Drone et Oculus Rift

Rapport de Projet

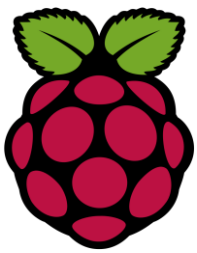
Filière IMA4-SC

Encadrant :

Jérémy Dequidt

Élèves :

Geoffrey Piekacz / Nathan Richez



Sommaire

Introduction

Depuis quelques années, le développement des drones à usage civil s'est largement popularisé. Avec la miniaturisation de l'électronique, ils sont de plus en plus performants et embarquent de plus en plus de fonctionnalités. Dans le cadre de notre projet de 4eme année, département Informatique-Microélectronique-Automatique à Polytech Lille, nous fûmes amenés à travailler sur une nouvelle fonctionnalité qui se répand de plus en plus dans le domaine des drones.

Dans ce projet, nous avons utilisé nos connaissances théoriques acquises ces deux dernières années au sein de l'école, ainsi que des ressources disponibles sur internet que ce soit pour de la documentation technique ou pour l'utilisation d'un outil plus complexe.

Pour commencer, nous présenterons le projet dans son ensemble, puis nous développerons en détails la conception de chaque module, pour enfin finir sur nos impressions.

I- Présentation du projet

1- Objectif du projet

L'objectif premier du projet était de contrôler un drone à l'aide de lunettes de réalité virtuelle et un système fixe de caméras positionné sous le drone. Pour des raisons de confort avec l'utilisation de la réalité virtuelle, il a fallu repenser le projet pour donner aux caméras la même amplitude de mouvement que les Oculus Rift. Ainsi, on envoie à l'opérateur qui contrôle le drone, une vue type jeu-vidéo « première personne » pour lui faciliter la tâche. Un procédé qui tend à se développer mais limité par son coût encore trop élevé.
Notre but est donc de concevoir un système embarqué sur le drone. Permettant de restituer deux flux vidéos, de deux caméras différentes, afin de reconstruire une image stéréoscopique, utilisable par les Oculus Rift.

2- Description du projet

Nous devons récupérer deux flux vidéo provenant d'un système de deux caméras qui sera, pour des raisons techniques, situé sous le drone. Ce système devra permettre aux caméras de pouvoir retranscrire le mouvement des lunettes, positionnées sur la tête de l'utilisateur. Le délai entre la restitution de ces mouvements et de l'image doit être assez court pour maximiser l'expérience de l'utilisateur. Les données doivent être envoyées sur un réseau pour s'affranchir de tout câble.

Nous avons donc réalisé ce système composé de deux caméras, piloté par deux servomoteurs, capable de mouvoir les caméras dans un demi-plan se situant devant le drone. Il est asservi par une carte électronique embarquant linux, rendant possible une communication sur un réseau. Il a fallu imaginer une structure embarquant tout le matériel en respectant certaines contraintes. Puis se documenter sur le fonctionnement du casque de réalité virtuelle (partie programme) afin de récupérer différentes valeurs de capteurs issues du boitier connecté en USB.

3- Matériel et stratégie

Pour ce projet, nous aurons besoin d'un RaspberryPi 2, qui est un nano-ordinateur mono-carte à processeur ARM sous Linux. Il a pour avantage d'être facilement programmable, de petite taille (taille d'une carte de crédit), et embarquant 4 ports USB, une connexion wifi et disposant de plusieurs pins pour contrôler différents composants. Nous aurons besoin de deux servomoteurs, pas forcément très puissants car la distribution de courant du RaspberryPi 2 est limitée à 2 ampères.

Nous avons évidemment besoin de deux caméras. La résolution des Oculus Rift (DK1) étant de 800*600 pixels par œil, il n'est pas nécessaire de dépasser cette valeur. Pour rendre autonome le système, on utilisera une batterie externe type « Smartphone », choisie à partir des caractéristiques d'alimentation du RaspberryPi 2. Ces composants sont détaillés en annexe. Pour le reste, il a fallu imaginer et utiliser différents outils (imprimante 3D, Découpeuse laser) pour façonner les différentes pièces.

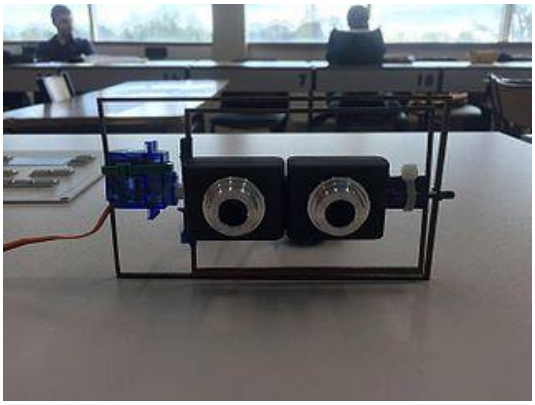
II- Conception des différents modules

Nous allons maintenant développer en détails, les parties essentielles du projet. Dans un premier temps, nous détaillerons la conception du système de caméras autour d'un RaspberryPi, puis la gestion des Oculus Rift, pour enchainer sur la communication entre ces deux modules, pour enfin finir sur la partie « Support plastique ».

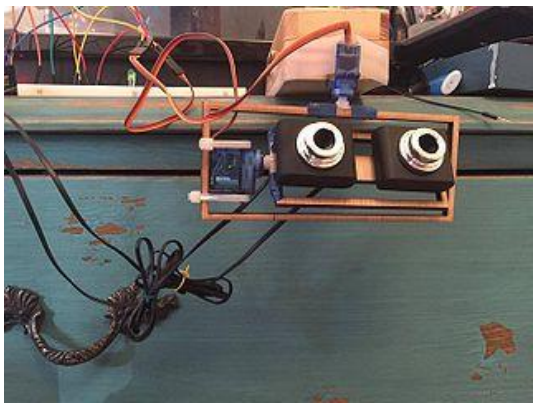
1- Système de caméra

Nous avons conçu un système composé de deux caméras, capable de restituer une image stéréoscopique dans un casque de réalité virtuelle. Il est aussi capable de restituer les mouvements de la tête de l'utilisateur portant les Oculus Rift. Dans cette partie, nous nous intéresserons au contrôle des servomoteurs et à la restitution des images.

La conception de la structure a été plutôt rapide (une à deux séances). Nous avons réalisé deux prototypes car le premier était beaucoup trop fragile. Il se compose de 4 pièces en bois (choix du bois car léger et facilement façonnable avec la découpeuse laser).



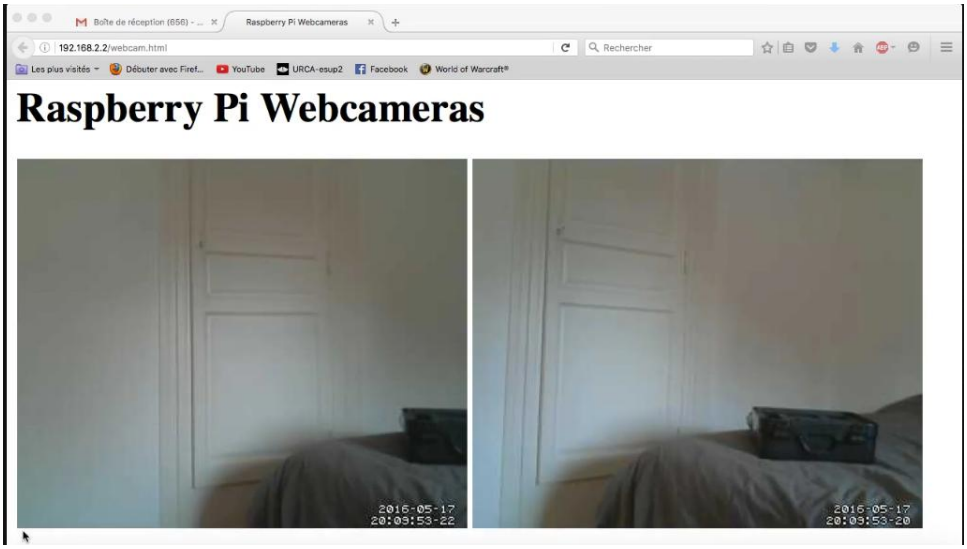
1er prototype



2nd Prototype

Pour motoriser les caméras, on a décidé d'utiliser des microservos SG90 (disponibles en annexe). Par choix, ils n'ont pas beaucoup de couple (pas besoin d'un énorme couple pour bouger des caméras de quelques grammes), donc moins gourmands en énergie. Ainsi, on essaye de préserver la batterie externe. Ces servomoteurs ont besoin de signaux spécifiques (PWM) pour les faire aller à une certaine position.
Ils sont directement branchés sur les broches du RaspberryPi. En effet, nous n'avons pas eu besoin d'utiliser une extension de carte pour contrôler nos deux servomoteurs. Le RaspberryPi 2 possède assez de broches pour pouvoir en piloter au moins deux. Ces servomoteurs sont contrôlés via WiringPi, une bibliothèque écrite en C, à l'origine disponible pour arduino, permettant de générer différentes PWM (Pulse With Modulation) sur certaines broches de la carte. Pour la suite, il a fallu lier les valeurs des accéléromètres à différentes PWM correspondant à différentes positions pour les servomoteurs.

La récupération du flux vidéo, quant à elle, se fait via un utilitaire disponible sur Raspbian: motion. C'est un logiciel permettant de contrôler un périphérique vidéo branché en USB. Son principal problème est d'être très gourmand en ressources pour le RaspberryPi 2. Outre le fait que cet utilitaire fasse de la détection de mouvement (en se basant sur le nombre de pixels changeant), il convertit en temps réel les images au format MPEG. En nous familiarisant avec son fonctionnement, nous sommes parvenus à désactiver toutes ces fonctions superflues et utiliser motion dans un but : la redirection d'un flux vidéo, sans traitement préalable.



La dernière partie consiste à rediriger les ports des flux vidéo sur un site en HTML via appache2. Ces flux sont disponibles en tapant l'adresse du raspberryPi sur le réseau et en y ajoutant soit le port d'écoute du serveur web, soit en y ajoutant le nom du fichier HTML. Cette page est directement utilisable avec les lunettes de réalité virtuelle.

2- Oculus Rift

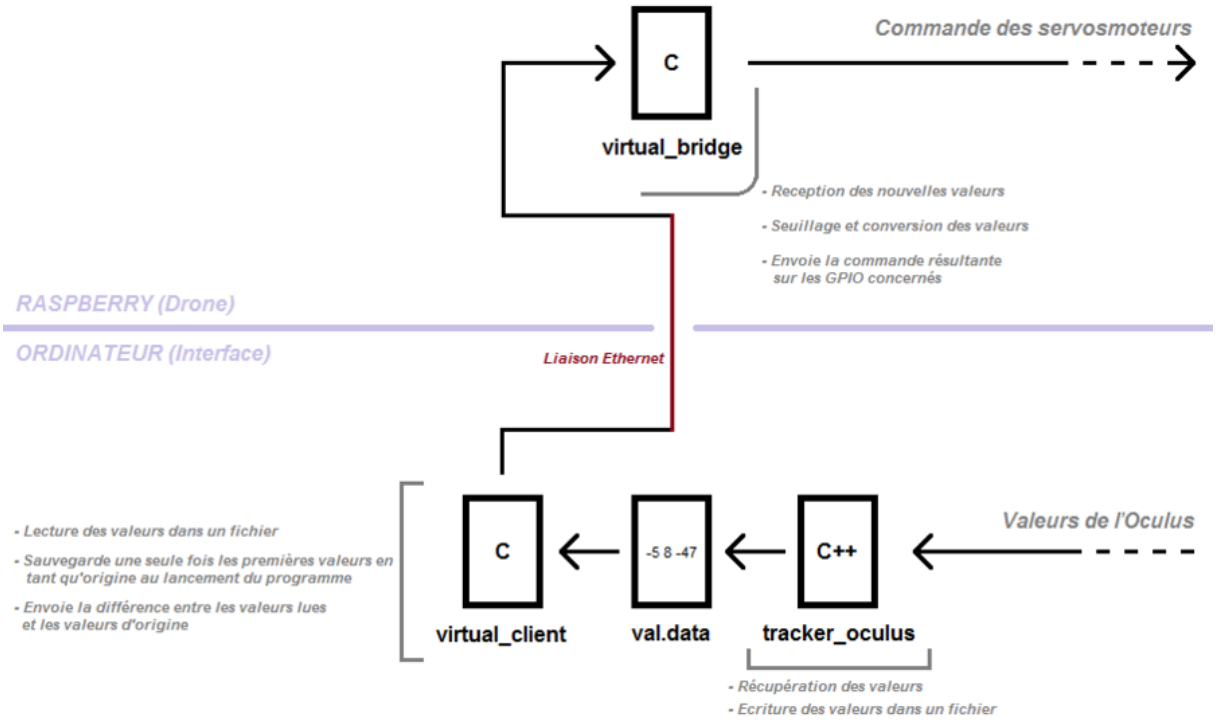
Pour pouvoir utiliser les Oculus Rift, il a fallu trouver un environnement possédant les différents outils nécessaires à sa programmation. Le SDK proposé par le constructeur n'étant disponible que sur Windows, nous avons du trouver une version exploitable sous système UNIX. Fort heureusement, un projet lancé à l'initiative de Brad Davis nous permet de manipuler tous les capteurs du boitier de l'Oculus Rift. Il contient toutes les bibliothèques et tous les fichiers requis à la manipulation et la programmation du masque de réalité virtuelle. De ce fait, nous avons pu récupérer les valeurs des accéléromètres, indispensables pour la suite de notre projet dans un terminal.

```
Manufacturer:'Oculus VR, Inc.'
Product:'Tracker DK'
Serial#:'L94GH3TGQG37'
[From Service] [TrackingManager]
Broadcasting new HMD count = 1
Current orientation - roll 0.00,
pitch 0.00, yaw -0.00
Current orientation - roll 8.15,
pitch -21.77, yaw -2.99
Current orientation - roll 3.78,
pitch -21.68, yaw -1.30
Current orientation - roll 8.99,
pitch -24.24, yaw -9.25
Current orientation - roll 9.90,
pitch -21.64, yaw -5.71
Current orientation - roll 10.81,
pitch -21.71, yaw -5.54
```

Les mesures des différents capteurs du boitier étant exploitables, il faudrait maintenant réussir à les utiliser dans notre projet. Notamment envoyer ces valeurs vers le système de caméras, et construire des positions à partir de la lecture courante des capteurs.

3- Communication

Nous avons eu besoin de définir un moyen de communication entre l'Oculus et le système embarqué sur le drone. Pour cela, nous avons modifié le programme codé en C++ permettant de récupérer les valeurs des accéléromètres liés à l'Oculus.
Ce programme en plus de récupérer les valeurs, écrit désormais les valeurs récupérées dans un fichier. En effet, nous nous sommes inspirés d'un pont réseau développé en C pendant les cours de cryptographie.

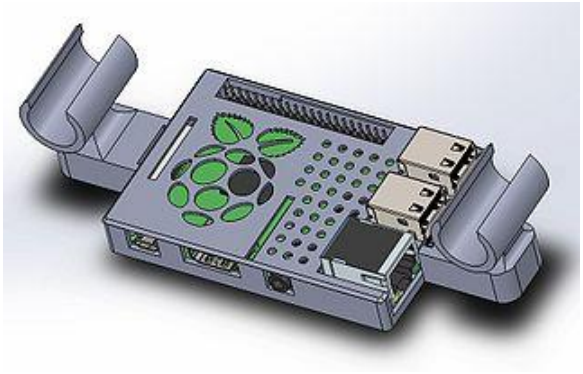


Grâce à ce fichier (val.data) contenant les valeurs des accéléromètres, n'importe quel type de programme capable de lire un fichier peut récupérer ces valeurs. De cette manière, nous avons pu coder en C un programme qui lit une première fois ces valeurs, les enregistre en tant que "valeurs d'origine" puis envoie sur le réseau la différence entre ces valeurs et les nouvelles valeurs lues. Cela permet au démarrage du programme de définir une position d'origine et de traiter tous les déplacements à partir de cette position.
Un programme sur le RaspberryPi récupère les valeurs envoyées sur le réseau, puis effectue un seuillage et une conversion des valeurs afin de commander les servomoteurs en conséquence.

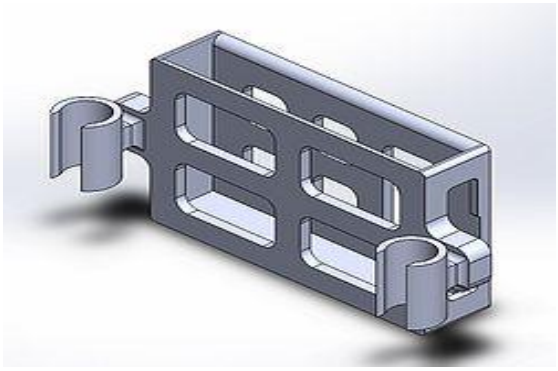
4- Support Plastique

Pour pouvoir embarquer nos modules sur le drone, il a fallu créer une structure capable d'embarquer tout notre équipement. Au début, nous avons conçu une nacelle à fixer sur les attaches d'une caméra native au drone. Cette nacelle a été pensée de manière à ce qu'elle soit la plus légère possible, tout en étant capable de contenir l'électronique, ainsi que de supporter le système de caméra. De plus, ce système avait pour avantage de centrer parfaitement la masse et garantir une certaine stabilité en vol, puisqu'elle se situait en dessous du drone (Prototype de nacelle disponible en annexe).

Plus tard, nous nous sommes rendu compte que cette solution n'était pas envisageable, car deux fixations de la précédente caméra étaient indémontables. De ce fait, nous avons donc repensé la structure en optant pour plusieurs compartiments : un pour la batterie et un pour le RaspberryPi.



Support RaspberryPi



Support Batterie externe

Chaque fixation est conçue de telle façon à ce qu'elle s'accroche facilement au niveau des pieds du drone. Le système de caméra, quant à lui, est fixé à l'avant. Voici une vue globale du système :

On notera que cette solution de fixation est beaucoup contraignante pour le drone car les supports demandent à être plus robustes (donc plus lourd). De plus, les charges étant placées de chaque côté du drone, un problème de centrage est à prévoir. On compte sur les systèmes gyroscopiques du drone pour compenser ce déséquilibre.

III- Impression sur le projet

1- Difficultés rencontrées

Nous avons, pendant ce projet, rencontré plusieurs difficultés. Notamment en choisissant le langage C comme langage de base pour notre système. En effet, la majorité des points évoqués et détaillés précédemment ont été développés à partir de documentation disponible sur internet. Les seuls exemples disponibles ne l'étaient que en langage de plus haut niveau (Python, JavaScript). Néanmoins, ce choix se justifie de part notre capacité à utiliser le langage depuis quelques temps, il est plus proche de l'électronique et nous permet d'écrire beaucoup plus proprement.

Une seconde difficulté a été de savoir gérer son temps pendant le projet. Beaucoup de modules évoqués lors de la description du projet, ont été, soit surévalués, soit sous-évalués (en temps). Cela a fortement impacté son déroulement. En effet, nous avions prévu de réaliser des mesures sur chaque module, que ce soit en terme de consommation électrique ou en terme de poids.

Le travail d'implémentation des modules a été une partie éprouvante. Quelques modules fonctionnaient sous différentes OS. Leur union a engendré quelques bogues.

Pour finir, l'utilisation et la programmation des Oculus Rift fut très compliqué à mettre en place. Le nombre de développeurs étant déjà limité sur ce type d'appareil, il a fallu redoubler d'effort pour trouver un environnement utilisable et compréhensible pour des développeurs novices.

2- Résultats

Nous finissons le projet sur un bilan globalement positif, puisque nous répondons en grande partie à la problématique de départ. Le gros travail de ce projet, mis à part trouver une solution pour chacun de nos modules, est un travail d' « implémentation ». Nous avons essayé au mieux, de développer des modules avec des conceptions simples et indépendantes leur permettant d'évoluer de leur côté. Même si certains modules requièrent des améliorations, nous récupérons bien à partir d'un drone une image en 3D stéréoscopique envoyée sur un serveur web. Nous arrivons à retranscrire les mouvements des lunettes sur notre système de caméras en envoyant des données depuis un couple serveur/client. Enfin, nous avons réussi à observer le rendu 3D en visualisant notre page web par le biais des Oculus Rift.

3- Améliorations

De nombreuses pistes d'amélioration sont possibles pour notre projet. Elles se situent en grande partie sur notre wiki. Dans un premier temps, nous pourrions remplacer motion par un programme (en C) réalisant les mêmes actions. Nous pourrions alors contrôler plus efficacement les ressources utilisées par le programme et limiter son fonctionnement au strict minimum.

Un deuxième point d'amélioration serait de réécrire le serveur et le client en C++ pour s'aligner avec l'environnement de programmation de l'Oculus Rift. On pourrait aussi améliorer la qualité de la vidéo diffusée dans les lunettes. En effet, si un groupe décide d'utiliser les versions plus récentes des Oculus Rift ou un autre fournisseur, il est tout à fait envisageable d'utiliser des caméras beaucoup plus performantes. Il serait alors nécessaire d'utiliser soit la dernière version du RaspberryPi (plus performante) ou une autre carte spécialisée dans le traitement vidéo.

Pour l'amélioration de la commande des servomoteurs, on pourrait s'affranchir de l'utilisation de WiringPi en utilisant la librairie bcm2835. C'est une libraire qui permet de contrôler les pins du RaspberryPi directement au niveau du processeur. C'est une librairie développée en C. Attention, elle a été développée sur les premiers modèles de RaspberryPi. Cela impliquerait quelques changements, principalement au niveau des dénominations des broches.

Nous pourrions aussi donner une structure plastique à notre système de caméras. Imprimée en 3D, elle aurait eu le même type de fixation que les deux autres supports. On pourrait gagner quelques grammes. Mais ce changement est principalement pratique et esthétique. Une modification pour le support du RaspberryPi est aussi à prévoir. En effet la fixation côté prise USB réduit le nombre de périphériques utilisables. Il avait été trouvé comme solution d'utiliser un Dongle wifi beaucoup plus petit que l'original.

Un dernier point d'amélioration serait de faire évoluer la communication entre le bloc Pc/Oculus Rift et le système de caméras. Nous utilisions un câble Ethernet pour simuler un réseau en local. Mais il est tout à fait possible et envisageable de passer par d'autres technologies sans fil, comme la radio télécommunication, ou encore le wifi, en créant un programme capable d'encapsuler les valeurs des accéléromètres, et les restituant sur un réseau avec une interface configurée pour récupérer ces valeurs.

Conclusion

Pour conclure, nous avons réalisé, dans le cadre d'un projet de 4ème année à Polytech Lille, un système de caméra stéréoscopique, capable de restituer des images et des mouvements à partir de lunettes de réalité virtuelle.

Durant ces 12 semaines, nous avons pu mettre en pratique nos connaissances théoriques et en développer certaines par le biais des technologies auxquelles nous avons été confrontés. Nous avons également du apprendre à gérer un projet dans son ensemble.

De nombreuses améliorations sont à prévoir pour optimiser au maximum les ressources de notre système embarqué. Néanmoins, nous sommes globalement satisfaits de ce projet de part les nombreux aspects qu'il nous a fait découvrir.

Annexes

Matériel :

Mini caméra *2 alimentation via le port USB
- Résolution 800*600 pixels
- Capteur CMOS
- Dimensions 38*15*35 mm
- Longueur câble 60 cm
- 12g

Chargeur portable Power Bak
- Puissance d'entrée 500 mA @ +5V
- Courant de sortie max 2,1 A @ +5V
- Capacité 5000 mAh
- Dimensions 102*66*15,8 mm
- Poids 154g

SG90 micro servo *2
- Poids 9g
- Dimensions 22.2*11.8*31 mm
- Couple 1,8 Kgf.cm
- Tension d'alimentation 4,8V
- Vitesse de fonctionnement 0,1s/60 degrés

Différents types de câbles Arduino.

Code : La majorité des codes seront disponibles directement sur le wiki dans une rubrique spécialisée. Ici seront mis les codes les plus pertinents et les fichiers d'inclusion.

Code contrôlant les servomoteurs sur RaspberryPi :

ctrl_servo.c

```c
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#include <softPwm.h>
#include <time.h>

#define PIN_0 1
#define PIN_1 0


void viderbuffer(){
        int c = 0;
        while(c!= '\n' && c != EOF){
                c = getchar();
        }
}

void aller_0(int nb){
  softPwmWrite(PIN_0,nb);
  delay(400);
}

void aller_1(int nb){
  softPwmWrite(PIN_1,nb);
  delay(400);
}

void aller_a(int nb1, int nb2){
  aller_0(nb1);
  aller_1(nb2);
}

int main(int argc, char *argv[]){
  int pos = 0;

  if(wiringPiSetup() == -1){
                printf("Impossible d'ouvrir wiringPi\n");
                exit(1);
        }
        //init PIN_0
        pinMode(PIN_0,OUTPUT);
        digitalWrite(PIN_0,LOW);
        softPwmCreate(PIN_0,0,500);

        //init PIN_1
        pinMode(PIN_1,OUTPUT);
        digitalWrite(PIN_1,LOW);
  softPwmCreate(PIN_1,0,500);

      int cmd_x, cmd_y;

  int tab1[13]={10,11,12,13,14,15,16,17,18,19,20,21,22};
  int tab2[13]={10,11,12,13,14,15,16,17,18,19,20,21,22};

  while(1){
    delay(1000);

            if(val_x > 50){
                  cmd_x = 12;
            }
            if(val_x < -50){
                  cmd_x = 0;
            }
            if(val_x > -50 && val_x < 50){
                  cmd_x = ((val_x +50)*12)/100;
            }

            if(val_y > 50){
                  cmd_y = 12;
            }
            if(val_y < -50){
                  cmd_y = 0;
            }
            if(val_y > -50 && val_y < 50){
                  cmd_y = ((val_y +50)*12)/100;
            }

            aller_a(tab1[cmd_x],tab2[cmd_y]);
  }
  return 0;
}
```

Line 79, Column 4 Spaces: 2 C

Fichier récupérant les valeurs des accéléromètres dans un fichier temporaire et envoyant ces dernières au Raspberry Pi :

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <poll.h>
#include <signal.h>
#include "libnet.h"
#include <time.h>

/** Variables globales */

float val_x, val_y, val_z;
float val_x_init, val_y_init, val_z_init;
int run = 1;

sigset_t ens1, ens2;
int sig;

/** Fonctions **/

/* Fonction principale */
int main(int argc,char *argv[])
{
    // Analyse des arguments
    if(argc!=3)
    {
        fprintf(stderr,"Syntaxe : client <serveur> <port>\n");
        exit(-1);
    }
    char *serveur=argv[1];
    char *port=argv[2];

    #ifdef DEBUG
    fprintf(stdout,"Pont sur %s port %s\n",serveur,port);
    #endif

    // Connexion au serveur
    printf("Connexion au serveur... ");
    int s= connexionServeur(serveur,port);
    if (s<0){perror("connexionServeur.socket");
    exit(EXIT_FAILURE);}

    // Communication avec le serveur
    FILE *dialogue=fdopen(s,"a+");
    FILE* fd = NULL;
    if(dialogue==NULL){ perror("gestionClient.fdopen"); exit(EXIT_FAILURE); }
    printf("OK\n");

    sigemptyset(&ens1);
    sigaddset(&ens1, SIGINT);
    sigaddset(&ens1, SIGQUIT);

    sigprocmask(SIG_SETMASK, &ens1, NULL);


    printf("Initialisation des valeurs de l'Oculus... ");
    fd = fopen("val.data","r");

    if(fd == NULL)
    {
        fclose(dialogue);
        shutdown(s,SHUT_RDWR);
        perror("Impossible d'ouvrir le fichier val.data pour l'Oculus\n");
        exit(EXIT_FAILURE);
    }

    fscanf(fd,"%f %f %f",&val_x_init, &val_y_init, &val_z_init);

    #ifdef DEBUG
    fprintf(stdout,"Lecture Fichier (BRUT) : %0.2f %0.2f %0.2f\n",val_x_init, val_y_init, val_z_init);
    #endif

    fclose(fd);
    usleep(200000);

    printf("OK\n");

    printf("Transmission des valeurs, CTRL_C pour arrêter\n");
    while(run)
    {
        sigpending(&ens2);
        if(sigismember(&ens2,SIGINT) || sigismember(&ens2,SIGQUIT))
        {
            val_x = 404.404;
            val_y = 404.404;
            val_z = 404.404;

```

```
            write(s,&val_x,sizeof(val_x));
            write(s,&val_y,sizeof(val_y));
            write(s,&val_z,sizeof(val_z));
            #ifdef DEBUG
            fprintf(stdout,"(W) %0.3f %0.3f %0.3f\n",val_x, val_y, val_z);
            #endif
            run = 0;
        }
        else
        {
            fd = NULL;
            fd = fopen("val.data","r");

            if(fd == NULL)
            {
                fclose(dialogue);
                shutdown(s,SHUT_RDWR);
                perror("Impossible d'ouvrir le fichier val.data pour l'Oculus\n");
                exit(EXIT_FAILURE);
            }

            fscanf(fd,"%f %f %f",&val_x, &val_y, &val_z);
            #ifdef DEBUG
            fprintf(stdout,"Lecture Fichier : %0.2f %0.2f %0.2f\n",val_x, val_y, val_z);
            #endif

            val_x = val_x_init - val_x;
            val_y = val_y_init - val_y;
            val_z = val_z_init - val_z;

            write(s,&val_x,sizeof(val_x));
            write(s,&val_y,sizeof(val_y));
            write(s,&val_z,sizeof(val_z));

            #ifdef DEBUG
            fprintf(stdout,"(W) %0.2f %0.2f %0.2f\n",val_x, val_y, val_z);
            #endif

            fclose(fd);
            usleep(200000);
        }
    }

    sigemptyset(&ens1);
    sigprocmask(SIG_SETMASK, &ens1, NULL);

    fclose(dialogue);

    /* On termine la connexion */
    shutdown(s,SHUT_RDWR);

    return 0;
}

Line 1, Column 1                                    Tab Size: 4    C
```

Fichier d'inclusion du serveur et du client :

libnet.c | libnet.h | Makefile | virtual_bridge.c

```
/** fichier libnet.h **/

/*************************************************************************/
/** Ce fichier decrit les structures et les constantes utilisees **/
/** par les fonctions reseau.                                   **/
/*************************************************************************/

/**** Constantes ****/

/** Nombre maximum de connexions tamponnees pour le serveur **/

#define MAX_CONNEXIONS  32
#define MAX_TAMPON 1600
/**** Fonctions ****/

void socketVersClient(int s,char **hote,char **service);
int connexionServeur(char *hote,char *service);
int initialisationServeur(char *service,int connexions);
int boucleServeur(int ecoute,int (*traitement)(int));
int read_fixed(int descripteur,unsigned char *array,int size);
int creationInterfaceVirtuelle(char *nom);


```

Ancienne nacelle :