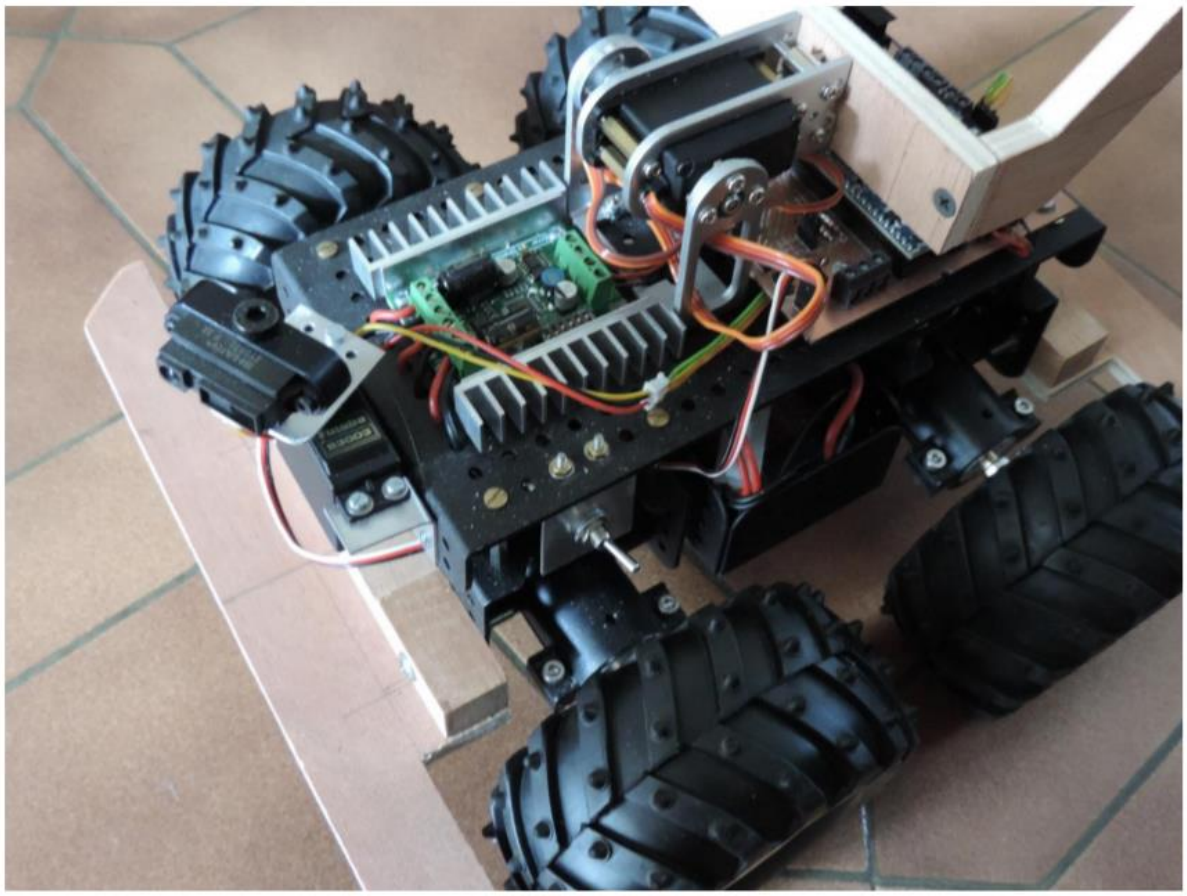
Projet tuteuré 2ème année DUT GEII

Robot Autonome



Nicolas BAËLE

Table des matières

Introduction

Nous avons choisi comme projet pour le semestre 3 de DUT GEII le robot autonome. Ce projet consiste à la programmation d'un module Mbed LPC1768 pour permettre à un robot 4 roues motrices de se déplacer en évitant les obstacles. Il est aussi possible de le programmer pour la contrôler à distance grâce au Bluetooth de son téléphone et à un module Bluetooth branché sur le module Mbed.

Le projet se distingue en 4 grandes parties, tout d'abord il faut s'attacher à la prise en main de Mbed et à la création des fonctions de base pour notre projet. Ensuite il faut utiliser le module Bluetooth pour contrôler le robot depuis son téléphone et grâce aux fonctions créées précédemment. Une fois cela fait il faut apprendre à gérer les différents capteurs (optique et ultrason), pour repérer avec précision un obstacle dans l'espace. Et enfin l'addition de toutes les parties précédentes pour arriver au but du projet et avois un robot qui peut se déplacer en évitant les obstacles, un robot autonome.

Prise en main Mbed

Il faut bien commencer quelque part, alors en suivant le sujet du projet on nous invite à découvrir tous les composants qui composent notre projet, la carte Sabertooth, le module Bluetooth, le capteur optique, et le plus important de tous, Mbed qui est notre compilateur et là où nous écrirons l'entièreté du code. Tout d'abord nous avons effectué 2 programme très simple pour apprendre la gestion des entrées/sorties sur Mbed, ainsi que la liaison série pour le débogage principalement.

Le premier programme est le suivant et permet de faire clignoter une led du module Mbed :

```
1
2  #include "mbed.h"                        // on inclu la librairie mbed.h
3
4  DigitalOut led1(LED1);                   // déclaration de la sortie digitale correspondant à la led 1
5
6  int main()                               // initialisation du main
7  {
8      while(1)                             // on créer une boucle infinie pour que le programme tourne en continu
9      {
10         led1 = !led1;                    // si la led1 est allumée, alors elle s'éteind et inversement
11         wait(1);                         // on attend une seconde avant de continuer
12     }
13 }                                        // fermeture du main
```

Le deuxième programme quant à lui permet d'afficher sur TeraTerm les touches appuyées par l'utilisateur sur le clavier en utilisant la liaison série, ce programme nous apprends comment déboguer, car on peut voir les valeurs que prennent les différentes variables du projet :

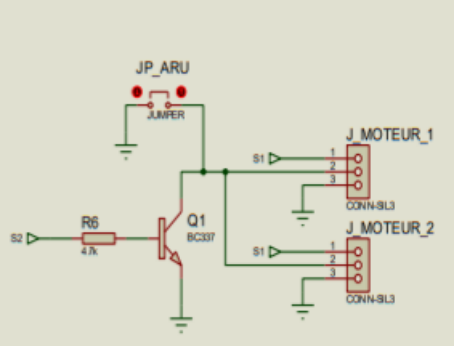
```
1
2  #include "mbed.h"                                    // on inclu la librairie mbed.h
3
4  Serial pc (USBTX, USBRX);                           // déclaration port serie pc qui utilise les entrées/sorties (tx, rx)
5
6  int main()                                          // initialisation du main
7  {
8      pc.printf("Renvois ce que vous tapez\n");       // printf affiche sur teraterm de "Renvois ce que vous tazpez"
9
10     while(1) {                                      // on créer une boucle infinie pour que le programme tourne en continu
11         pc.putc(pc.getc());                         //putc envoi sur teraterm la valeur demandée, getc lit la valeur demandée
12     }
13 }
```

Ces programmes ne sont que des simples bases sur le fonctionnement d'Mbed, mais ils sont néanmoins essentiels, notamment celui en liaison série car son application nous permettra par la suite de tester le bon fonctionnement de notre programme ainsi que les données perçues par les différents capteurs.

Cependant, notre projet a besoin de fonctions plus élaborées que celles-ci, c'est ce que nous verrons dans la partie suivante avec la gestion de la carte Sabertooth.

Gestion de la carte Sabertooth

La carte Sabertooth communique avec le module Mbed par liaison série sur les pins p9 et p10 respectivement tx et rx, ainsi qu'avec la pin p11 qui correspond à la sortie S2 sur le schéma ci-dessous. Cette sortie S2 est indispensable au fonctionnement de la carte Sabertooth et doit être mise à 0 pour que cette dernière puisse recevoir des informations de la part du module Mbed, elle sera utilisée par la suite pour la fonction d'arrêt d'urgence.



Sur le schéma, le JUMPER correspond à un cavalier que l'on doit enlever manuellement pour que la carte puise recevoir des informations.

La carte Sabertooth reçois des informations par le port série et à l'aide de la fonction putc, elle prend plusieurs paramètres tels que :

```
Putc(address);
Putc(0);
Putc(speed);
Putc((address + 0 + speed) & 127);
```

address : l'adresse sur laquelle la carte est paramétrée pour qu'elle sache que les informations lui sont destinées, ici ce sera 128.

0 : c'est une adresse qui correspond à une action définie par la carte directement il y en a 17, ici 0 correspond à faire avancer le moteur 1.

speed : la vitesse souhaitée pour le moteur, qui varie de 0 à 127.

Il nous était demandé dans le sujet de réaliser plusieurs fonctions basiques pour pouvoir commencer à commander notre robot par le programme. Je vais donc vous les présenter.

void avancer (void)

Cette fonction permet de faire avancer les deux moteurs dans la même direction et donc de faire avancer le robot. Elle utilise les fonctions 0 et 4 qui font respectivement avancer les moteurs 1 et 2. On fixe la vitesse à 50 pour des raisons techniques, le moteur 1 à une vitesse de 48 car les deux moteurs ne vont pas exactement à la même allure.

```
1  void avancer(void)
2  {
3      device.putc(128);
4      device.putc(0);
5      device.putc (48);
6      device.putc((128 + 0 + 48) & 127);
7      device.putc(128);
8      device.putc(4);
9      device.putc (50);
10     device.putc((128 + 4 + 50) & 127);
11 }
```

void arreter (void)

Cette fonction permet de stopper les deux moteurs et donc de faire arrêter la voiture.

On fixe donc la vitesse des moteurs sur 0, et on utilise les mêmes commandes 0 et 4, la carte Sabertooth comprends donc quelle doit faire avancer les deux moteurs à une vitesse nulle et donc arrêter les moteurs.

```
1  void arreter(void)
2  {
3      device.putc(128);
4      device.putc(0);
5      device.putc (0);
6      device.putc((128 + 0 + 0) & 127);
7      device.putc(128);
8      device.putc(4);
9      device.putc (0);
10     device.putc((128 + 4 + 0) & 127);
11 }
```

void tourner_droite (void)

Cette fonction permet de faire faire à notre robot un huitième de tour vers la droite. Ce qui consister à faire tourner le moteur gauche (1) vers l'avant et le moteur droit (2) vers l'arrière. On utilise donc la fonction 0 et la fonction 5 qui permet de faire reculer le moteur 2. On fixe la vitesse des deux moteurs à 30 pendant 1 seconde puis on fixe la vitesse à 0 pour qu'il s'arrête.

```
void tourner_droite(void){

        device.putc(128);
        device.putc(0);
        device.putc (34);
        device.putc((128 + 0 + 34) & 127);
        device.putc(128);
        device.putc(5);
        device.putc (30);
        device.putc((128 + 5 + 30) & 127);

        wait(1);

        device.putc(128);
        device.putc(0);
        device.putc (0);
        device.putc((128 + 0 + 0) & 127);
        device.putc(128);
        device.putc(5);
        device.putc (0);
        device.putc((128 + 5 + 0) & 127);
        }
```

void tourner_gauche (void)

Cette fonction a pour but de faire tourner le robot d'un huitième de tour vers la gauche, le code est sensiblement similaire à celui de la fonction pour tourner à droite mais il utilise la fonction 4 déjà vu précédemment et la fonction 1 qui fait reculer le moteur 1.

```
void tourner_gauche(void){

        device.putc(128);
        device.putc(1);
        device.putc (30);
        device.putc((128 + 1 + 30) & 127);
        device.putc(128);
        device.putc(4);
        device.putc (36);
        device.putc((128 + 4 + 36) & 127);

        wait(1);

        device.putc(128);
        device.putc(1);
        device.putc (0);
        device.putc((128 + 1 + 0) & 127);
        device.putc(128);
        device.putc(4);
        device.putc (0);
        device.putc((128 + 4 + 0) & 127);
        }
```

void reculer (void)

Cette fonction a pour but de faire reculer le robot en utilisant les fonctions 1 et 5 vus précédemment pour faire reculer respectivement les moteurs 1 et 2.

```
void reculer()
{
    device.putc(128);
    device.putc(1);
    device.putc (48);
    device.putc((128 + 1 + 48) & 127);
    device.putc(128);
    device.putc(5);
    device.putc (50);
    device.putc((128 + 5 + 50) & 127);
}
```

void demi_tour (void)

Cette fonction utilise la même logique que la fonction tourner_gauche, cependant le wait est plus long pour laisser le temps à la voiture de tourner assez. Elle utilise les fonctions 1 et 4.

```
void demi_tour(){

        device.putc(128);
        device.putc(1);
        device.putc (30);
        device.putc((128 + 1 + 30) & 127);
        device.putc(128);
        device.putc(4);
        device.putc (30);
        device.putc((128 + 4 + 30) & 127);

        wait(5);

        device.putc(128);
        device.putc(1);
        device.putc (0);
        device.putc((128 + 1 + 0) & 127);
        device.putc(128);
        device.putc(4);
        device.putc (0);
        device.putc((128 + 4 + 0) & 127);
        }
```

void arret_urgence (void)

Cette fonction utilise par sortie S2 sur la pin p11 vu précédemment qui doit être à 0 pour que le robot reçoive les informations, son but est donc de mettre S2 à 1, ceci a pour effet de stopper le robot et de le mettre en attente d'une réinitialisation.

```
void arret_urgence()
{
    S2 = 1;
}
```

Ensuite, nous avons modifiés ces fonctions pour pouvoir entrer une valeur de vitesse, ici la variable vitesse, et avoir une montée et une descente progressive en vitesse.

Pour la montée :

```
void avancer(int vitesse)
{
    for(int i = 0; i <= vitesse; i++;)
    {
        device.putc(128);
        device.putc(0);
        device.putc (i);
        device.putc((128 + 0 + i) & 127);
        device.putc(128);
        device.putc(4);
        device.putc (i);
        device.putc((128 + 4 + i) & 127);
    }
}
```

Pour la descente :

```
void avancer(int vitesse)
{
    for(int i = vitesse; i >= 0; i--;)
    {
        device.putc(128);
        device.putc(0);
        device.putc (i);
        device.putc((128 + 0 + i) & 127);
        device.putc(128);
        device.putc(4);
        device.putc (i);
        device.putc((128 + 4 + i) & 127);
    }
}
```

Toutes ces fonctions sont les fonctions basiques que nous utiliserons par la suite, avec parfois quelques modifications suivant les besoins.

Bluetooth

La gestion du Bluetooth est plus simple qu'il n'y paraît. En effet, on communique avec une carte Bluetooth qui elle communique avec le module Mbed par liaison série par les ports p13 et p14.

On peut donc approcher le Bluetooth avec des fonctions simple telles que putc(), qui envois la valeur souhaitée dans la liaison série, ou encore getc(), qui permet de récupérer la valeur présente dans la liaison série.

Nous allons donc scinder notre programme en deux parties, la première sera un mode automatique avec les capteurs, la seconde sera un mode dit manuel qui sera gérée avec le téléphone. On passera de l'une à l'autre grâce à la variable manuel, où manuel == 1 veut dire manuel et manuel == 0 veut dire automatique.

Sur l'application Bluetooth SPP pro on définit des boutons qui envoi des caractères, ces derniers sont alors traités par le programme et on tous une fonction attitrée.

Avec la fonction readable(), on attend qu'un caractère arrive sur la liaison série, si c'est le cas, on entre ce caractère dans la variable rblue qui nous permettra de faire nos tests par la suite (if).

```
while(1)
    {
            if(blue.readable())
            {
                rblue = blue.getc();
                printf("%c",rblue);
            }
            if(rblue == 'q')
            {
               manuel = 0;
            }
            if(rblue == 'm')
            {
                manuel = 1;
            }
            if(manuel == 1)
            {
                if(rblue == 'a')
                {
                    avancerb();
                }
                if(rblue == 'g')
                {
                    tourner_gauche();
                }
                if(rblue == 'd')

                {
                    tourner_droite();
                }
                if(rblue == 'r')
                {
                    reculer();
                }
                if(rblue == 's')
                {
                    arreter();
                }
                if(rblue == 't')
                {
                    demi_tour();
                }
                if(rblue == 'u')
                {
                    arret_urgence();
                }
            }
        }
```

Capteur optique GP2Y0A21YK

La deuxième partie du projet étant d'avoir un robot autonome, il faut donc s'attaquer à la gestion des capteurs.

Nous avons donc en premier lieu essayer de gérer le capteur optique qui offre une information si un obstacle est présent devant lui, cependant, il a un rayon de visibilité très court et devra donc être assisté par le capteur SRF qui lui, a un champ de détection très large. Le capteur optique délivre une plage de tension de [0 ; 3.3] V nous avons donc décidé, dans le mode automatique, que le robot se stop lorsque le capteur envoi un voltage qui correspond à peu près a 30cm sur la pin p15. C'est-à-dire, dans la fonction avancer (), nous avons mis une condition telle que s'il y a un obstacle à 30 cm, le robot s'arrête et effectue une rotation pour chercher une voie libre.

```
184         if(manuel == 0)
185         {
186             if (obs == 0)
187             {
188                 avancer(capt);
189             }
190             else if ( obs == 1 && ( gauche == 1 | first== 0))
191             {
192                 tourner_droite();
193                 tourner_droite();
194                 obs = 2;
195             }
196             else if ( obs == 1 && ( droite == 1 ))
197             {
198                 tourner_gauche();
199                 tourner_gauche();
200                 obs = 2;
201             }
202             else if ((capt > 0.3f) && ( obs == 2))
203             {
204                 demi_tour();
205                 obs = 0;
206             }
207             else obs = 0;
208         }
```

Les variables droite et gauche permettent au robot de savoir dans quel sens il a tourné précédemment. En effet si le robot rencontre un obstacle, il va effectuer un ¼ de tour vers la droite en premier lieu, s'il y a toujours un obstacle, il effectuera un demi-tour, et enfin, s'il y a toujours un obstacle il effectuera un ¼ de tour vers la gauche. La variable obs étant la variable liée a la présence d'un obstacle.

Main Code

Voilà le code total actuel, certaines modifications sur les fonctions ont été réalisée pour pouvoir utiliser le capteur optique.

```
#include "mbed.h"

DigitalOut S2 (p11);                    //sortie pour bon fonctionnement a 0
Serial device (p9, p10);                //ports série tx, rx
AnalogIn capt (p15);                    //entrée capteur optique
Serial pc(USBTX, USBRX);
Serial blue(p13,p14);

char rblue = 'q';
int manuel = 1;
int obs;
int start ;
int gauche ;
int droite ;
int first = 0;

void tourner_gauche()                                   //tourner de 1/8 a gauche
{
        device.putc(128);
        device.putc(1);
        device.putc (30);
        device.putc((128 + 1 + 30) & 127);
        device.putc(128);
        device.putc(4);
        device.putc (36);
        device.putc((128 + 4 + 36) & 127);
        wait(1);
        device.putc(128);
        device.putc(1);
        device.putc (0);
        device.putc((128 + 1 + 0) & 127);
        device.putc(128);
        device.putc(4);
        device.putc (0);
```

```
35          device.putc((128 + 4 + 0) & 127);
36          gauche = 1;
37          droite = 0;
38  }
39
40  void tourner_droite()                                   //tourner de 1/8 a droite
41  {
42          device.putc(128);
43          device.putc(0);
44          device.putc (34);
45          device.putc((128 + 0 + 34) & 127);
46          device.putc(128);
47          device.putc(5);
48          device.putc (30);
49          device.putc((128 + 5 + 30) & 127);
50          wait(1);
51          device.putc(128);
52          device.putc(0);
53          device.putc (0);
54          device.putc((128 + 0 + 0) & 127);
55          device.putc(128);
56          device.putc(5);
57          device.putc (0);
58          device.putc((128 + 5 + 0) & 127);
59          droite = 1;
60          gauche = 0;
61          first = 1;
62  }
63
64  void demi_tour()                                    //demi tour
65  {
66          device.putc(128);
67          device.putc(1);
```

```
        device.putc (30);
        device.putc((128 + 1 + 30) & 127);
        device.putc(128);
        device.putc(4);
        device.putc (30);
        device.putc((128 + 4 + 30) & 127);
        wait(5);
        device.putc(128);
        device.putc(1);
        device.putc (0);
        device.putc((128 + 1 + 0) & 127);
        device.putc(128);
        device.putc(4);
        device.putc (0);
        device.putc((128 + 4 + 0) & 127);
}

void reculer()
{
    device.putc(128);
    device.putc(1);
    device.putc (48);
    device.putc((128 + 1 + 48) & 127);
    device.putc(128);
    device.putc(5);
    device.putc (50);
    device.putc((128 + 5 + 50) & 127);
}


void arret_urgence()
{
    S2 = 1;
}
```

```
102
103 void avancer(float speed)
104 {
105     if(speed > 0.3f)
106     {
107         device.putc(128);
108         device.putc(0);
109         device.putc (0);
110         device.putc((128 + 0 + 0) & 127);
111         device.putc(128);
112         device.putc(4);
113         device.putc (0);
114         device.putc((128 + 4 + 0) & 127);
115         obs = 1;
116     }
117     else
118     {
119         device.putc(128);
120         device.putc(0);
121         device.putc (48);
122         device.putc((128 + 0 + 48) & 127);
123         device.putc(128);
124         device.putc(4);
125         device.putc (50);
126         device.putc((128 + 4 + 50) & 127);
127     }
128 }
129
130 void avancerb()
131 {
132     device.putc(128);
133     device.putc(0);
134     device.putc (48);
```

```
135     device.putc((128 + 0 + 48) & 127);
136     device.putc(128);
137     device.putc(4);
138     device.putc (50);
139     device.putc((128 + 4 + 50) & 127);
140 }
141
142 void arreter()
143 {
144     device.putc(128);
145     device.putc(0);
146     device.putc (0);
147     device.putc((128 + 0 + 0) & 127);
148     device.putc(128);
149     device.putc(4);
150     device.putc (0);
151     device.putc((128 + 4 + 0) & 127);
152 }
153
154 void init()
155 {
156     S2 = 0;
157     device.putc(0XAA);
158     start = 0;
159     gauche = 0;
160     droite = 0;
161 }
162
163 int main()
164 {
165     init();
166     blue.baud(57600);
167
```

```
168     while(1)
169     {
170         if(blue.readable())
171         {
172             rblue = blue.getc();
173             printf("%c",rblue);
174 
175         }
176         if(rblue == 'q')
177         {
178            manuel = 0;
179         }
180         if(rblue == 'm')
181         {
182             manuel = 1;
183         }
184         if(manuel == 0)
185         {
186             if (obs == 0)
187             {
188                 avancer(capt);
189             }
190             else if ( obs == 1 && ( gauche == 1 | first== 0))
191             {
192                 tourner_droite();
193                 tourner_droite();
194                 obs = 2;
195             }
196             else if ( obs == 1 && ( droite == 1 ))
197             {
198                 tourner_gauche();
199                 tourner_gauche();
200                 obs = 2;
```

```
            }
            else if ((capt > 0.3f) && ( obs == 2))
            {
                demi_tour();
                obs = 0;
            }
            else obs = 0;
        }
            if(manuel == 1)
            {
                if(rblue == 'a')
                {
                    avancerb();
                }
                if(rblue == 'g')
                {
                    tourner_gauche();
                }
                if(rblue == 'd')
                {
                    tourner_droite();
                }
                if(rblue == 'r')
                {
                    reculer();
                }
                if(rblue == 's')
                {
                    arreter();
                }
                if(rblue == 't')
                {
                    demi_tour();
                }

                if(rblue == 'u')
                {
                    arret_urgence();
                }
            }
    }
}

```

Conclusion

Ce projet nous permet de nous familiariser avec Mbed et la gestion de la liaison numérique-analogique, et nous permet de voir que l'on peut facilement créer son propre robot, cependant le coût de ce robot n'est pas à négliger car :

_SRF10 : 32$

_Sabertooth 2x12 : 82$

_ Dagu Wild Thumper 4WD : 175$

_ Sharp GP2Y0A21YK0F : 11$

_ LPC1768 : 60$

Ce qui fait un total aux alentours des 360$

Le projet n'est pas encore fini car il manque encore la bonne gestion des capteurs et le pilotage 100% autonome du robot, cependant, nous estimons notre avancée a 60% par rapport au cahier des charges.