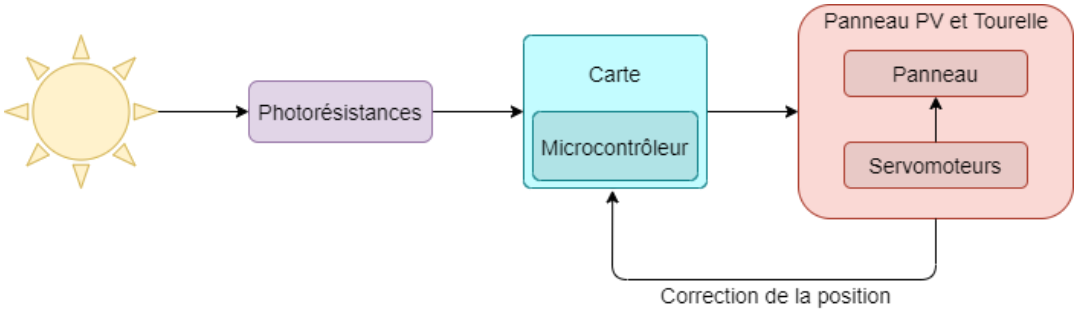
BARKOUDEH Julian
DOUZET Camille
EI2I3 II
14/06/2021

Compte Rendu Projet Elec Info

1. Introduction

Le but de ce projet est de concevoir un système **Solar Tracker** ou Tournesol Solaire en français. Il s'agit d'un **système de production électrique solaire asservi** au mouvement du soleil ou tout autre source lumineuse en mouvement. Le but est donc de simuler le mouvement de la Terre autour du soleil, cela permettra donc d'optimiser le rendement du panneau solaire. Pour se faire, le système réalisera des mesures grâce à des **photorésistances** de manière **autonome**. Voici un premier schéma qui permet de comprendre rapidement les enjeux du projet:



Nous avons bien mis en évidence la particularité du projet : la **correction de la position** de façon autonome et constante par rapport au soleil et grâce aux servomoteurs et la tourelle.

2. Présentation du projet

a. Cahier des charges

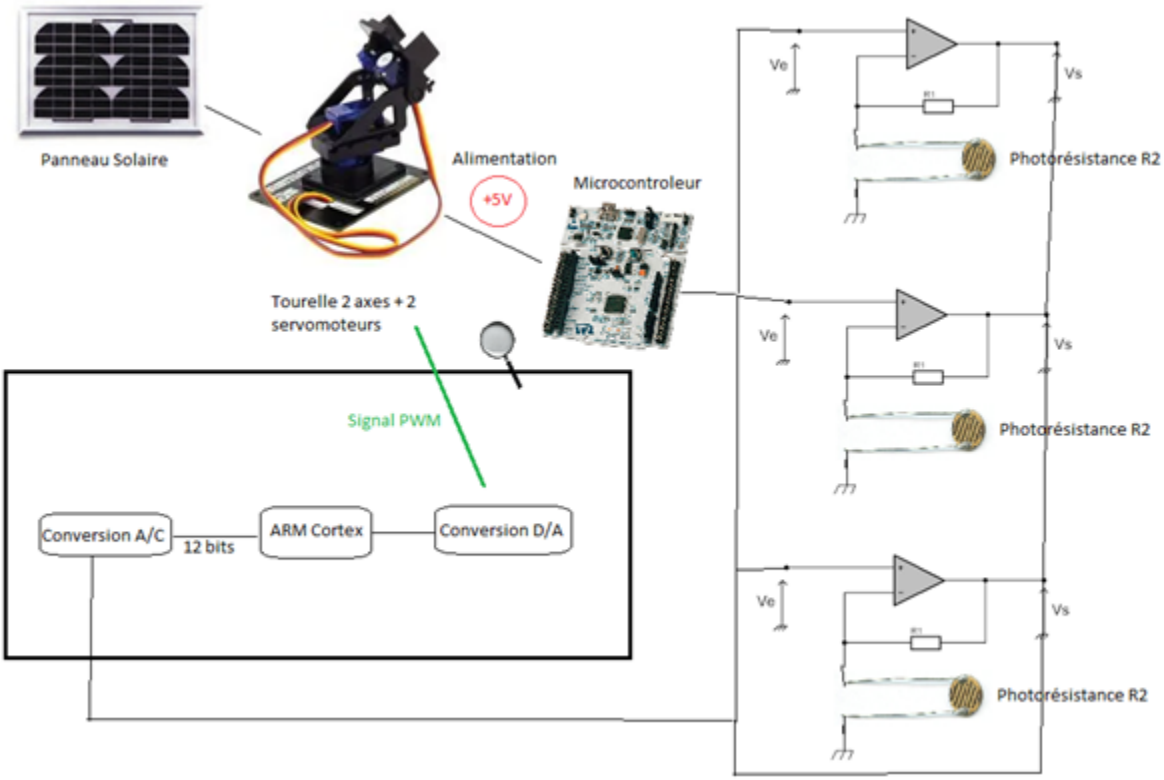
Voici une description fonctionnelle synthétique du projet :

Qui ?	Binôme de deux étudiants apprentis
Quoi ?	Conception d'un système Solar Tracker - Comprendre le fonctionnement de la carte - Comprendre le système asservis du moteur et de la tourelle - Exploiter les données des capteurs - Programmer le microcontrôleur
Où ?	En présentiel et distanciel
Quand ?	60 heures
Comment ?	A l'aide du matériel fourni par Polytech Sorbonne : - Carte STM32F072 - Servomoteurs PIMORONI PIM 183 - Capteurs NSL-19M51 - Microcontrôleur STM32

Afin de concevoir ce système Solar Tracker, nous devons traiter la **partie hardware** ainsi que la **partie software** de ce projet. En premier lieu, nous devons exploiter les données reçues par les **photorésistances**. Cela implique de traiter la **conversion** entre données analogique en numérique, de **stocker** ces données pour pouvoir ensuite les utiliser. L'élaboration préalable de **l'algorithme** est donc importante pour **commander les servomoteurs**. Il prendra aussi en compte de manière **autonome** les nouvelles mesures des photorésistances et adaptera la position des servomoteurs et donc du panneau solaire.

3. Hardware : Présentation des blocs

a. Schéma bloc



La schéma ci-dessus présente le projet dans son ensemble. On a donc représenté tous les modules hardware que nous avons utilisés. Nous avons aussi représenté les signaux importants comme le signal PWM et l'alimentation.

b. Fonctionnement d'une cellule photovoltaïque

Inventée en 1838, le principe de la **cellule photovoltaïque** est d'exploiter **l'énergie lumineuse** afin de produire de **l'électricité**. La cellule photovoltaïque est composée principalement d'une **couche de semi-conducteur**. Une couche supérieure d'un semi-conducteur dopé N, qui est plus riches en électrons libres que la couche dopé P. Quand la

lumière percute la cellule, les **électrons** sont excités et migrent vers la couche N, alors que les trous migrent vers la couche P. Ce mouvement d'électrons crée un **courant électrique**.

Il existe différents types de cellules photovoltaïques selon les semi-conducteurs utilisés ;

- Cellule au silicium monocristallin :

Ce type de cellule est très répandu et utilisé dans les panneaux solaires. Après que le silicium est extrait, il est traité à froid, donc forme des cristaux. Il sera fondu et reconstruit pour former un grand cristal. Ce type de silicium peut atteindre un rendement de 23%.

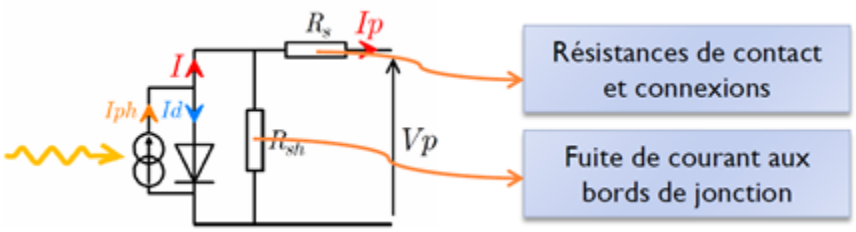
- Cellule au silicium polycristallin :

Ce matériau est obtenu après fusion et cristallisation et sciage du silicium. Ce type est moins efficace, et peut atteindre un rendement de 19% seulement.

- Cellule au silicium amorphe :

Cette cellule est formée par le silicium sous forme allotropique non cristalline. Les électrons dans ce semi-conducteur forment un réseau aléatoire et peuvent atteindre un rendement de 13%.

Le schéma équivalent d'une cellule photovoltaïque peut être présenté de la façon suivante :



L'espace entre les deux couches de semi-conducteurs N et P est appelé jonction PN. Une petite perte de courant peut être présente dans cette jonction, et présentée par la résistance Rsh.
D'après le schéma équivalent le courant peut être présenté dans l'expression :

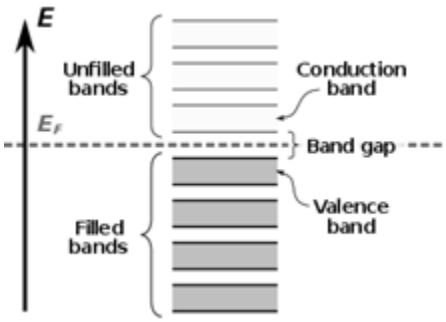
$$Iph \approx I_{cc} = Iph_0 \times \frac{E_{[W/m^2]}}{1000} \times (1 + coeff_\theta(\theta - 25°C))$$

On en déduit que le courant est proportionnel à la puissance par unité de surface de l'intensité lumineuse.

c. Fonctionnement des photorésistances

Le composant principal d'une **photorésistance** est le **composant semiconducteur**.

Afin de comprendre le fonctionnement d'une photorésistance nous devons voir le **théorème des bandes**. En effet, les **électrons** sont classés selon leurs niveaux d'énergies comme on le voit dans le schéma ci- dessous.



En ayant beaucoup d'électrons haute énergie, le semi-conducteur agit comme un **conducteur de courant**. Quand les photons percutent la surface de la photorésistance, les électrons sont **excités** et passent au niveau d'énergie conducteur. Ceci produit un courant électrique qui est traduit par la formule suivante ;

$$I = q\mu n \frac{A}{L} V$$

On remarque que le **courant** est **proportionnel** avec la surface de la photorésistance. Or, afin d'avoir un dispositif optimal, la surface de la photorésistance est en forme de ruban. Ceci maximise la surface de contact avec les électrodes et permet de conserver la conduction du semi-conducteur. Il existe différentes façons pour monter un système de photorésistances. Ceci est possible en **pont diviseur** ou en pont.

Pour les besoins de ce projet nous allons utiliser des photorésistance **NSL-19M5**.

Cette photorésistance est montée sur une **céramique** de type TO-18 et la surface est isolée avec une couche en plastique. Ceci permet d'avoir un bon isolement et une résistance contre l'humidité. La photorésistance peut fournir une tension maximale de 100 V, tout en garantissant une dissipation d'énergie de 50 mW. Cette photorésistance peut opérer sous des températures pouvant atteindre 75°.

WWW.LUNAINC.COM Precision – Control – Results

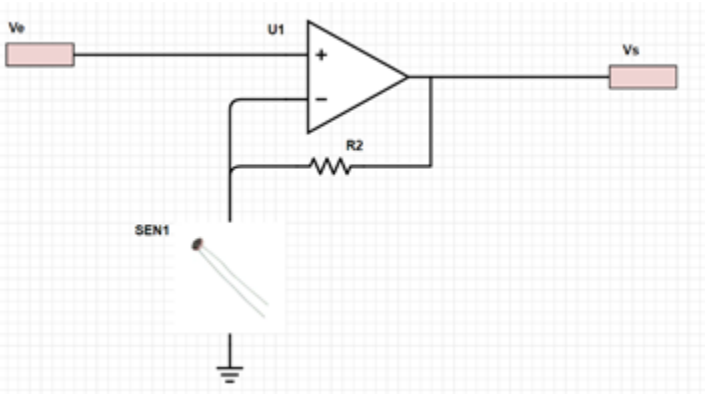
OPTO-ELECTRICAL PARAMETERS

T_a = 23°C unless noted otherwise

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNITS
Light Resistance	10 lux., 2854°K[3]	20	-	100	KΩ
	100 lux., 2854°K[3]	-	5	-	
Dark Resistance	10 sec after removal of test light.	20	-	-	MΩ
Spectral Peak	-	-	550	-	nm
Gamma	1-10 Lux	-	0.7	-	-
Gamma	10-100 Lux	-	0.7	-	-

NOTE:
3. Cells light adapted at 30 to 50 Ftc for 16 hrs minimum prior to electrical tests.

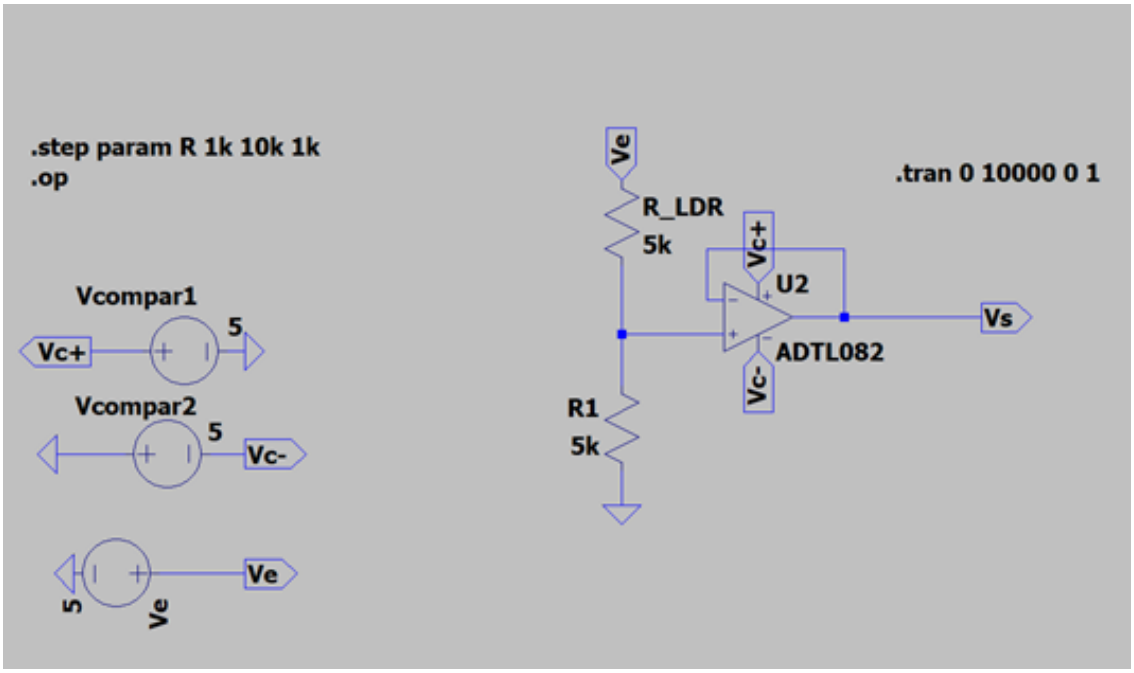
Ce **schéma** est un schéma électrique simplifié du dispositif. L'utilisation **d'amplificateur opérationnel non-inverseur** est important pour notre dispositif. En effet, afin de fournir plusieurs niveaux de tensions pour les valeurs différentes lus par la photorésistance, nous mettons en place un montage AOP non inverseur pour obtenir un **gain positif**. La **tension de sortie** de ce système peut s'écrire de la forme :



$$u_s = (1 + \frac{R_2}{R_1} u_e)$$

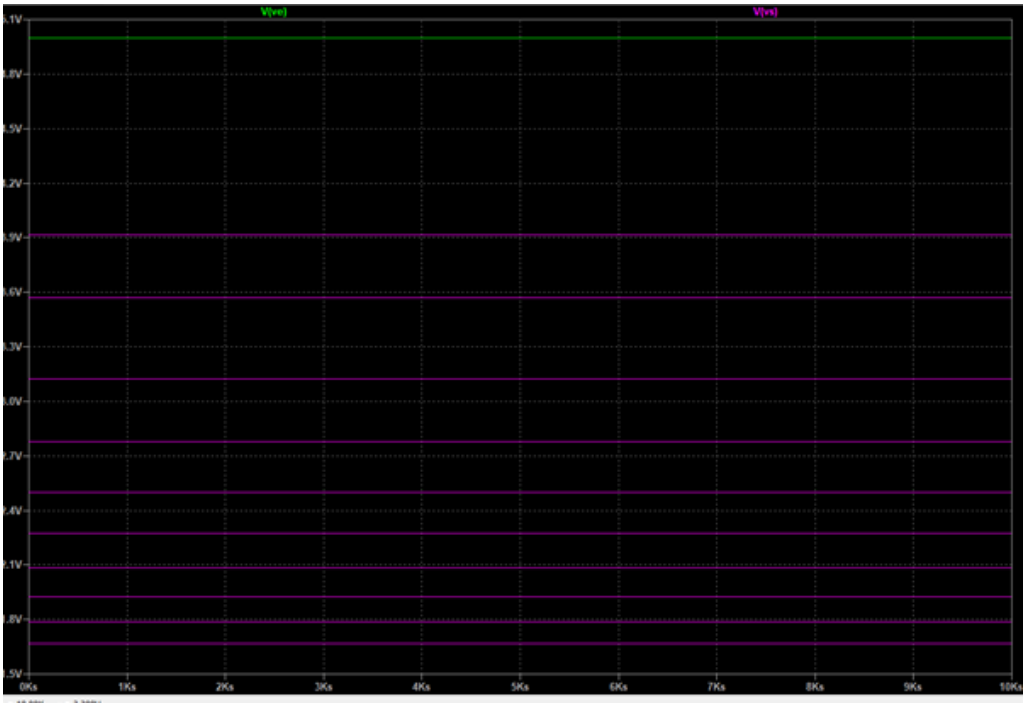
Afin de mieux adapter ces matériels sur notre projet, nous avons réalisé des **simulations** du comportement électrique de ce système. Nous avons simulé 3 cas :

- Cas R_LDR constant :



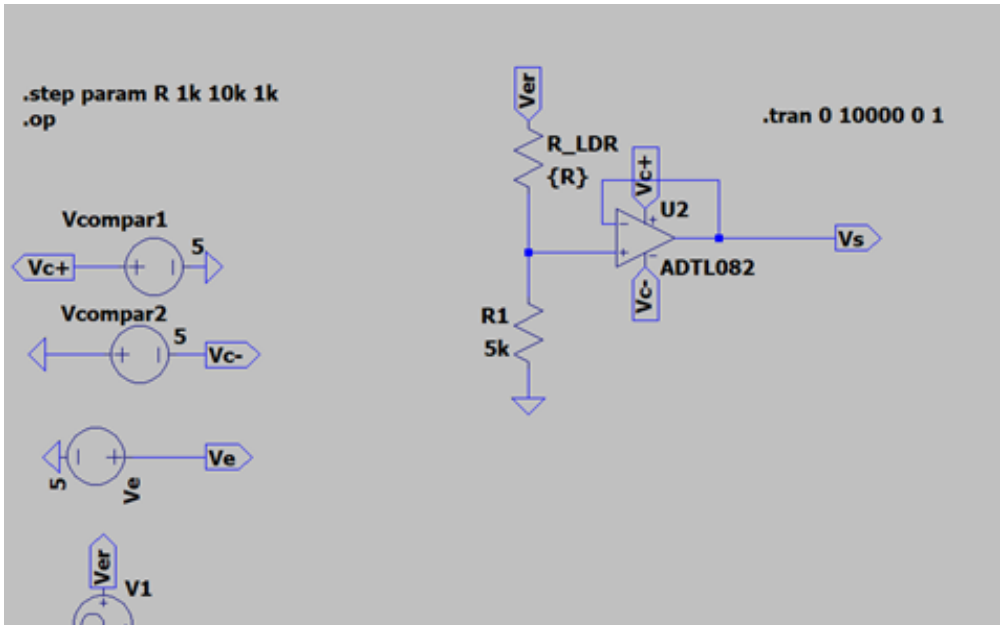
On remarque une légère perte de tension de sortie par rapport à l'entrée. Ceci peut être expliqué par des pertes dans l'AOP.

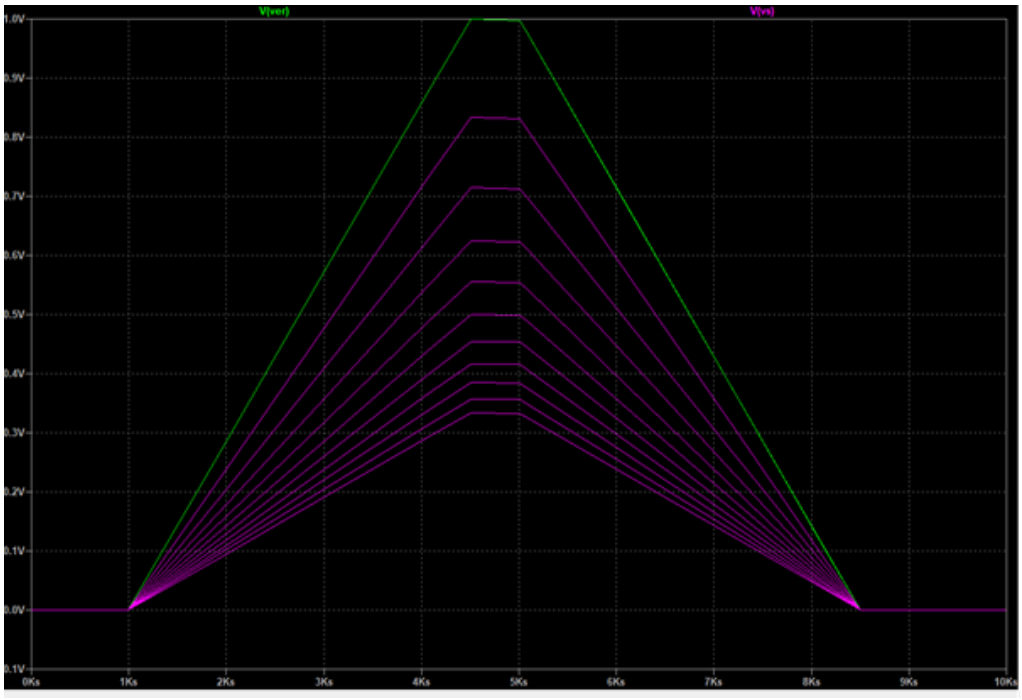
- Cas R_DLR variable :



On remarque qu'en augmentant la charge, la tension diminue de manière logarithmique. Nous avons 10 valeurs de charge entre 1k et 10k. Donc pour chaque charge, une tension différente de sortie est mesurée.

- Cas d'entrée variable





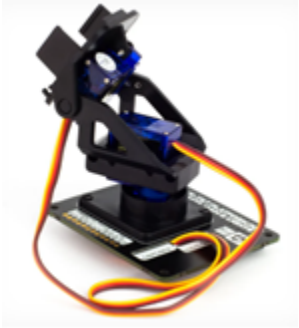
On observe qu'on obtient la même courbe en sortie, or la courbe baisse car la résistance augmente. On remarque que l'intérêt de ce montage est de fournir une tension variable en sortie avec un gain positif. Ainsi cette tension est comprise entre Ve et 0, donc l'interprétation de cette tension rendra plus facile la traduction de l'intensité lumineuse.

d. Fonctionnement d'un panneau solaire

Le panneau **SUNTECH STP005B-12/DEA**, est un panneau adapté à l'utilisation urbain et utilise un **semi-conducteur monocristallin**. Ce panneau de tension 17 Volt et puissance de 5 W. Ce panneau comporte 36 cellules PV, de la forme 4*9. Ce panneau est certifié IP 65, donc présente une bonne résistance à l'exposition à l'eau.

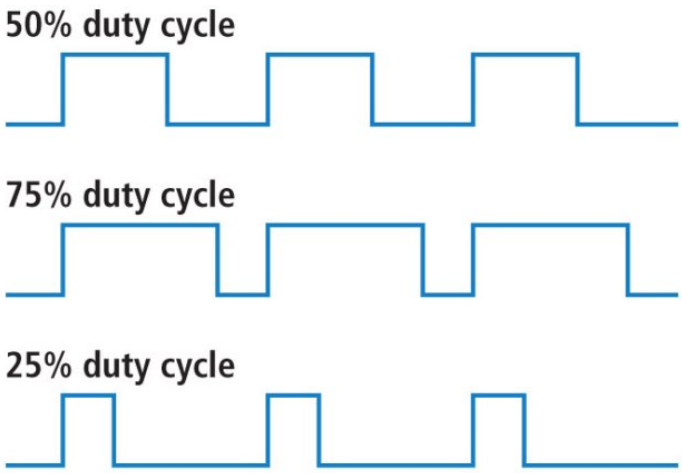
Ce panneau peut fournir une puissance maximale de 5W, et opère à 0.29 A.

e. Fonctionnement des servomoteurs



Le **servomoteur** fonctionne différemment d'un moteur normal. Ce dernier **tourne tant qu'il est alimenté** et effectue un **mouvement continu**, cela n'est donc pas très précis. En revanche, on peut voir sur l'image à gauche différents fils utiles au fonctionnement des servomoteurs. Traditionnellement, le fil rouge est utilisé pour l'alimention des moteurs et le noir pour la masse. Un autre fil est responsable de la commande du servomoteur, fournissant un signal PWM (Pulse Width Modulation) ou MLI en français.

Un signal PWM est appelé **pseudo analogique**. Il a la forme d'un signal carré d'une durée contrôlée permettant de limiter l'alimentation du moteur. L'image ci-dessous, permet de visualiser la forme de ce type de signal en comparaison avec un signal horloge.



Ce type de signal peut être **généré numériquement** grâce au microcontrôleur, qui génère une **pulsation de rapports cycliques** différents. Ensuite la moyenne de la tension est calculée par la formule en dessous afin de traduire ce signal en donnée que le servomoteur va utiliser pour contrôler l'alimentation des moteurs.

$$V_{moy} = \frac{1}{T}\left(\int_0^{T_{on}} E\,dt + \int_{T_{on}}^{T} 0\,dt\right) = \frac{1}{T} \cdot E \cdot T_{on} = E \cdot \frac{T_{on}}{T} = E \cdot \frac{N}{N_{max}}$$

Comme mentionné auparavant, le dispositif comporte deux moteurs pouvant assurer **deux axes de rotations**. Les deux moteurs peuvent tourner jusqu'à 180° dans chaque axe. Ceci est suffisant pour notre projet.

f. Carte STM32

Les caractéristiques techniques de la carte sont les suivantes :

- Microcontrôleur STM32F072RBT6 avec 256Ko de mémoire flash, 16Ko de RAM dans un boîtier LQFP64
- ST-LINK/V2 embarqué avec commutateur permettant d'utiliser le kit comme un ST-LINK/V2 autonome
- Alimentation de la carte par le bus USB ou à partir d'une tension d'alimentation de 5 V externe
- Alimentation externe pour application : 3V et 5V
- L3GD20, capteur de mouvement MEMS ST, gyroscope triaxial à sortie numérique
- Un capteur tactile linéaire ou quatre boutons tactiles
- USB utilisateur avec connecteur mini B
- Connecteur de carte fille RF-EEPROM
- Header d'extension pour toutes les E/S du LQFP64 permettant la connexion rapide avec la carte de prototypage et facilitant les mesures
- Deux boutons-poussoirs (utilisateur et reset)

Le composant le plus important est le microcontrôleur, est un ARM Cortex 32 bits et peut opérer jusqu'à 48 MHZ. Le choix de cette carte est justifié par le type d'alimentation externe qui est adapté aux capteurs. Ainsi, la présence de boutons-poussoirs et des boutons tactiles permet d'ajouter des fonctionnalités supplémentaires.
La présence de deux convertisseurs suivants est essentielle aussi pour ce projet :

- One 12-bit, 1.0 µs ADC (up to 16 channels)
  - Conversion range: 0 to 3.6 V
  - Separate analog supply: 2.4 V to 3.6 V
- One 12-bit D/A converter (with 2 channels)

En effet, afin d'exploiter les données analogiques reçues par les capteurs, il est nécessaire de convertir ces données en données digitales afin de pouvoir manipuler ces données dans les codes. La conversion Digitale\analogue est aussi importante pour la commande des moteurs.

En addition, la présence de plusieurs protocoles de transfert de données permet aussi d'avoir une flexibilité de choix selon le besoin et les capteurs utilisés.

- Communication interfaces
  - 2 I^2C interfaces supporting Fast Mode Plus (1 Mbit/s) with 20 mA current sink, one supporting SMBus/PMBus and wakeup
  - 4 USARTs supporting master synchronous SPI and modem control, two with ISO7816 interface, LIN, IrDA, auto baud rate detection and wakeup feature
  - 2 SPIs (18 Mbit/s) with 4 to 16 programmable bit frames, and with I^2S interface multiplexed
  - CAN interface
  - USB 2.0 full-speed interface, able to run from internal 48 MHz oscillator and with BCD and LPM support

4. Software : Présentation du code

a. Convertisseur Analogique-numérique

Le but de l'algorithme principal est de traiter toutes les données reçues sur la carte, afin de modifier les signal PWM de sortie.

C'est pour cela que la **conversion analogique-numérique** des données reçues par les PR est très importante pour le système, ainsi que pour la précision de la commande des servomoteurs.

On initialise tout d'abord tous les paramètres de la CAD dans la fonction **"`BSP_ADC_Init_IT`".** Cette fonction permet de configurer les entrées de conversion, ainsi que la résolution de la conversion.

La conversion analogique-numérique se fait sur **3 channels** consacrés pour chaque voie de RP. Ainsi, la conversion se passe par l'utilisation des interruptions. Cela se passe dans la fonction **"`ADC1_COMP_IRQHandler`" .**
Nous déclarons tout d'abord des variables externes qu'on les données les valeurs de la conversion de chaque channel.

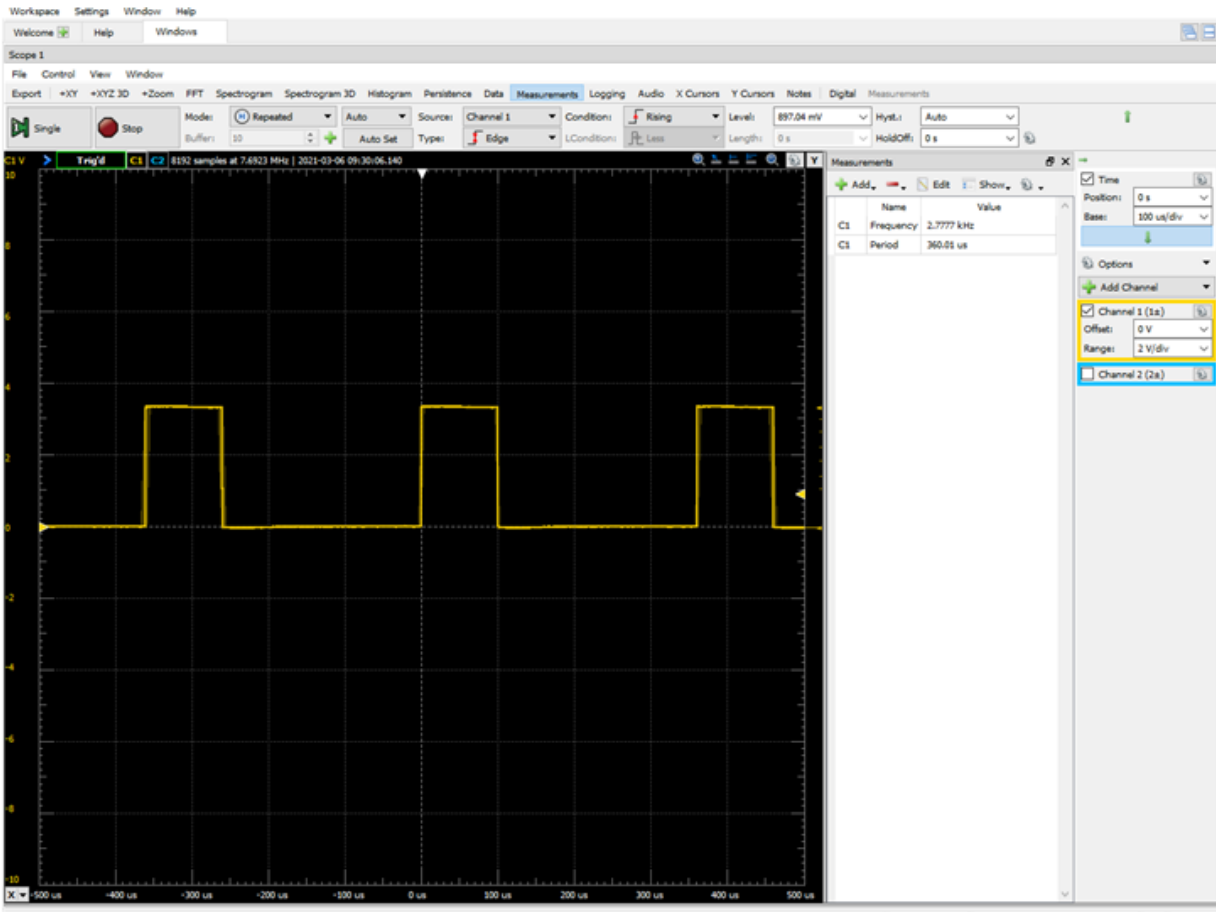
b. Signal PWM

Afin de générer un signal PWM il faut régler **trois paramètres**. **L'amplitude** de ce signal, ainsi que la **période** entre deux fronts sont importantes à définir. Ainsi, le paramètre qui va nous permettre de commander les servomoteurs est le **rapport cyclique** de ce signal.

Dans notre projet nous avons besoin de générer deux signaux PWM, et les modifier indépendamment.

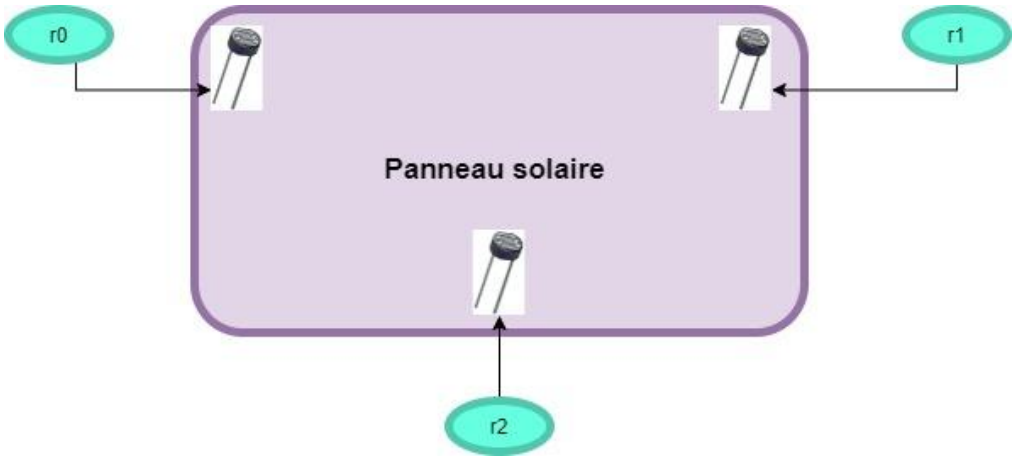
La fonction "**BSP_TIMER_PWM_Init**" permet d'initialiser tous les paramètres mentionnées auparavant pour générer deux signaux PWM. Nous déclarons les signaux sur les canaux PA9 et PA8.

Nous avons ensuite simulé un signal PWM afin de savoir les valeurs minimums et maximums à mettre pour les registres de rapports cycliques. Ces valeurs minimums et maximums dépendent des servomoteurs et de l'angle qu'ils peuvent couvrir.



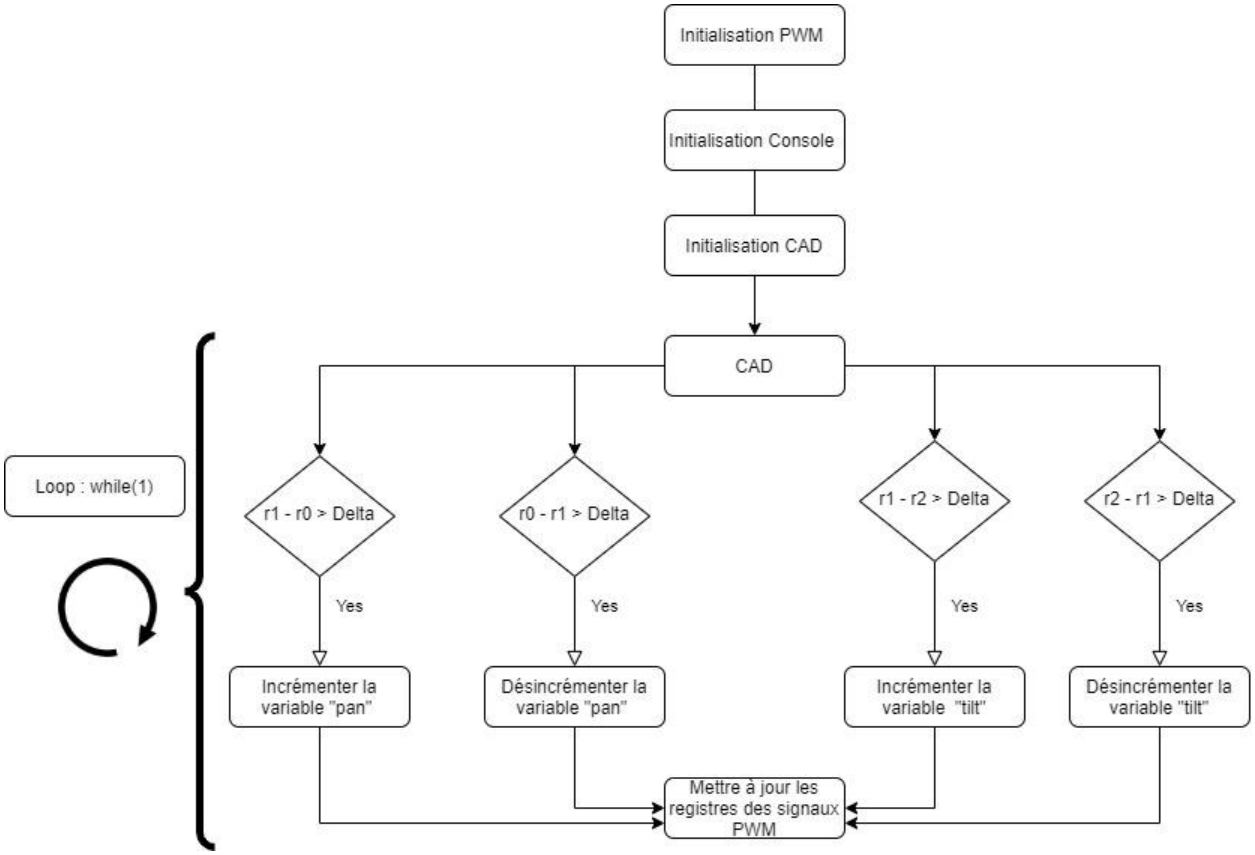
c. Organigramme de l’algorithme

Comme mentionnée auparavant nous allons déclarer **3 variables globales,** chacune portant la valeur d’un canal de conversion. Les variables vont être appelées “r0”, “r1” et “r2”. Ces variables représente les capteurs de la manière suivante :



Nous allons examiner ces valeurs dans une série de boucles “if”. En effet, nous allons comparer ces valeurs à une valeur **“delta”** que nous fixons.

Ensuite, nous allons déclarer deux variables **“tilt”** et **“pan”**, pour les modifier selon les conditions qui ont été vérifiées. Les valeurs de ces deux variables vont être utilisées à la fin de la boucle **“while”** pour modifier les registre commandant les rapport cycliques des signaux PWM.



Nous commençons notre algorithme par une **phase d'initialisation**. En effet, il est nécessaire d'initialiser les signaux PWM, c'est-à-dire définir les différentes variables comme la période, les canaux etc. Nous devons ensuite initialiser la console : pour ce faire, nous utilisons Putty et établissons une connexion USART qui permet d'afficher tous les messages que l'on souhaite.Ils vont permettre de savoir en permanence dans quel cas nous sommes de l'algorithme. Finalement, nous devons initialiser la conversion analogique digitale (ou numérique). Cela consiste en l'initialisation d'un tableau permettant de stocker les données utiles pour les différents traitements des cas d'utilisation.

Une fois la phase d'initialisation finie, nous rentrons dans une **boucle infinie**. Dans cette boucle on peut faire face à **4 cas différents** :

- (r1-r0 > delta) :
  Nous comparons tout d'abord les lectures de la CAD pour les photorésistances droite et gauche par rapport au delta. Si cette condition est vraie, la lumière est à droite, donc on incrémente la variable "pan" avec des pas de 6. Cela permet d'augmenter l'angle du servomoteur d'un degré à chaque tour de boucle.
  On regarde également la valeur de la variable "pan" si elle n'a pas encore dépassé la valeur maximum que nous avons fixée. Si cette condition est vraie, la variable "pan" ne sera plus incrémentée.
- (r0-r1 > delta) :
  Cette condition permet de tourner le servomoteur à gauche. En effet, si cette condition est vraie, on décrémente la variable "pan" avec des pas de 6. On regarde également si la variable "pan" n'est pas en dessous de la valeur minimum, afin de la décrémenter.
- (r1-r2 > delta) :
  Afin de savoir quand est ce qu'il faut incrémenter ou décrémenter la variable de la "tilt", il suffit de comparer la lecture de la CAD d'un canal en haut avec le canal en bas.
  En effet avec la valeur de "delta", la lumière sera plus ou moins centrée entre les deux PR r0 et r1, donc leurs lectures ne seront pas très différentes. Cela permet de simplifier le code, ainsi que de minimiser des mouvements du panneau non nécessaires.
  Dans ce cas, la lecture de la photorésistance en haut est supérieure à celle en bas. On incrémente la variable "tilt" afin de déplacer le panneau vers le haut. On regarde également si la valeur de la variable "tilt" est au-dessus de la valeur maximum.
- (r2-r1 > delta) :
  Cette condition permet de vérifier s'il faut baisser le panneau. On applique le même principe, tout en observant la valeur de ma variable "tilt" pour ne pas dépasser la valeur minimum.

- **Choix de delta :**

Après avoir écrit le code, nous avons choisi une valeur "delta" élevée afin d'éviter d'avoir des mouvements brusques. Nous avons choisi également un délai élevé entre chaque tour de boucle, afin de pouvoir observer les lectures de la CAN et les valeurs des variables "tilt" et "pan". Les différents tests réalisés nous ont permis de choisir une valeur de delta appropriée par rapport à la fiabilité des composants que nous avons.

5. Difficultés rencontrées et résolution

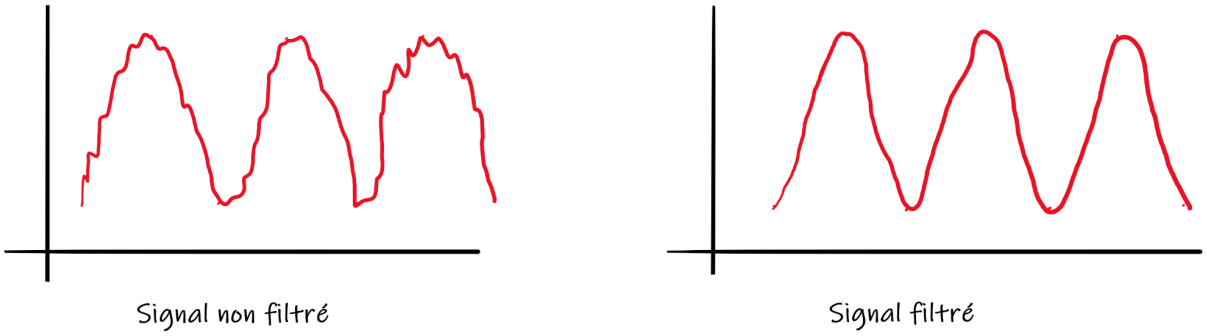
5.1 Difficulté liée à la conversion analogique-numérique

Une grande difficulté que nous avons rencontrée, a été liée à la CAD. Au début du projet nous avons choisi de faire la conversion à la demande de l'utilisateur, avec une boucle qui fait la conversion sur les 3 canaux. Ensuite, on stocke les lectures de la CAD dans un tableau.

Ayant obtenu des valeurs de conversion irréalistiques, et ne correspondant pas à la lumière, nous avons décidé de faire la conversion d'une autre manière. En effet, nous avons changé notre code afin de faire la conversion avec des interruptions.

5.2 Besoin de filtrage du signal

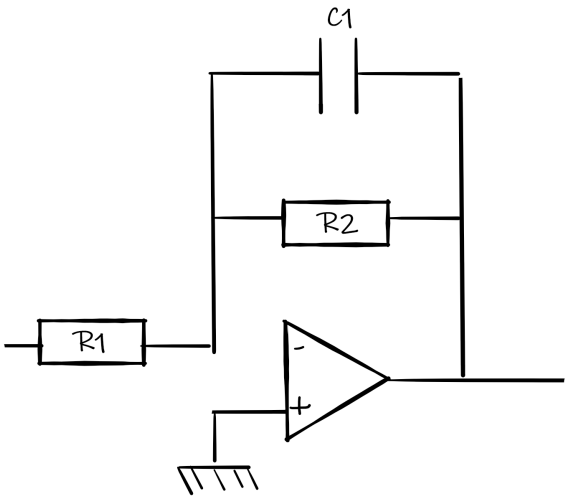
Nous avons aussi dû filtrer le signal avec un filtre passe bas aussi appelé filtre anti-repliement spectral dans notre cas. Pour ce faire, nous avons utilisé un condensateur. Voici un exemple de l'utilité d'un tel filtrage :



Le but du filtrage est de réduire le SNR (Signal Noise Ratio) :

En effet, comme l'unité $(S/B)^2$ est d'un ordre supérieur à (S/B), le SNR sera plus petit pour un signal filtré.

Voici le schéma du circuit électronique de notre cas de figure :



Le condensateur constitue le filtre passe bas du circuit. En effet, en haute fréquence, ce condensateur est équivalent à un fil, le module du gain vaut donc 0. On sait que : Bande Passante (de l'ordre de 10^6) x la Valeur Efficace du Bruit (de l'ordre de 10nV/Hz). Pour réduire la Valeur Efficace du bruit, on réduit donc la Bande Passante, ainsi on ne garde que le signal intéressant.

6. Conclusion

Pour conclure, malgré les quelques difficultés rencontrées nous avons réussi à mener à bien ce projet et à obtenir des résultats satisfaisants. Ce projet étant très complet, nous avons pu appliquer les connaissances acquises lors du cours de Microcontrôleur ainsi que de la programmation en C. Ces bases ainsi que l'étude préalable des composants et des enjeux du projet se sont montrés très utiles pour réaliser le projet et nous ont permis de gagner en efficacité. La gestion du projet en binôme s'est aussi très bien passée et nous sommes tous les deux satisfaits de notre production finale.

7. Annexe

Pour lancer le projet, il suffit d'avoir ouvert le projet et lancer le code main.c

Fonction main :

```
//Variables globales
uint8_t cmpt= 1;
int16_t r1,r0,r2;
//Definition des channel de conversion ADC
// r0                                                  r1
//                          r2
int main()
{

    int8_t delta = 50;
    int tilt = 1500;
    int pan = 1500;
    SystemClock_Config();
    //Initialisation PWM
    BSP_Console_Init();
    mon_printf("La Console est ready!\r\n");
        BSP_TIMER_PWM_Init();

    //Initaialisation de L'ADC
    BSP_ADC_Init_IT();
    //BSP_PWM_HS422_Init();
    // Boucle Infinie
    while(1)
    {
                mon_printf("r0 : %d\n\r", r0);
        mon_printf("r1 : %d\n\r", r1);
        mon_printf("r2 : %d\n\r", r2);
        mon_printf("Valeur pan : %d\n\r", pan);
        mon_printf("Valeur tilt : %d\n\r", tilt);

        //Reglage du Pan
        if(r1-r0 > delta)
        {
            if (pan < max_pan){
            mon_printf("**********Aller à Droite**********\r\n");
```

```
                pan = pan +6;
                        }
        }
        if(r0-r1 > delta)
        {
            if (pan > min_pan){
            mon_printf("***********Aller à Gauche************\r\n");
                pan = pan - 6;
                        }
        }

        //Reglage du Tilt
        if(r1-r2 > delta)
        {
            if (tilt < max_tilt){
            mon_printf("***********Monter************\r\n");
                tilt = tilt + 6;
                        }
        }
        if(r2-r1 > delta)
        {
            if (tilt > min_tilt){
            mon_printf("*********** Descendre ************\r\n");
                tilt  = tilt -6;
                        }
        }

        TIM1->CCR2 = pan;
        TIM1->CCR1 = tilt;

        BSP_DELAY_ms(20);
    }

}
```

Fonction d'initialisation du CAD :

```
void BSP_ADC_Init_IT(void)
{
```

```
    // Activation de horloge du GPIOC
    // Mettre a '1' le bit b19 du registre RCC_AHBENR
    RCC->AHBENR |= (1<<19);
    // Configure le pin PC1 en mode Analog
    // Mettre à "11" les bits b3b2 du registre GPIOC_MODER
    GPIOC->MODER |= (0x03)|(0x03<<2)|(0x03<<4);
    // Activation de horloge de ADC
    // Mettre a '1' le bit b9 du registre RCC_APB2ENR
    // voir page 130 pour les autres bits
    RCC->APB2ENR |= (1<<9);
    // Reset de la configuration de ADC
    // Mise a zero des regsitres de configuration de ADC
    ADC1->CR = 0x00000000;
    ADC1->CFGR1 = 0x00000000;
    ADC1->CFGR2 = 0x00000000;
    ADC1->CHSELR = 0x00000000;
    // Choix de la resolution (nombre de bits des data)
    // bits b4b3 (Data resolution)
    // 00: 12 bits
    // 01: 10 bits
    // 10: 8 bits
    // 11: 6 bits
    ADC1->CFGR1 &= ~(0x03 <<4);
    // Choix de la source horloge pour ADC
    // bits b31b30 CKMODE[1:0]: ADC clock mode
     // 00: ADCCLK (Asynchronous clock mode), generated at product level
(refer to RCC section)
    // 01: PCLK/2 (Synchronous clock mode)
    // 10: PCLK/4 (Synchronous clock mode)
    // 11: Reserved
    ADC1->CFGR2 |= (0x01 <<31UL);
    ADC1->CR |= ADC_CR_ADCAL; /* (3) */
        while ((ADC1->CR & ADC_CR_ADCAL) != 0); /* (4) */
    // Activer ADC
    // Mettre a '1' le bit b0 du registre ADC_CR
    ADC1->CR |= (1<<0);
    // Demarrer la conversion
    // Mettre a '1' le bit b2 du registre ADC_CR
    ADC1->CR |= (1<<2);
```

```
    /* This code example configures the AD conversion in continuous mode
and in
    backward scan. It also enable the interrupts. */
    /* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
    /* (2) Select the continuous mode and scanning direction */
    /* (3) Select CHSEL10 and CHSEL11 */
    /* (4) Select a sampling mode of 111 i.e. 239.5 ADC clk to be greater
than 17.1us */
    /* (5) Enable interrupts on EOC, EOSEQ and overrrun */
    /* (6) Wake-up the VREFINT (only for VBAT, Temp sensor and VRefInt) */
    ADC1->CFGR1 |= ADC_CFGR1_CONT & ~ADC_CFGR1_SCANDIR; /* (2) */
          ADC1->CHSELR  =  ADC_CHSELR_CHSEL10  |  ADC_CHSELR_CHSEL11  |
ADC_CHSELR_CHSEL12 ; /* (3) */
    ADC1->SMPR |= ADC_SMPR_SMP_0 | ADC_SMPR_SMP_1 | ADC_SMPR_SMP_2; /* (4)
*/
    ADC1->IER = ADC_IER_EOCIE ; /* (5) */
    ADC->CCR |= ADC_CCR_VREFEN; /* (6) */
    /* Configure NVIC for ADC */
    /* (7) Enable Interrupt on ADC */
    /* (8) Set priority for ADC */
    NVIC_EnableIRQ(ADC1_COMP_IRQn); /* (7) */
    NVIC_SetPriority(ADC1_COMP_IRQn,1); /* (8) */
}
```

Fonction de la CAD :

```
extern uint8_t cmpt;
extern uint16_t r1,r0,r2;
void ADC1_COMP_IRQHandler(void)
{
    if ((ADC1->ISR & (1<<3)) == (1<<3))
    {
        cmpt=1;
        ADC1->ISR |= ADC_ISR_EOSEQ;
        r2= ADC1->DR;
    }
    else if( cmpt == 1)
    {
        r1= ADC1->DR;
        cmpt = 2;
```

```
    }

    else if (cmpt == 2)
    {
        r0= ADC1->DR;
        cmpt = 3;
    }
}
```

Fonction initialisation du PWM :

```
void BSP_TIMER_PWM_Init()
{
    // Activer horloge du GPIOA
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
    // Configurer PA8 et PA9 en mode Alternate Function
    GPIOA->MODER &= ~(GPIO_MODER_MODER8_Msk | GPIO_MODER_MODER9_Msk);
    GPIOA->MODER |= (0x02 <<GPIO_MODER_MODER8_Pos) | (0x02
    <<GPIO_MODER_MODER9_Pos);
    // Choisir Alternate Function AF2 (TIM1)
    GPIOA->AFR[1] &= ~(0x000000FF);
    GPIOA->AFR[1] |= (0x00000022);
    // ACtiver horloge du Timer TIM1
    RCC -> APB2ENR |= RCC_APB2ENR_TIM1EN;
    // Faire un Reset de la configuration du TIM1
    TIM1->CR1 = 0x0000;
    TIM1->CR2 = 0x0000;
    TIM1->CCER = 0x0000;
    // Configuration frequence de comptage
    // Prescaler : registre TIM61PSC
    // Fck = 48MHz -> /48 = 1MHz frequence de comptage : 1ums
    TIM1->PSC = (uint16_t) 48 -1;
    // Configuration periode des evenements
    //ARR: registre TIM6_ARR
    // periode PWM = 11 ms
    TIM1->ARR = (uint16_t) 11000;
    // Activer le registre Auto-Reload Preload
    // prechargement
    TIM1->CR1 |= TIM_CR1_ARPE;
```

```
    // Reset de la configuration
    TIM1->CCMR1 = 0x0000;
    TIM1->CCMR2 = 0x0000;
    // Choix du mode PWM mode 1 pour les 2 cannaux
    TIM1->CCMR1 |= (0x06 <<TIM_CCMR1_OC1M_Pos) | TIM_CCMR1_OC1PE;
    TIM1->CCMR1 |= (0x06 <<TIM_CCMR1_OC2M_Pos) | TIM_CCMR1_OC2PE;
    // Definir les valeurs initiales pour le rapport cyclique des 2 PWM
    TIM1->CCR1 = 900;
    TIM1->CCR2 = 900;
    // Activer les sorties
    TIM1->CCER |= TIM_CCER_CC1E | TIM_CCER_CC2E;
    // Activer la sortie principale
    TIM1->BDTR |= TIM_BDTR_MOE;
    // Lancer le Timer TIM1
    TIM1->CR1 |= TIM_CR1_CEN;
}
```