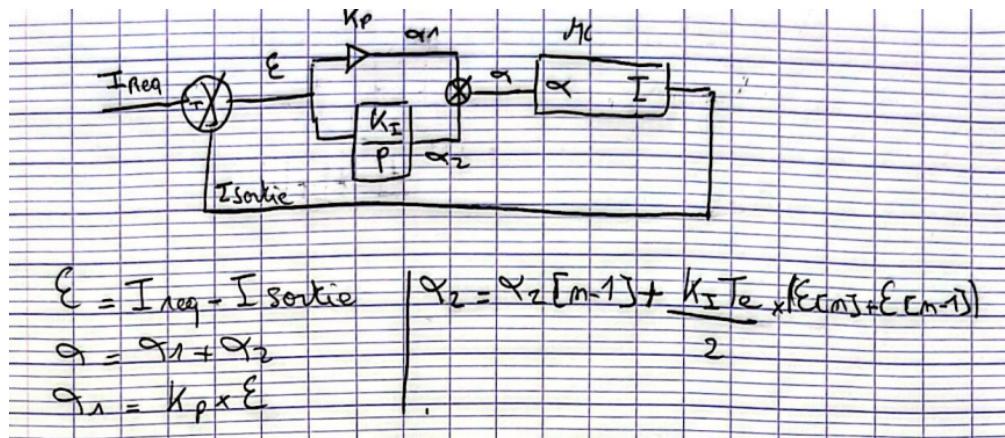
On relève une vitesse de 705 tr/min, ce qui est proche de la vitesse souhaitée.

On passe ensuite à l'asservissement en courant

4. Asservissement en courant

On implémente une nouvelle fonction dans notre shell. Qui prend en argument le courant de consigne

```
else if((strcmp(argv[0], "asserv")==0))
{
    flagA=1;
    ireq=atoi(argv[1]);
}
```



On crée un correcteur PI avec les paramètres suivants et on implement les equation ci-dessus :

$k_p=0.002$

$k_i=3$

$T_e=1/16000$

```

if(flagA==1) {

    eps_prev=eps; //stockage de l'ancienne valeur de l'erreur
    eps=ireq-value;//calcul de l'erreur
    alpha1=eps*kp;
    alpha2=alpha2prev+(ki*Te/2)*(eps+eps_prev);

    if(alpha2>1){ //anti windup
        alpha2=1;
    }
    if(alpha2<0){
        alpha2=0;
    }

    alpha2prev=alpha2;
    alf=alpha1+alpha2;
    if(alf>1){
        alf=1;
    }
    if(alf<0){
        alf=0;
    }

    alf=(int)(alf*5313);
    TIM1->CCR1=alf;
    TIM1->CCR2=5313-alf;
}

FLAG=0; // on remet le flag du DMA a zero

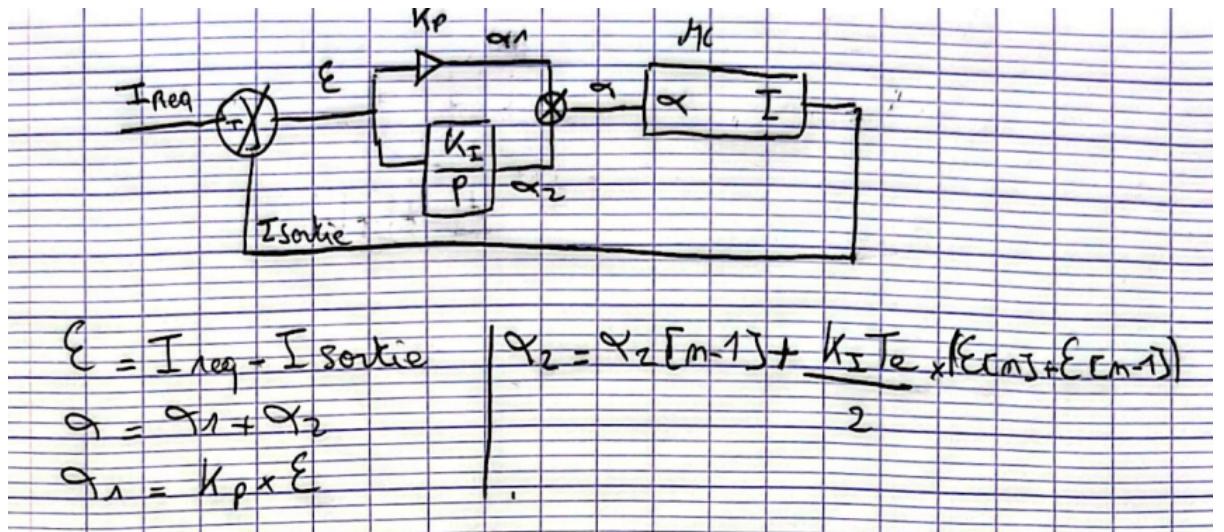
```

L'asservissement en courant fonctionne avec une consigne de 2A on observe bien un courant d'environ 2A.

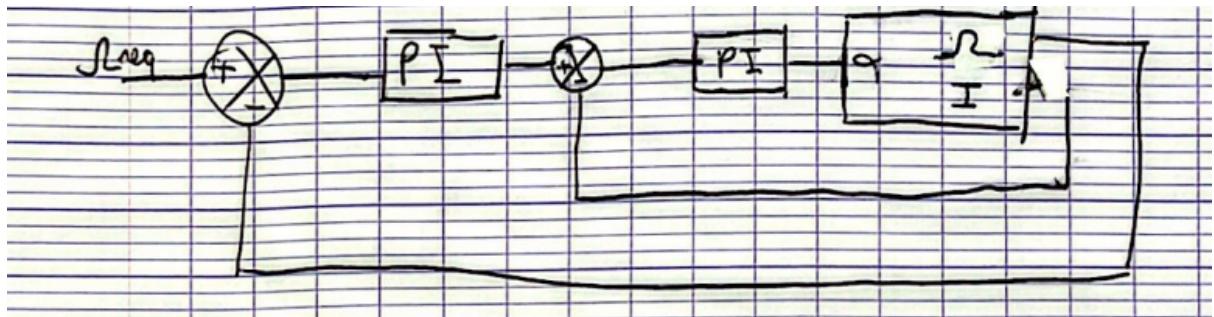
```

*-----*
| Welcome on Nucleo-STM32G431RB |
*-----*
user@Nucleo-STM32G4RB31>>asserv 2
user@Nucleo-STM32G4RB31>>mesure
courant: 2.39 A
user@Nucleo-STM32G4RB31>>mesure
courant: 2.35 A
user@Nucleo-STM32G4RB31>>mesure
courant: 2.02 A

```



5. Asservissement en vitesse



les équations restent les mêmes que dans la partie précédente:

Il faut également que le temps de réponse de la boucle de courant soit supérieur à celui de la boucle de vitesse nous devons donc utiliser des interruptions afin que la fréquence de la boucle de vitesse soit de 10kHz et celle de la boucle de courant 16kHz*

Par manque de temps nous n'avons pas fini l'asservissement en vitesse.