/////////////////////////////

/// Code Autono'Car ///

/////////////////////////////////

// RP Lidar - A2-M8 - Slamtec //

//////////////////////////////////////////

/// Institut d'optique Graduate School ///

//////////////////////////////////////////

/////// PIMS 2019 - 2020 /////////////

///////////////////////////////////////////////////////////////////////////////

// Kévin Falque - Alexis Kedzia - Kassim Hanane - Hugo Affaticati /////////////

// Zhe Wang - Benoit Paris- Cissé Mahawa - Aurélie Lebrun - Thomas Serre /////

//////////////////////////////////////////////////////////////////////////////

////////////////////////////

////// Bibliothèques ///////

 //////////////////////////

#include "mbed.h"

#include "rplidar.h"

///////////////////

///// Brochage ////

///////////////////

Serial moduleBT(PC\_10,PC\_11); // connection émetteur/récepteur bluetooth

DigitalOut my\_led(LED1);

Serial my\_pc(USBTX,USBRX);// connection émetteur/récepteur pc

RPLidar my\_lidar; // Définition du LiDAR

PwmOut my\_lidar\_pwm(A3); // Brcohe de connexion

PwmOut servo(PA\_15); //Broche de connection du servomoteur

PwmOut moteur(PB\_7); //Broche de connection du motor driver contrôlant la vitesse du moteur

BufferedSerial my\_rp(A0, A1); // connection émetteur/récepteur LiDAR (TX/RX)

rplidar\_response\_device\_info\_t lidar\_info;

///////////////////////////////////

/////////// VARIABLES /////////////

///////////////////////////////////

// DIRECTION //

double sumDist;

double sumAngle;

const int taille\_liste = 10; // Nombre de tours sur lequel on moyenne la distance max

double maxDistance, angleAtMaxDist; // Distance max et angle associé à chaque tour

int compteur=0;

double mAng; // Moyenne de l'angle de la distance max après 'taille\_liste' tour

double mDist; // Moyenne de la distance max de la distance max après 'taille\_liste' tour

double zin[taille\_liste][2]; // Liste stockant les distances max des 'taille\_liste' tours

double angle\_roues; // Commande envoyée au servomoteur

const double k = 6;

int s;

char data = 's'; // consigne bluetooth

// CORRECTION //

///////////////////////////////////////////////////////////////////////

// Variables pour l'optimisation de la tejectoire que nous n'avons ///

// pas eu le temps de traiter en raison du confinement ///

///////////////////////////////////////////////////////////////////////

double angleD;

double correction;

const double d90 = 1250;

const double d0 = 130;

const double largeur = 18 ;

///////////////////////////////////

/////////// FONCTIONS /////////////

//////////////////////////////////

// FONCTION NORMALISATION ANGLES //

////////////////////////////////////////////////////////////////////////////////

// Fonction qui passe les angles entre 270° et 360° en angles entre -90° et 0°//

////////////////////////////////////////////////////////////////////////////////

double normalisation(double angle){

 if ( angle > 270){

 angle = angle - 360 ;

 }

 return angle ;

 }

// FONCTION TEST DISTANCE //

////////////////////////////////////////////////////////////////////////////////////////////////

// Fonction qui test la zone de sécurité (proportionnelle: non testé en raison du confinement //

////////////////////////////////////////////////////////////////////////////////////////////////

double test\_distance(double angle, double distance){

 if ( angleD < 0 ){

 if ( distance < (angle\*(d0 - d90)/90) + d0 ){

 return 1 ;

 }

 else{

 return 0;

 }

 }

 else{

 if ( distance < (angle\*(d90 - d0)/90) + d0 ){

 return 1 ;

 }

 else{

 return 0;

 }

 }

 }

/////////////////////////////////////

////////////////////////////////////

////////////// MAIN ////////////////

///////////////////////////////////

int main() {

 // Initialisation moteur et servomoteur

 moduleBT.baud(9600); //vitesse de transmission du capteur BT à la carte

 servo.period\_ms(20); //période de réception du servomoteur

 servo.pulsewidth\_us(1300); //première commande du servomoteur -> tout droit

 moteur.period\_ms(20); //période de réception du moteur

 moteur.pulsewidth\_us(1500); // 1500 -> point mort

 wait(2); // 2s imposée par la voiture

 int alp = 0; //constante qui prend la valeur 1 quand la voiture est en marche

 float vit; //vitesse du moteur

 // VITESSE DE ROTATION LIDAR

 my\_lidar\_pwm.period\_us(40);

 my\_lidar\_pwm.pulsewidth\_us(20);

 // TRANSMISSION PC

 my\_pc.baud(115200);

 // INITIALISATION LiDAR //

 my\_lidar.begin(my\_rp);

 my\_lidar.setAngle(0,360);

 my\_led = 1;

 // Initialisation du régulateur //

 for(vit = 1500; vit < 1510; vit+=1){

 moteur.pulsewidth\_us(vit);

 wait(0.1);

 }

 vit = 1500;

 moteur.pulsewidth\_us(vit);

 wait(1);

 alp = 0;

 my\_led = alp;

 /////////////////////////////////

 while(1) {

 my\_led = alp;

 // CONNECTION BLUETOOTH : ALLUMER ET DEMARRER LA VOITURE

 if (moduleBT.readable()) { // car version avec fonction d'interruption non fonctionnelle

 data = moduleBT.getc(); // récupérer un caractère sur la carte venant du capteur BT

 if (data == 'g'){ // g pour go -> démarrer la voiture

 wait(1);

 moteur.pulsewidth\_us(1600); // 1600 -> vitesse de démarrage (la plus lente)

 alp=1; // stockage de l'état marche ou arrêt de la voiture

 data = '0';

 }

 if (data == 's') { // s pour stop -> arrêter la voiture

 moteur.pulsewidth\_us(1500);

 alp=0;

 data = '0';

 }

 }

 //RECUPERATION DONNEES LIDAR ET CALCUL DE LA MOYENNE

 if (IS\_OK(my\_lidar.waitPoint())) { // ATTENTE D'UNE DONNEE

 float distance = my\_lidar.getCurrentPoint().distance; // RECUPERATION DE LA DISTANCE

 float angle = my\_lidar.getCurrentPoint().angle; // RECUPERATION DE L'ANGLE

 if (my\_lidar.getCurrentPoint().startBit) { // CHECK D'UN TOUR DU LIDAR : if(1)

 if (compteur==taille\_liste){ // VIDE LA LISTE : if(2)

 sumDist=1; //

 sumAngle=1; //

 compteur=0; //

 int s = 0; //

 while(s<taille\_liste+1){ // MOYENNAGE SUR taille\_liste VALEURS

 sumDist += zin[s][1]; //

 sumAngle += zin[s][0]; //

 s++; //

 } //

 mAng=(double)sumAngle/taille\_liste; // CALCUL DE L'ANGLE MOYEN

 mDist=(double)sumDist/taille\_liste; // CALCUL DE LA DISTANCE MAX MOYENNE

 moduleBT.printf("angle = %lf et distance = %lf \r\n", mAng , mDist); // AFFICHE LES VALEURS MOYENNES SUR LE TELEPHONE

 //CONSIGNE SERVOMOTEUR

 angle\_roues = 1300 + k\*angleAtMaxDist; // APPROCHE DE DIRECTION PROPORTIONNELLE

 servo.pulsewidth\_us(angle\_roues);

 }

 compteur++; // REMPLISSAGE DE LA LISTE TEMPORAIRE

 zin[compteur][0]=angleAtMaxDist; //

 zin[compteur][1]=maxDistance; //

 // REINITIALISATION ANGLE ET DISTANCE

 maxDistance = 0;

 angleAtMaxDist = 0;

 }

 else { // SI ON EST PAS AU POINT DE DEPART (IF(1)): RECHERCHE DU MAX SUR LE TOUR EN QUESTION

 //

 if ((angle < 90) or (angle >270)){ // ON GARDE QUE LES ANGLES DE DEVANT

 if ( (distance > 0) && (distance > maxDistance)) { // RECHERCHE DE LA DISTANCE MAX

 maxDistance = distance; //

 angle = normalisation(angle); // ANGLES ENTRE 270 ET 360 DEVIENNENT ENTRE 0 ET -90

 angleAtMaxDist = angle;

 }

 }

 }

 } else {

 if (IS\_OK(my\_lidar.getDeviceInfo(lidar\_info, 100))) { // SI ON NE RECOIT PAS DE DONEES DU LIDAR

 my\_lidar.startScan(); //

 wait(1); //

 my\_pc.printf("START SCAN\r\n"); //

 } //

 }

 //CONSIGNE MOTEUR

 if(alp==1){

 vit = 1600;

 moteur.pulsewidth\_us(vit);

 }

 // ASSERVISSEMENT DE LA VITESSE

 // Nous n'avons pas eu la possibilité d'ajouter cette partie pour le LiDAR //

 // en raison de problèmes rencontrés durant le confinement et non résolus //

 }

}