
Rapport technique de IETI

Robot autonome :

S6 2021

Jules-Théo Tang, Nicolas Guénaux, Laurent Forges et Pierre Chauvin-Buthaud

---

Ce document comporte une page de présentation, 8 pages de synthèse et 4 pages d'annexe.

Nous attestons que ce travail est original et qu'il ne comporte pas de plagiat.

A- Comprendre le projet

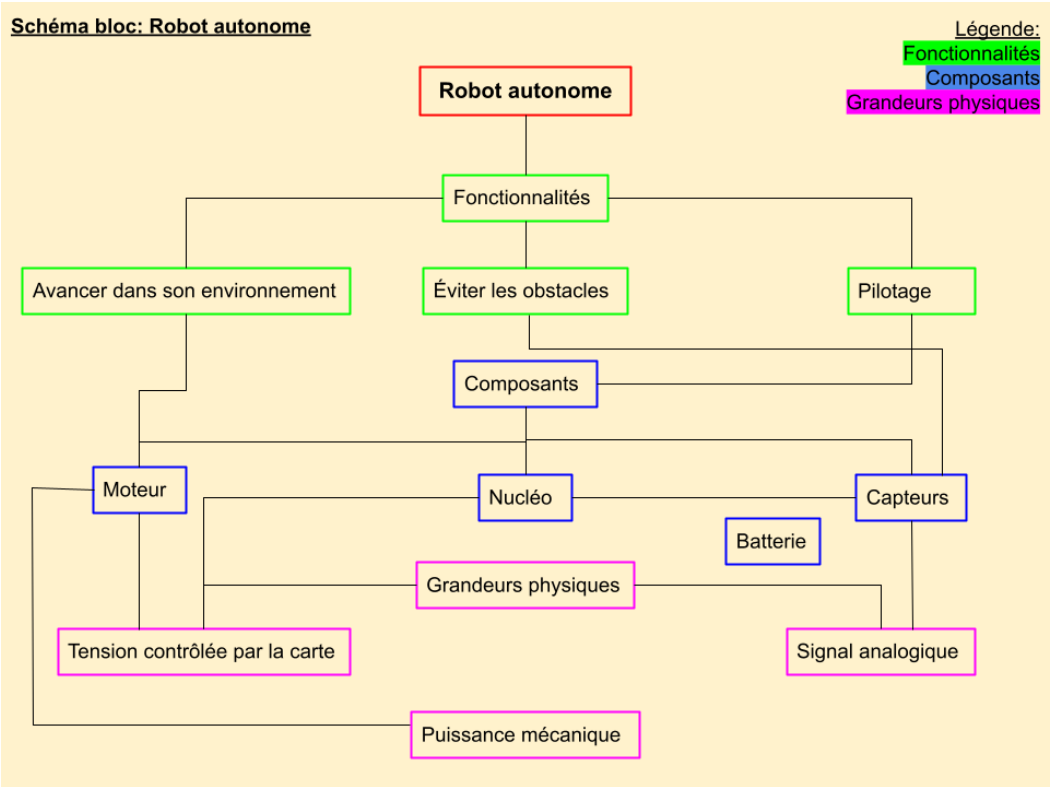
Introduction :

Au cours des dernières années, un nouveau type de robot a fait surface : les robots autonomes. On peut notamment penser aux robots aspirateurs ou tondeuses à gazon, qui fonctionnent sans aucune supervision humaine. Un point commun de ces robots est de pouvoir évoluer dans leur environnement en évitant les obstacles. C'est ce que nous allons essayer de réaliser dans ce projet.

Pour réaliser notre robot, nous disposons d'une plateforme dotée de 3 roues et deux moteurs. Ce robot sera un véhicule à propulsion : parmi ses 3 roues, deux sont motrices et situées à l'arrière. Avoir 3 roues permettra également au robot de pouvoir pivoter facilement. Ainsi, on peut aisément faire demi-tour devant un obstacle. Les éventuels obstacles seront détectés par des capteurs de distance qu'il faudra installer à l'avant du véhicule. Le but final est alors que le robot se déplace et évolue dans son environnement tout en évitant les obstacles. Son comportement sera dicté par un code informatique et implémenté via une carte nucléo.

Découpage fonctionnel :

Trouver des informations dans la documentation
Créer les fonctions correspondantes sous mbed en C
Récupérer un signal analogique Faire des actions à intervalle régulier Mettre en place un asservissement numérique
Piloter une LED Faire des actions à intervalle régulier
Faire communiquer 2 systèmes
Mettre en place un asservissement numérique



B- Réaliser le prototype

I- Schémas électriques:

La principale partie du projet est l'asservissement des deux moteurs. Ces deux moteurs ont préalablement été fixés au bâti qui est notre châssis. Chaque moteur est piloté par une tension que l'on module via un transistor contrôlé par la carte Nucléo.
Cela permet de "moduler" la rotation du moteur qui est proportionnelle à la tension appliquée. On se sert d'une carte Nucleo pour envoyer des échelons en série qui modélisent les différentes commandes: avancer, s'arrêter, tourner. Les échelons sont caractérisés dans le code par un rapport cyclique à choisir (dans [0,1]). En pratique, on choisira un rapport cyclique de 0,5; à la fois pour tester notre projet et éviter les chocs violents lors d'essais.

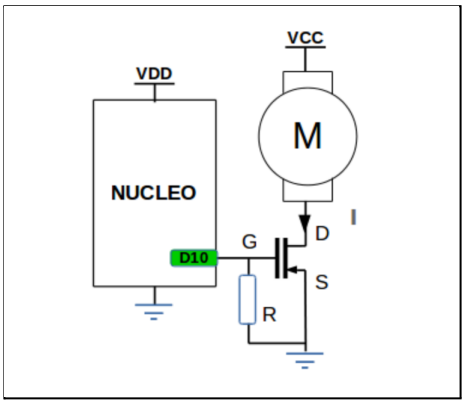
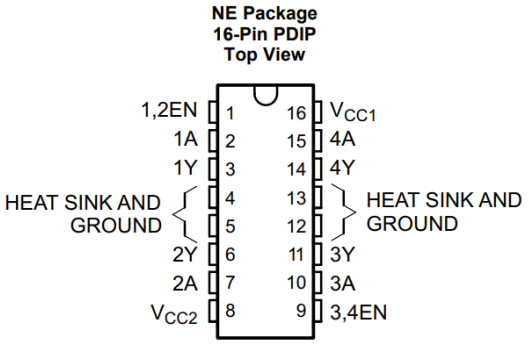


Figure 1 - Circuit à monter pour un seul moteur i.e. une roue qui tourne dans un seul sens

En pratique, il faudra contrôler les deux roues et chacune dans les deux sens de rotation. On utilisera alors un transistor qui permet de gérer ces options: le sjl

Afin de contrôler les roues dans les deux sens et d'alimenter le circuit et la carte nucleo on utilise un transistor en pont en H (voir détails sur la figure ci-dessous).



Pin Functions

PIN		TYPE	DESCRIPTION
NAME	NO.		
1,2EN	1	I	Enable driver channels 1 and 2 (active high input)
<1:4>A	2, 7, 10, 15	I	Driver inputs, noninverting
<1:4>Y	3, 6, 11, 14	O	Driver outputs
3,4EN	9	I	Enable driver channels 3 and 4 (active high input)
GROUND	4, 5, 12, 13	—	Device ground and heat sink pin. Connect to printed-circuit-board ground plane with multiple solid vias
V_{CC1}	16	—	5-V supply for internal logic translation
V_{CC2}	8	—	Power VCC for drivers 4.5 V to 36 V

Figure 2 : Schéma du transistor (pont en H)

II - A propo de l'algorithme

Les actions et le fonctionnement de notre robot sont régis par un code informatique en C/C++. Ce code permet au robot d'évoluer dans son environnement en adaptant le fonctionnement de son moteur et sa direction en fonction de l'information délivrée par ses 3 capteurs.
L'algorithme proposé est composé de trois procédures majeures expliquées ci-après.
Nous n'expliciterons pas ici le détail des lignes de codes, mais le développeur averti s'apercevra à sa vue, que l'algorithme utilisé, pour répondre aux contraintes du robot (inertie des moteurs, blocages), est le fruit d'un nombre important de tests et d'aller retour entre le code et la pratique.
NB : les deux moteurs propulsant le robot (en actionnant respectivement les deux roues) ont un temps de réponse assez élevé. Ils sont donc contrôlés par modulation en largeur d'impulsions.

II.1 - Procédure Initiale : Le mode croisière

Le robot exécute cette procédure lorsque son champ est libre : ses 3 capteurs ne repèrent aucun obstacle assez près pour être dangereux. Pour être plus spécifique : est dangereux un obstacle qui se situe à une distance inférieur à une distance seuil définie empiriquement.
Dans ce premier mode de fonctionnement, le robot avance donc tout droit (aux défauts de parallélisme des roues près) et sa vitesse est proportionnelle aux distances séparant le robot de l'obstacle le plus près, renvoyée par chaque capteur. Ainsi le robot peut amorcer un freinage ou accélérer selon la situation. Il dispose d'une certaine capacité d'anticipation.

II.2 - Procédure Intermédiaire : La mise à l'arrêt

Si un des 3 capteurs détecte un obstacle dangereux, le robot reçoit l'ordre de s'arrêter : ses 2 roues sont mises à l'arrêt pendant un certain temps, de manière à mettre fin à l'inertie du robot.
Seulement, pour avoir le temps de s'arrêter, le robot ne doit pas rouler trop vite en amont, ou bien détecter l'obstacle d'assez loin. Pour cela, les limites des capteurs de distance ont été étudiées. Elles ont permis de fixer, en complément d'un certain nombre de tests, un triplet de valeurs
$(vitesse\ max,\ distance\ seuil\ d'alerte,\ temps\ de\ mise\ à\ l'arrêt\ des\ roues)$ qui permet d'éviter le crash dans l'immense majorité des cas. Pour être plus précis, les seuls obstacles pouvant poser problème au robot sont ceux qui sont mal repérés par les capteurs car trop fins ou situés trop bas.

II.3 - Procédure Finale : Le changement de cap

Une fois le véhicule à l'arrêt et le choc avec l'obstacle évité, le robot doit déterminer sa position relative vis à vis de cet obstacle afin de le contourner en pivotant dans un sens ou dans l'autre.
Cela est rendu possible par les informations renvoyées par les 3 capteurs et notamment ceux sur les côtés : En effet, si un capteur de côté indique une plus grande proximité de l'obstacle que le second, on va donner l'ordre au robot de s'orienter dans la direction du second capteur.
Pour se faire, on fait tourner le robot sur lui-même : la roue de gauche tourne dans un sens et celle de droite dans l'autre (à la même vitesse). Ainsi le robot pivote. Il continue de pivoter tant que les capteurs détectent un obstacle trop proche pour reprendre la marche.
Lors de cette procédure, le robot peut éventuellement rencontrer un second obstacle plus proche que le premier et dans la direction opposée. Il s'arrêtera alors un court instant et recommencera la procédure, mais cette fois-ci il pivotera dans l'autre sens.

Une fois tous les obstacles dangereux contournés, le robot retourne dans son état initial : il reprend sa marche.

NB : Une copie du code est à retrouver en annexe.

C- Valider et caractériser le système

Tests et validation :

A chaque début de séance nous prenons soin de bien rebrancher chaque élément comme il se doit et en reprenant le code Mbed qui fonctionnait la fois d'avant. Si le prototype fonctionne, on reprend le système et on essaie de l'améliorer (notamment en ajoutant des fonctions au code).
À chaque ajout de fonctionnalité nouvelle, on teste le système afin de savoir dans quelle mesure il répond correctement à nos attentes. Tant que le résultat n'est pas satisfaisant, on retourne ajuster la fonctionnalité pour ensuite la remettre à l'essai. On converge ainsi vers un résultat cohérent avec la théorie.

Mise en rotation des roues: On lit la datasheet des moteurs à disposition pour comprendre comment les brancher et connaître leurs capacités théoriques. On définit de façon relative les côtés gauches et droits et les sens horaires et anti-horaire. Après la rédaction d'un code simple sur Mbed, on vérifie si les côtés et sens de rotation correspondent aux noms des variables associées; si ce n'est pas le cas, on ajuste les nom des variables du code.

Les capteurs : On lit la datasheet des capteurs à disposition (SHARP GP2YA21) pour comprendre comment les brancher et connaître leurs capacités théoriques. Afin d'optimiser leur utilisation, on branche un capteur au circuit et "print" le signal analogique reçu par le système. Grâce au retour de TeraTerm on étudie l'évolution du signal qui s'affiche à l'écran en fonction de la distance à un objet (on prend un objet d'une grande surface).
Plusieurs remarques découlent de ces observations :

- Le système n'étant pas parfait, les valeurs qui s'affichent à l'écran ne sont pas identiques pour un objet à distance fixe.
- Des anomalies se produisent régulièrement : lorsque l'objet est loin du capteur il arrive que ce dernier affiche un signal très fort parmi une série de valeurs faibles et inversement.
- La tension seuil des capteurs (distance à partir de laquelle les capteurs ne réagissent plus correctement) est définie pour une certaine distance qui correspond à une certaine valeur de tension.

Cette étude nous a permis de déterminer le seuil de fonctionnement réel du capteur ainsi que le rapport entre la tension reçue par le système et la distance à un objet. Comme on s'appuie également sur un système de seuils pour piloter le robot, on est amené à effectuer une moyenne sur plusieurs valeurs de la distance entre le capteur et l'obstacle afin de limiter les effets parasites dus aux imperfections du système.

Système de contrôle à LED:
Afin de pouvoir vérifier rapidement si notre système répond comme nous le souhaitons sans avoir à faire des tests en situation réelle, nous avons mis en place un système de contrôle à LED. Nous utilisons 4 LED (Rouge, Verte, Jaune, Bleue), de facon à pouvoir différencier les actions de notre robot (s'arrêter, tourner à gauche, tourner à droite, avancer). A chaque entrée dans une boucle d'action du robot, on éteint les LED qui ne servent pas et on allume la LED correspondant au régime de fonctionnement. La mise en situation du robot sur un support surélevé nous permet de vérifier et valider la corrélation entre le régime de fonctionnement et le système à LED.

Adaptation de la vitesse en fonction de la distance à un objet :
Afin de déterminer les seuils de fonctionnement du robot (s'arrêter, tourner) et donc déterminer les rapports cycliques à appliquer aux moteurs, on effectue une série de tests en situation réel pour ajuster vitesse et distance de freinage.
Avant d'arrêter complètement les moteurs, on réduit leur vitesse de rotation en fonction de la distance à laquelle elle se trouve de l'obstacle. (CF B-II.1-Mode croisière)
La portée des capteurs étant trop faible, nous choisissons de ne pas faire rouler le robot à un rapport cyclique de 1 (l'inertie du robot est trop importante pour qu'il s'arrête avant l'obstacle). (CF B-II.2-Mise à l'arrêt)

Contourner un obstacle :
Afin de réussir à contourner un obstacle; on code la procédure décrite en B-III.3-changement de cap.
Pour déterminer les seuils avec lesquels travailler et les rapports cycliques à imposer au système, on effectue des tests en situation réelle. La corrélation entre capteur et sens de rotation de chaque moteur ayant été préalablement établie, on s'assure que les rotations du robot (pratiquement sur place) n'entraînent pas de collision avec les côtés du robot; auquel cas on ajuste les seuils des capteurs latéraux.

D- Comprendre les étapes de réalisation

I - Planning :

Séance 1 - 25/01/2021:
- compréhension du système et câblage
- lecture de la datasheet des transistors
- compréhension du code fourni
- mise en rotation d'une roue puis d'une deuxième dans les deux sens

Séance 2 - 01/02/2021:
- fonctionnement du capteur de proximité
- corrélation capteurs de distance/roues
- mise en place du programme pour le capteur de distance
- indépendance du système par rapport à l'ordinateur et à l'alimentation → système embarqué
- test du système : problème de parallélisme, fonctionnement correct du système mais distance de freinage trop courte lors de la détection d'obstacle, le système s'arrête lorsque sa vitesse est assez faible

Séance 3 - 10/03/2021:
- Recablage du système et vérification du fonctionnement et des interactions entre moteurs et capteurs
- Embarcation du système total (Nucléo + sa pile 9V, la breadboard, l'alimentation pour les deux moteurs)
- Modification des seuils des capteurs afin d'améliorer la distance d'arrêt (système avec limitation du rapport cyclique. Le véhicule s'arrête devant un obstacle même à pleine vitesse

Séance 4 - 24/03/2021:

- Ajout de 2 nouveaux capteurs portant leur nombre à 3 : 1 à l'avant, 1 en position avant gauche, 1 dernier en position avant droite. Le capteur central est choisi de telle sorte que sa portée soit supérieure. Ainsi, on accède à un autre niveau d'anticipation et d'autonomie du robot.
- Câblage des nouveaux capteurs
- Etude des nouveaux capteurs (tension min / max, distance min / max)
- Nous avons réfléchi à un système de socle et de protection pour les capteurs à l'avant.

Séance 5 - 12/04/2021:

- Prise en compte des nouveaux capteurs dans le code. Implémentation de nouvelles procédures. Avec 3 fois plus d'informations, on peut alors faire prendre des décisions au robot, le rendant effectivement autonome : L'obstacle faisant face au véhicule est localisé en 2D.
- Le robot pivote dans le bon sens.
- Design du bâtis sur le logiciel inkscape

Séance 6 - 10/05/2021:

- Mise en place de tests exhaustifs pour optimiser les paramètres du robot : distances seuils auxquelles le robot doit mettre un terme à son mouvement pour chaque capteur (gauche, droite et centre). On veut le RC le plus grand possible qui ne conduise pas à un crash du robot.
- Découpe laser et création du bâtis
- Embarcation du système (carte nucleo, batterie, breadboard & capteurs) sur le nouveau batis.

Séance 7 - 19/05/2021:

- Derniers tests et affinage des paramètres du système.
- Nouveau système d'adaptation de la vitesse (RC) en fonction des 3 distances fournies par les 3 capteurs. Conduit à une nouvelle réduction des situations dangereuses.
- Correction partielle du défaut de parallélisme des deux roues.
- Nous convergeons vers un système autonome et stable : le robot se déplace à une vitesse convenable tout en anticipant les obstacles et en s'orientant dans la bonne direction.

Conclusion & Ouverture du projet

En quelques séances, nous avons réussi à concevoir un robot autonome. Son comportement est satisfaisant, dans la mesure où il évolue dans son environnement à une allure dynamique tout en esquivant chaque obstacle. Les objectifs sont donc remplis. Toutefois, on ne peut que regretter le léger défaut de sa trajectoire, lui-même dû au défaut de parallélisme des deux roues motrices.

En outre, si ce robot doit servir de base pour un robot aspirateur ou tondeuse, on s'aperçoit rapidement qu'il n'est pas bien efficace car il peut passer plusieurs fois au même endroit sans jamais couvrir une ou plusieurs zones.
Ces différents challenges peuvent être relevés en asservissant les moteurs avec le reste du système et en conservant en mémoire les différents obstacles rencontrés, de manière à cartographier l'environnement.

Annexe:

```
// Code du robot autonome
//
// Auteurs : Nicolas Guénaux, Jules-Théo Tang, Laurent Forges, Pierre Chauvin Buthaud
//
// Date : S2 de l'année 2020/2021

/* Prototype des fonctions */

double TensionToDistance(double sensor, int capteur);

/* TensionToDistance : Convertit la tension reçue par un capteur en distance
        Entrées :  sensor : la tension, floatant
        capteur : booléen correspondant au différent capteur :
        si capteur = 1 il s'agit d'un des deux capteur de côté, si capteur = 0 il s'agit du capteur central
        On les différencie car on a spécialement choisi un capteur central avec de meilleures caractéristiques
        Sortie : la distance correspondante
*/

#include "mbed.h"
#include "math.h"

PwmOut moteur_ccg1(D6);     //Sortie numérique correspondant à la roue gauche contrôlée dans le sens horaire
PwmOut moteur_ccg2(D5);     //Sortie numérique correspondant à la roue gauche contrôlée dans le sens anti-horaire
PwmOut moteur_ccd1(D9);     //Idem pour la roue droite
PwmOut moteur_ccd2(D10);

PwmOut led_bleue(D11);      //La LED bleue, si allumée, signifie que le robot a reçu pour ordre d'avancer
PwmOut led_rouge(D12);      //La LED rouge, si allumée, signifie que le robot a reçu pour ordre de s'arrêter
PwmOut led_d(D13);          //La LED_d (jaune), si allumée, signifie que le robot a reçu pour ordre de pivoter à droite
PwmOut led_g(D14);          //La LED_g (verte), si allumée, signifie que le robot a reçu pour ordre de pivoter à gauche

Serial pc(USBTX, USBRX);    //Utile pour évaluer l'acuité du code avec le robot sur TeraTerm

AnalogIn analog_in_m(A0);   //Tension délivrée par le capteur central
AnalogIn analog_in_g(A1);   //Tension délivrée par le capteur gauche
AnalogIn analog_in_d(A2);   //Tension délivrée par le capteur droit


int main() {

        double sensor_m, sensor_g, sensor_d;   //Variables qui stockeront la valeur des tensions relevées respectivement
par
                                //le capteur du milieu, de gauche et de droite.

        int first = 0;          //Booléen qui indique le régime de fonctionnement : 0 <=> on est à l'arret; 1 <=> on avance/on
pivote.
                //'first = 1' permet d'indiquer si c'est la première fois que l'on repère un obstacle. Auquel cas on arrête
complètement le véhicule.
                //Si à l'inverse 'first = 0', c'est que l'on s'est déja arrêté et qu'on est en train de contourner l'obstacle
                //ou bien qu'il n'y a plus d'obstacle et que l'on avance en ligne droite

        int direction = 0;  //Indique la direction suivie en cas de contournement d'obstacle : utile pour les changements de
direction
                //'direction' peut prendre 3 valeurs possibles : 0, 1 ou 2; 'direction = 0' <=> On amorce le virage pour la
première fois;
                //'direction = 1' <=> Jusqu'à maintenant on a tourné vers la gauche; direction = 2 <=> Jusqu'à maintenant
on a tourné vers la droite.


        pc.baud(115200);    //Utile pour évaluer l'acuité du code avec le robot sur TeraTerm
```

```
	moteur_ccg1.period_ms(10);  //On fixe la période pour la modulation en largeur d'impulsions (PWM) des deux sens pour chacune des deux roues
	moteur_ccg2.period_ms(10);  //Une période de 10ms suffit car le moteur alimenté en courant continu met du temps à réagir
	moteur_ccg1.write(0);		//Puis on initialise leur rapport cyclique (RC) à 0
	moteur_ccg2.write(0);
	moteur_ccd1.period_ms(10);
	moteur_ccd2.period_ms(10);
	moteur_ccd1.write(0);
	moteur_ccd2.write(0);


	double d_m,d1_m,d2_m,d3_m, d_g,d1_g,d2_g,d3_g, d_d,d1_d,d2_d,d3_d ;   // On définit plusieurs variables représentant des distances (unité proche du cm)


	// On définit des valeurs seuilles de distance à partir desquelles le robot recevra l'ordre de s'arrêter net.
	double seuil_m = 75;		//Le capteur central étant plus puissant, on aura le temps d'adapter la vitesse, avant de stoper net
	double seuil_g = 80;		//Les seuils correspondant aux capteurs gauche et droit sont donc un peu plus élevé
	double seuil_d = 80;		//Ce sont des valeurs déterminées de manière empirique

	double RC, RC_eff; //Les rapports cycliques

	while(1){

	//Pour chaque capteur on fait plusieurs acquisition des valeurs de tension délivrée, pour lisser ces valeurs et diminuer la probabilité de fausse alerte (qui arreterait le robot pour rien)
	sensor_m = analog_in_m.read();		//Acquisition de la tension délivrée par le capteur central
	d1_m = TensionToDistance(sensor_m, 0); //Calcul de la distance associée
	sensor_g = analog_in_g.read();		//Idem pour le capteur gauche
	d1_g = TensionToDistance(sensor_g, 1);
	sensor_d = analog_in_d.read();		//Puis le droit
	d1_d = TensionToDistance(sensor_d, 1);
	wait_us(50000);			//Temps de pause

	sensor_m = analog_in_m.read();		//On réitère
	d2_m = TensionToDistance(sensor_m, 0);
	sensor_g = analog_in_g.read();
	d2_g = TensionToDistance(sensor_g, 1);
	sensor_d = analog_in_d.read();
	d2_d = TensionToDistance(sensor_d, 1);
	wait_us(50000);

	sensor_m = analog_in_m.read();		//On réitère
	d3_m = TensionToDistance(sensor_m, 0);
	sensor_g = analog_in_g.read();
	d3_g = TensionToDistance(sensor_g, 1);
	sensor_d = analog_in_d.read();
	d3_d = TensionToDistance(sensor_d, 1);
	wait_us(50000);

	d_m=(d1_m+d2_m+d3_m)/3;		//On travaillera par la suite avec cette valeur moyenne de la distance relevé pour ce capteur central
	d_g=(d1_g+d2_g+d3_g)/3;		//Idem pour les capteurs gauche et droit
	d_d=(d1_d+d2_d+d3_d)/3;

	//Cette procédure s'execute en 150ms : ce temps ne nuit pas au bon fonctionnement du robot



	RC = 0.51;		//Valeur établie de manière empirique : ce RC allie vitesse raisonnable et une probabilité de crash très faible

	//"Début" procédure initiale (Marche)
	if(d_m>seuil_m && d_g>seuil_g && d_d>seuil_d){  //S'il n'est détecté aucun obstacle trop proche par les 3 capteurs
	if (first==0){		//signifie une fin de procédure d'esquivement (ou première itération de la boucle infinie car cette valeur est initiée à 0)
			first = 1;   //On avance
```

```
                led_rouge.write(0);  //Seule la LED bleue s'allume. Les autres s'éteignent. Car on initie la trajectoire vers
l'avant.
                led_d.write(0);
                led_g.write(0);
                led_bleue.write(0.75);
                moteur_ccg2.write(0);   //On s'assure que les roues ne tournent pas encore à l'envers
                moteur_ccd2.write(0);   //Permet d'éviter des bugs
                }

        //Dans tous les cas (fin de procédure d'esquivement OU continuité de la trajectoire vers l'avant) :

        RC_eff = RC*(d_m/100.0)*(d_g/90.0)*(d_d/90.0);  //Le rapport cyclique effectif permet de prendre en compte
                                        //les distances aux obstacles détectés par les 3 capteurs
                                        //La distance max repérable par les capteurs gauche et droit est 90(unité de distance)
                                        //La distance max repérable par le capteur central est 100 (il est plus puissant, d'où le
seuil moins élevé)
        moteur_ccg1.write(RC_eff);    //Les deux roues tournent dans le sens horaire au même rythme
        moteur_ccd1.write(RC_eff);

        pc.printf("GO");       //Pour TeraTerm (optionel)
        }
        //"Fin" procédure initiale (Marche)


        else{       //Si un des capteurs détecte un obstacle

        //"Début" de la procédure intermédiaire (Arrêt)
        if (first){   //Si c'est la première fois qu'on détecte cet obstacle
                led_bleue.write(0);
                led_rouge.write(1);  //On s'arrête
                led_d.write(0);
                led_g.write(0);

                moteur_ccg1.write(0);
                moteur_ccg2.write(0);

                moteur_ccd1.write(0);
                moteur_ccd2.write(0);

                wait_us(1000000);               //On attend un temps que le robot s'arrête complètement (rendu nécessaire
par l'inertie acquise)
                first = 0;

                pc.printf("STOP");  //Pour TeraTerm (optionel)
                }
        //"Fin" de la procédure intermédiaire (Arrêt)

        //"Début" de la procédure finale (Esquive)
        if (d_g>d_d && (direction == 0 || direction == 1)){ //Si l'obstacle est à droite ET (que l'on initie une nouvelle rotation
OU que l'on tournait déjà à gauche)
                led_g.write(1);         // indique que l'on tourne à gauche
                led_d.write(0);
                led_rouge.write(0);
                led_bleue.write(0);

                moteur_ccg2.write(RC);  //RC>RC_eff : on peut pivoter plus vite que l'on avance car on ne risque pas de
cogner quoi que ce soit en pivotant
                moteur_ccd1.write(RC);  //Les roues tournent au même rythme MAIS en sens inverse (d'ou le pivotement)

                // indique que l'on tourne à gauche
                pc.printf("GAUCHE");
                direction = 1;
                }

        if (d_g<d_d && (direction == 0 || direction == 2)){ //Exactement la même procédure mais cette fois pour tourner à
droite
                led_d.write(1);         // indique que l'on tourne à droite
                led_g.write(0);
                led_rouge.write(0);
```

```
                led_bleue.write(0);

                moteur_ccg1.write(RC);
                moteur_ccd2.write(RC);

                pc.printf("DROITE");
                direction=2;
                }

        if ((d_g>d_d && direction == 2)||(d_g<d_d && direction == 1)) { //Si les capteurs indique qu'il y a finalement un
obstacle plus proche de l'autre côté

                //Procédure de changement de cap

                //Procédure permettant d'éviter un blocage du robot dans le cas où [il cherchait à esquiver un
                //premier obstacle à gauche en tournant à droite('direction=2')] mais [trouve finalement sur sa course un
obstacle plus près à droite ('d_g>d_d')]
                //Et vice versa

                //On arrête la course du robot
                moteur_ccg1.write(0);
                moteur_ccg2.write(0);

                moteur_ccd1.write(0);
                moteur_ccd2.write(0);


                //Indicateur de changement de cap
                led_g.write(1);
                led_d.write(1);
                led_rouge.write(0);
                led_bleue.write(0);

                //On attend un temps que le robot s'arrête complètement (rendu nécessaire par l'inertie acquise)
                wait_us(200000);

                //On va initier un nouveau virage
                direction = 0;

                }
        //"Fin" de la procédure finale
        }
        //Les mots 'Début' et 'Fin' sont mis entre guillemet car les procédures durent généralement plus d'un tour de boucle...
        }

        }


double TensionToDistance(double sensor, int capteur){

        double Dmin, Dmax;

        // Valeurs déterminées empiriquement : le capteur central est plus puissant : il voit plus loin, il faut donc distinguer les
cas
        if (capteur){           // Il s'agit d'un capteur de côté
        Dmin = 5;               // en cm
        Dmax = 85;
        }
        else{                   // Il s'agit du capteur central
        Dmin = 5;               // en cm
        Dmax = 130;
        }

        double Vmin = 0.07;
        double Vmax = 1;

        double dist = Dmax - ((Dmax - Dmin)/(Vmax - Vmin))*(sensor-Vmin);
        return dist;
}
```