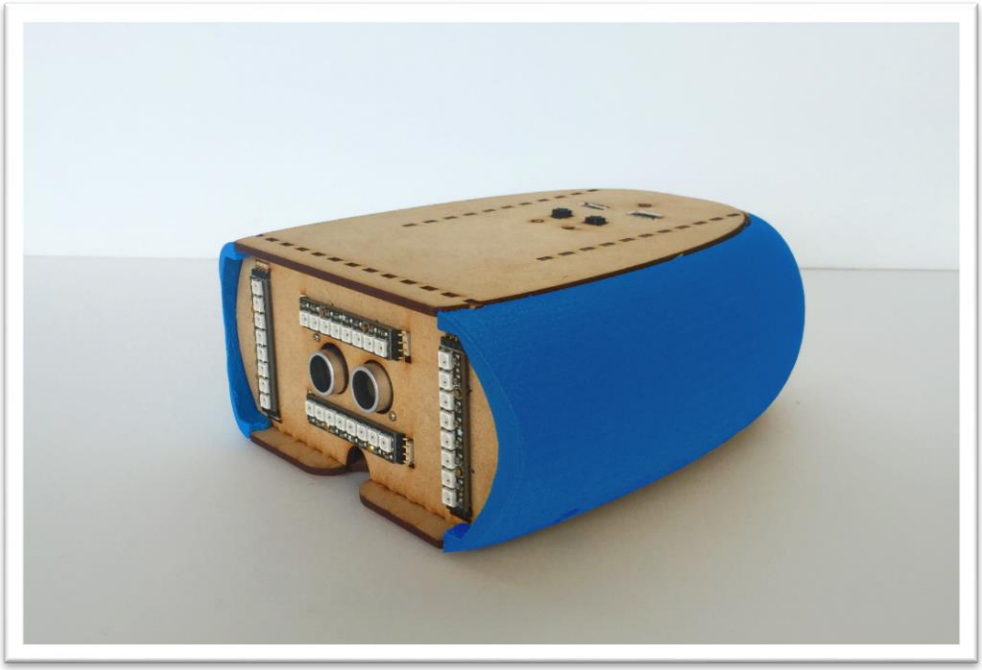
CLAVERIE Martin

CHARNET Victor

Polytech Lille – 4ème année

Informatique, Microélectronique, Automatique



Rapport de projet : « Zone de sécurité pour vélo »

2015 / 2016

Responsables : M. BOE & M. VANTROYS

Sommaire

1. Introduction

Dans le cadre de notre quatrième année à l'école d'ingénieur Polytech Lille en formation IMA (Informatique, Microélectronique, Automatique) nous avons dû réaliser un projet technologique.

Le but premier de ce projet est d'augmenter la sécurité des cyclistes grâce à un système autonome qui devra être adaptable sur tout type de vélos.

La sécurité des personnes à vélo dépend pour beaucoup des autres utilisateurs et notamment des conducteurs d'engins mécanisés.

Ce projet propose d'aider les autres conducteurs à respecter une zone de sécurité lorsqu'ils doublent ou suivent un vélo. Pour cela, après détection d'un véhicule qui suit ou double, des modules laser doivent délimiter au sol la zone d'environ 1m autour du cycliste.

De plus, un système de clignotants est disponible si le système est appairé à un smartphone.

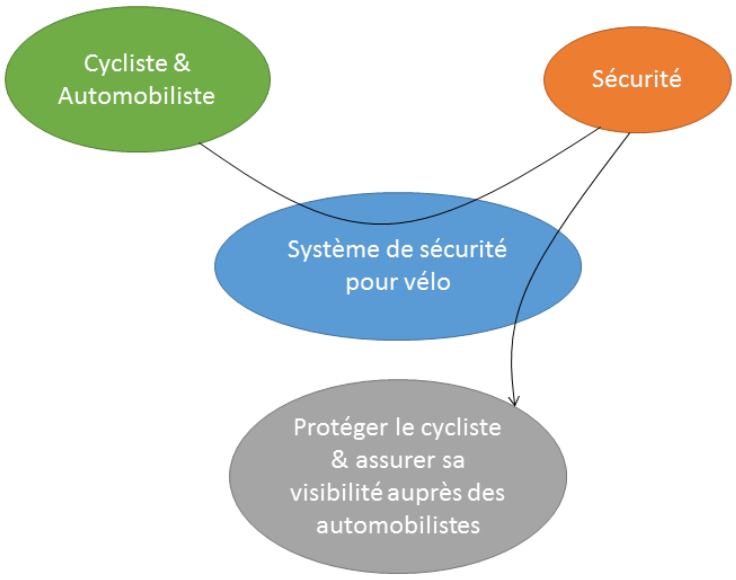
Les points essentiels sont :

- Avertir d'un changement de direction
- Informer le cycliste de la présence d'un automobiliste
- Informer l'automobiliste des distances de sécurité

Pour bien définir les contraintes à respecter pour le projet, nous avons réalisé différentes analyses du projet.

2. Analyses du projet

a. Analyse bête à cornes



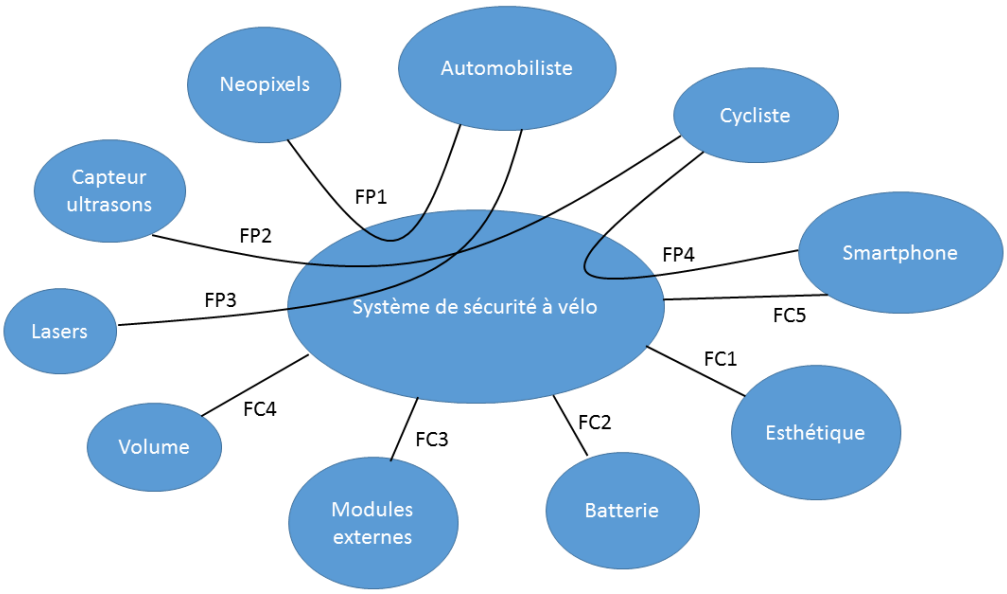
b. Diagramme pieuvre

Fonctions principales :

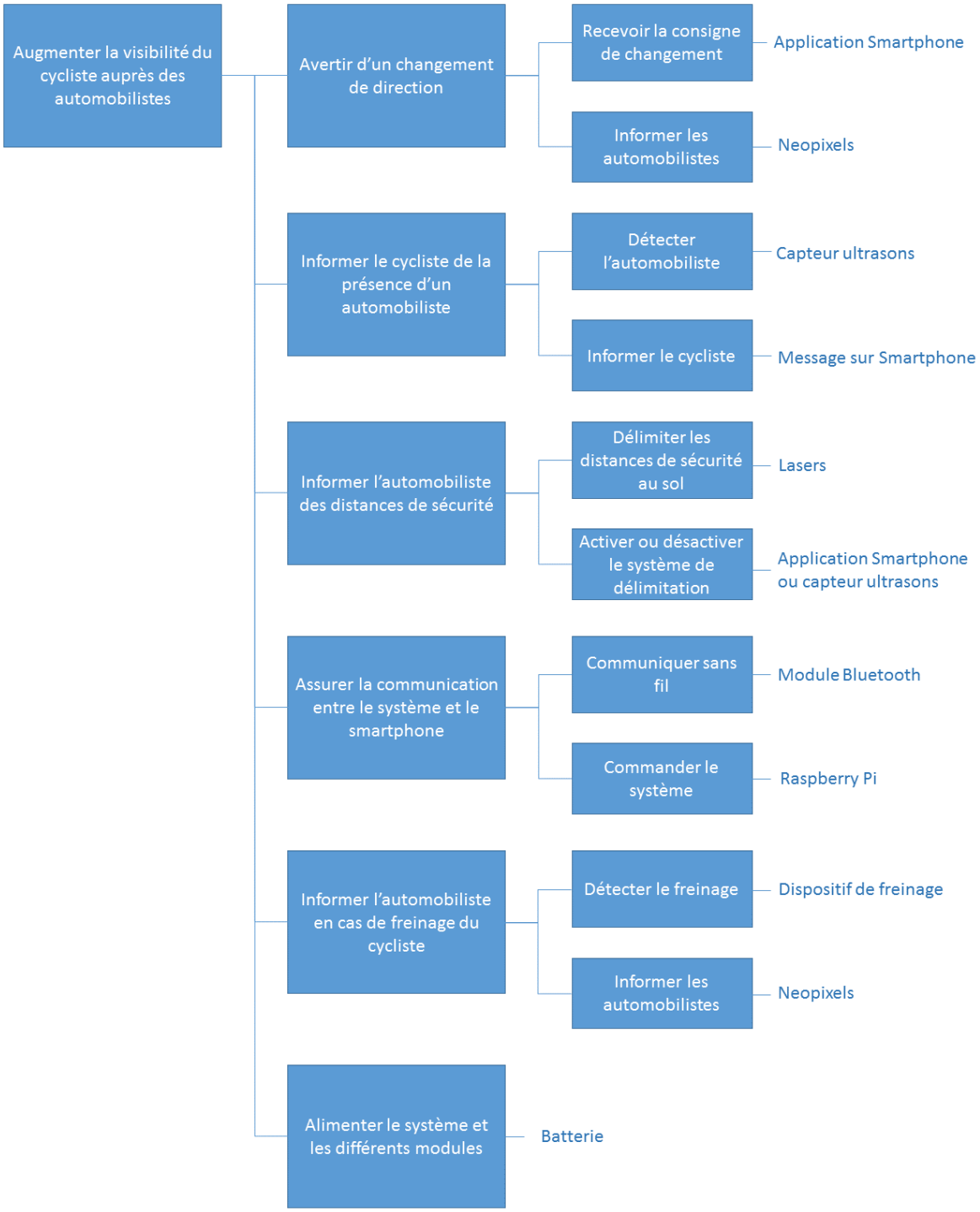
- **FP1** : Avertir les automobilistes d'un changement de direction du vélo
- **FP2** : Informer le cycliste de la présence d'un automobiliste derrière lui
- **FP3** : Informer l'automobiliste des distances de sécurité
- **FP4** : Assurer une communication entre le smartphone et le système

Fonctions contraintes :

- **FC1** : Esthétique
- **FC2** : Bonne gestion énergétique
- **FC3** : Installation de modules supplémentaires simple
- **FC4** : Volume réduit
- **FC5** : Interface smartphone simple et intuitive



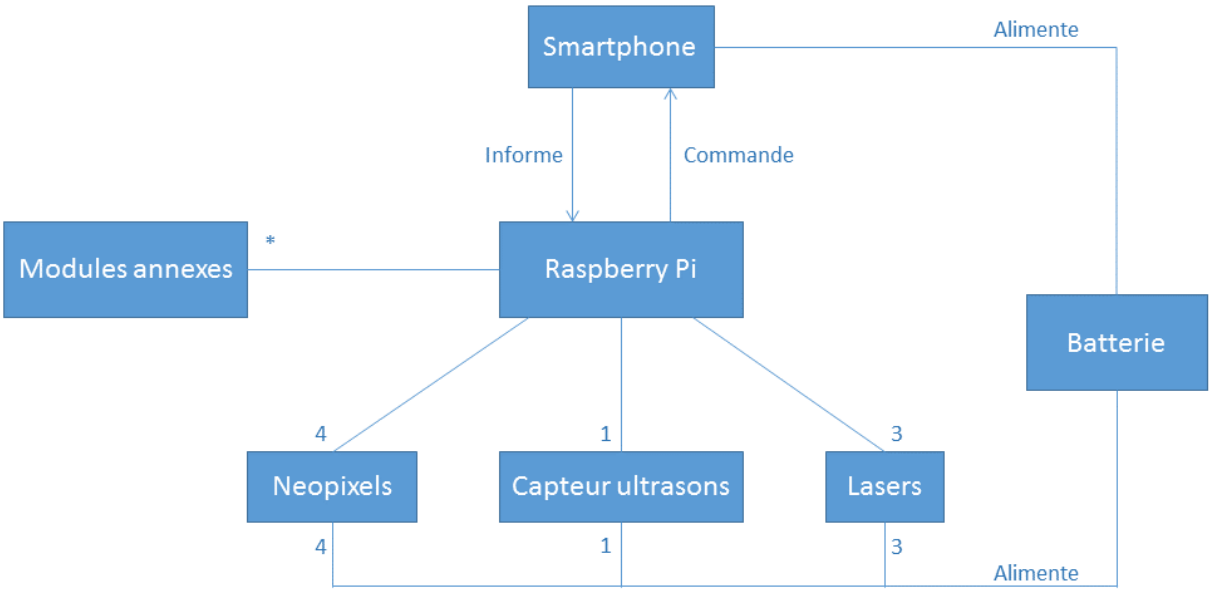
c. Analyse fonctionnelle



Il est à noter que la fonctionnalité « avertir l'automobiliste en cas de freinage du cycliste » a été retenue comme optionnelle et sera disponible avec l'ajout de modules. De ce fait, elle n'est pas présente sur la version finale présentée.

d. Représentation UML

L'intérêt de la représentation UML est qu'elle permet d'établir un modèle en définissant sous forme de diagramme les différentes classes, objets et relations au sein du système.



3. Solutions techniques envisagées

Dès le début du projet, nous avons décidé de partir sur un système sous forme de boitier, se plaçant sur le porte bagage des vélos. La zone de sécurité à respecter autour du vélo sera représentée par des modules lasers lignes.

A l'origine du projet, nous avons choisi la Raspberry Pi car nous comptions utiliser un module de caméra en plus du système d'ultrasons pour la détection des véhicules à l'arrière ; notamment en utilisant un système de reconnaissance de plaques d'immatriculation. Mais très vite nous avons réalisé que de nombreuses contraintes se seraient présentées à nous, notamment pour la circulation de nuit. Si le projet était à recommencer, nous utiliserions surement un Arduino, qui est plus adapté dans cette application.

La solution retenue pour détecter les véhicules à l'arrière est le module ultrason.

4. Présentation des modules

a. Stick NeoPixel

Le sticks NeoPixel sont utilisés au nombre de 4 dans notre projet. Il s'agit de sticks de 1*8 LEDs RGB réglables par microcontrôleur (Raspberry Pi dans notre cas).

La particularité de ces composants sont qu'ils peuvent se brancher en série. Ainsi il ne suffit que d'un seul microcontrôleur pour piloter l'ensemble.

La tension nominale des NeoPixel est de 5V, et leur alimentation est garantie par le circuit électronique associé.

Dans notre application, les Neopixel ont deux applications. Ils servent tout d'abord à informer l'automobiliste qu'il se situe trop près du cycliste en clignotant d'un rouge vif. Si le système est connecté à un Smartphone, les NeoPixel servent également de clignotants sur les côtés.

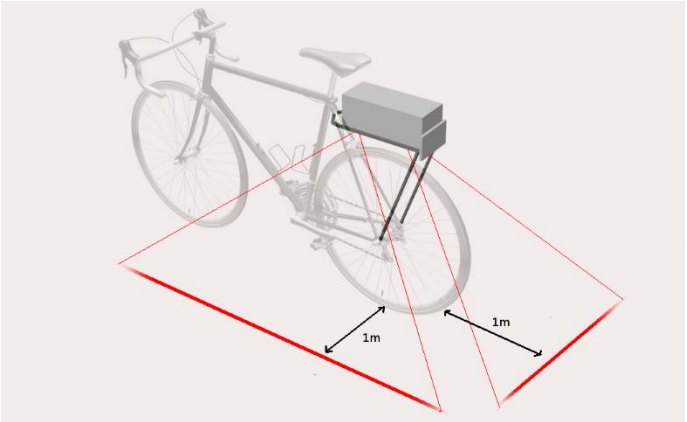
b. Capteur ultrason SR04

Le capteur ultrason utilisé dans notre projet permet de détecter la présence d'un véhicule à l'arrière du vélo.

En fonction de la distance à laquelle celui-ci se situe du vélo, le système réagit différemment. Tout d'abord, le boitier va dessiner une zone de sécurité que le véhicule devra respecter avec le vélo. Si jamais cette zone est franchie, les NeoPixel émettront un clignotement lumineux rouge pour informer d'une trop grande proximité avec le vélo.

c. Module laser HLM1230

Les 3 lasers présents sur les côtés et à l'arrière du système permettent de délimiter la zone de sécurité à respecter avec le vélo lorsqu'un véhicule est trop proche. Ils ont la particularité d'émettre « une ligne » au lieu d'un point lumineux comme la plupart des lasers.



5. Programmation de la Raspberry Pi

a. Présentation générale

Le programme principal embarqué sur le système exécuté par la Raspberry Pi est programmé en C. Il gère la communication Bluetooth avec le téléphone, le capteur ultrason et les NeoPixel.

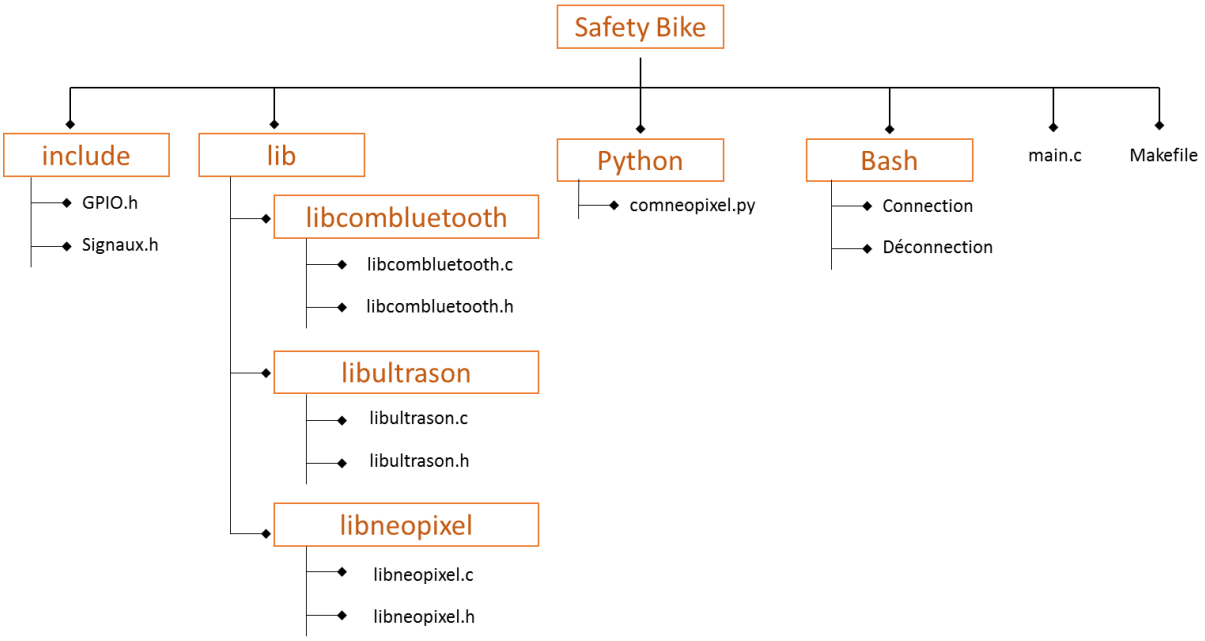


Fig. 1 : Arborescence du dossier *SafetyBike*

Le dossier Safety Bike contient :

- Le programme principal : main.c
- Le Makefile
- Le dossier *include*, qui contient les headers dans lesquels sont définis les broches GPIO et les constantes utilisées par les différentes librairies.
- Le dossier *lib*, qui contient les librairies utilisées : libcombluetooth, libultrason et libnéopixel. [cf 5.b]

Les NeoPixel sont normalement développés pour fonctionner sur des microcontrôleurs temps réel tel que Arduino, AVR, PIC, etc... En effet leur fonctionnement réside dans un protocole très sensible au timing de l'ordre de la milliseconde. Néanmoins une librairie développée par la communauté existe et permet de les contrôler sur Raspberry Pi. Cependant elle est développée en Python (et non en C) et utilise une broche PWM.

- Ce script se situe dans le dossier *Python*.
- Le dossier *Bash*, qui contient les commandes permettant d'activer (et de désactiver) le Bluetooth et de la gérer comme une communication série.

b. Les librairies

b.1. Libcombluetooth

Cette librairie permet la gestion de la communication Bluetooth avec un smartphone Android. Elle lance un script Bash pour lancer le daemon Bluetooth [le code est 1234] après appuie sur le bouton « connectivité », et attend l'appairage.

Une fois l'appairage fait, le programme lit en boucle le fichier /dev/rfcomm0. Le smartphone envoie un octet à chaque fois que l'utilisateur appuie sur un clignotant qui sera traité par le programme.

S'il reçoit :

- 0x61 ['a' en caractère] : il allume/éteint le clignotant droit.
- 0x62 ['b' en carcatère] : il allume/éteint le clignotant gauche.

Ce fonctionnement normal peut être interrompu par des signaux :

- SIGINT : arrête le programme (cf 1.4)
- SIGUSR1 : *[émit par libultrason]* s'il y a une procédure de dépassement, envoie un octet au smartphone 0x57 ['W'] pour signaler le début et 0x58 ['X'] pour signaler la fin.

b.2. Libultrason

Cette librairie permet la gestion du capteur ultrason.

Pour effectuer une mesure on envoie un pulse de 10us sur la broche TRIG et on attend une interruption sur la broche ECHO.

Il calcul le temps mis entre le pulse TRIG et l'ECHO et connaissant la vitesse du son dans l'air on en déduit à quelle distance se situe l'obstacle.

On a 2 seuils de mesure qui déclenche différents systèmes :

- DISTANCE_DETECTION : est la distance à laquelle les lasers s'allument.
- DISTANCE_WARNING : est la distance à laquelle les avertisseurs lumineux se mettent en marche. Le programme envoie un signal SIGUSR1 à la libcombluetooth et SIGUSR2 à libneopixel.

Pour éviter que des valeurs erronées viennent déclencher notre système de façon accidentelle, on lisse la valeur sur une dizaine de mesures.

b.3. Libneopixel

Cette librairie gère la communication avec le script python.

Elle lance le script comneopixel.py au démarrage et dialogue avec lui en fonction des différents signaux qu'elle reçoit :

- SIGINT : arrête le programme et le script (cf 1.4)
- SIGURS1 : *[émit par libcombluetooth]* lit le message reçu par la communication IPC, et envoie la commande au script.
- SIGUSR2 : *[émit par libultrason]* envoie une commande WARNING au script.

c. Démarrage

Au démarrage de la Raspberry Pi, le programme se lance automatiquement grâce à la modification du fichier /etc/rc.local

.

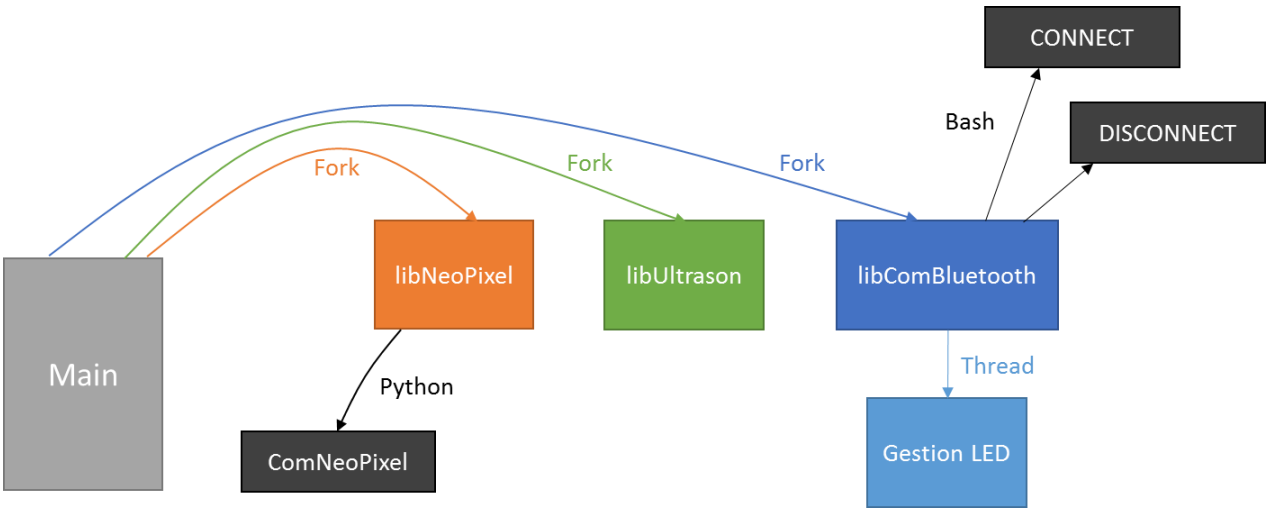


Fig. 2 : Schéma général de la partie programmation sur Raspberry PI

Le programme principal va créer plusieurs processus : un pour la gestion des NeoPixel, un pour le capteur ultrasons et un dernier pour la communication Bluetooth. Cela permet de traiter toute les informations en « parallèle ».

Une fois lancée, la libneopixel lance le script comneopixel.py qu'elle va commander par la suite.

La libcombluetooth, elle, lance le Bash connection permettant de démarrer un appairage Bluetooth. Elle créer aussi un thread pour la gestion de l'interface (état des LED).

d. Gestion des signaux

d.1 SIGINT

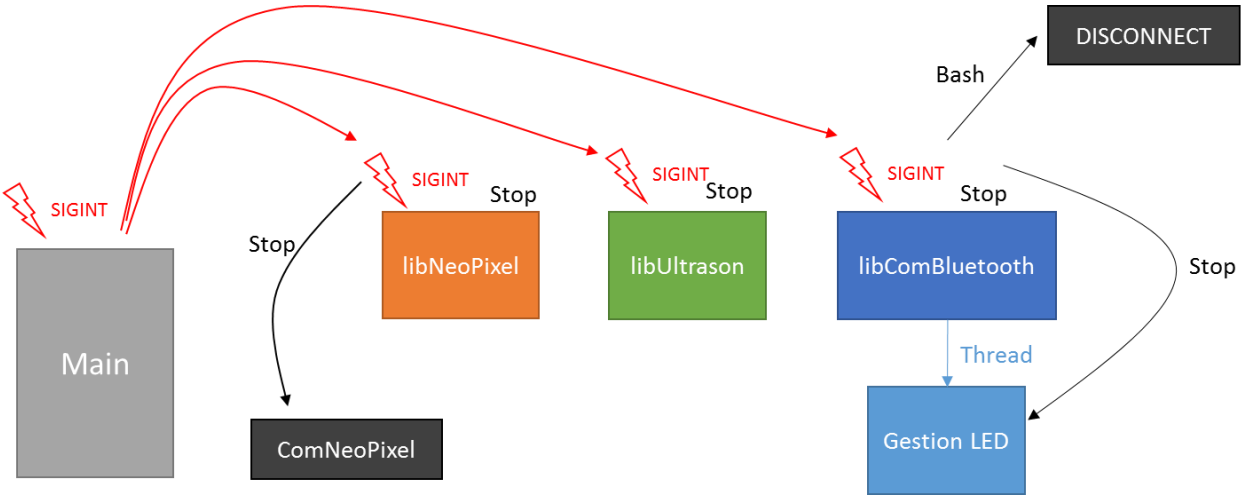


Fig. 3 : Schéma de réception du signal SIGINT

Lorsque le programme principal récupère le signal SIGINT il les transfère aux différents processus qu'il a créés et attends que ceux-ci se termine.

- Libneopixel : envoie une commande d'arrêt au script et s'arrête.
- Libcombluetooth : Arrête son thread, lance le script Bash déconnexion et s'arrête.
- Libultrason : s'arrête.

1.1.1. SIGUSR

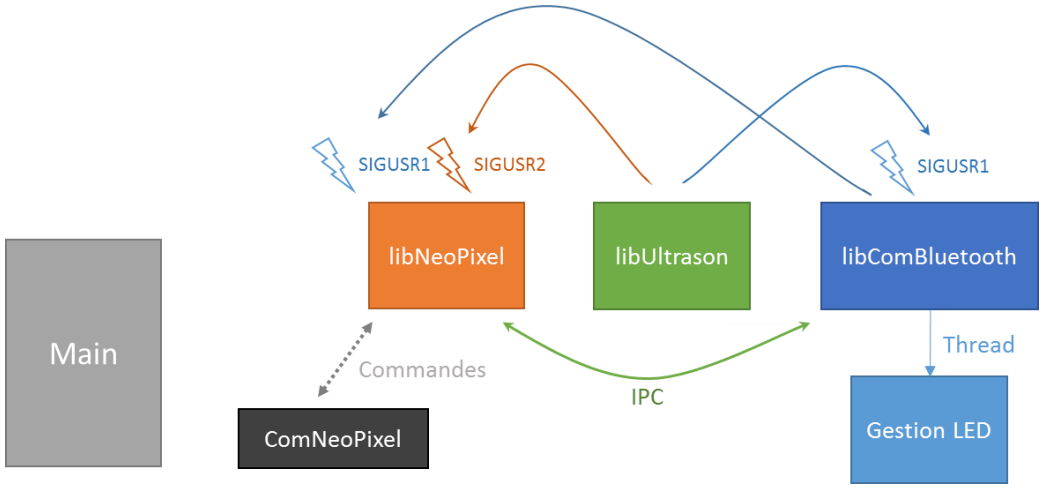


Fig. 4 : Schéma de réception du signal SIGUSR

6. Programmation Android

a. Présentation de l'interface

Nous avons développé une application Android permettant d'avertir l'utilisateur d'une procédure de dépassement. Nous avons aussi ajouté la fonctionnalité de clignotants.

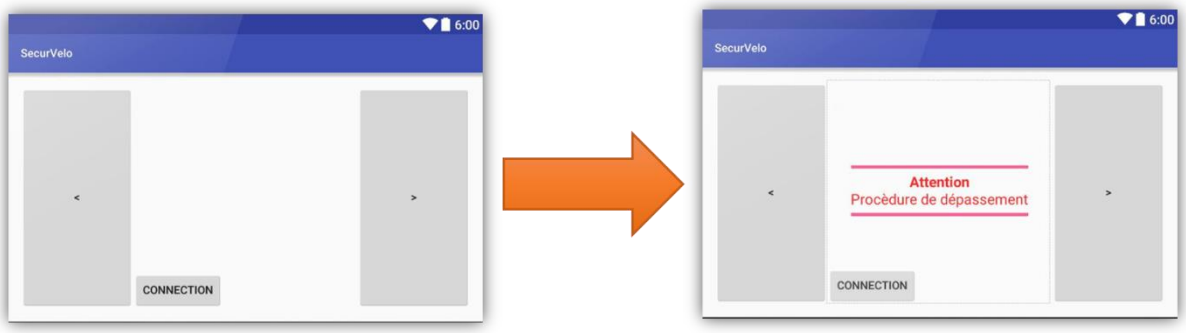


Fig. 5 : Interface de l'application

L'interface est simple et intuitive. Il y a 3 boutons :

- Connection : permet l'appairage avec le dispositif.
- 2x Clignotants : allume/éteint le clignotant sélectionné.

b. Communication Bluetooth

Après de nombreuses recherches sur internet nous avons trouvé une classe java de gestion des sockets en Bluetooth appelée BtInterface.

Cette classe contient 5 méthodes :

- public BluetoothDevice getDevice()

  Retourne un objet du type 'BluetoothDevice' de la liste des appareils appairés.

- public void connect()

  Permet d'ouvrir une connexion socket avec l'appareil en handler du type 'BluetoothDevice'.

- public void close()

  Permet de se fermer la connexion avec le handler.

- public void sendData(String data)

  Permet d'envoyer un string via la connexion établie.

Ainsi qu'une classe de réception dans un Thread :

- private class ReceiverThread extends Thread {...}
  Lit en boucle le buffer de réception

On implémente l'interface View.OnClickListener à la classe MainActivity afin de pouvoir recevoir les interruptions relatives à l'appui d'un bouton.

```
public class MainActivity extends Activity implements View.OnClickListener {

    /* ...
       Déclaration des différents widgets (Button, TextView...)
       ...
    */

    private Button connect;
    private BtInterface bt;

    //Dans le constructeur
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity main);

        /* ...
           Initialisation des widgets
           ...
        */

        //Allocation et construction d'une BtInterface
        bt = new BtInterface(handlerStatus, handler);

        //Initialisation du bouton connect (que l'on récupère du fichier XML)
        connect = (Button)findViewById(R.id.connect);

        //On lui fixe un Listener -> interruption
        connect.setOnClickListener(this);

        client = new GoogleApiClient.Builder(this).addApi(AppIndex.API).build();
    }

    //Dans la méthode onClick appeler lors d'un Button
    public void onClick(View v) {

        //On regarde quel bouton à été appuyé
        switch(v.getId()) {

            //Si c'est le bouton connect
            case R.id.connect:
                //Alors on appel la méthode connect de la class BtInterface
                bt.connect();
                break;
        }
    }
```

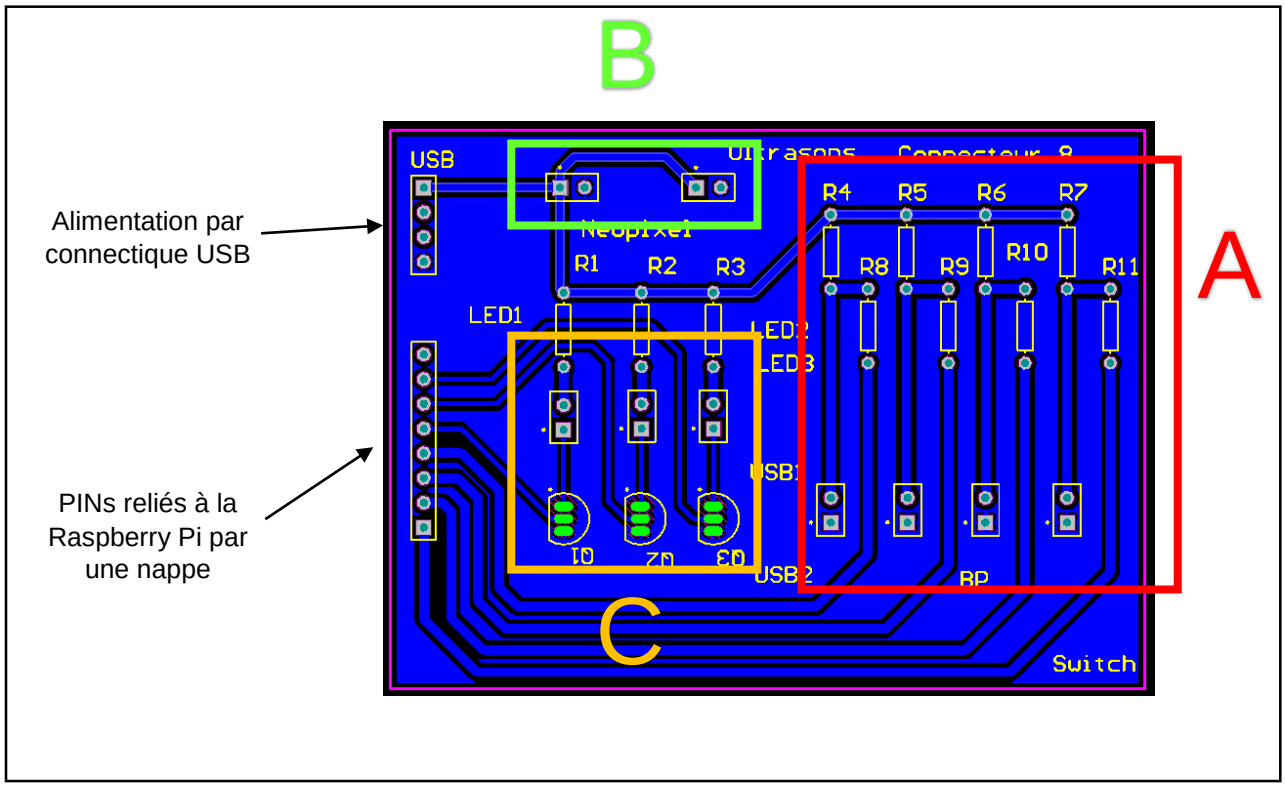
7. Partie électronique

a. Présentation de la batterie

Ce système ayant pour finalité d'être embarqué, il fallait gérer toute la partie alimentation et le circuit associé.

La source d'énergie choisie est une batterie RS PB-A5200, pouvant fournir 5000mAh avec une tension de 5V. ces caractéristiques sont adaptées à notre utilisation. L'avantage de cette batterie est qu'elle peut être rechargée par micro-USB, ce qui est assez accessible comme connectivité (identique aux prises standards des téléphones portables actuels). Mais nous avons également choisi cette batterie car elle possède deux sorties USB. De ce fait nous pouvons en utiliser une pour alimenter directement la Raspberry Pi, et une autre qui sera reliée par câble USB au circuit électronique pour alimenter les différents modules du système.

b. PCB gérant l'alimentation

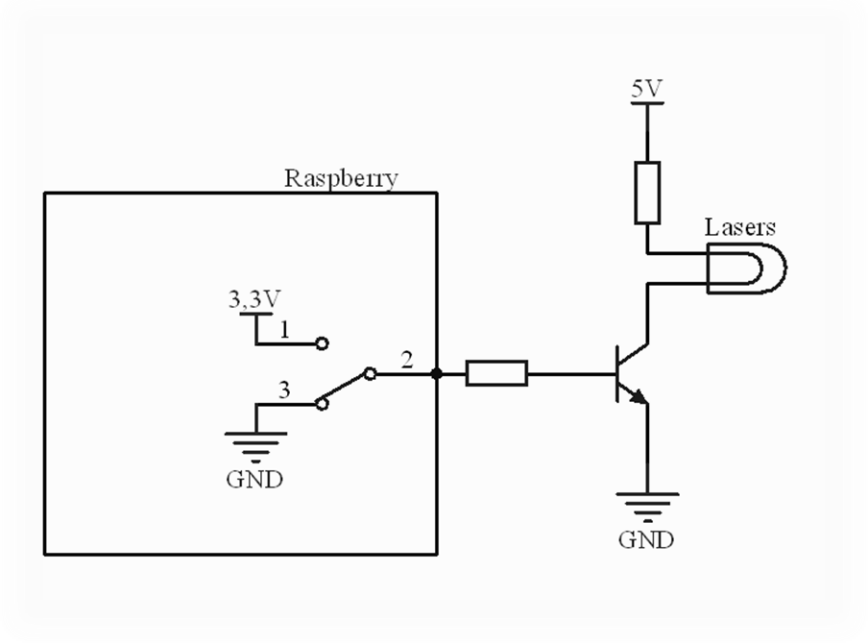


Zone A : Commande des boutons et ports USB sur l'interface physique du système

Zone B : Alimentation (+5V/GND) des modules NeoPixel et Ultrasons

Zone C : Commande des modules lasers & LED pour information utilisateur

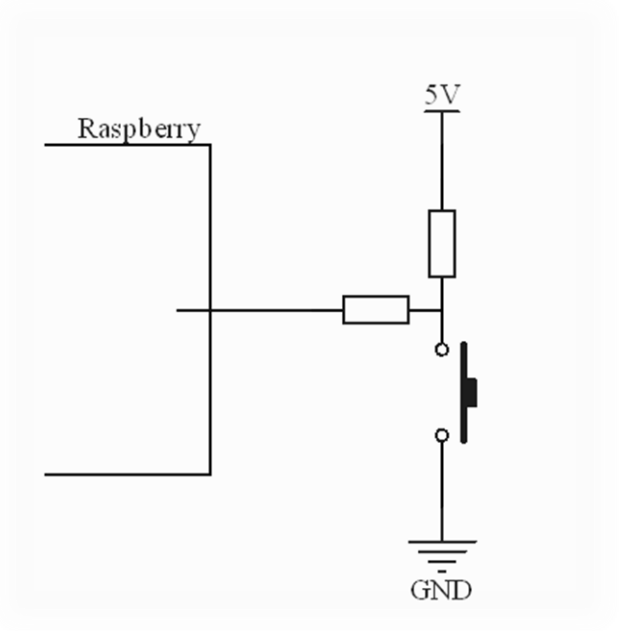
c. Commande des lasers



La commande des lasers est réalisée grâce à un montage simple à l'aide d'un transistor NPN. Le schéma présenté est simplifié, en effet, les 3 modules sont placés en parallèle. En effet, nous commandons les trois de manière synchronisée. Lorsque nous souhaitons déclencher le fonctionnement des lasers, la Raspberry envoie un signal haut sur la pin associée aux laser, qui correspond à la base du transistor. Ainsi, le transistor sera saturé et se comportera comme un interrupteur fermé, ce qui provoquera l'allumage des lasers. En revanche, lorsque la base n'est pas alimentée, le transistor reste dans l'état bloqué, il se comporte donc comme un interrupteur ouvert et les lasers restent éteints.

d. Commande boutons & modules complémentaies

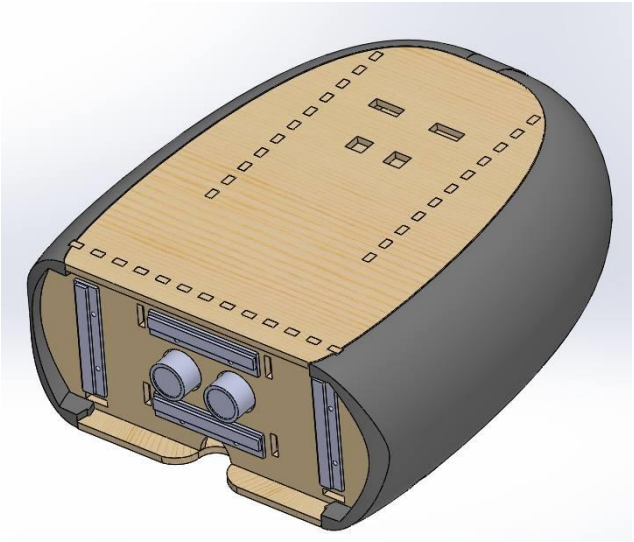
Le montage que nous avons utilisé pour l'utilisation des boutons et ports USB disponibles sur l'interface physique du système est un montage avec une résistance de tirage (Pull-UP). Ce montage permet de pas laisser flottante l'entrée du circuit. Dans ce cas, la résistance de tirage impose un niveau haut (5V) alors que l'interrupteur ou module présente un niveau bas. Sans résistance de tirage, l'entrée serait flottante et ainsi à un niveau logique indéfini et pourrait causer un dysfonctionnement sur le circuit logique d'entrée.



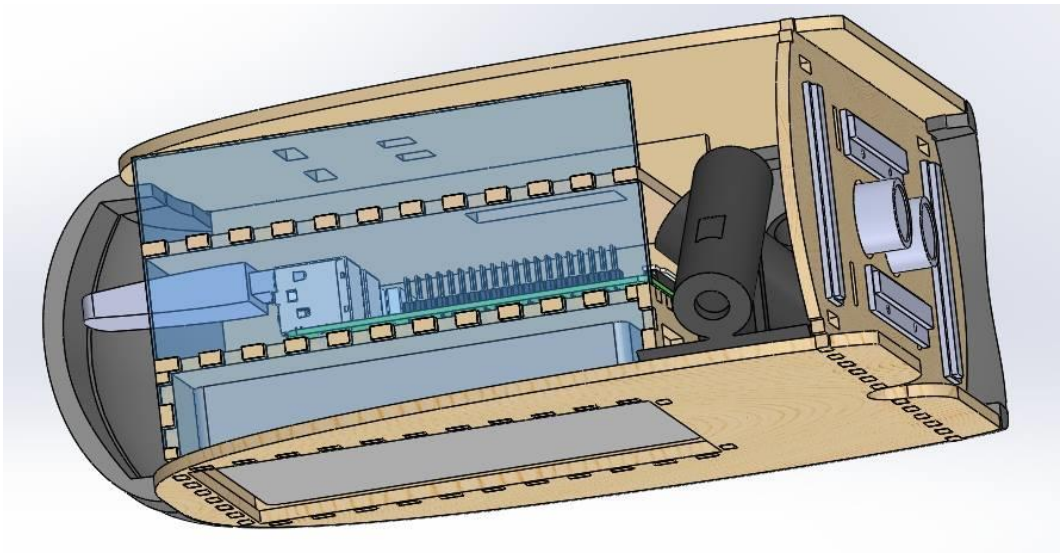
Note suite à une question posée en soutenance : la résistance présente sur la PIN n'est pas nécessaire dans notre cas, car elle est déjà incluse dans la Raspberry Pi, mais il aurait fallu l'ajouter si le circuit logique associé était plus rudimentaire.

8. Modélisation 3D du boitier

Pour ce projet, nous nous sommes focalisés sur l'objectif d'avoir un vrai prototype à présenter pour la soutenance. Il s'agissait donc de travailler également l'aspect visuel. La principale contrainte était de réunir tous les composants du système dans un minimum de place. Le boitier final a été conçu en 3D sur le logiciel SolidWorks. Afin de définir l'occupation de chaque composant il a fallu les modéliser un par un. Une fois ce travail effectué, nous avons élaboré le boitier final afin qu'il soit fonctionnel (interface physique intuitive et accès à la batterie simplifié) mais également esthétique.



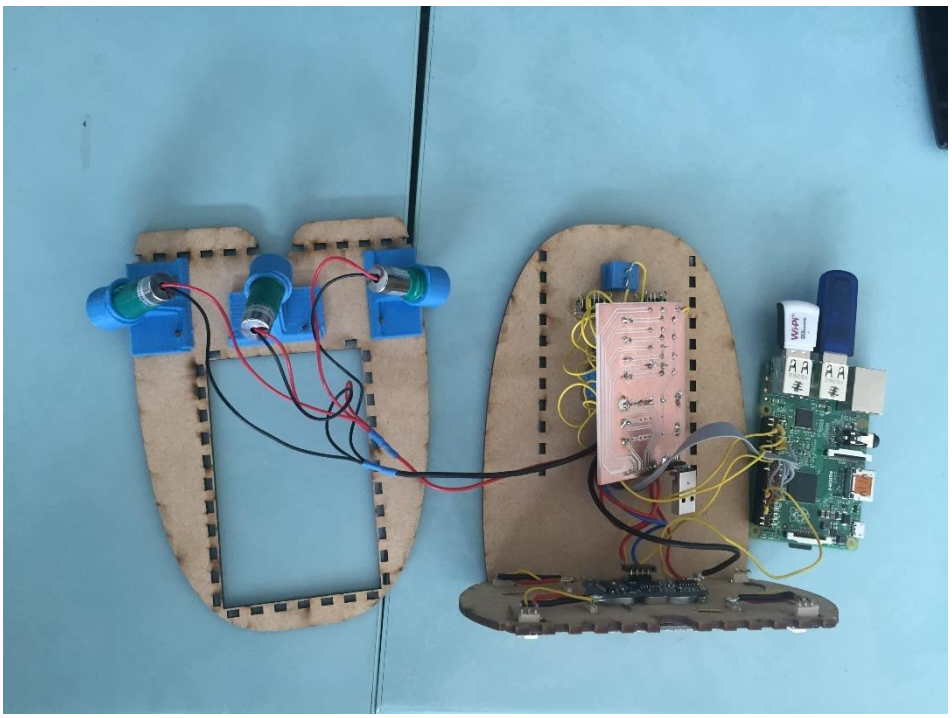
Toutes les parties planes sont en bois (épaisseur 3mm), et ont été réalisées à la découpeuse laser (Polytech). Les parties latérales du boitier ainsi que les supports lasers ont été réalisés en impression 3D (Polytech et TopOffice).



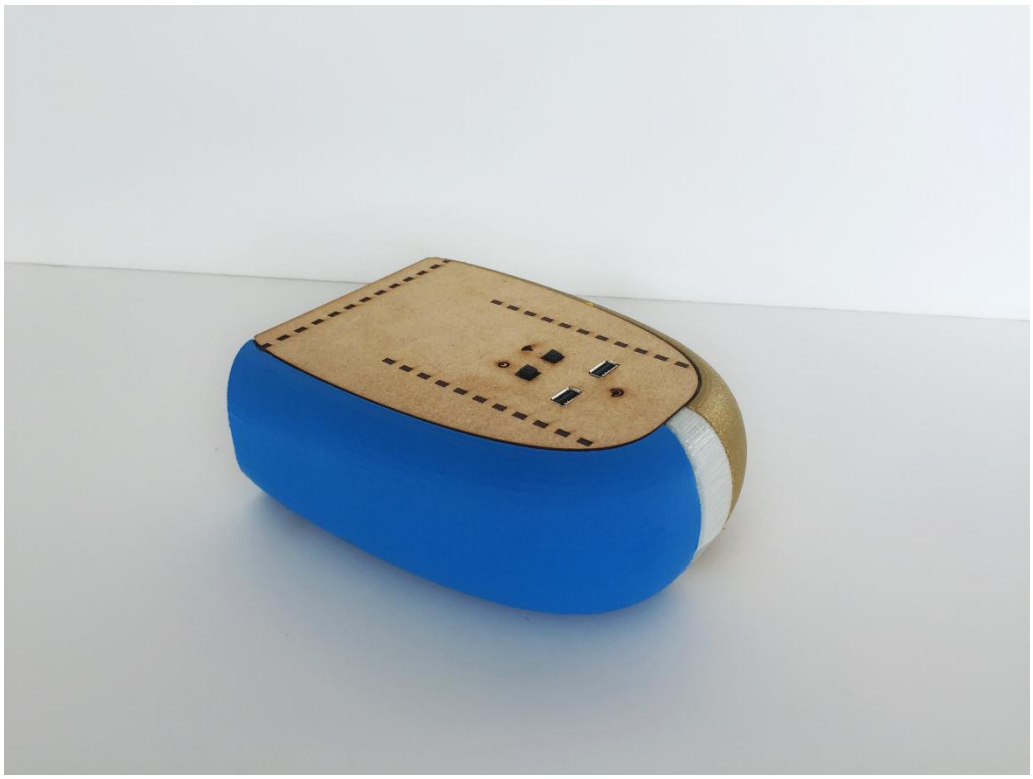
Au niveau des supports pour les lasers, il s'agissait de trouver un système permettant de fixer les lasers tout en leur imposant un certain angle, ce qui permettait la projection des lignes à une distance d'environ 1m du vélo.

9. Assemblage du boitier et réalisation finale

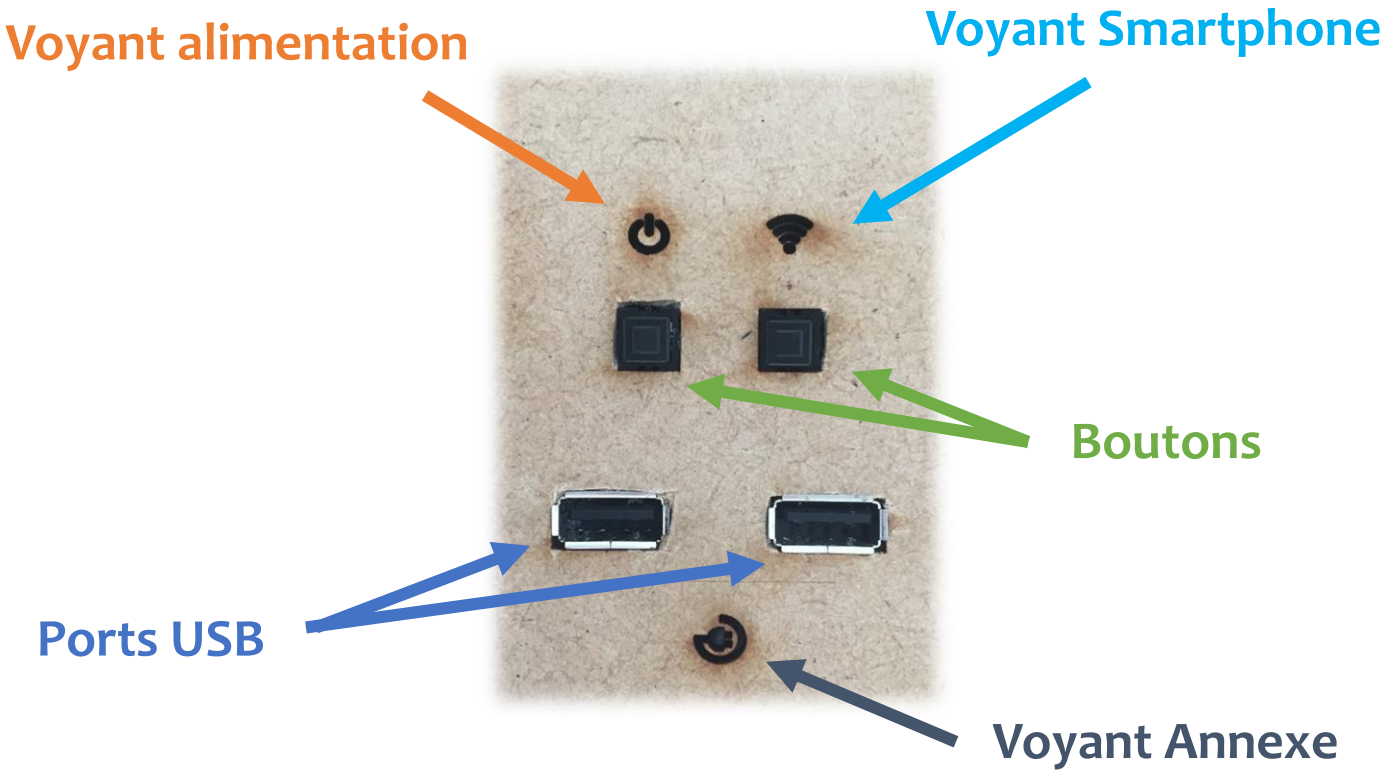
Suite à la réalisation du boitier, du PCB et des soudures des composants, nous avons pu procéder à l'assemblage du système. Les principales contraintes étaient de tout réunir dans le même boitier. Car même en ayant pris en compte l'occupation de chaque composant lors de la conception du boitier, l'espace disponible était très réduit. De plus, nous n'avions pas forcement pris en compte la place occupée par les fils et différents câbles



Rendu final du final du prototype :



10. Présentation de l'interface utilisateur physique

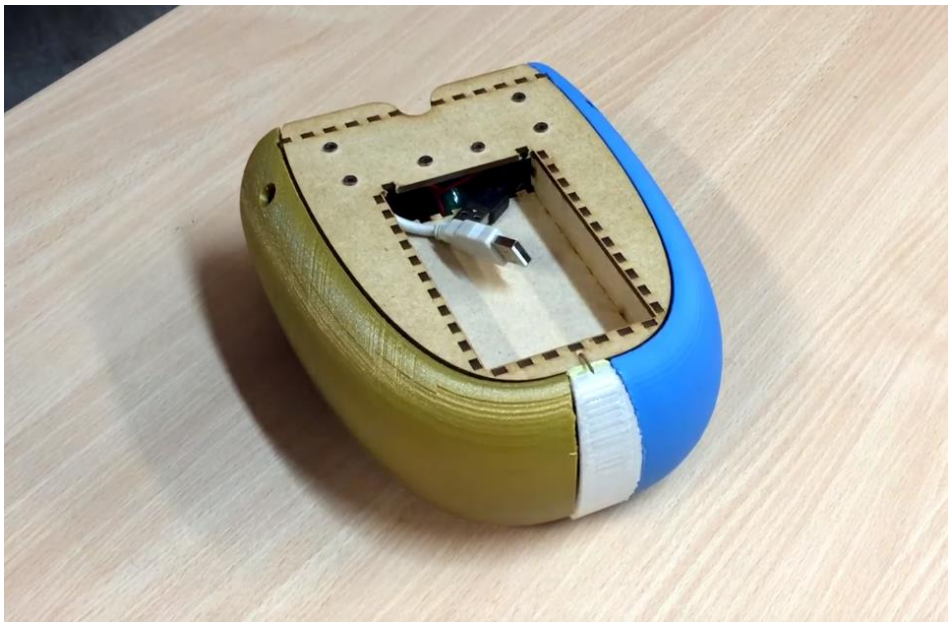


L'interface physique est voulue simple et accessible pour l'utilisateur. Elle a été développée pour permettre le fonctionnement du système en autonomie, c'est-à-dire sans utilisation obligatoire du Smartphone. Elle présente différents voyants, boutons et connectique utilisables par l'utilisateur.

- Le voyant d'alimentation permet d'informer l'utilisateur si le système est en fonctionnement.
- Le voyant Smartphone informe de l'état de la connectivité entre le téléphone et le système. S'il est éteint, le système est en fonctionnement autonome. S'il clignote, le système lance la procédure d'appairage. S'il est allumé, les deux appareils sont liés et fonctionnent ensemble.
- Le bouton de gauche permet de réinitialiser le programme, alors que celui de droite lance la procédure d'appairage avec le téléphone.
- Les ports USB permettent l'ajout futur de modules complémentaires au système, tels qu'un système de feux stops (en cas de freinage), un antivol connecté, ou un compteur de vitesse.
- Le voyant annexe permet d'informer l'utilisateur si un module est connecté et alimenté.

11. Accessibilité à la batterie

La batterie, dont l'accès est placé sous le système est amovible pour la recharge. Elle est connectée à la Raspberry et à la carte électronique via deux câbles USB. Il était important de garantir une accessibilité aisée à l'alimentation du système pour une recharge simple pour l'utilisateur.



12. Résultats obtenus

A la fin du projet, une majorité des points du cahier des charges ont été respecté, il ne reste à l'heure actuelle plus que la correction de quelques points mineurs et l'ajout de modules à effectuer.

Opérationnel	Améliorations
▪ Boitier esthétique ▪ Appairage avec Smartphone ▪ Clignotants ▪ Lasers (zone de sécurité) ▪ Warnings ▪ Module ultrasons	▪ Matériel (plus performant) ▪ Correction du code et connectique (notamment Bluetooth) ▪ Installation de modules supplémentaires ▪ Système de fixation au vélo pas encore réalisé ▪ Exploitation plus poussée de l'utilisation du Smartphone (utilisation du GPS)

13. Conclusion

Ce projet, réalisé dans le cadre de notre quatrième année à Polytech Lille a quasiment été mené à son terme.

Nous avons été en mesure de fournir un prototype quasi fonctionnel à la fin de ce projet. Tant sur l'aspect électronique qu'informatique, mais également au niveau du boitier qui a été soigneusement travaillé.

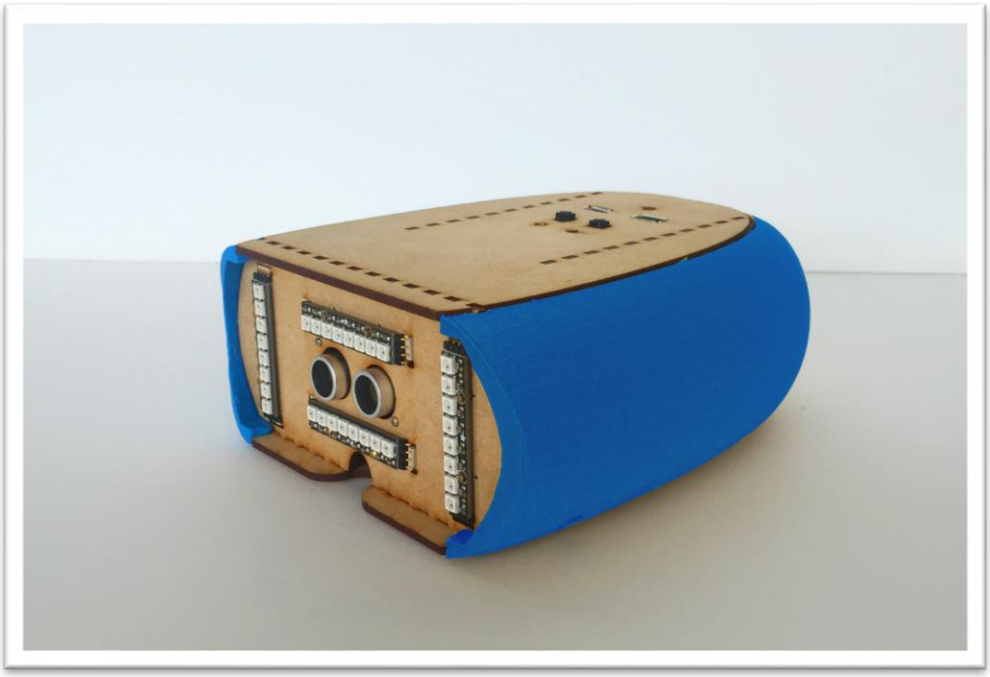
Les fonctionnalités de la zone de sécurité, de l'avertissement lumineux ainsi que la communication avec le Smartphone et le système de clignotants sont fonctionnelles.

Néanmoins, le projet reste perfectible, que ce soit au niveau de la programmation et de la connectivité, mais également par l'ajout de modules complémentaires que nous n'avons pas eu le temps de développer, comme le système de feu stop, d'antivol connecté ou bien de compteur de vitesse.

Nous avons appris à travailler en binôme sur un projet de longue durée et sommes fiers d'avoir pu présenter un prototype honorable lors de notre soutenance.

CLAVERIE Martin

CHARNET Victor



- ANNEXES -

1.1. Programmes

1.1.1. MAKEFILE

```
CFLAGS += -Wall -DDEBUG
lib += -lcombluetooth -lneopixel -lultrason -lpthread -lwiringPi

all: libcombluetooth libneopixel libultrason main

clean:
	rm -f core *.o SecuriteVelo
	rm -f core lib/*.a

main:
	$(CC) $(CFLAGS) -I include -L lib -O main.c -o SecuriteVelo $(lib)

libcombluetooth:
	$(CC) $(CFLAGS) -c lib/libcombluetooth/libcombluetooth.c -o
lib/libcombluetooth/libcombluetooth.o
	ar r lib/libcombluetooth.a lib/libcombluetooth/libcombluetooth.o
	ranlib lib/libcombluetooth.a

libneopixel:
	$(CC) $(CFLAGS) -c lib/libneopixel/libneopixel.c -o
lib/libneopixel/libneopixel.o
	ar r lib/libneopixel.a lib/libneopixel/libneopixel.o
	ranlib lib/libneopixel.a

libultrason:
	$(CC) $(CFLAGS) -c lib/libultrason/libultrason.c -o
lib/libultrason/libultrason.o
	ar r lib/libultrason.a lib/libultrason/libultrason.o
	ranlib lib/libultrason.a
```

1.1.2. MAIN.C

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>

#include "define/signaux.h"
#include "define/gpio.h"

#include <libcombluetooth.h>
#include <libneopixel.h>
#include <libultrason.h>
```

```
pid_t combluetooth;
pid_t neopixel;
pid t ultrason;
int status;
int stop;
int wait_warning;

void distribution_signal(int signal)
{
        if(signal == SIGINT)
        {
                stop = 1;
                kill(combluetooth, SIGINT);
                kill(neopixel, SIGINT);
                kill(ultrason, SIGINT);
        }
}

void arret()
{
        distribution_signal(SIGINT);
}

int main(void)
{
        int status;
        stop = 0;
        wait_warning = 0;

        struct sigaction action;

    action.sa_handler = distribution_signal;
    sigaction(SIGINT, &action, NULL);

    if (wiringPiSetup () == -1)
        exit(0) ;

    digitalWrite(GPIO_LED_ON,1);
    wiringPiISR(GPIO_BOUTON_ON,INT_EDGE_FALLING,arret);


    pid_t pid_pere = getpid() ;

    neopixel = fork();

        if(neopixel == 0)
                fork_neopixel();

        if(getpid() == pid_pere)
        {
                combluetooth = fork();

                if(combluetooth == 0)
                        fork_combluetooth(neopixel);
        }

        if(getpid() == pid_pere)
        {
                ultrason = fork();
```

```
                if(ultrason == 0)
                        fork_ultrason(neopixel, combluetooth);
        }

        if(getpid() == pid_pere)
        {
                #ifdef DEBUG
                        printf("pid combluetooth: %d\n", combluetooth);
                        printf("pid neopixel:     %d\n", neopixel);
                        printf("pid ultrason:     %d\n", ultrason);
                #endif

                while(!stop)
                        printf(".");

                sleep(3);
                #ifdef DEBUG
                        printf("[STOP] main\n");
                #endif

                digitalWrite(GPIO_LED_ON,0);
        }

        return 0;
}
```

1.1.3. GPIO.H

```
#include <wiringPi.h>

#define GPIO_BOUTON_CONNECT           5
#define GPIO_BOUTON_ON                        4

#define GPIO_LASER                            0
#define GPIO_LED_ON                           3
#define GPIO_LED_BLUETOOTH            2
#define GPIO_LED_USB                  25

#define GPIO_ECHO_ULTRASON            11
#define GPIO_TRIG_ULTRASON            27

#define GPIO_NEOPIXELS                        1

extern int wait_warning;
```

1.1.4. SIGNAUX.H

```
#include <signal.h>

#define SIGOBSTACLE 10


//IPC
#define RAS             0
#define STOP            1
#define ARRET_STOP  2
```

```
#define CLIG_DROIT     3
#define CLIG_GAUCHE 4
#define WARNING        5
```

1.1.5. LIBCOMBLUETOOTH.C

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#include <signal.h>

#include "../../define/signaux.h"
#include "../../define/gpio.h"

#define NON_CONNECTE 0
#define RECHERCHE 1
#define CONNECTE 2

#define CLE_MSG (key_t) 50

typedef struct  {
long type;
char texte[1];
} Msg_requete;

int msgid;

FILE* socket_bluetooth = NULL;
pthread_t tid;

int pid;
int boucle;
int connect;
int anti_rebond;
int libcom_warning;
Msg_requete message;

void init_gpio()
{
        pinMode (GPIO_LED_ON, OUTPUT) ;
        pinMode (GPIO_LED_BLUETOOTH, OUTPUT) ;
    pinMode (GPIO_LED_USB, OUTPUT) ;
}

void deconnexion()
{
    boucle = 0;

    if(socket_bluetooth != NULL)
        fclose(socket_bluetooth);

    digitalWrite(GPIO_LED_ON,0);
    digitalWrite(GPIO_LED_BLUETOOTH,0);
```

```
    digitalWrite(GPIO_LED_USB,0);

    pthread_join(tid, NULL);

    system("bash bash/disconnect > log");
    msgctl(msgid,IPC_RMID,0);

    #ifdef DEBUG
    printf("[STOP] lcombluetooth: fork_combluetooth\n");
    #endif

    exit(0);
}

void libcombluetoothHandler_gestionSignaux(int signal)
{
    if(signal == SIGINT)
        deconnexion();

    else if(signal == SIGUSR1 && connect == CONNECTE)
    {
        FILE* socket_bluetooth_2 = fopen("/dev/rfcomm0", "w");

        if(socket_bluetooth != NULL)
        {
            if(libcom_warning == 0){
                fprintf(socket_bluetooth_2, "%c", 'W');
                libcom_warning = 1;
            }
            else if(libcom_warning == 1){
                fprintf(socket_bluetooth_2, "%c", 'X');
                libcom_warning = 0;
            }
            fclose(socket_bluetooth_2);
        }
    }
}

void connection()
{
    if(anti_rebond == 0){
    connect = RECHERCHE;
    system("bash bash/connect & > log");
    anti rebond = 1;
    }
}

void* thread_gestionLED(void* arg)
{
    while(boucle){
         switch(connect)
         {
            case RECHERCHE:
                    digitalWrite(GPIO_LED_BLUETOOTH,1);
                    sleep(1);
                    digitalWrite(GPIO_LED_BLUETOOTH,0);
                    sleep(1);
            break;

            case CONNECTE:
                    digitalWrite(GPIO_LED_BLUETOOTH,1);
```

```
                    sleep(1);
            break;

            case NON_CONNECTE:
                    digitalWrite(GPIO_LED_BLUETOOTH,0);
                    sleep(1);
            break;
        }
    }

    #ifdef DEBUG
    printf("[STOP] lcombluetooth: thread_gestionLED \n");
    #endif


        pthread_exit(NULL);
}

void lectureBluetooth(int pid_neopixel)
{
    socket_bluetooth = fopen("/dev/rfcomm0", "r");

    if (socket_bluetooth != NULL)
    {
        int caractereActuel;
        do
        {
            caractereActuel = fgetc(socket_bluetooth); // On lit le
caractère
            if(caractereActuel == 'a')
            {
                wait_warning = 1;
                printf("[+] Receive: CLIGNOTANT DROIT\n");

                sprintf(message.texte,"%d",CLIG_DROIT);
                message.type = 1;
                msgsnd(msgid, &message,  1, 0);

                kill(pid_neopixel, SIGUSR1);
                connect = CONNECTE;
            }
            else if(caractereActuel == 'b')
            {
                wait warning = 1;
                printf("[+] Receive: CLIGNOTANT GAUCHE\n");

                sprintf(message.texte,"%d",CLIG_GAUCHE);
                message.type = 1;
                msgsnd(msgid, &message,  1, 0);

                kill(pid_neopixel, SIGUSR1);
                connect = CONNECTE;
            }


        }while(caractereActuel != EOF);

        fclose(socket_bluetooth);
    }
}
```

```
void fork_combluetooth(int pid_neopixel)
{

    pid = getpid();
    socket_bluetooth = NULL;

    anti_rebond = 0;
    boucle = 1;
    libcom_warning = 0;
    connect = NON_CONNECTE;

    pthread_create(&tid, NULL, thread_gestionLED, NULL);

    struct sigaction action;

    action.sa_handler = libcombluetoothHandler_gestionSignaux;
    sigaction(SIGINT, &action, NULL);
    sigaction(SIGUSR1, &action, NULL);

    if((msgid=msgget(CLE_MSG,0))==-1){
        perror ("msgget");
        exit(1);
    }

    if (wiringPiSetup () == -1)
        exit(0) ;

    init_gpio();

    wiringPiISR(GPIO_BOUTON_CONNECT,INT_EDGE_FALLING,connection);

    while(connect == 0 && boucle == 1) {
        printf(".");
    }

    do{
        lectureBluetooth(pid_neopixel);
    }while(boucle);

}
```

1.1.6. LIBNEOPIXEL.C

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <signal.h>

#include "../../define/gpio.h"
#include "../../define/signaux.h"

#define CLE_MSG (key_t) 50
```

```
typedef struct  {
long type;
char texte[1];
} Msg requete;

int pid;
int neo_warning;
int msgid;
int cmd;
FILE* commande = NULL;

Msg_requete message;

void* start_neopixel(void* arg)
{
        commande = fopen("python/config", "w");

        if(commande != NULL){
                fprintf(commande, "%d", RAS);
                fclose(commande);
        }

        system("python python/comneopixel.py &");
        pthread_exit(NULL);
}

void gestionSignaux(int signal)
{
        if(signal == SIGINT){

                        #ifdef DEBUG
                        printf("[STOP] lneopixel: neopixel_init\n");
                    #endif

                    commande = fopen("python/config", "w");

                    if(commande != NULL){
                        fprintf(commande, "S");
                        fclose(commande);
                    }
                        msgctl(msgid,IPC_RMID,0);
                        exit(0);
        }
        else if(signal == SIGUSR1)
        {
                msgrcv(msgid, &message, 1, 1, 0);
                printf("Neopixel: %s\n", message.texte);
                commande = fopen("python/config", "w");

                if(commande != NULL){

                        switch(message.texte[0] - 48){
                                case STOP:
                                        cmd = STOP;
                                break;

                                case ARRET_STOP:
                                        cmd = ARRET_STOP;
                                break;

                                case CLIG_DROIT:
```

```
                                        if(cmd != STOP){
                                                if(cmd != CLIG_DROIT)
                                                        cmd = CLIG_DROIT;
                                                else{
                                                        cmd = RAS;
                                                        wait_warning = 0;
                                                }
                                        }
                                break;

                                case CLIG_GAUCHE:
                                        if(cmd != STOP){
                                                if(cmd != CLIG_GAUCHE)
                                                        cmd = CLIG_GAUCHE;
                                                else{
                                                        cmd = RAS;
                                                        wait_warning = 0;
                                                }
                                        }
                                break;
                        }

                        fprintf(commande, "%d", cmd);

                        fclose(commande);

                }
        }
        else if(signal == SIGUSR2)
        {
                commande = fopen("python/config", "w");

                if(commande != NULL){
                        if(neo_warning == 0){
                                neo_warning = 1;
                                cmd = WARNING;
                                fprintf(commande, "%d", cmd);
                        }
                        else if( neo_warning == 1){
                                neo_warning = 0;
                                cmd = RAS;
                                fprintf(commande, "%d", cmd);
                        }
                        fclose(commande);
                }
        }
}

void fork_neopixel()
{
        pid = getpid();
        neo_warning = 0;

        if((msgid=msgget(CLE_MSG,IPC_CREAT|IPC_EXCL|0666))==-1){
          perror ("msgget");
          exit(1);
        }

        struct sigaction action;

    action.sa_handler = gestionSignaux;
```

```
    sigaction(SIGINT, &action, NULL);
    sigaction(SIGUSR1, &action, NULL);
    sigaction(SIGUSR2, &action, NULL);

    pthread_t tid;
        pthread_create(&tid, NULL, start_neopixel, NULL);

        while(1)
                sleep(1);
}
```

1.1.7. LIBULTRASON.C

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>
#include <signal.h>
#include <sys/time.h>

#include "../../define/gpio.h"
#include "../../define/signaux.h"

#define DISTANCE_DETECTION 50
#define WARNING_DISTANCE 20

#define NB_MOYENNE 3
#define TEMPS_SLEEP_US 100000
#define TEMPS_TRIG_US 10


int pid;
int boucle;
int cpy_pid_neopixel;
int cpy_pid_libcombluetooth;

int warning;

struct timeval tbegin,tend;
long double texec;
float d[NB_MOYENNE];
float  moyDistance;


void stop_handler(int signal)
{
    if(signal == SIGINT){
        boucle = 0;
    }
}

void addValeur(float v)
{
    if(v>0){
        int i;
        for(i=1; i<NB_MOYENNE; i++)
        {
            d[i-1] = d[i];
        }
```

```
            d[NB_MOYENNE-1] = v;
    }
}

void calculMoyenne()
{
    float sum = 0;
    int i;
    for(i=1; i<NB_MOYENNE; i++)
    {
        if(d[i] > sum)
        sum = d[i];
    }
    moyDistance = sum;
}

void calculDistance()
{

        gettimeofday(&tend,NULL);

        texec=(tend.tv_usec-tbegin.tv_usec);

        float valeur = texec/58;
        addValeur(valeur);
        calculMoyenne();


    if(!wait_warning){
        if(warning == 0 && moyDistance < WARNING_DISTANCE)
        {
            kill(cpy_pid_libcombluetooth, SIGUSR1);
            kill(cpy_pid_neopixel, SIGUSR2);
            warning = 1;
        }

        if(warning == 1 && moyDistance > WARNING_DISTANCE)
        {
            kill(cpy_pid_libcombluetooth, SIGUSR1);
            kill(cpy_pid_neopixel, SIGUSR2);
            warning = 0;
        }
    }
}

void fork_ultrason(int pid_neopixel, int pid_libcombluetooth)
{
    wiringPiSetup () ;

    boucle = 1;
    warning = 0;

    cpy_pid_neopixel = pid_neopixel;
    cpy_pid_libcombluetooth = pid_libcombluetooth;

    struct sigaction action;

    action.sa_handler = stop_handler;
    sigaction(SIGINT, &action, NULL);

        pinMode (GPIO_ECHO_ULTRASON, INPUT) ;
```

```
        pinMode (GPIO_TRIG_ULTRASON, OUTPUT) ;
        pinMode (0, OUTPUT) ;

        digitalWrite(GPIO TRIG ULTRASON,0);

       wiringPiISR(GPIO_ECHO_ULTRASON,INT_EDGE_FALLING,calculDistance);
        texec=0.;

      do {
           if(!wait_warning){
               digitalWrite(GPIO_TRIG_ULTRASON,1);
               usleep(TEMPS_TRIG_US);
               digitalWrite(GPIO_TRIG_ULTRASON,0);

               // Start timer
               gettimeofday(&tbegin,NULL);
               usleep(TEMPS_SLEEP_US);


               if(moyDistance<DISTANCE_DETECTION)
                   digitalWrite(GPIO_LASER,1);
               else
                   digitalWrite(GPIO_LASER,0);
           }
           else
               digitalWrite(GPIO_LASER,0);

       }while(boucle);


    #ifdef DEBUG
        printf("[STOP] lultrason: fork_ultrason\n");
    #endif

    digitalWrite(GPIO_LASER,0);

}
COMNEOPIXEL.PY# NeoPixel library strandtest example
# Author: Tony DiCola (tony@tonydicola.com)
#
# Direct port of the Arduino NeoPixel library strandtest example.
Showcases
# various animations on a strip of NeoPixels.
import time

from neopixel import *

# LED strip configuration:
LED_COUNT      = 8*4      # Number of LED pixels.
LED_PIN        = 18      # GPIO pin connected to the pixels (must support
PWM!).
LED_FREQ_HZ    = 800000  # LED signal frequency in hertz (usually 800khz)
LED_DMA        = 5      # DMA channel to use for generating signal (try 5)
LED_BRIGHTNESS = 50     # Set to 0 for darkest and 255 for brightest
LED_INVERT     = False   # True to invert the signal (when using NPN
transistor level shift)

CLIGNOTANT_DROITE = 1
CLIGNOTANT_GAUCHE = 2

# Define functions which animate LEDs in various ways.
```

```
def colorWipe(strip, color, wait_ms=0):
        """Wipe color across display a pixel at a time."""
        for i in range(strip.numPixels()):
                strip.setPixelColor(i, color)
                strip.show()
                time.sleep(wait_ms/1000.0)


def setCligotants(strip, color, type):

        for i in range(strip.numPixels()):
                strip.setPixelColor(i,Color(0,0,0))
                strip.show()

        if type == 1 or type == 3:
                for i in range(3*8,4*8):
                        strip.setPixelColor(i, color)
                        strip.show()

        if type == 2 or type == 3:
                for i in range(1*8,2*8):
                        strip.setPixelColor(i, color)
                        strip.show()

def clignotant(type):
        setCligotants(strip,Color(60, 255, 0), type)
        time.sleep(400/1000.0)
        setCligotants(strip,Color(0, 0, 0), type)
        time.sleep(400/1000.0)

def stop():
        colorWipe(strip, Color(0, 255, 0))

def warning():
        colorWipe(strip, Color(0, 255, 0))
        time.sleep(100/1000.0)
        colorWipe(strip, Color(0, 0, 0))
        time.sleep(100/1000.0)

def RAS():
        colorWipe(strip, Color(0, 0, 0))


# Main program logic follows:
if __name__ == '__main__':
        # Create NeoPixel object with appropriate configuration.
        strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA,
LED_INVERT, LED_BRIGHTNESS)
        # Intialize the library (must be called once before other
functions).
        strip.begin()

        boucle = 1;
        print ('Press Ctrl-C to quit.')
        while boucle:

                mon_fichier = open("python/config", "r");
                contenu = mon_fichier.read()
                mon_fichier.close()
```

```
                if contenu =="3":
                        clignotant(CLIGNOTANT_GAUCHE)
                if contenu == "4":
                        clignotant(CLIGNOTANT DROITE)
                if contenu == "1":
                        stop();
                if contenu=="0":
                        RAS();
                if contenu=="5":
                        warning();
                if contenu=="S":
                        boucle = 0;
```

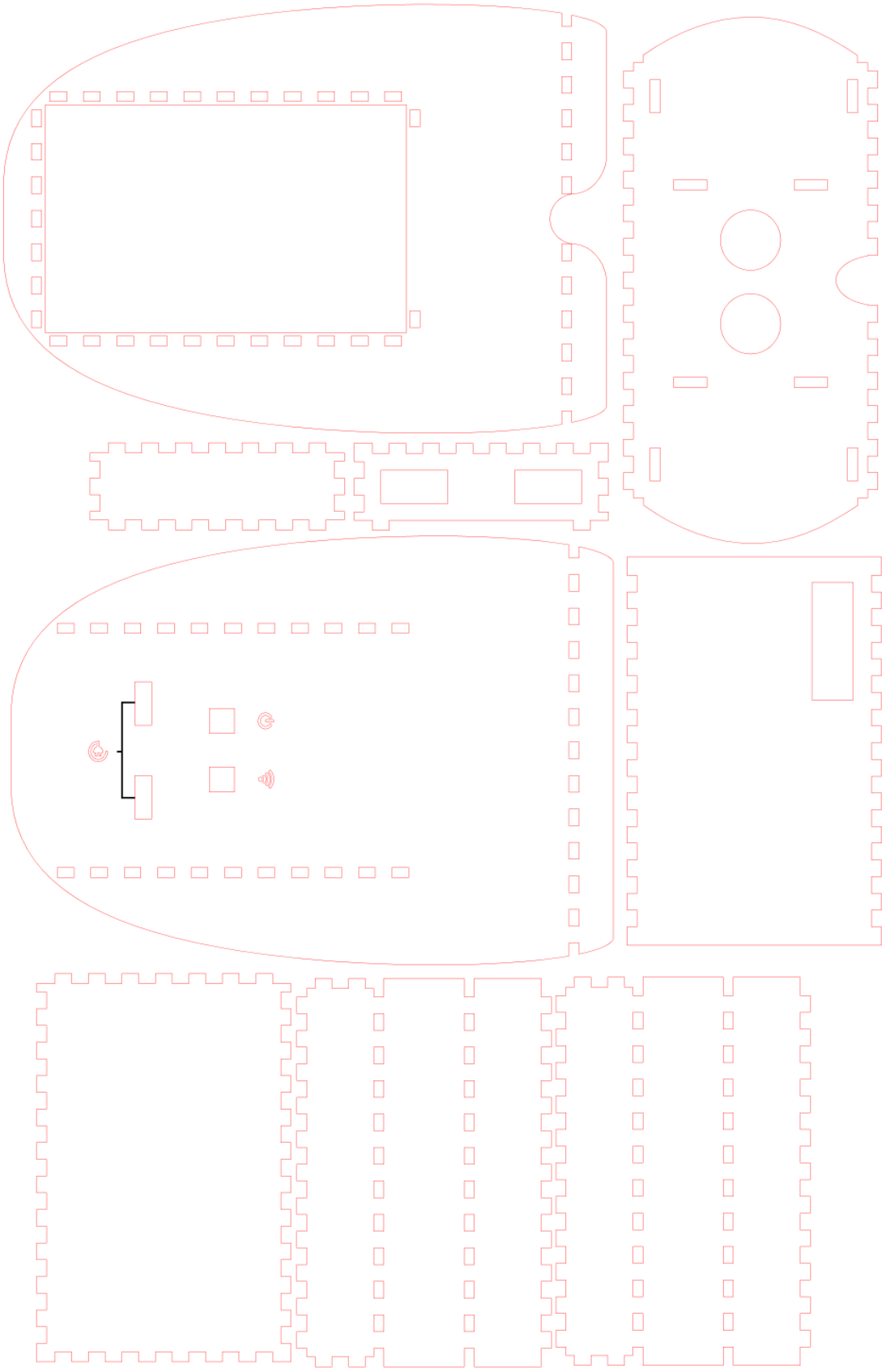
1.1.8. CONNEXION

```
#!/bin/bash

bluetooth-agent 1234 &
sdptool add --channel=22 SP > rapport
rfcomm listen /dev/rfcomm0 22 & >rapport
```

1.1.9. DECONNEXION

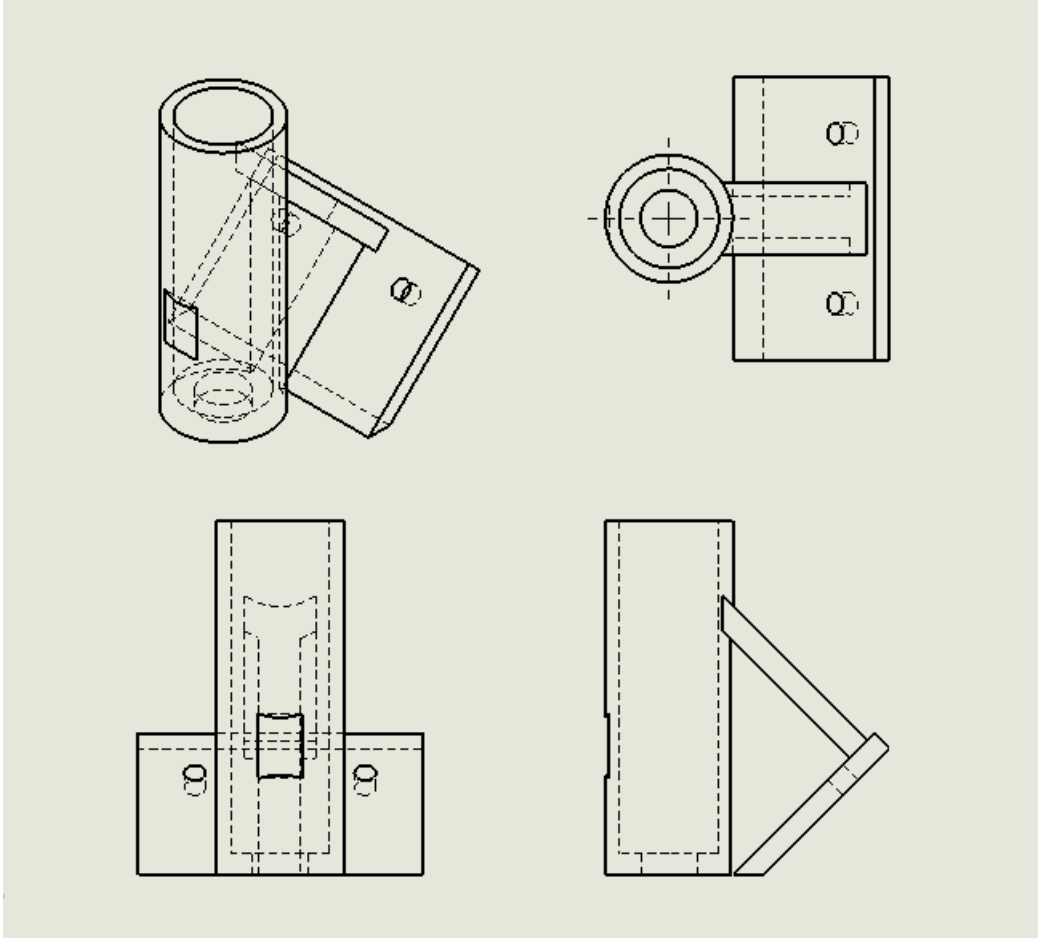
```
#!/bin/bash

rfcomm release 0
```

1.2. Plan découpe laser

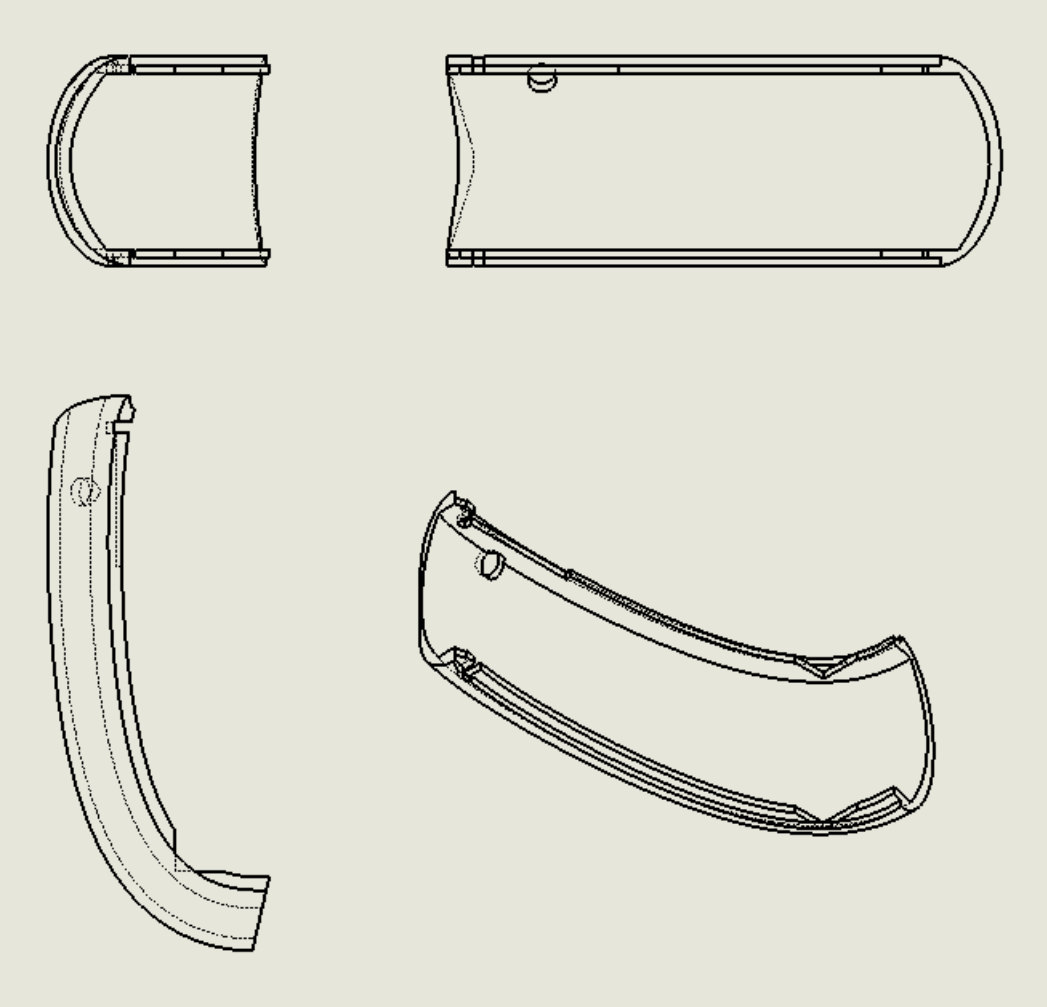


1.3. Pièce imprimée en 3D

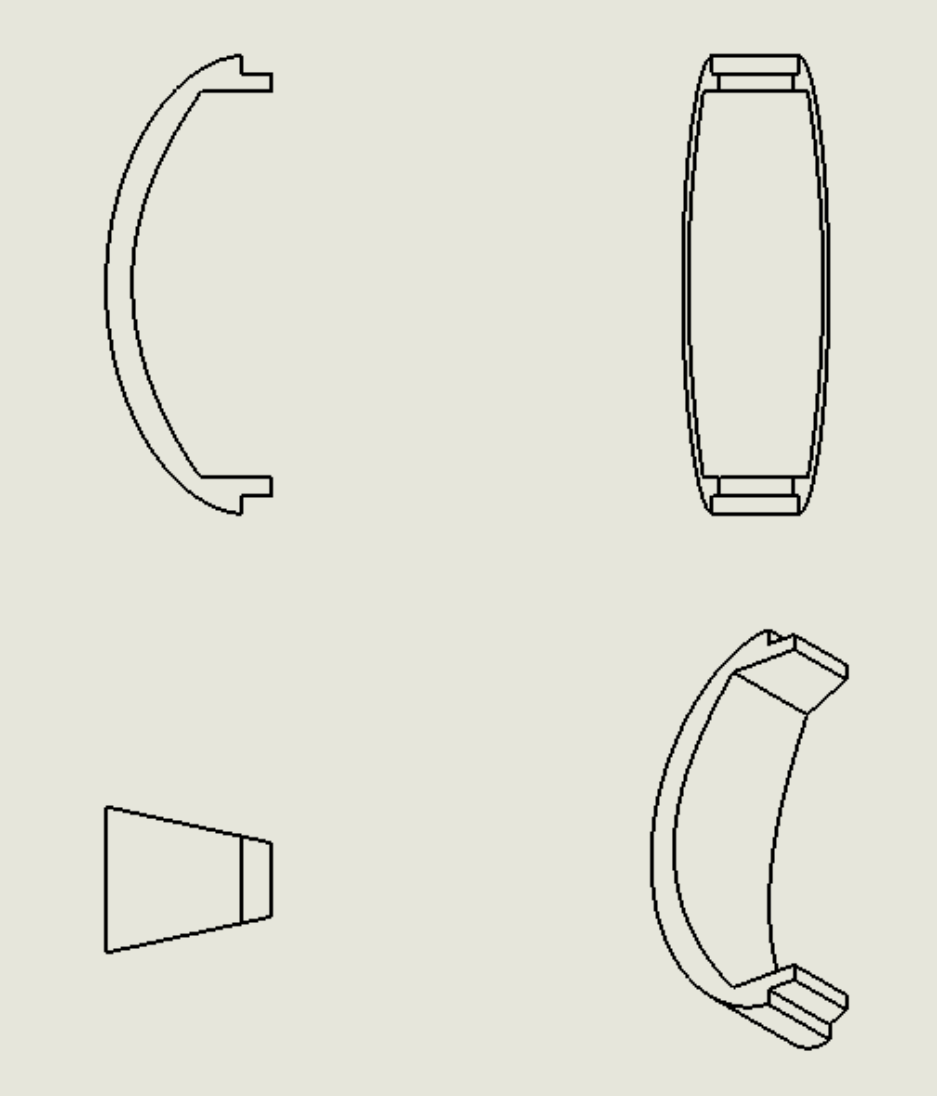
1.3.1. SUPPORT LASER



1.3.2. COTER DROIT / GAUCHE



1.3.3. COTER FIN



1.4. ASSEMBLAGE FINAL