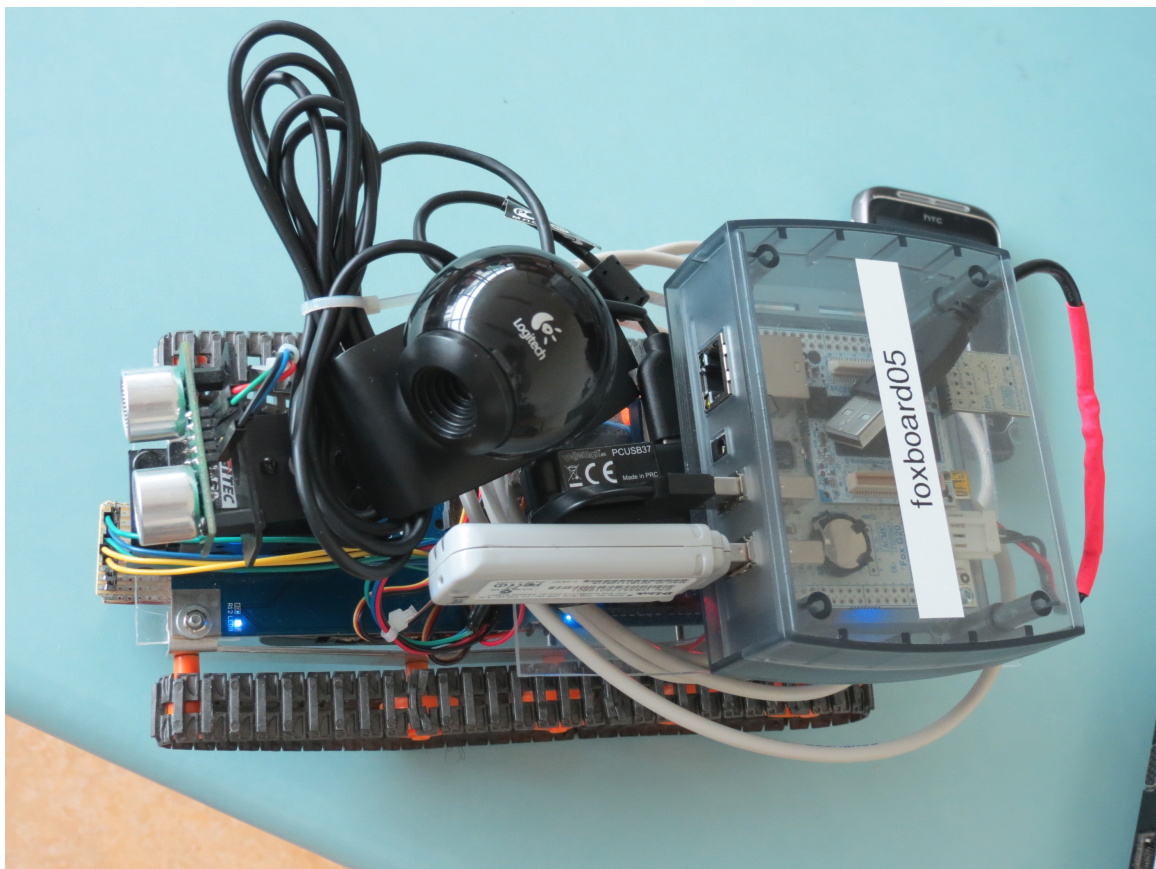


ROBOTS MOBILES



Élèves : FEI Juntao

LIU Zhen

Tuteur : Xavier REDON

2012—2013

INTRODUCTION.....	3
MATÉRIEL ET OBJECTIFS.....	4
OBJECTIFS.....	4
MATÉRIEL ET MONTAGE.....	5
AVANCEMENT DU PROJET.....	7
PARTIE D' ARDUINO.....	7
<i>Contrôlé par le clavier.....</i>	7
<i>Contrôlé par le sonar.....</i>	7
<i>Contrôlé par le capteur couleur.....</i>	7
PARTIE DE L'INTERFACE DE COMMANDE.....	7
<i>Structure du système.....</i>	7
<i>Avant de faire la configuration, il faut d'abord constituer le rapport la-dessous.....</i>	8
<i>Installation du système.....</i>	8
<i>Connexion des périphériques.....</i>	8
<i>Accès au serveur Web de la FoxBoard.....</i>	9
<i>Configuration de la FoxBoard.....</i>	9
<i>Programmer le robot MindStorm.....</i>	9
<i>Adapter l'interface Web.....</i>	10
<i>Communication inter-FoxBoard.....</i>	10
<i>Arrêter la FoxBoard.....</i>	10
DIFFICULTÉS RENCONTRÉS.....	10
FONCTIONNEMENT ET RÉSULTAT DE NOTRE ROBOT.....	10
MODE WIFI.....	11
MODE AUTOMATIQUE.....	11
MODE SUIVANT UNE LIGNE.....	11
MODE À CHOISIR PAR L'INTERFACE PAGE WEB.....	11
CONCEPTION DES PROGRAMMES.....	11
PARTIE DU CHÂSSIS.....	11
<i>Partie la librairie et les configurations des interfaces.....</i>	11
<i>Partie des sous programmes.....</i>	14
<i>Partie du programme principal.....</i>	15
PARTIE DE PAGE WEB.....	17
<i>Partie de la foxboard.....</i>	17
La liaison série.....	18
La mode serveur et la mode client.....	19
Structure à définir pour les deux modes.....	19
CONCLUSION.....	21

Introduction

Pendant le projet de robotmobile, nous avons progressé le châssis Arduino et le châssis Phidget. Le but du projet est de concevoir trois modes de fonctionnement pour les deux châssis. La première mode est que l'utilisateur peut commander les châssis par toucher l'écran tactile en connexion de wifi qui est émis par la foxboard. En plus, dans la deuxième mode, les châssis déplacent en évitant les obstacles de toutes les sens grâce à un capteur d'obstacle SRF05. Par contre, la troisième mode permet aux deux châssis de bouger suivant une ligne en couleur grâce à un capteur de couleur CR999.

La mode de fonctionnement peut être changée par l'utilisateur sur l'interface page web.

Mots clés : Arduino, Wifi, Phidget, Foxboard, Interface page web

Matériel et objectifs

Objectifs

Le but est de concevoir un robot capable de se déplacer dans un environnement ayant des obstacles. Le robot est contrôlé par utilisateur. Le robot doit pouvoir être contrôlé idéalement à distance par une interface page web, sur laquelle l'environnement est visualisé grâce à une webcam.

La mission se compose de quatre parties:

Création de mode de fonctionnement Wifi

L'utilisateur peut commander les châssis par toucher l'écran tactile en connexion de wifi qui est émis par la foxboard. La command est transmis par l'interface page web, ensuite la foxboard la reçoit et finalement elle est transmise vers un châssis.

Création de mode de fonctionnement automatique

Dans la deuxième mode, les châssis déplacent en évitant les obstacles de tous les sens grâce à un capteur d'obstacle SRF05. Une fois le châssis est animé, il se déplace dépendant de la valeur mesuré par le capteur sonar. Le servomoteur HS-422 tourne pour bien positionner le sens de capteur, puisque le capteur est mis sur le servomoteur.

Création de mode de fonctionnement suivant une ligne en couleur

Elle permet aux deux châssis de bouger suivant une ligne en couleur grâce à un capteur de couleur CR999. Le capteur de couleur nous passe les valeurs R, G, B et C, que nous devons utiliser pour faire bouger les châssis. Nous attachons une grande importance à la précision des valeurs que le capteur avait mesurées. Le quatrième paramètre de luminosité sert à compenser l'influence de la lumière.

Integration les trois modes

La mode de fonctionnement peut être changée par l'utilisateur sur l'interface page web. L'utilisateur peut toucher l'interface graphique sur page web pour choisir et changer la mode fonctionnement.

Matériel et montage

Châssis DFRobots Rover V1.5

Arduino ATMega328P

Moteurs Tamiya L293

Servomoteur HS-422

Sonar SRF-05

webcam

Bus I2C (Inter Integrated Circuit)

Capteur de couleurs ADJD-S371

Foxboards

Clé wifi

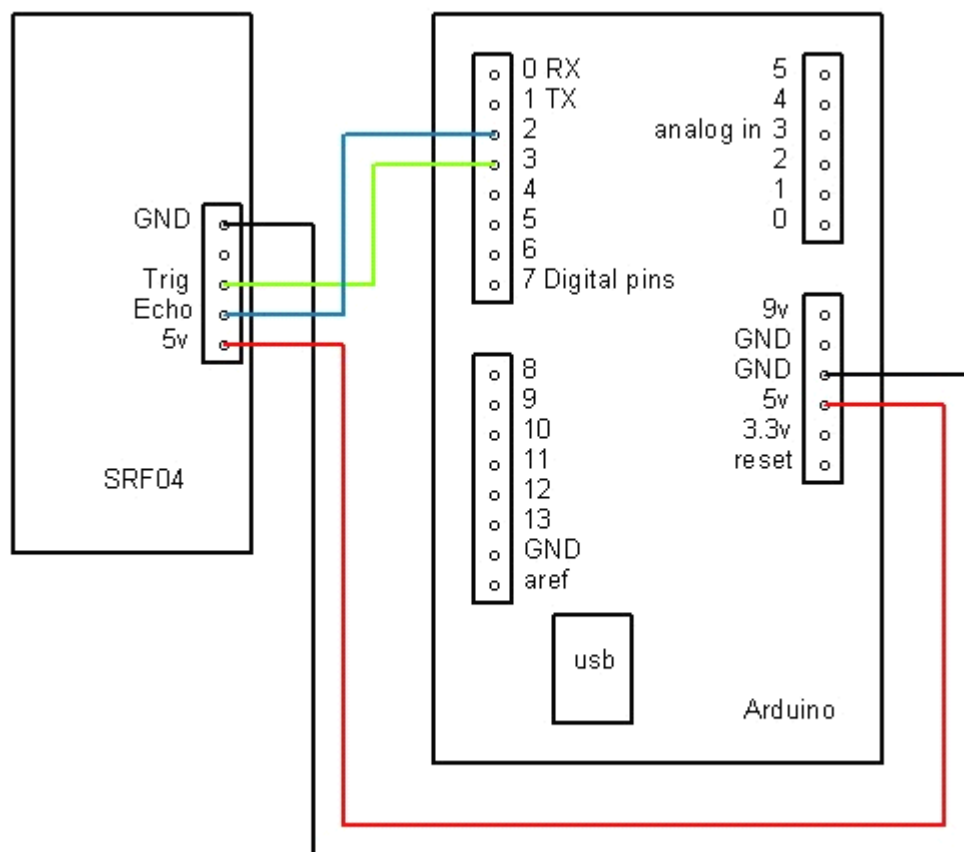
Second châssis fonctionnant par Phidgets

Câbles variés

capteur de couleur ADJD S371 CR999.

Batterie (4 piles) et rechargeur

L'écran tactile muni de fonction de wifi (Ipad ou le téléphone portable)



Montage du sonar SRF05

Avancement du projet

Partie d' Arduino

Cette partie sert à contrôler le châssis par l'ordinateur. Il peut être séparé par trois étapes.

Contrôlé par le clavier

Premièrement, nous avons contrôlé le châssis par le clavier. Il peut être utilisé directement dans la partie l'interface de commande.

Contrôlé par le sonar

Deuxièmement, nous avons réalisé l'étape "Marche automatique". Il y a deux parties dans cette étape. Au début, nous avons configuré les interfaces de sonar et Servomoteur HS-422 pour scanner et trouver le chemin sans obstacles. Le sonar va mesurer et imprimer la distance entre l'obstacle et le châssis. Si la distance est très petite, le châssis va être arrêté et Servomoteur va tourner le sonar en 180°(depuis 90° à 180° et depuis 90° à 0°). Si le sonar a trouvé le direction où il n'y a pas d'obstacle, il va retourner un valeur(pour décider si le châssis doit tourner à gauche ou à droite). À la fin, nous avons lié la partie scanner et la partie châssis. Donc, le châssis peut recevoir le signal du sonar pour éviter les obstacles.

En fait, nous avons mis le sonar d'un certain hauteur par rapport à la terre, donc il n'a pas pu bien détecter des petits obstacles.

Contrôlé par le capteur couleur

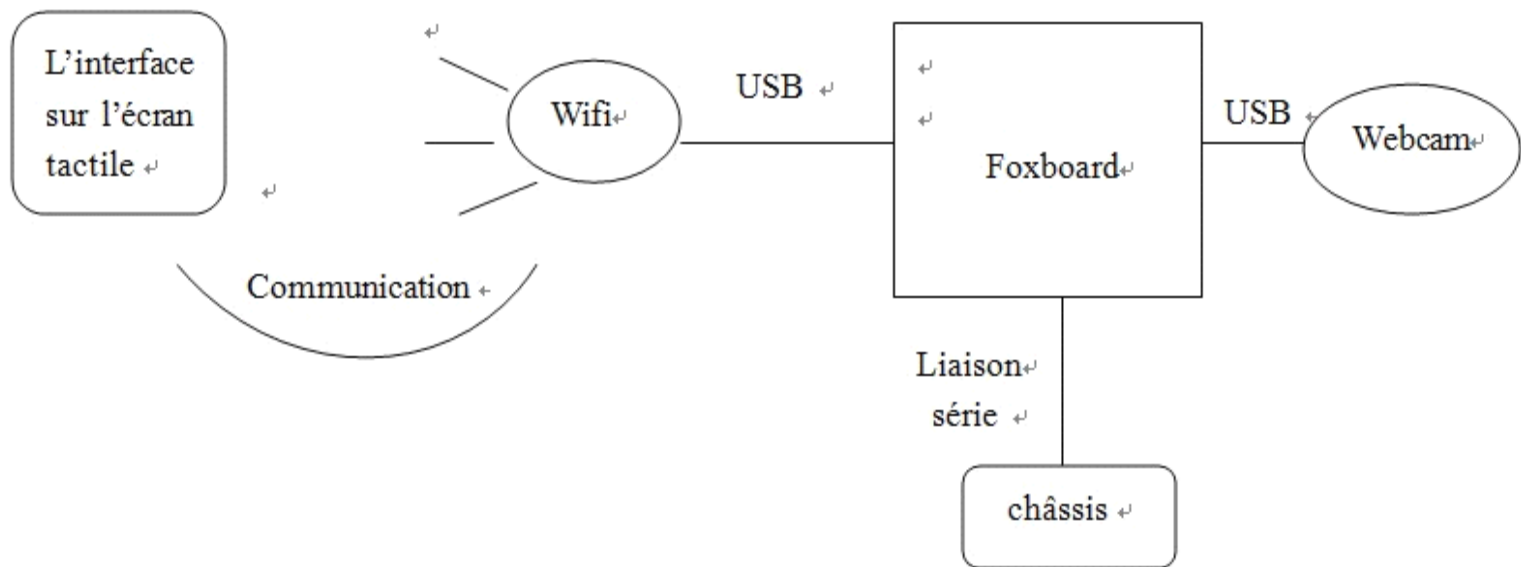
Troisièmement, nous avons utilisé le capteur couleur pour réaliser notre but : suivant une ligne. Dans cette partie, le capteur couleur va envoyer quatre valeurs (R, G, B et C) pour exprimer la couleur du chemin sous le châssis ('C' est mis pour éviter l'effet de la lumière). Dans notre cas, le châssis va marcher suivant une ligne verte et la terre est jaune. Grâce à cela, le châssis va recevoir le message pour assurer qu'il marche sur la ligne. Si le capteur ne trouve pas la ligne verte, le servomoteur bouge de gauche à droite en détectant la ligne jusqu'à il peut trouver la ligne.

Partie de l'interface de commande

Cette partie sert à relier la communication entre les pages web et le châssis via la foxboard.

Structure du système

Avant de faire la configuration, il faut d'abord constituer le rapport la-dessous



Structure du système communicant

Installation du système

Pour commencer il faut installer le système sur la micro-SD servant de disque à la FoxBoard. Commencer par récupérer l'image de ce système. Insérer la carte micro-SD dans l'adaptateur USB. Repérer le nom du périphérique USB en tapant la commande `df` (par exemple `/dev/sdb`). Avant de copier l'image sur votre micro-SD démonter les montages indiqués par la commande `df` et commençant par `/media` (par exemple `/media/kernel`). Pour démonter il faut taper des commandes sur le modèle suivant.

```
$ umount /media/kernel
```


Alors copier l'image sur votre micro-SD.

```
$ dd if=foxlego.img of=/dev/sdb
```

Insérer la micro-SD dans la FoxBoard, maintenant c'est prêt à la démarrer. Pour alimenter la FoxBoard utiliser les câbles USB et brancher sur l'ordinateur fixe.

Connexion des périphériques

Deux périphériques USB doivent être connectés à la FoxBoard; une interface réseau WiFi et une caméra USB

Accès au serveur Web de la FoxBoard

La FoxBoard est configurée avec une adresse IP sur l'interface Ethernet de 192.168.100.1. Pour la première configuration, connecter le port Ethernet de la FoxBoard au port Ethernet de la machine Linux via un câble croisé. Configurer la machine Linux pour configurer sa carte Ethernet de façon automatique (protocole DHCP). Maintenant contacter la FoxBoard en filaire. Taper simplement l'URL <http://192.168.100.1/> dans un navigateur en ayant pris soin de retirer le mandataire Web. Alors configurer l'accès WiFi à votre FoxBoard, préciser un SSID original et un canal radio de façon à éviter les conflits. Vérifier que vous pouvez vous connecter sur votre FoxBoard en WiFi.

Configuration de la FoxBoard

Il faut maintenant adapter la FoxBoard à un robot MindStorm. Sur le site Web de la FoxBoard, choisir l'onglet *Configuration*. Trouvez le nom de Wifi.

Programmer le robot MindStorm

Nous allons maintenant pouvoir contrôler le robot par l'interface Web de la FoxBoard (à moyenne distance en WiFi). Nous pouvons l'onglet *Contrôle* pour cela. Les instructions sont envoyés dans la boîte numéro 3 de la brique MindStorm et la brique doit envoyer une réponse dans la boîte numéro 4. Avec le sous-onglet "basic", l'interface Web fournie transmet le code 1 pour la flèche vers le haut, le code 2 pour la flèche vers le bas, le code 3 pour tourner dans un sens, le code 4 pour tourner en sens inverse et le code 0 pour le panneau de stop. Avec le sous-onglet "total" l'interface Web va envoyer le code 5 dans la boîte numéro 3 en ayant envoyé auparavant les valeurs pour les moteurs dans les boîtes numéro 5 et 6. L'onglet "lignes" permet de mettre le robot en mode suivi de ligne et l'onglet "positions" affiche une carte et doit permettre de positionner les robots sur cette carte.

En plus, nous avons ajouté la nouvelle fonction pour choisir le mode de fonctionnement dans la rubrique "total".

Adapter l'interface Web

Il est possible d'ajouter de nouvelles pages Web sur le site de la FoxBoard en utilisant l'onglet *Téléchargement*. Nous pouvons nous inspirer des sources du site. Nous pouvons trouver les scripts utilisés dans ces pages dans cette archive

Communication inter-FoxBoard

A partir de là nous devons mettre au point un système de communication entre les FoxBoard. Le dit programme reçoit des informations en provenance de toutes les FoxBoard (lorsqu'elles sont connectée à un même réseau WiFi). C'est le script PHP `ordonne.php` qui permet d'envoyer ces informations, nous pouvons voir comment ce script est appelé par le site Web en lisant la page HTML `conduire.html` et en particulier le code JavaScript qu'elle contient. Pour vous aider, les sources des programmes déjà écrits pour la FoxBoard sont disponibles

Arrêter la FoxBoard

La FoxBoard doit toujours s'arrêter proprement via l'onglet *Configuration* de son site Web ou par la commande `halt` lorsque que vous êtes connectés dessus.

Difficultés rencontrés

Il faut exactement suivre les étapes de configuration. Une fois, la foxboard est lancée, nous ne pouvons pas arrêter par débrancher la ressource électrique, sinon il y a un risque de abîmer la carte.

Les messages reçus et envoyés doivent être de même type, nous expliquons dans la partie <<Conception des programmes>>.

Même si nous avons modifié notre programme, le fonctionnement en mode capteur couleur dépend beaucoup de la tension de batterie. L'instabilité est causée par la régulation entre la vitesse et le sens. En fait, si nous ajoutions une boussole pour retourner le degré de rotation, le fonctionnement serait mieux.

Fonctionnement et résultat de notre robot

Il faut lancer la foxboard, ensuite nous pouvons uniquement utiliser l'interface page web pour contrôler le châssis

Mode wifi

Sur la rubrique Contrôle, les flèches représentent les sens. Stop sert à arrêter le châssis

Mode automatique

Déplacement du châssis bien évitant un ou plusieurs obstacles devants, à droite ou à gauche

Mode suivant une ligne

Déplacement exactement sur la ligne collée par terre.

Mode à choisir par l'interface page web

La mode de fonctionnement peut être changée par l'utilisateur sur l'interface page web. L'utilisateur peut toucher l'interface graphique sur page web pour choisir et changer la mode fonctionnement.

Conception des programmes

Partie du châssis

Il y a trois modes du fonctionnement de notre châssis. Et nous avons les bien organisés dans un seule programme. Et nous allons expliquer dans l'ordre: la librairie et les configurations des interfaces, les sous programmes et le programme principal.

Partie la librairie et les configurations des interfaces

//pour la partie librairie

```
#include <Wire.h>
```

```
#include<Servo.h>
```

//pour la partie moteur

int E1 = 6; //M1 Speed Control

int E2 = 5; //M2 Speed Control

int M1 = 8; //M1 Direction Control

int M2 = 7; //M2 Direction Control

//pour la partie sonar

const int numOfReadings = 10; // number of readings to take/ items in the array

int readings[numOfReadings]; // stores the distance readings in an array

int arrayIndex = 0; // arrayIndex of the current item in the array

int total = 0; // stores the cumulative total

int averageDistance = 0; // stores the average value

// setup pins and variables for SRF05 sonar device

int echoPin = 2; // SRF05 echo pin (digital 2)

int initPin = 3; // SRF05 trigger pin (digital 3)

unsigned long pulseTime = 0; // stores the pulse in Micro Seconds

unsigned long distance = 0; // variable for storin

//define servo stuff

#define SERVOPIN 9 //digital pin 9

#define SERVOMIN 0 //minimum angle 0 degrees

#define SERVOMAX 45 //maximum angle 45 degrees

//define rotation sensor stuff

#define ROTATIONPIN 0 //analog pin 0

// ADJD-S311's I2C address, don't change

#define ADJD_S311_ADDRESS 0x74

#define RED 0

```
#define GREEN 1
```

```
#define BLUE 2
```

```
#define CLEAR 3
```

// ADJD-S311's register list

```
#define CTRL 0x00
```

```
#define CONFIG 0x01
```

```
#define CAP_RED 0x06
```

```
#define CAP_GREEN 0x07
```

```
#define CAP_BLUE 0x08
```

```
#define CAP_CLEAR 0x09
```

```
#define INT_RED_LO 0xA
```

```
#define INT_RED_HI 0xB
```

```
#define INT_GREEN_LO 0xC
```

```
#define INT_GREEN_HI 0xD
```

```
#define INT_BLUE_LO 0xE
```

```
#define INT_BLUE_HI 0xF
```

```
#define INT_CLEAR_LO 0x10
```

```
#define INT_CLEAR_HI 0x11
```

```
#define DATA_RED_LO 0x40
```

```
#define DATA_RED_HI 0x41
```

```
#define DATA_GREEN_LO 0x42
```

```
#define DATA_GREEN_HI 0x43
```

```
#define DATA_BLUE_LO 0x44
```

```
#define DATA_BLUE_HI 0x45
```

```
#define DATA_CLEAR_LO 0x46
```

```
#define DATA_CLEAR_HI 0x47
```

```
#define OFFSET_RED 0x48
```

```
#define OFFSET_GREEN 0x49
```

```
#define OFFSET_BLUE 0x4A
```

```
#define OFFSET_CLEAR 0x4B
```

// Pin definitions:

```
int sdaPin = A4; // serial data, hardwired, can't change
```

```
int sclPin = A5; // serial clock, hardwired, can't change
```

```
int ledPin = 4; // LED light source pin, any unused pin will work
```

// RGB LED pins, should all be PWM output pins:

```
int redledPin = 9;
```

```
int greenledPin = 10;
```

```
int blueledPin = 11;
```

```
int rgbPins[3] = {redledPin, greenledPin, blueledPin};
```

// initial values for integration time registers

```
unsigned char colorCap[4] = {9, 9, 2, 5}; // values must be between 0 and 15
```

```
unsigned int colorInt[4] = {2048, 2048, 2048, 2048}; // max value for these is 4095
```

```
unsigned int colorData[4]; // This is where we store the RGB and C data values
```

```
signed char colorOffset[4]; // Stores RGB and C offset values
```

//Les variables globales

```
Servo myServo;
```

```
int pos = 0;
```

```
int com=0;
```

```
char val = '6';
```

```
int r=0;
```

```
int g=0;
```

```
int b=0;
```

```
int c=0;
```

```
int t1=0;
```

Partie des sous programmes

En première temps, nous avons créé cinq sous programmes pour réaliser que le

châssis peut conduire normalement.(a et b sont les vitesses des moteurs à gauche et à droite)

```
void stop(void) //Arrêter  
  
void forward(char a,char b) //Avancer  
  
void reverse (char a,char b) //Reculer  
  
void right (char a,char b) //À droit  
  
void left (char a,char b) //À gauche
```

Ensuite, nous avons écrite les sous programmes pour mesurer la distance et scanner pour éviter les obstacles.

```
void sonar_dist(void) //pour mesurer la distance  
  
char scan(void) //pour scanner
```

Et puis, ils sont les sous programmes pour le capteur couleur donc il peut bien exprimer la couleur.

```
void initADJD_S311() /* initADJD_S311() - This function initializes the ADJD-S311 and its  
capacitor and integration registers The vaules for those registers are defined near the top of the code.  
the colorCap[] array defines all capacitor values, colorInt[] defines all integration values.*/  
  
void printADJD_S311Values()/* printADJD_S311Values() reads, formats, and prints all important registers  
of the ADJD-S311. It doesn't perform any measurements, so you'll need to call getRGBC() to print  
new values.*/
```

```
void performMeasurement()
```

```
void getRGBC()/* getRGBC() - This function reads all of the ADJD-S311's data registers and stores them into  
colorData[]. To get the most up-to-date data make sure you call performMeasurement() before calling this  
function.*/
```

À la fin, c'est les programmes pour réaliser les trois modes de fonctionnement.

```
void handcontrol()  
  
void suivligne()  
  
void autodirection()
```

Partie du programme principal

Dans Arduino, il y a deux programmes nécessaires. Ils sont *setup* et *loop*. Voici sont les programmes.

```
void setup()
{
    pinMode(ledPin, OUTPUT); // Set the sensor's LED as output
    digitalWrite(ledPin, HIGH); // Initially turn LED light source on

    for (int i=0; i<3; i++)
    { // Set up the RGB LED pins
        pinMode(rgbPins[i], OUTPUT);
        digitalWrite(rgbPins[i], LOW);
    }

    //setup moteur

    int i;

    for(i=5;i<=8;i++)

        pinMode(i, OUTPUT);

    myServo.attach(SERVOPIN);

    Serial.begin(9600);

    //setup color capture

    Wire.begin();

    delay(1); // Wait for ADJD reset sequence

    initADJD_S311(); // Initialize the ADJD-S311, sets up cap and int registers
}

void loop(void){
    while (Serial.available() < 1) {
        } // Wait until a character is received

    char val= Serial.read();

    Serial.println(val, DEC);
}
```



```
switch(val)
{
    case '6'://wifi_control

        handcontrol();

        break;

    case '7'://suivre une ligne

        suivligne();

        break;

    case '8'://marche automatiquement

        autodirection() ;

        break;

    case '0'://arret

        stop();

        break;

}
}
```

Partie de page web

Conduire.html sert à afficher les icônes sur la page web, dans la rubrique control, nous avons ajouté trois icônes pour choisir la mode fonctionnement, la flèche bleue sert à contrôler par toucher les icônes(Up, down, ringht, stop ou left). La flèche de droite rouge sert à passer en mode automatique et une autre sert à passer en mode suivant une ligne en couleur.

Ordonne.php sert à gérer la page web en ayant la commande de controle.c. Pour trouver Ordonne.php, faire cd /var/www. Pour l'ouvrir, fair vim ordonne.php

```
$sortie=shell_exec(COMMANDE.' '.$cmd".BOITE_ORDRE.BOITE_REPONSE);
```

La virable \$cmd est la commande à transmettre.

Si on essaye en mode serveur (ssh -l root 192.168.100.1):

```
read(5, "1 3 4\n", 4096)          = 6
```

```
write(3, "1", 1)                  = 1
```

'1' est un caractère qui est le message à communiquer.

Partie de la foxboard

Le fichier Controle.c se trouve dans FoxLego/Controle/ sert à offrir la commande, principalement il y a trois sous parties:

La liaison série

Les deux fonctions à écrire: envoyerInstruction(int fd,unsigned char instruction), recevoirReponse(int fd), où instruction est un char.

Avant de utiliser la liaison série dans le programme principal, il faut faire l'initialisation

```
int init_serial(char *device,int speed){  
    struct termios new;  
  
    int fd=open(device,O_RDWR|O_NOCTTY);  
  
    if(fd<0){perror(device); exit(-1);}  
  
    tcgetattr(fd,&saveterm); /* save current port settings */  
  
    bzero(&new,sizeof(new));  
  
    new.c_cflag=CLOCAL|CREAD|speed|CS8;  
  
    new.c_iflag=0;  
  
    new.c_oflag=0;  
  
    new.c_lflag=0; /* set input mode (non-canonical, no echo,...) */  
  
    new.c_cc[VTIME]=0; /* inter-character timer unused */  
  
    new.c_cc[VMIN]=1; /* blocking read until 1 char received */  
  
    tcflush(fd, TCIFLUSH);  
  
    tcsetattr(fd,TCSANOW,&new);  
}
```

```

        return fd;
    }

```

Et finale il faut arrêter la communication liaison serie

```

void close_serial(int fd)
{
    tcsetattr(fd,TCSANOW,&saveterm);

    close(fd);
}

```

La mode serveur et la mode client

Structure à définir pour les deux modes

```

static struct option long_options[]={

    {"server",0,0,'s'},

    {"client",0,0,'c'},

    {0,0,0,0}

};

```

En mode client, il faut recevoir le message de l'interface, dans notre cas, il y a **instruction** est importante :

```

// Mode Client

if(mode==MODE_CLIENT){

    char message[MAX_CHAINE];

    int ds=connexionServeur(CHEMIN_SOCKET);

    sprintf(message,"%c %c\n",instruction,answer==1?'1':'0');

    int taille=strlen(message);

    if(write(ds,message,taille)==taille){

        taille=read(ds,message,MAX_CHAINE);

        message[taille]='\0';

        if(taille>0) printf("%s\n",message);

    }

}

```

En mode serveur, la foxboard doit envoyer le message vers le châssis via la liaison série:

```
// Mode serveur

if(mode==MODE_SERVEUR){

    // Initialise la connexion liaison serie

    int fd=init_serial(device,SPEED);

#ifdef DEBUG

    //fprintf(stdout,"Liaison serie connected to %s \n", address);

#endif

    // Lecture des commandes et envoi des commandes liaison serie

    unlink(CHEMIN_SOCKET);

    initialisationServeur(CHEMIN_SOCKET,MAX_CONNEXIONS);

    //Enoyer le message

    if(fgets(message,MAX_CHAINE,flux)==NULL)

        { perror("main.fgets"); exit(-1); }

    if(sscanf(message,"%c %c",&instruction,&wait)!=2) break;

    envoyerInstruction(fd,instruction);
```

Conclusion

Grâce à l'aide et les conseils de Monsieur Xavier REDON, nous avons pu nous progresser pas à pas et finalement atteindre le but que nous avons fixé au début de projet. Nous le remercions.

Pendant ce projet, nous avons eu la chance de pratiquer la connaissance sur réseaux, système, les langage C, HTML/PHP, Arduino et Phidget, et aussi la compétence d'analyser le montage électronique et de résoudre les bugs dans les programmes. A part, nous croyons que la collaboration et la communication était beaucoup importante pour l'avancement et le réussite du projet.