Projet Saé-31 : Cible de carabine laser

Compte rendu de conception

Conception et réalisation de la partie électronique d'une cible de carabine laser

Gr 308
Maxime PHRAVIXAY
Thomas FERNANDEZ
2022-2023
IUT Lyon 1 – département GEII
Responsable (professeurs) : ROBIN Gael & LACHARMOISE Cédric

Sommaire

Introduction

Des étudiants en GMP ont fabriqué, lors de leur projet de deuxième année, la partie mécanique d'un stand de tir pour une carabine laser et leur professeur a conçu la carabine laser. Cependant la partie électronique de ce stand de tir nécessite des changements afin de le rendre complètement fonctionnel.

Dans le cadre du projet GEII de deuxième année, nous devons étudier le fonctionnement du stand de tir pour imaginer, concevoir et réaliser les changements nécessaires afin de pouvoir utiliser le stand de tir avec la carabine laser.

Voici un synoptique simplifié du fonctionnement global du stand de tir :

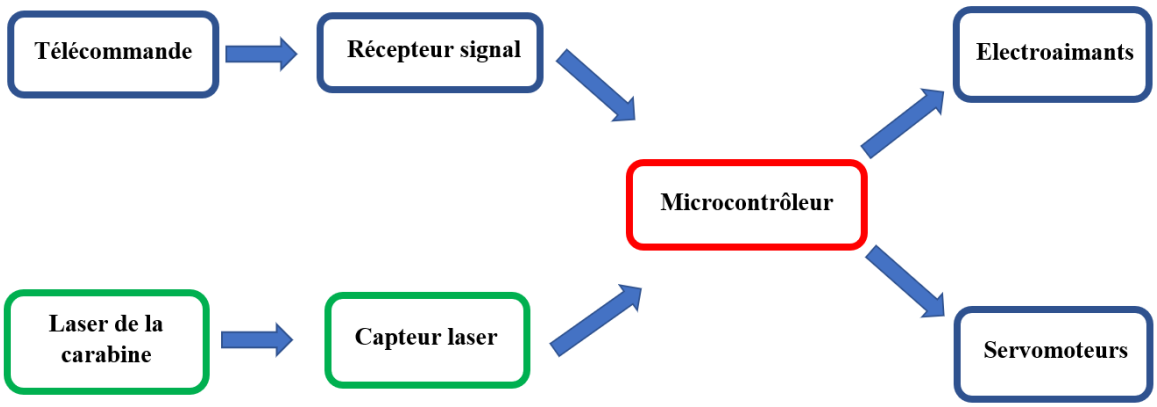


Figure 1: Synoptique simplifié

Cahier des charges

Ce stand est composé :

- D'une télécommande avec 4 boutons,
- D'un récepteur de signaux, associer à la télécommande
- De cinq capteurs laser
- Des verres convexes concentrant les rayons incidents sur chacun des capteurs, cela afin d'obtenir une meilleure précision malgré la circonférence moyenne des cibles.
- De deux servomoteurs, (le 1er pour faire descendre les panneaux des cibles, le 2nd pour changer de mode entre tir debout/couché)
- De cinq électroaimants,
- D'une matrice de transistor
- D'un microcontrôleur

Présentation approfondie du projet & Schéma synoptique détaillé

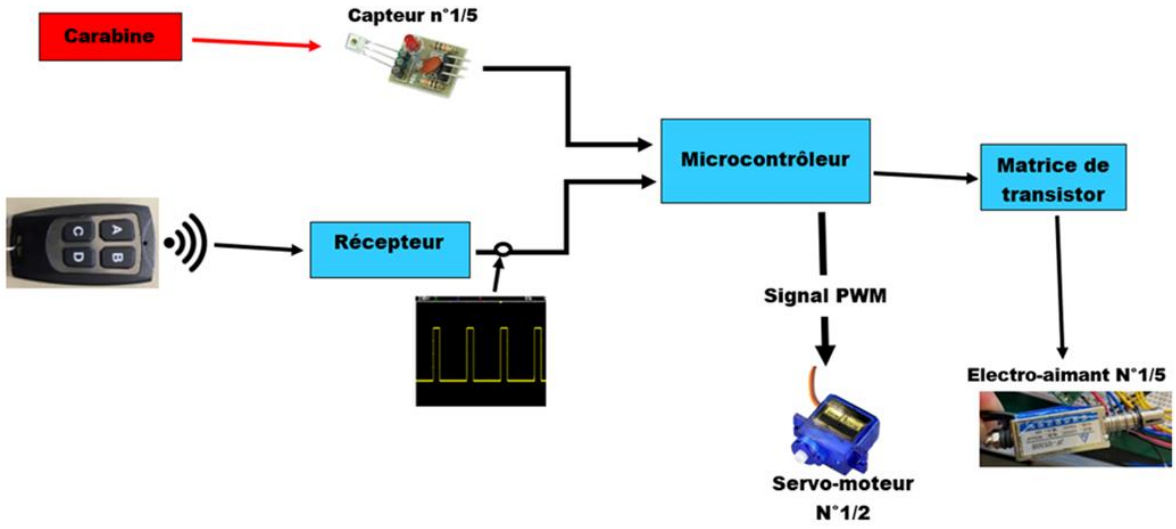


Figure 2: Synoptique détaillé

Fonctionnement du stand de tir

A l'état initial, les fentes de chaque cible sont obstruées par des panneaux.

A l'appui du bouton "A" de la télécommande, les panneaux sont abaissés simultanément par un servomoteur (← n°1) (voir image n°3) pour découvrir le centre des cibles.

Le joueur peut alors commencer à tirer à la carabine. Lorsqu'une des cibles est touchée par un laser, que ce soit précisément sur le capteur ou bien suffisamment proche de ce dernier, le panneau associé vient le recouvrir.

Le stand a également deux configurations, tir debout qui est la configuration de base et tir couché dans laquelle la taille du centre de la cible est réduite.

Pour passer de la configuration tir debout à tir couché ou inversement, on appuiera sur le bouton "B" de la télécommande. Cela envoie un signal au récepteur, qui le transmet ensuite au microcontrôleur. Le microcontrôleur va alors générer et envoyer un signal PWM au servomoteur (←n°2) (voir image n°3) qui par une action mécanique va faire glisser une plaque percée de fente de taille égale à celle de la base de la cible. Et cela jusqu'à ce que les cinq autres fentes de plus petite taille, se superposent aux lentilles convexes.

Images du stand de tir :

Figure 3 : Image n°1 : face avant du stand

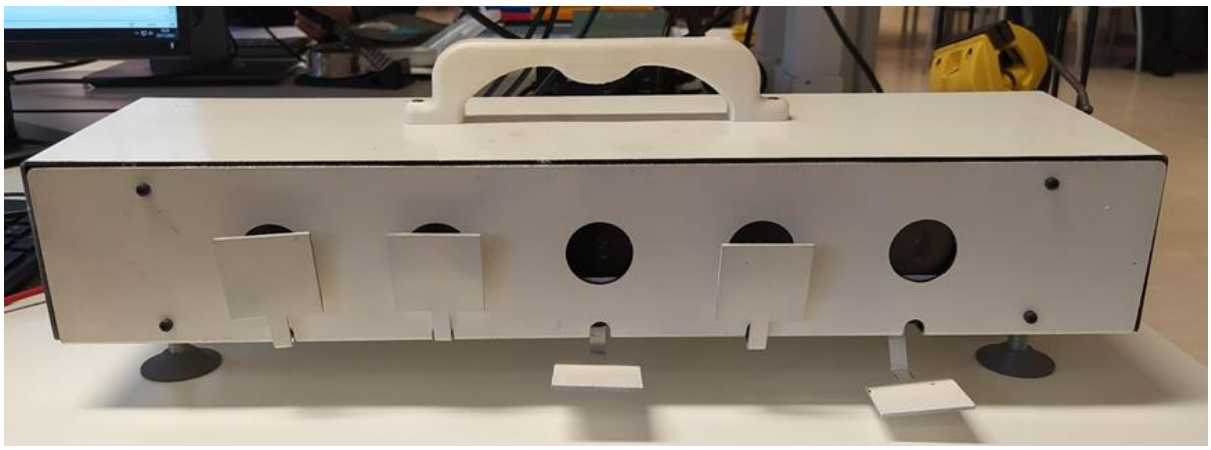


Figure 4 : Image n°2 :

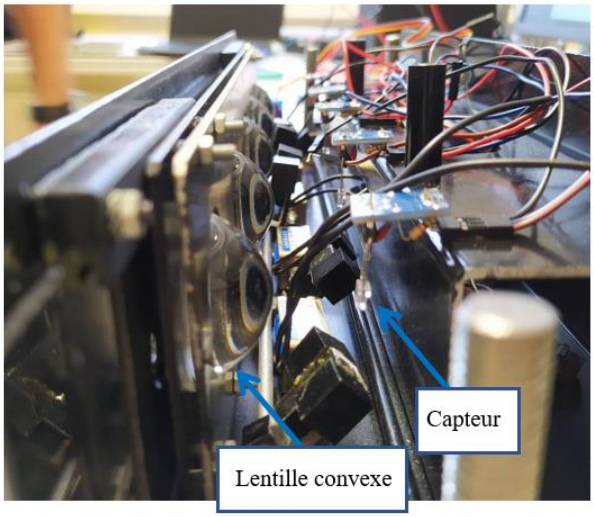
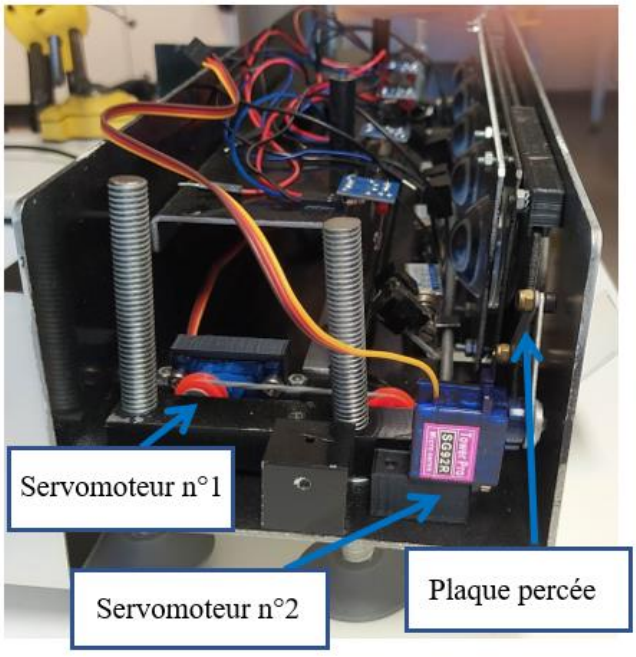


Figure 5 : Image n°3 :



Contraintes

- Délais maximums de six mois pour la réalisation du projet
- Utilisation d'une télécommande pour commander les différentes fonctionnalités
- Compatibilité des capteurs avec les lasers tirés par la carabine (longueur d'onde avec une même plage de valeur)
- Gestion de cinq capteurs
- Prise en compte du mode d'alimentation des électro-aimants afin de pouvoir les activer aux moments désirés

Documents de gestion de projet

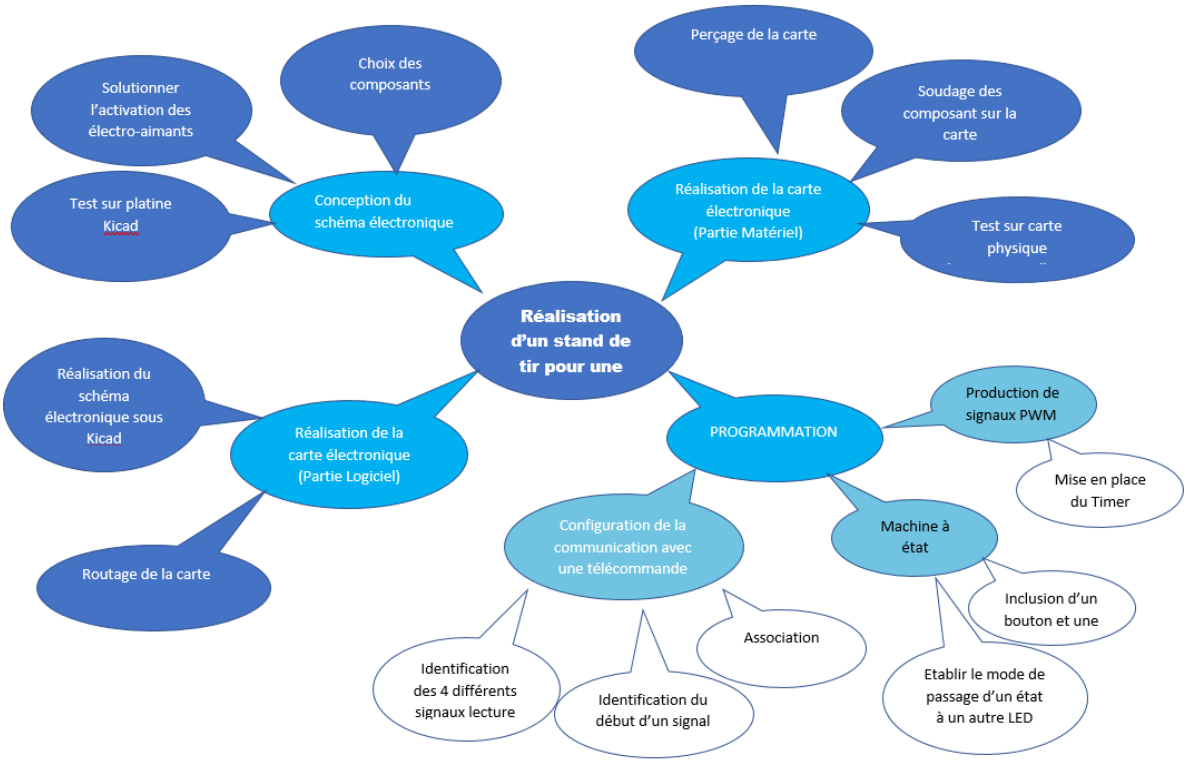
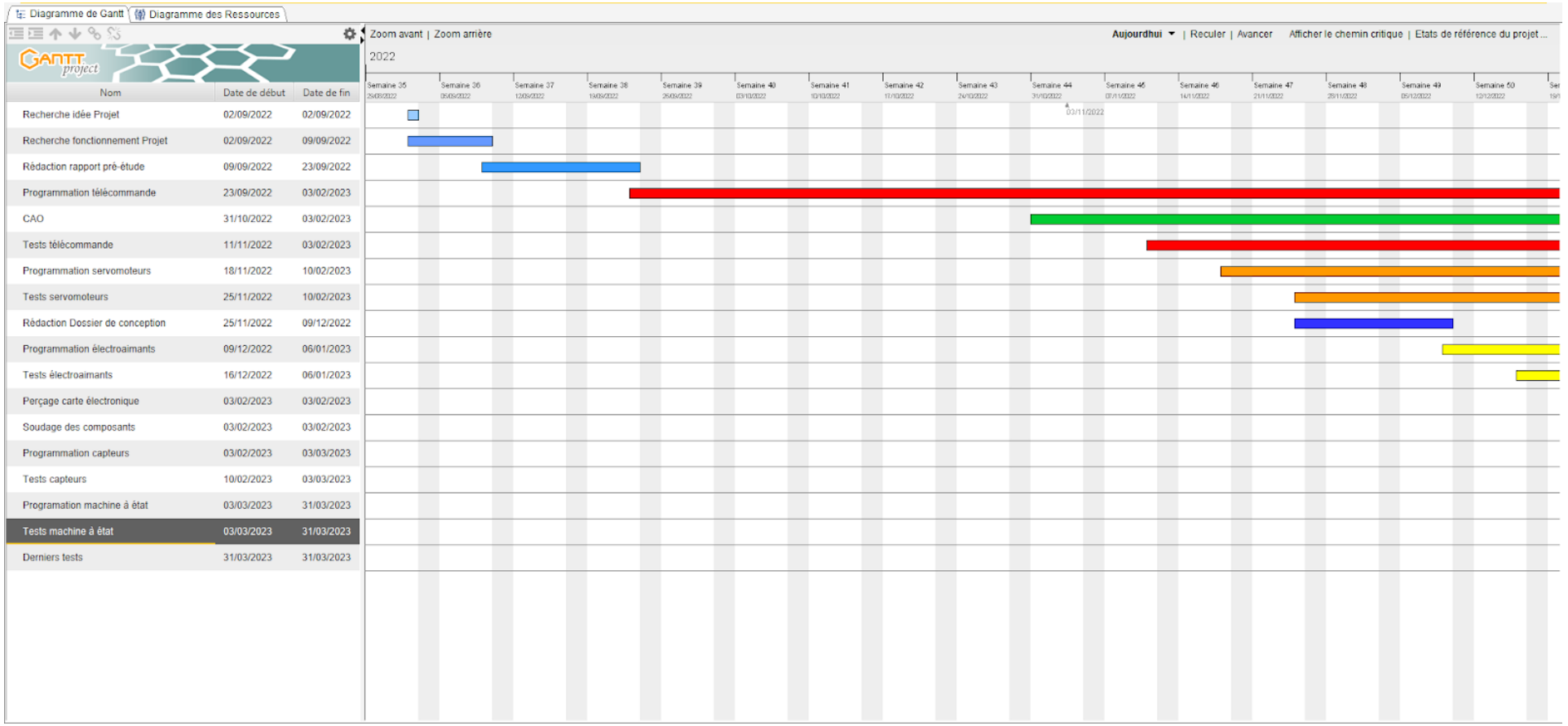
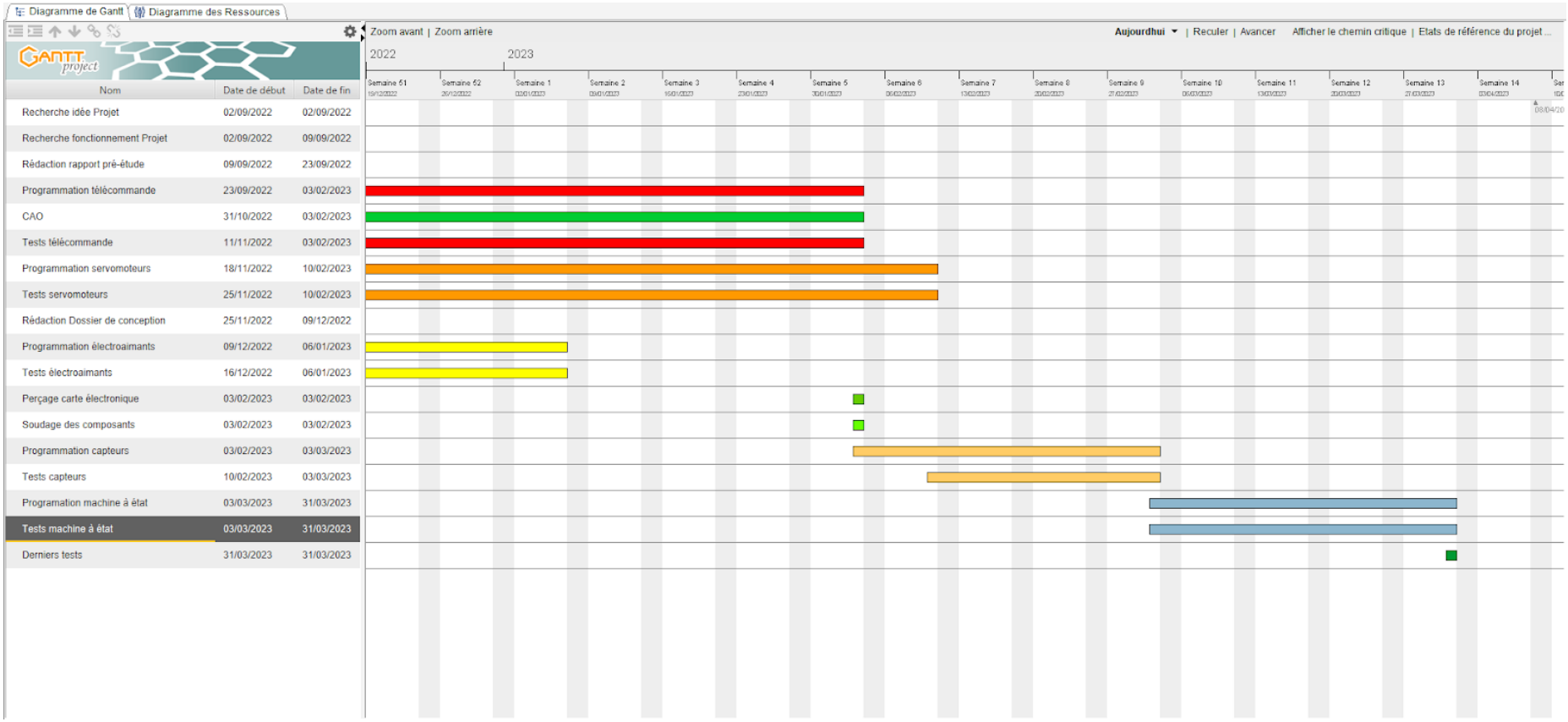


Figure 6 : Organigramme



Le Gantt ci-dessus récapitule toutes les tâches effectuées entre septembre et décembre 2022 pour ce projet.

Figure 7 : Gantt du Projet



Le Gantt ci-dessus récapitule toutes les tâches effectuées entre décembre 2022 et mars 2023 pour ce projet.

Schéma structurel détaillé et ses explications

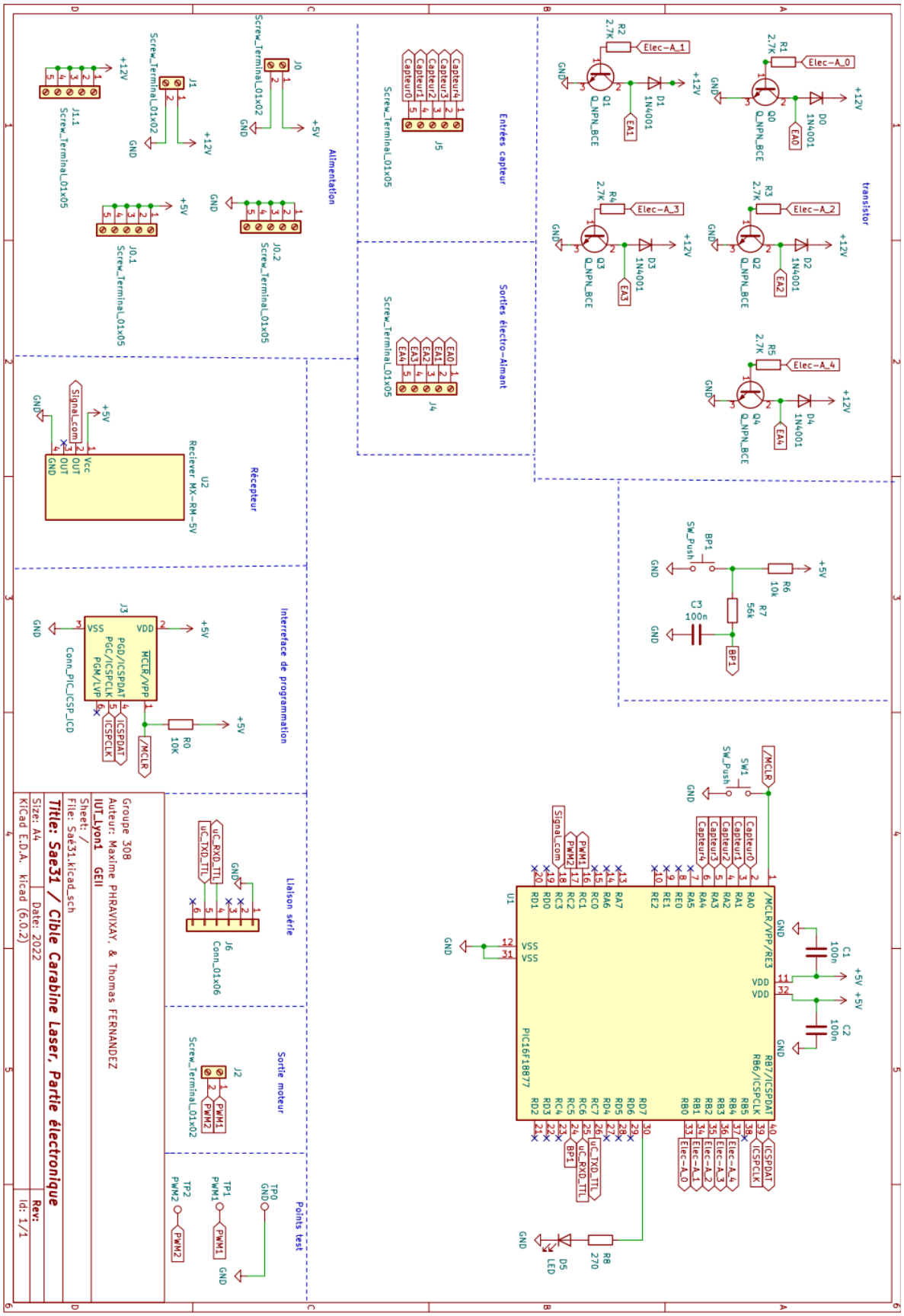


Figure 8 : Schéma réalisé sous Kicad

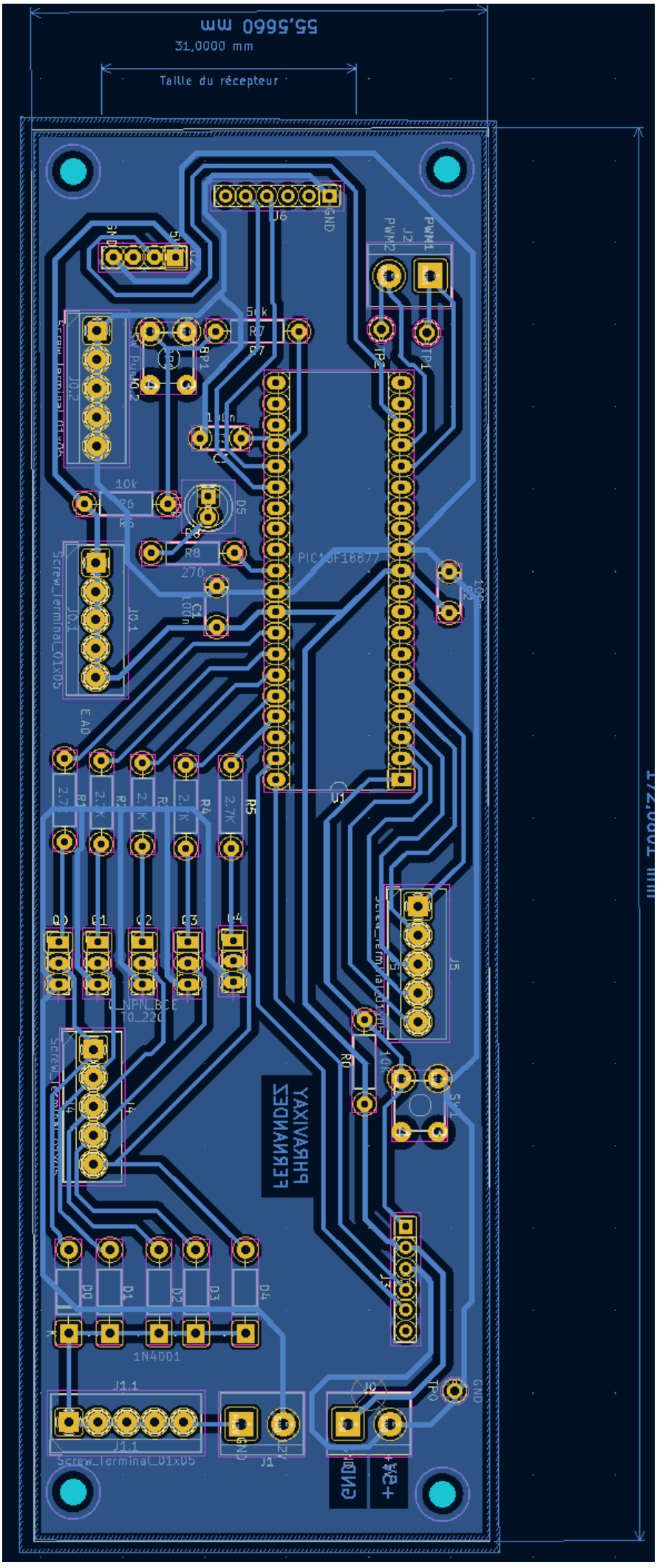
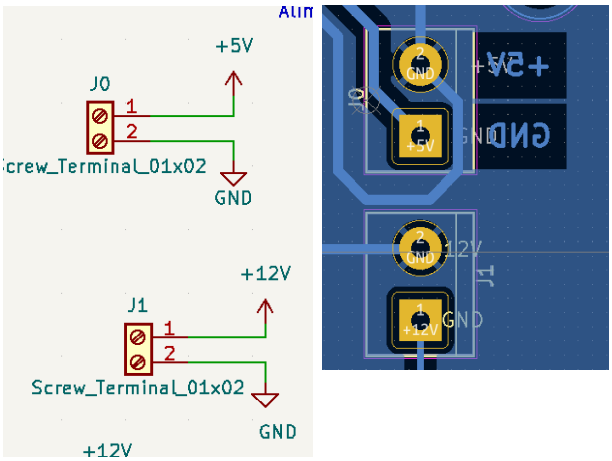


Figure 9 : PCB réalisé sur le logiciel

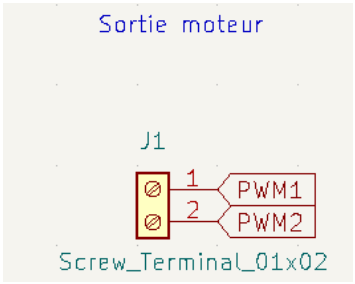
Pour les détails sur les empreintes utilisées voir partie “Nomenclature” page…

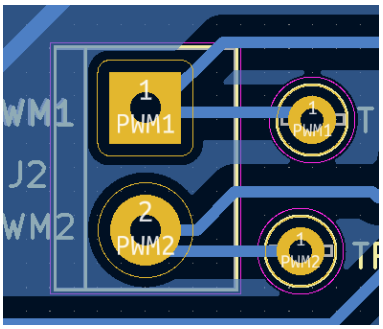
L’alimentation



Il s’agit de deux borniers à visses composé de deux entrées. Les entrées du bornier située en haut de la figure sont reliés respectivement aux GND et à une tension de cinq volts. Quant aux entrées du bornier située en bas de la figure sont reliés, l’un au GND et le 2nd au 12 volts.

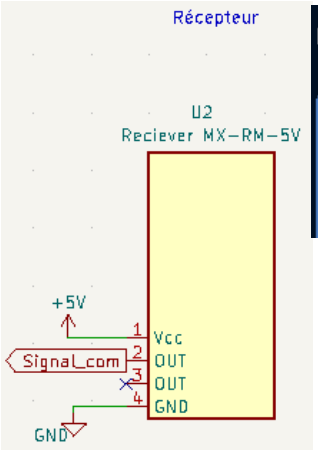
PWM

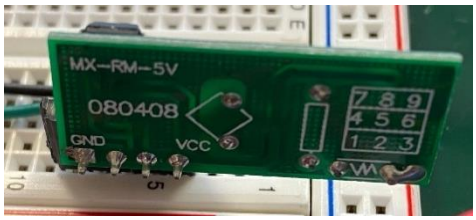




Il s’agit d’un bornier à visse composé de deux sorties. Ils doivent être reliées aux entrée PWM (fil jaune) des servomoteurs.

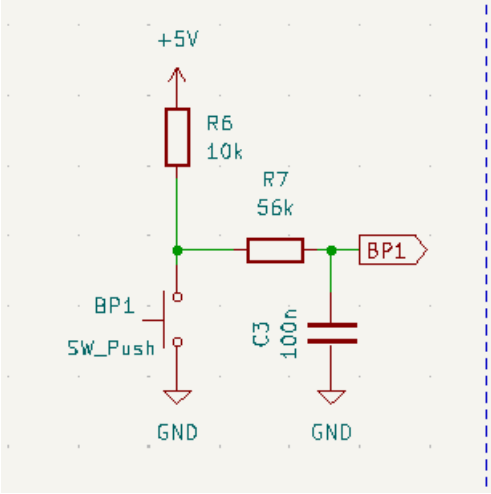
Récepteur (←←télécommande)

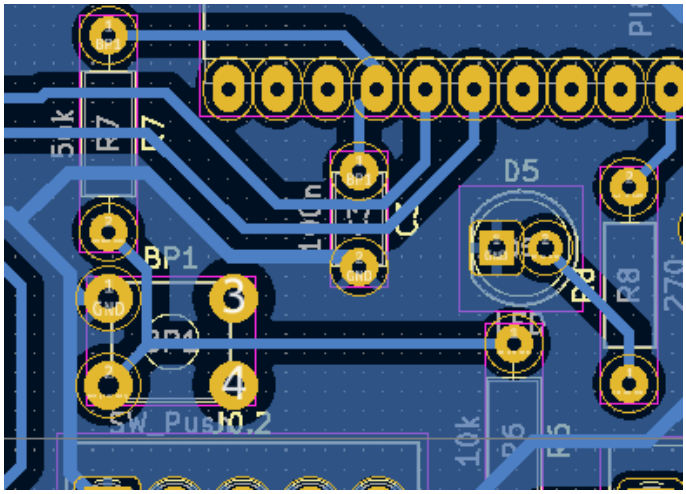




Sur la carte électronique il s'agit d'un support 4 broches sur lequel est assemblé le récepteur MX-RM-5V.
Le symbole utilisé dans le schéma a été créé avec l'éditeur de symbole se trouvant dans la page d'exploitation du logiciel KICAD.

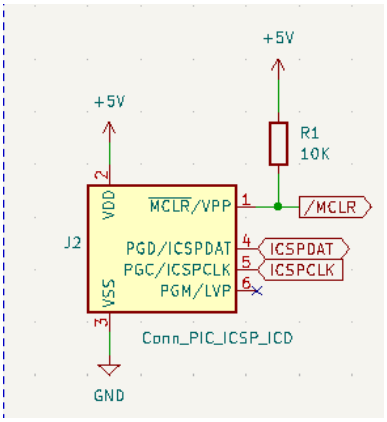
Machine à état

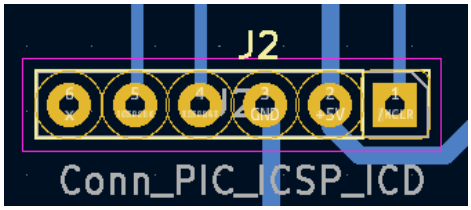




A la demande du client nous avons incorporé a notre système, un bouton poussoir (BP1) et une LED (D5) afin de pouvoir crée une machine à état.

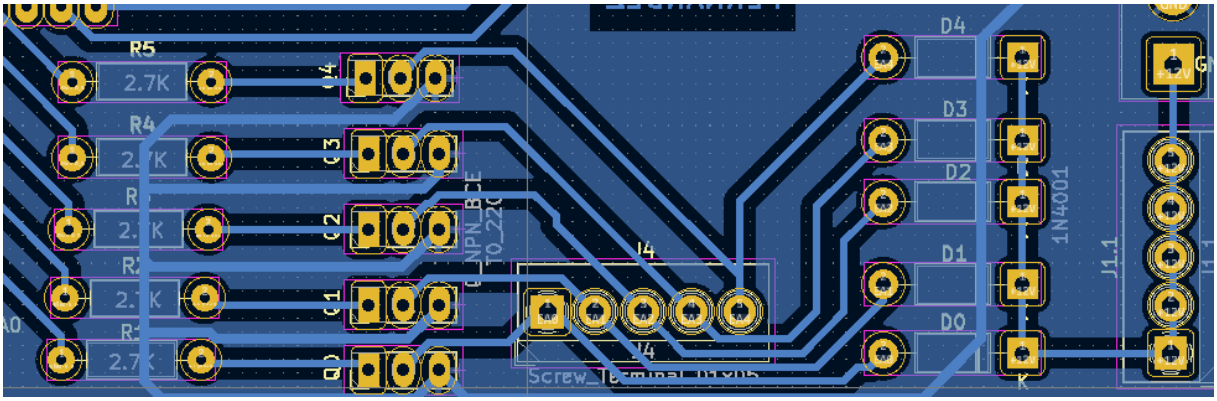
Interface de programmation

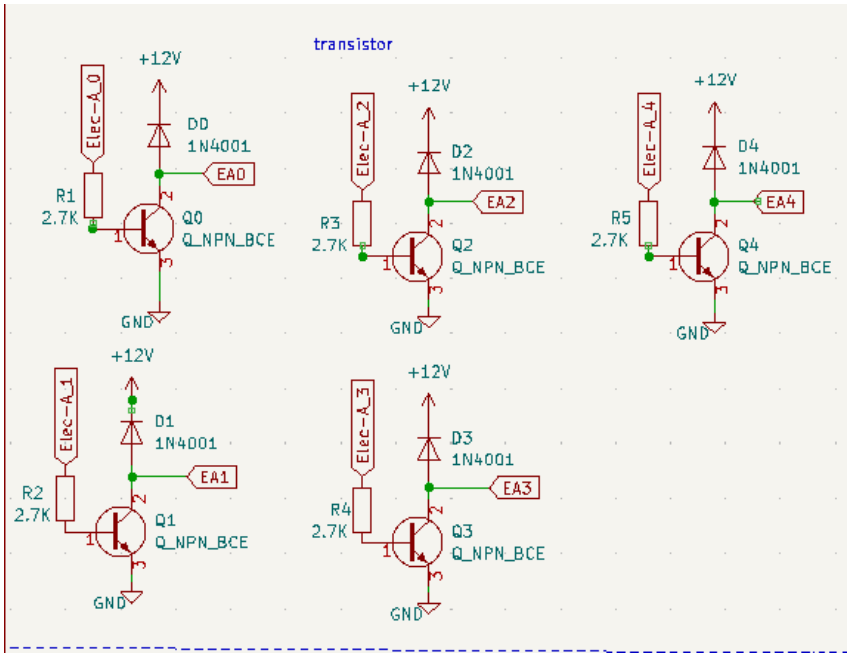




Ici il s'agit d'un connecteur six broches, que l'on connecte à un PIC KIT qui est relié à un PC par un câble USB, afin de transférer des programmes au microcontrôleur.

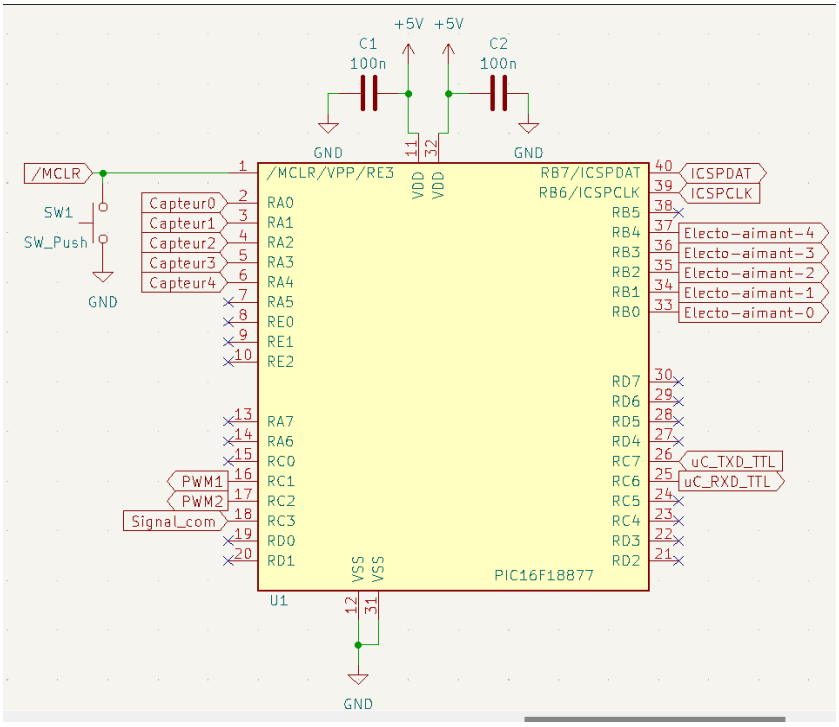
Réseau de transistor

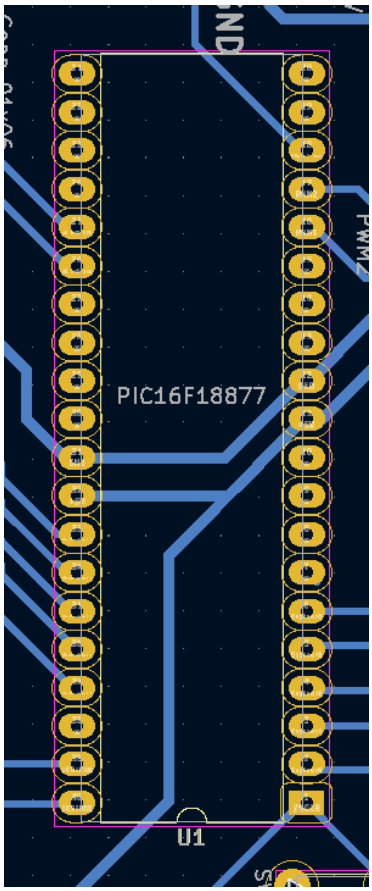




A l'origine nous avions repris de câblage composé d'une matrice de transistors Darlington, mais nous nous sommes rendu compte que cela ne convenait tout simplement pas. Car nous ne réussissions pas à avoir un courant suffisant pour actionner les nombreux électro-aimants. Nous avons alors finalement opté pour un câblage à base de transistor et de résistance. Câblage que l'on peut voir sur le PCB si dessus.

Microcontrôleur





Sachant que l'on a besoin d'un microcontrôleur avec :

- cinq entrées capteurs,
-cinq sorties pour commander les électroaimants
-deux sorties pouvant générer des signaux PWM
-et une entrer supplémentaire sur laquelle on peut appliquer un Timer2 et un Timer4.

Cette entrer est relier au récepteur de signaux.

➔ Le microcontrôleur nécessaire doit donc contenir un Timer2, un Timer4 et aussi un Timer6 (pour générer des signaux PWM).

Notre choix s'est donc porté vers le PIC16F18877

Nomenclature

Référence	Valeur	Type	Empreinte
D5		LED	LED_THT:LED_D5.0mm
BP1		SW_Push	Library_Empreintes_Sae31:SW_TH_Tactile_Omron_B3F-10xx
SW1		SW_Push	Library_Empreintes_Sae31:SW_TH_Tactile_Omron_B3F-10xx
D0-D4	1N4001		Diode_THT:D_DO-41_SOD81_P10.16mm_Horizontal
C1-C3	100n	Polyester	Library_Empreintes_Sae31:C_Disc_D4.7mm_W2.5mm_P5.00mm
R1-R5	2.7K	Couche de carbone	Library_Empreintes_Sae31:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
R0, R6	10K	Couche de carbone	Library_Empreintes_Sae31:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
R7	56K	Couche de carbone	Library_Empreintes_Sae31:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
R8	270	Couche de carbone	Library_Empreintes_Sae31:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
U2		Reciever MX-RM-5V	Library_Empreintes_Sae31:PinHeader_1x04_P2.54mm_Vertical
U1		PIC16F18877	Package_DIP:DIP-40_W15.24mm_LongPads
J0-J2		Screw_Terminal_01x02	Library_Empreintes_Sae31:TerminalBlock_bornier-2_P5.08mm
J1.1		Screw_Terminal_01x05	TerminalBlock_4Ucon_1x05_P3.50mm_Horizontal
J0.1, J0.2		Screw_Terminal_01x05	TerminalBlock_4Ucon_1x05_P3.50mm_Horizontal
J3		Conn_01x06	Library_Empreintes_Sae31:PinHeader_1x06_P2.54mm_Vertical
J4, J5		Screw_Terminal_01x05	TerminalBlock_4Ucon_1x05_P3.50mm_Horizontal
J6		Conn_01x06	Library_Empreintes_Sae31:PinHeader_1x06_P2.54mm_Vertical
Q0-Q4		Q_NPN_BCE	Library_Empreintes_Sae31:transistor
TP0	GND		TestPoint:TestPoint_Loop_D2.50mm_Drill1.0mm

TP1	**PWM1**		**TestPoint:TestPoint_Loop_D2.50mm_Drill1.0mm**
TP2	**PWM**		**TestPoint:TestPoint_Loop_D2.50mm_Drill1.0mm**

Tension d'alimentation

Référence CAO	**Référence**	**Composant**	**Tension de service**		**Tension en sortie**	
U1	**Microcontrôleur**	**PIC16F8877**	**5V**			
U3	**Matrice de transistor**	**ULN2003**	**Max :50V Saturation :1.7V**		**GND**	
U2	**Récepteur**	**Receiver MX-RM-5V**	**5V**		**Low**	**High**
					0V	**1V**
J5	**Capteur *5**	**DS1 8820**	**5V**		**Low**	**High**
					+ou-0V	**+ou-5V**
Connecter aux sorties de U3	**Electro-aimant*5**	**JF-0530B**	**12V**			
Connecter aux bornier J1	**Servomoteur**	**SG90R**	**Min**	**Max**		
			4.8V	**6V**		

Organigrammes des programmes (Programmation du microcontrôleur)

Nous avons programmé notre microcontrôleur de façon qu'il puisse :

- contrôler les différents composants de notre stand de tir (servomoteurs, capteurs, ...)
- reconnaitre et différencier les signaux reçus par le récepteur
- associer à chaque bouton de la télécommande une action
- mettre en place une machine à état
- mettre en place les différents modes de jeu du biathlon (tir debout /assis)

De plus, nous avons divisé notre code en plusieurs fichier pour améliorer sa lisibilité et séparer les fonctions liées au contrôle des **composants**, les fonctions de la **télécommande** et les fonctions de la **machine à état**.

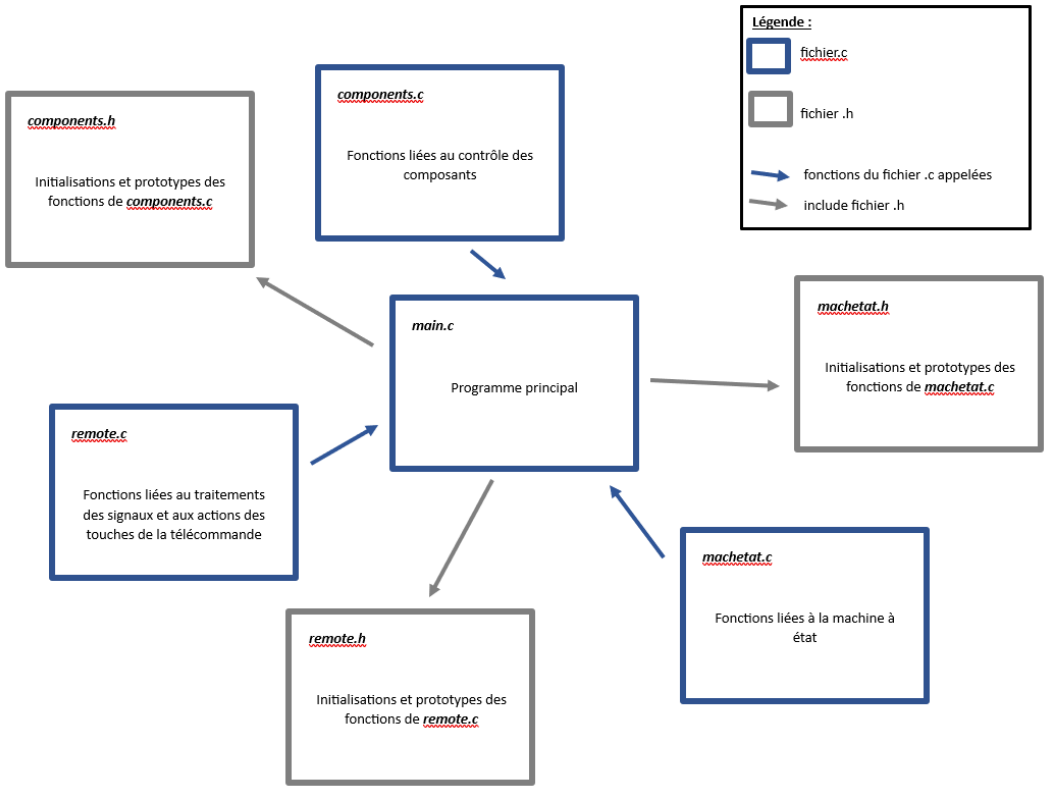


Figure 10 : Schéma montrant l'organisation de notre programme

Ainsi, comme on peut le voir dans le schéma ci- dessus, dans le fichier *remote.c*, on retrouve les fonctions liées au traitement des signaux reçus par le récepteur, ainsi que la fonction qui permet d'associer à chaque bouton de la télécommande une action. Dans le fichier *components.c*, on retrouve les fonctions liées au contrôle des composants du stand de tir. Et dans le fichier *machetat.c*, on retrouve les fonctions liées à la machine à état du stand de tir.

De plus, chaque initialisation de variables ainsi que les prototypes des fonctions sont présent dans les fichiers.h de *remote.c, components.c* et *machetat.c.*

Et dans notre programme principal, nous avons juste appelé la fonction qui concerne l'utilisation de la télécommande et celle qui concerne la machine à état.

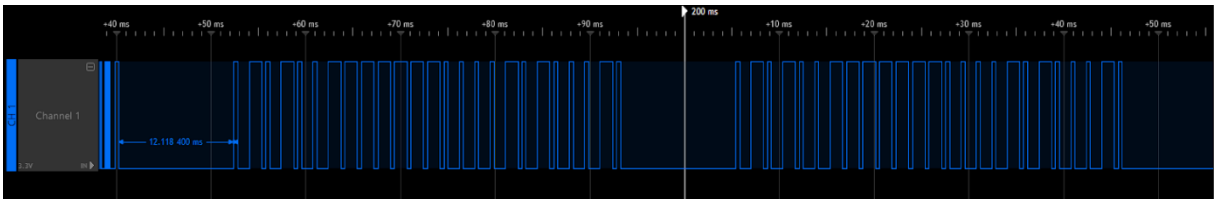
Par la suite, nous allons vous expliquer en détail les fonctions que nous avons créé pour faire fonctionner notre stand de tir :

Programmation télécommande : (fichier *remote.c*)

En septembre 2022, lorsque nous avions reçu le stand de tir, il possédait une télécommande et une petite carte électronique sur laquelle il y avait un récepteur. Cependant nous ne savions pas ce que le récepteur recevait de la télécommande et ce qu'il envoyait au microcontrôleur.

Nous avons donc décidé de séparer le récepteur de sa carte électronique pour pouvoir traiter les données directement avec le microcontrôleur.

Pour le traitement des signaux, nous avons donc visualisé les signaux de chaque bouton et on a écrit un programme (*remote.c*) permettant différencier les bits « 1 » et « 0 » (selon une convention établie en annexe 3) :



Exemple de signal émis par la télécommande

Pour commencer, nous avons utilisés le Timer 2 et le Timer 4 du PIC16F18877 pour le traitement des signaux reçus par le récepteur.

Nous avons fait les configurations des timers avec MCC qui nous a générer des fonctions que nous pouvons utiliser. Nous avons utilisé pour chacun des deux timers la fonction TMRX_SetInterruptHandler(fonction appelée), qui permet d'appeler les fonctions permettant de traiter les signaux envoyés par le récepteur. On a appelé ces fonctions dans la fonction *remote_initialize()* pour éviter de charger le *main.c* .

D'abord, nous avons utilisé le Timer 4 pour détecter les périodes qui marquent le début des signaux qui nous intéressent. Durant cette période, le signal est à l'état bas pendant 12 ms. Nous avons configuré le Timer 4 en Roll over pulse de façon qu'il démarre et se réinitialise aux fronts descendants au bout d'une période de 10 ms (supérieure au temps entre chaque signal qui est de 12 ms).

Ainsi si l'on a une durée à l'état bas supérieure à la période du timer, alors l'interruption du timer appellera la fonction qui réinitialise les variables associées au comptage et à la sauvegarde des bits reçu par le récepteur. Soit la fonction " start_data ()" (voir l'organigramme) :

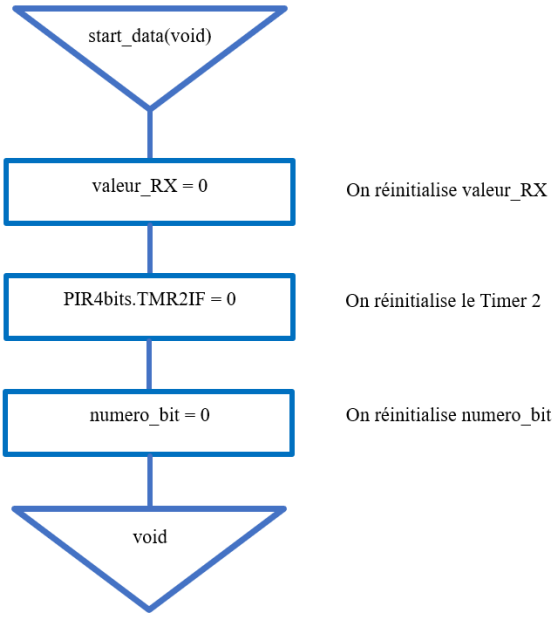


Figure 11 :Organigramme de la fonction *start_data()*

Ensuite, nous avons utilisé le Timer 2. Nous l'avons configuré en Monostable et de façon qu'il démarre et se réinitialise aux fronts montants et on a mis la période de débordement à 800 µs. Ainsi chaque débordement l'interruption du Timer 2 appellera la fonction servant à sauvegarder les bits un par un, à chaque débordement. Soit la fonction "lecture_bit ".

Cette fonction associée au Timer 2 nous permet de stocker les données renvoyées par le Timer 2 en les stockant dans la variable valeur_RX (voir l'organigramme ci-dessous) :

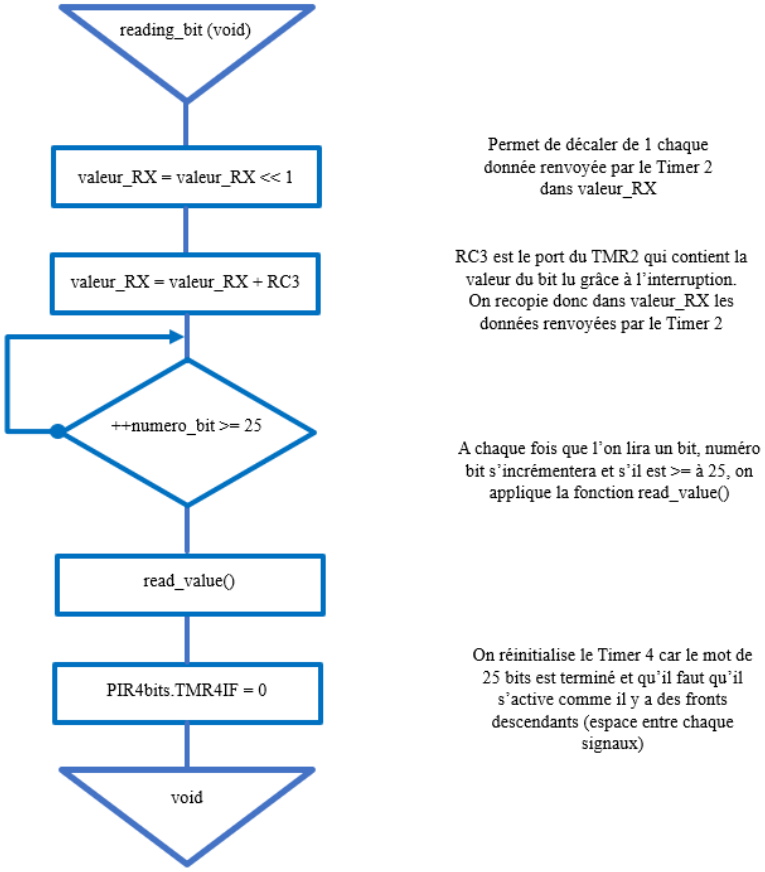


Figure 12: Organigramme de la fonction *reading_bit ()*

Après on a créé read_value () qui est une fonction qui permet de vérifier que les 26 premiers bits de la variable valeur_RX correspondent à ceux de la télécommande :

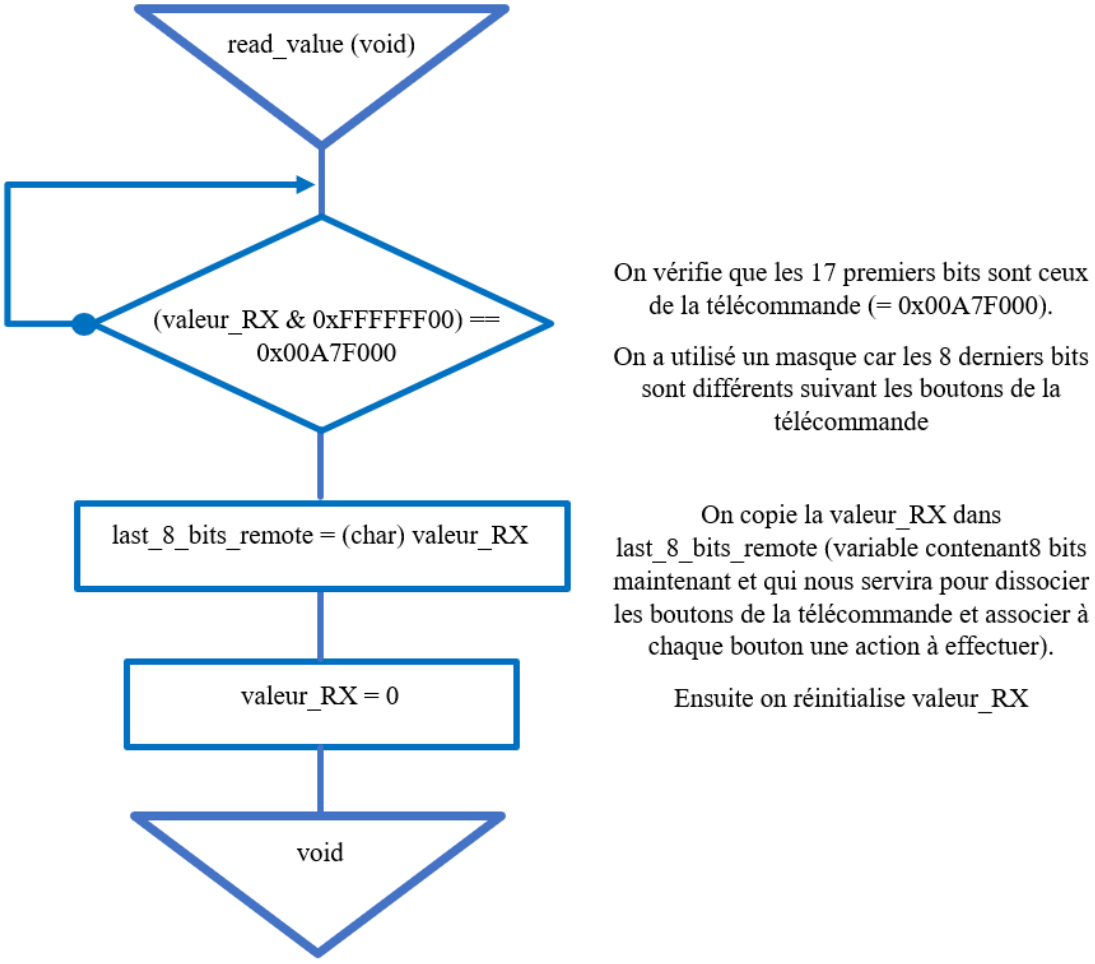


Figure 13 : Organigramme de la fonction *read_value()*

Ainsi nous avons des fonctions permettant d'enregistrer les bits des signaux reçus par notre récepteur et permettant de les comparer afin de déterminer s'ils correspondent à des signaux émis par notre télécommande. Et grâce à la fonction *read_value()*, on a une fonction qui permet de récupérer les 8 derniers bits du signal de 25 bits enregistré et les copies dans la variable *last_8_bits_remote.*

On va donc se servir de cette variable pour attribuer une action à chaque bouton. Au préalable, nous avions relevé les valeurs des 8 derniers bits de chaque bouton (voir annexe 3), et le switch case dans notre fonction *remote_ctrl_buttons* compare la valeur de *last_8_bits_remote* et attribut une série d'action en fonction de cette valeur.

Voici un organigramme pour comprendre le fonctionnement complet de la fonction *remote ctrl buttons :*

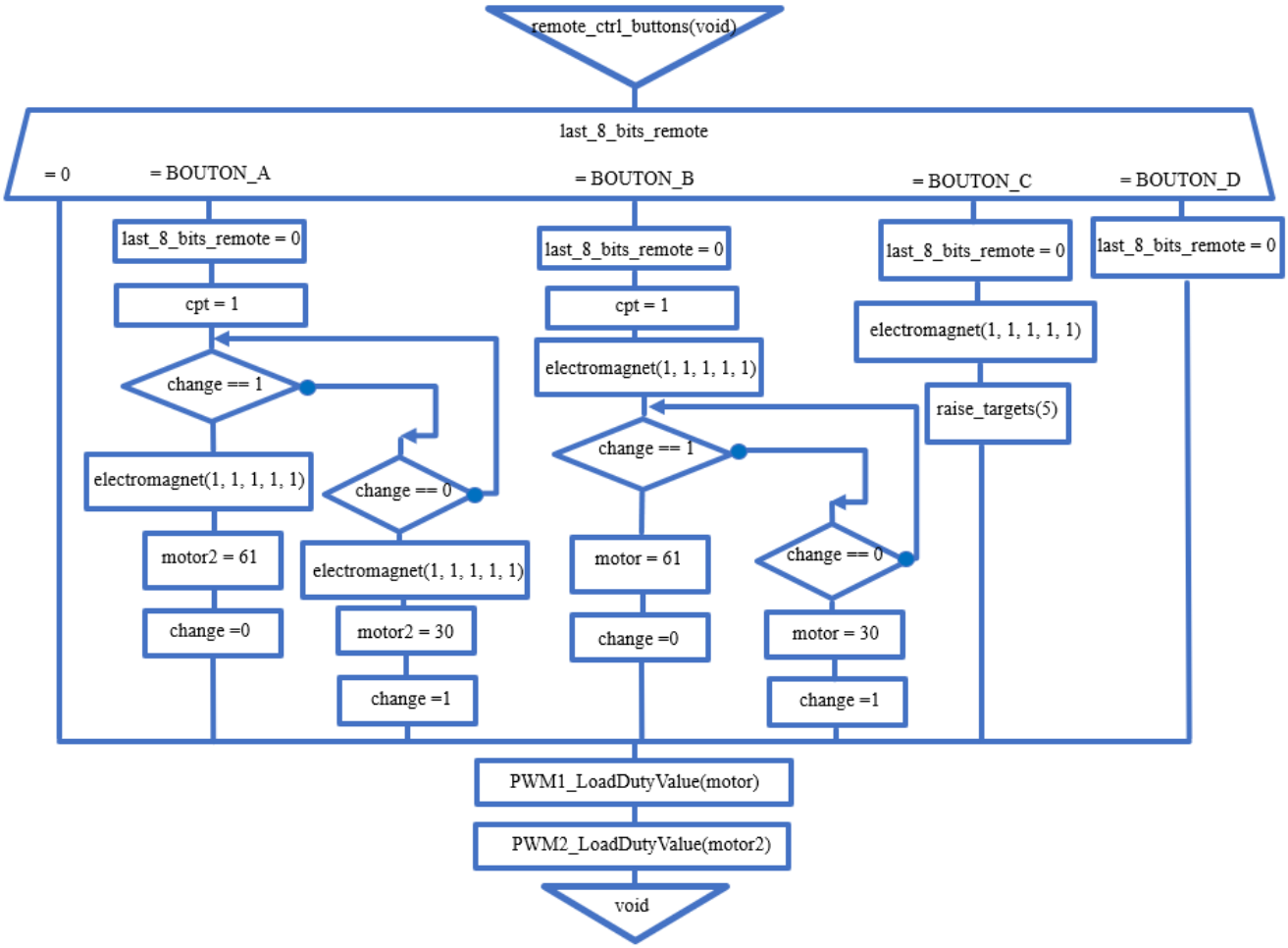


Figure 14: Organigramme de la fonction *remote_ctrl_buttons ()*

Contrôle des servomoteurs :

Dans la fonction *remote_ctrl_buttons* (et dans la machine à état par la suite) on utilise la fonction *PWM1_LoadDutyValue(motor)* voici pourquoi:

Pour contrôler les servomoteurs avec un microcontrôleur, il faut que l'on génère un signal PWM. Le signal PWM à un rapport cyclique, et en fonction de celui-ci, nous pouvons changer la position du servomoteur. Nous sommes partis sur une période de PWM d'environ 20 ms, générée à l'aide du Timer 6. Et nous avons généré à l'aide de MCC un signal PWM sur deux PORTS de notre PIC correspondants aux deux connexions de nos deux servomoteurs.

On obtient donc deux fonctions de MCC : ***PWM1_LoadDutyValue(motor)*** et ***PWM2_LoadDutyValue(motor2)*** auxquelles nous avons ajouté les variables *motor* et *motor2* car ce sont les variables qui nous permettent de changer le rapport cyclique du signal PWM de nos deux servomoteurs.

Lorsque l'on met la valeur 30, cela correspond à un rapport cyclique de 5 % c'est-à-dire un angle de 0° effectué par le servomoteur. Et lorsque l'on met la valeur 61, cela correspond à un rapport cyclique de 10 % c'est-à-dire un angle de 90° effectué par le servomoteur.

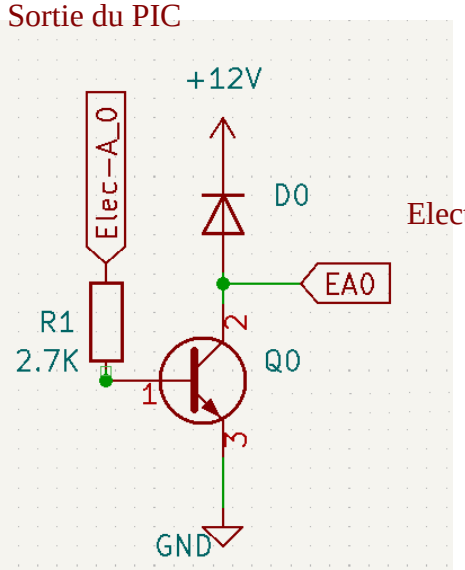
Programmation des composants : (fichier *components.c*)

Dans le fichier *components.c*, nous avons mis toutes les fonctions qui permettent de contrôler nos composants de notre stand de tir : les électro-aimants, les capteurs et le bouton poussoir de la machine à état.

Contrôle des électro-aimants :

Nos électroaimants ont besoin d'être alimenté en 12V et ils ont besoin d'un courant plus important que le reste de notre circuit. Cependant cela pose un problème car un courant trop élevé risque d'endommager le reste du circuit. Nous avons donc réalisé un circuit de commande pour chaque électroaimant car ce circuit peut supporter des intensités plus importantes.

Sortie du PIC



Electro-aimant

Ce circuit est composé d'un transistor MOSFET NPN (Q0), d'une diode placée en diode de roue libre (D0), d'une résistance de 2,7 kΩ et on a intégré l'électroaimant au circuit.

Pour contrôler l'électro-aimants avec le PIC on doit envoyer un signal de commande. Lorsque l'on envoie un « 1 », l'électroaimant se rétracte et lorsque l'on envoie un « 0 », l'électroaimant se déploie.

On a donc fait une fonction *electromagnet()* pour contrôler chaque électroaimant. Cette fonction est construite pour que lorsque l'on l'utilise on peut soit contrôler chaque électroaimant un par un, grâce aux 5 paramètres *elct* correspondant aux 5 électroaimants.

Nous avons aussi rajouté une sécurité qui désactive nos électroaimants 3 secondes après leur activation car il commence chauffer si on les active trop longtemps.

Voici donc un organigramme qui montre la fonction *electromagnet()* plus en détail :

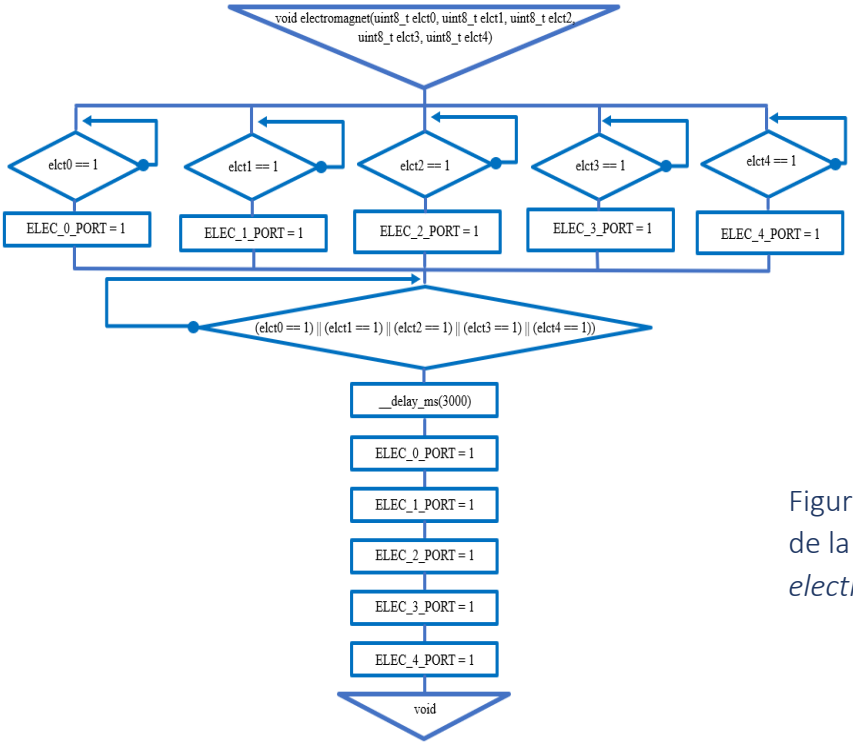


Figure 15: Organigramme de la fonction *electromagnet()*

Contrôle des capteurs :

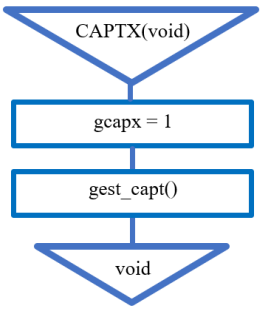
Les capteurs laser de notre stand de tir renvoie un état haut (5V) lorsqu'il capte un laser et un état bas (0 V) lorsqu'il ne capte rien.

Nous avons donc connecté nos capteurs à des ports du PIC qui possèdent une IOC, et grâce à la fonction IOCAF0_SetInterruptHandler(fonction) générée par MCC, à chaque fois qu'il y aura une interruption (passage à l'état haut) sur le PORT d'un des capteurs, la fonction entre parenthèse (fonction) sera appelée. Nous avons donc créé 5 fonctions *CAPTX* correspondant aux 5 capteurs de notre stand de tir.

Cependant dans les fonctions utilisant des interruptions nous ne pouvons pas utiliser la commande *printf()* permettant d'envoyer un message à l'aide de la liaison série que nous avions installée. Nous avons donc créé une autre fonction *gest_capt*. En appelant cette fonction dans nos fonction *CAPTX* indirectement à chaque fois que le capteur détecte un laser il fera la série d'actions que l'on veut pour notre stand de tir (relever les trappes, ...)

Aussi pour éviter de charger le *main.c* on a appelé les fonctions *IOCAF0_SetInterruptHandler(fonction)* dans la fonction *components_initialize()* pour éviter de charger le *main.c*)

Voici deux organigrammes des fonctions *CAPTX* et de *gest_capt* pour illustrer nos propos :



Il y 5 copies de cette fonction car nous avons 5 capteurs

gcapx = 1 car c'est un des paramètres de gest_capt() et s'il est égal à 1 lorsque l'on appelle gest_capt(), dans cette dernière chaque capteur x enclencheront chacun une série d'actions

Figure 16: Organigramme de la fonction *CAPTX()*

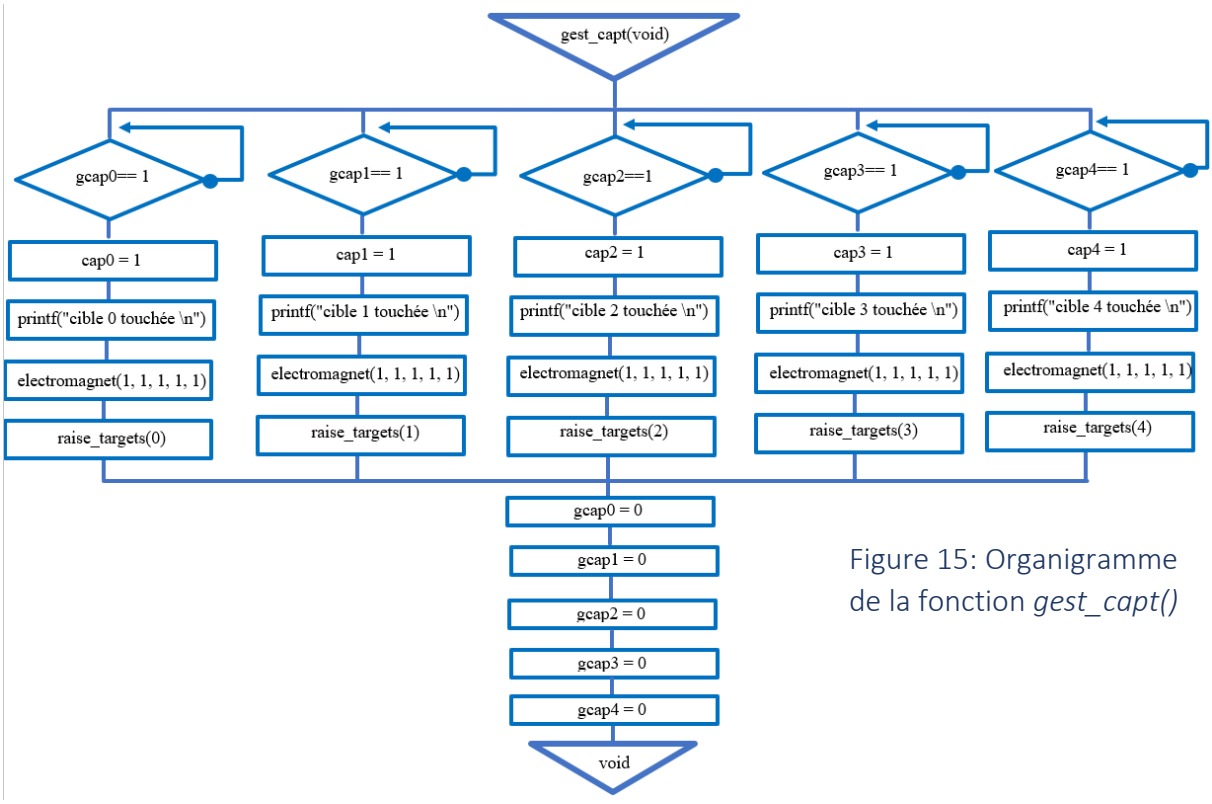


Figure 15: Organigramme de la fonction *gest_capt()*

Remontée des trappes :

Mécaniquement, le stand de tir a un problème car il ne possède pas de système pour remonter les trappes cachant les cibles. Pour contourner ce problème mécanique nous avons donc simulé cette remontée mécanique avec des messages que l'on envoie sur la liaison série que nous avons installé. On a donc créé la fonction *raise_targets* qui est appelée à chaque fois qu'une des cibles est touchée et permet de renvoyer un message à l'aide de la commande *printf ()*. Voici un organigramme pour montrer le fonctionnement de *raise_targets* :

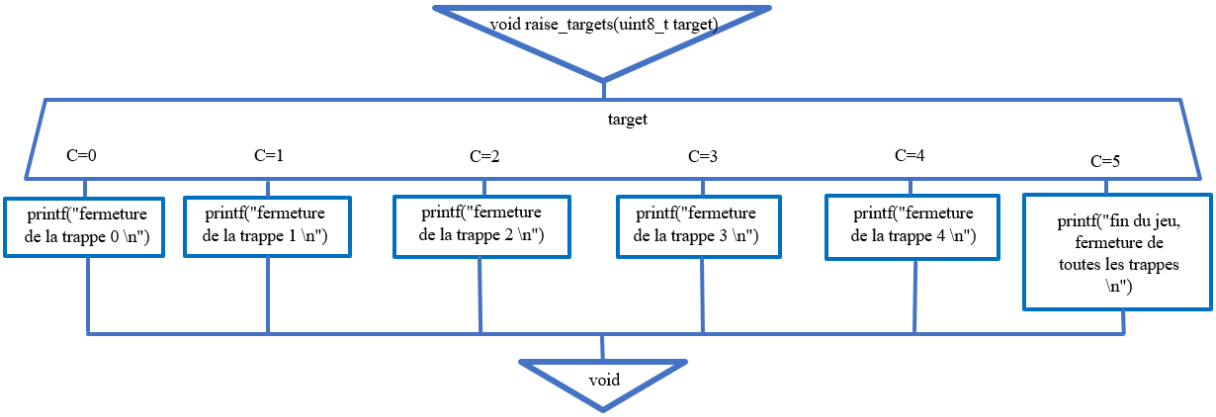


Figure 16: Organigramme de la fonction *raise_targets()*

Programmation de la machine à état : (fichier *machetat.c*)

Les clients de notre projet (professeurs) voulaient que nous rajoutions une machine à état pour que l'on puisse faire fonctionner le stand de tir même s'il l'on perdaient la télécommande. Cette machine à état est contrôlable à l'aide d'un bouton poussoir.

Bouton poussoir (BP) :

A chaque appui sur le bouton poussoir, la machine à état change d'état. Pour réaliser ce fonctionnement nous avons connecté le bouton poussoir sur un des ports du PIC qui possèdent une IOC, et grâce à la fonction IOCCF5_SetInterruptHandler(fonction) générée par MCC, à chaque fois qu'il y aura une interruption (passage à l'état bas = appui sur BP) sur le PORT du bouton poussoir, la fonction entre parenthèse (fonction) sera appelée. Cette fonction c'est la fonction BP. Elle contient des lignes de codes permettant de faire un compteur (cpt). On utilisera la variable cpt dans notre fonction liée à la machine à état.

Voici un organigramme pour illustrer le fonctionnement de *BP* () :

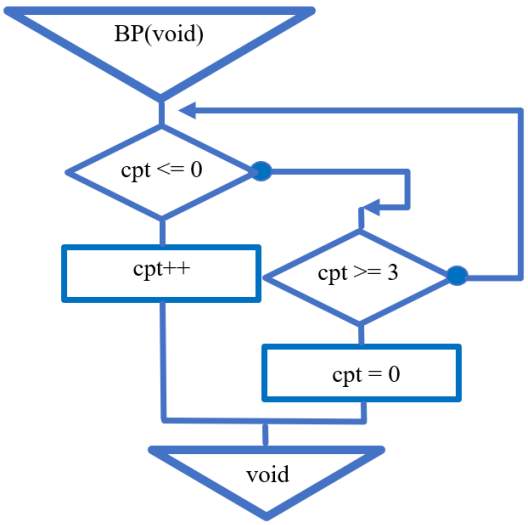


Figure 15: Organigramme de la fonction *BP()*

Machine à état : (fichier *machetat.c*)

La machine à état fonctionne dans le même principe que le système utilisé lorsque l'on utilise la télécommande. C'est-à-dire que l'on a un état de base lorsque les cibles sont fermées, mode tir debout et un mode tir assis ave des fentes plus petites et une fin du jeu. Et dans le système du jeu, on prend aussi en compte les capteurs qui indiquent lorsqu'une cible est touchée. (voir extrait du code ci-dessous :)

```
if (cpt == 0) {

    debut1 = 0;
    debut2 = 0;
    if (change == 1) {
        LED_SetHigh();
        change = 0;
    } else if (change == 0) {

        LED_SetLow();
        change = 1;
    }

}

if (cpt == 1) {
    LED_SetHigh();
    motor2 = 30;
    if (debut1 == 0) {
        electromagnet(1, 1, 1, 1, 1);
        motor2 = 60; //descente
        motor = 30; //mode debout

        debut1 = 1;
        debut2 = 0;

    } else {
        motor2 = 30;
    }


}
if (cpt == 2) {
    LED_SetLow();
    motor2 = 30;
    if (debut2 == 0) {
        electromagnet(1, 1, 1, 1, 1);
        motor2 = 60;
        motor = 60; //mode assis

        debut1 = 0;
        debut2 = 1;
    }
}
```

```
    if (cpt == 3) {
        LED_SetHigh();
        electromagnet(1, 1, 1, 1, 1);
        raise_targets(0);
        raise_targets(1);
        raise_targets(2);
        raise_targets(3);
        raise_targets(4);
        raise_targets(5);
        //fermeture des trapes

        debut1 = 0;
        debut2 = 0;
        cpt = 0;
        motor = 30;
        motor2 = 30;
    }
    PWM1_LoadDutyValue(motor);
    PWM2_LoadDutyValue(motor2);
}
```

Pour lancer la machine à état, on appelle *machetat.c* dans le main.c. Et si le jeu est lancé et que les capteurs sont tous touchés, on incrément le compteur ce qui fait changer d'état la machine à état.

```
        /* lancement de la machine à état  */
        if (cpt != memo) {
            cpt = memo;
            machetat();

        }

        if ((cpt == 1) || (cpt == 2)) {
            if ((cap0 == 1) && (cap1 == 1)&&(cap2 == 1) && (cap3 == 1) &&(cap4 == 1)) {
                cpt++;
            }
        }
    }
}
```

Etapes de l’avancement du projet

Pour ce projet, nous avons tout d’abord, réalisé un premier câblage sur une platine, et écrit un programme permettant au microcontrôleur d’une part de traiter les signaux envoyés par le récepteur, afin de pouvoir identifier sur quel bouton on appuie. Et cela, pour à terme pouvoir associer l’appui sur certain bouton à une action qui lui sera attribuée. Comme par exemple, la descente des panneaux du stand ou bien le passage du mode « tir assis » à « tir debout ».

Et d’autre part de générer des signaux PWM afin de commander deux servomoteurs.

Ensuite, nous nous sommes penchés sur la mise en œuvre des électroaimants et la gestion des capteurs. Et avec des tests nous avons procédé à quelques modifications et remplacé la matrice de transistors par un réseau de transistors car ce composant était défectueux.

Et pour finir, les clients (professeurs), nous ont fait part d’une requête supplémentaire qui concernaient l’ajout d’une machine à état actionnée par un bouton et représentée par une led.

Conclusion

Au terme de notre projet, nous avons une carte électronique capable d'identifier et de traiter les signaux reçus d'une télécommande, auxquels on a pu associer différentes actions.

De plus, elle permet de piloter deux servomoteurs, l'un permet de faire pivoter une barre faisant tomber simultanément tous les panneaux du stand et l'autre permet de passer de la configuration « Tir debout » à « Tir assis ».

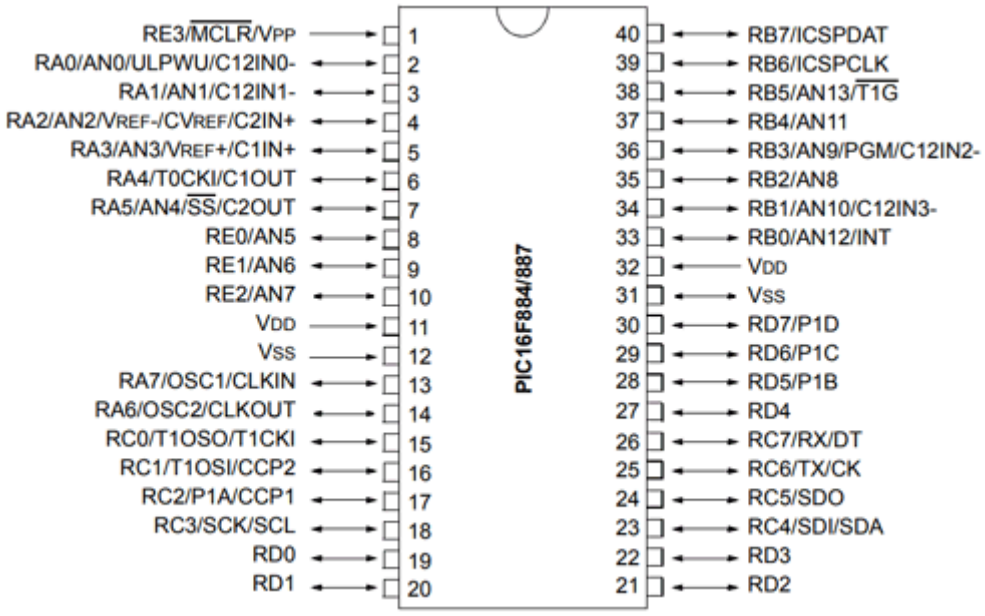
Nous avons aussi mis en place une machine à état qui fonctionne avec un bouton-poussoir et une LED.

Cette carte permet également d'actionner des électro-aimants bloquant les panneaux précédemment cités, quand ils sont en position relevée ou abaissée. Sachant qu'ils ne sont actionnés qu'au moment précédant la descente des panneaux ou lorsqu'un capteur laser est touché.

Cependant, nous rencontrons des problèmes dû soit à des erreurs dans le programme, soit dû à des courts-circuits, qui font que les électro-aimants et le servomoteur servant à faire descendre les panneaux de la cible, s'activent même quand l'instruction n'a pas été envoyer à la carte.

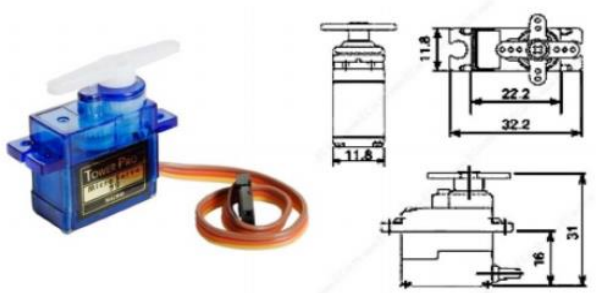
Annexes

Annexe1 : *extrait de la datasheet du PIC 16F18877*



Annexe2 : *extrait de la datasheet du servomoteur SG90*

SG90 9 g Micro Servo



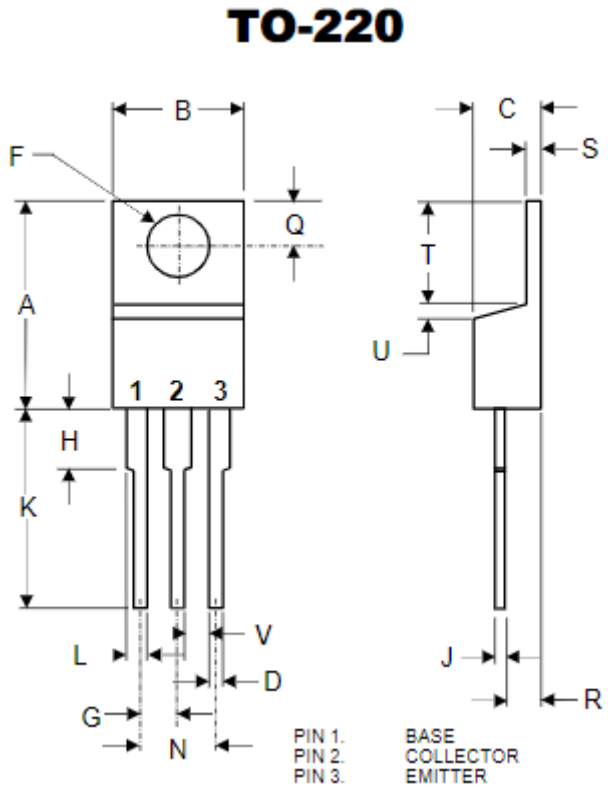
Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but *smaller*. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.

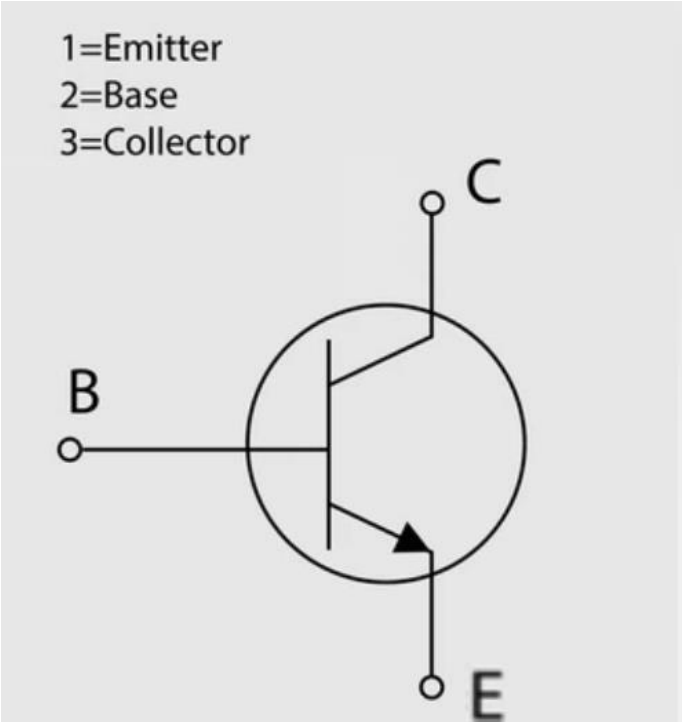
Specifications

- Weight: 9 g
- Dimension: 22.2 x 11.8 x 31 mm approx.
- Stall torque: 1.8 kgf·cm
- Operating speed: 0.1 s/60 degree
- Operating voltage: 4.8 V (~5V)
- Dead band width: 10 µs
- Temperature range: 0 ºC – 55 ºC

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is all the way to the left. ms pulse) is all the way to the right, ""-90" (~1ms pulse) is all the way to the left.

Annexe3 : *extrait de la datasheet du transistor Mofset NPN BDX33*

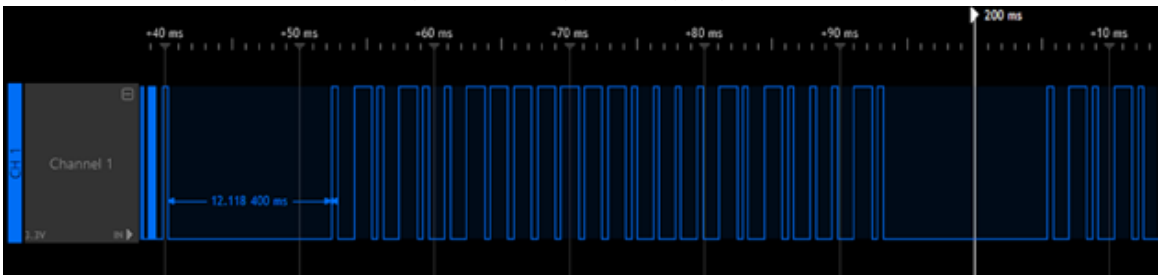




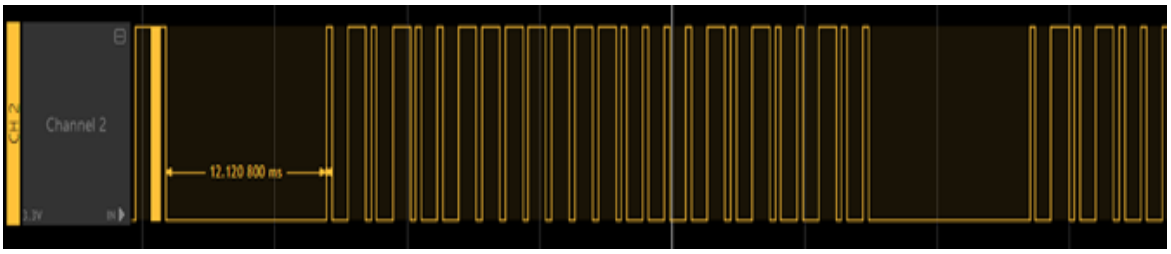
Signaux télécommande : Annexe4

Ci- dessous sont présents les signaux émis par la télécommande lors de l'appui sur chacun des boutons. Comme on peut le voir sur les captures d'écran ci-dessous chaque signal est constitué d'un motif de 53 ms. Chaque motif est composé de 12 ms pendant lesquelles le signal est à l'état bas, puis suit les 25 bits.
On a défini qu'un bit est à 1 quand l'état haut dure plus de 800 microsecondes et il est à 0 quand l'état haut ne dure que 0.4millisecondes.
Les 17 premiers bits de chaque signal sont identiques, seuls les 8 derniers permettent de les différencier.

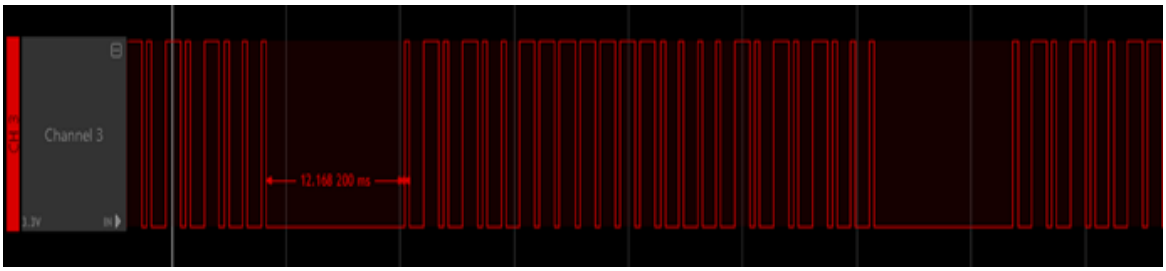
Bouton A : 8 bits finaux (10100010) b → (A2) h



Bouton B :8 bits finaux (10100100) b → (A4) h



Bouton C :8 bits finaux (10101000) b → (A8) h



Bouton D :8 bits finaux (10110000) b → (B0) h