BEI NextHEV 2017-2018

Rapport technique :

Évolution de la maquette

Ce rapport technique concerne les différentes évolutions liées à la maquette présente au laboratoire Laplace. Le développement de ces évolutions se décompose en deux parties : les **tests concernant le bus CAN B**, et l'**amélioration de la simulation** proposée par l'équipe 2016-2017.

I. Tests du bus CAN B

Jusqu'à présent, seul le bus CAN A était utilisé. Nous avons donc cherché à utiliser le bus CAN B. Après avoir correctement installé le driver des bus CAN sur les deux PC (voir Tutorial Linux machine 2017-2018), nous avons utilisé les **fonctions *recvcan* et *sendcan***, disponibles dans le **dossier « Janus-CAN-Linux-driver-BETA/application »**. Leur documentation est disponible en exécutant dans un terminal « *./recvcan -h* » et « *./sendcan -h* ».

Afin de tester la communication via l'un des deux bus (par exemple le CAN B), il suffit de brancher le CAN report au câble (voir Tutorial CAN-report 2017-2018), et d'exécuter la commande « *./sendcan canB 12345* » : le PC envoie alors le nombre 12345 sur le bus CAN B. Si tout fonctionne, le message devrait apparaître dans la fenêtre du CAN report.
Il est alors possible de tester la communication entre les deux PC (par exemple via le CAN A), en réalisant les étapes suivantes :
- sur le premier PC, exécuter la commande « *./recvcan canA* » : le PC attend alors de recevoir une trame sur le bus CAN A
- sur le second PC, exécuter la commande « *./sendcan canA 12345* » : le PC envoie alors le nombre 12345 sur le bus CAN A
Si tout fonctionne, le message devrait devrait apparaître sur le terminal du premier PC.

Nous avons donc testé le bus CAN B avec le CAN report, sans succès. En effet, **il semblerait que la carte Janus du PC 2 présente un défaut au niveau du bus CAN B**. Nous avons suivi la démarche décrite par le **fichier texte *Démarche_test_CAN.txt*** pour arriver à cette conclusion. Ainsi, **seul le bus CAN A est fonctionnel sur les PC 1 et 2**. Lorsque le PC 3 sera mis à disposition, il sera nécessaire de tester le bon fonctionnement de la carte Janus. Cette démarche nous a également permis d'observer que **l'interrupteur côté CAN report doit forcément être sur ON pour pouvoir recevoir un message**.

Afin de faciliter la compréhension du fichier Démarche_test_CAN.txt, voici quelques informations :
- Les **cartes Janus**, qui permettent l'interface CAN, sont **configurées à l'aide de jumpers** placés pour définir notamment l'**adresse du bus CAN**, les **interruptions associées**, ainsi que la **terminaison de ligne**. Ainsi, le positionnement des jumpers concernant l'adresse du bus CAN et les

interruptions associées doit correspondre aux adresses et interruptions entrées comme argument dans le fichier « install.sh » permettant d'installer le driver des bus CAN sur chaque PC. La documentation liée à ces jumpers est disponible proche de la maquette.
- La maquette présente deux câbles (un pour chaque bus CAN). Les terminaisons de ligne, nécessaires à la communication, sont gérées différemment selon le câble utilisé. **Dans le cas du gros câble, leur activation est gérée par la commutation des interrupteurs oranges** situés sur chaque fiche du câble. De ce fait, **les jumpers correspondants doivent être absents**. **Dans le cas du petit câble, ce sont les jumpers placés sur la carte Janus qui gèrent la terminaison**.

Si rien n'a changé depuis l'écriture de ce rapport, la configuration actuelle est la suivante :
- Le **bus CAN A** est relié par **le plus petit des câbles**, tandis que le **bus CAN B** est relié par **le plus gros**.
- Les **jumpers sur les deux cartes Janus** sont placés aux **mêmes endroits**.
- Les **jumpers** concernant la **terminaison de ligne** pour le **CAN A** sont **présents**. Ceux pour le **CAN B** sont **absents**.
- Les **jumpers** concernant la **configuration de l'adresse** des bus CAN sont placés de sorte à avoir une adresse de **0xD7000** pour le CAN A, et 0xD7200 (= 0xD7000 + 0x200) pour le CAN B.
- Les **jumpers** concernant la **configuration des interruptions** des bus CAN sont placés de sorte à avoir l'**interruption 5 pour le CAN A**, et l'i**nterruption 7 pour le CAN B**.
- La **configuration logicielle** des deux PC **correspond** aux adresses et interruptions ci-dessus.

II. Amélioration de la simulation proposée par l'équipe 2016-2017

Nous sommes partis de la **version v1C de la simulation complète du véhicule**, développée par nos collègues au cours de la session 2016-2017. Pour rappel, cette simulation est composée du **superviseur** d'une part, et de la **plateforme** d'autre part. Connaissant le profil de vitesse souhaité, le superviseur envoie la commande à la plateforme via le bus CAN A. Puis la plateforme calcul l'état suivant du système, et envoie au superviseur la vitesse du véhicule résultante, toujours via le bus CAN A. Nous avons donc utilisés les codes présents dans les **dossiers « Simulateur_Platform_v1C »** et « **Superviseur_Platform_v1C ».**

Nous avons modifié les codes proposés, de sorte à réaliser deux objectifs :
- **pouvoir utiliser facilement l'un ou l'autre des deux bus CAN (version v2a)**
- **envoyer les données sur le bus CAN en virgule fixe (versions v2b et v2c)**

Le **fonctionnement de la dernière version v2c** est décrit ci-après. Veuillez lancer dans un premier temps la plateforme, puis dans un second temps le superviseur.
- Le **superviseur** envoie à la plateforme les **consignes de couples** pour les axes avant et arrière à travers le **bus CAN**.
- Ces consignes sont converties en **16 bits non signé**, mais sont **transmises en tant que données ASCII** sur le bus CAN. Ce sont donc au maximum 5*16 = **80 bits et non 16 bits qui transitent** à chaque fois.
- La **plateforme** reçoit les **consignes en tant que données ASCII**, et les **convertit en 16 bits non signé, puis en flottant**.
- La **plateforme** effectue alors un **pas de calcul**, afin de déterminer les **vitesses de la plateforme et des roues** résultantes.

- La **plateforme** envoie alors au superviseur ces **vitesses** à travers le **bus CAN**.
- Comme pour les consignes, ces vitesses sont converties en **16 bits non signé**, mais sont **transmises en tant que données ASCII** sur le bus CAN.
- Le **superviseur** reçoit les **vitesses en tant que données ASCII**, et les **convertit en 16 bits non signé, puis en flottant**.

Plusieurs défauts sont à noter concernant cette dernière version v2c :
- Il semble que l'**identificateur de certaines trames** reçues par l'un ou l'autre des PC **ne corresponde pas à l'identificateur attendu**, ce qui provoque la perte de la donnée. Ce défaut est hérité de la version de départ, à savoir la version v1C.
- Les **données qui transitent sur le bus CAN** ne sont pas en 16 bits mais en **80 bits**, puisque les données sont converties en **caractères ASCII**.
- Les **programmes d'envoi et de réception des données sur le bus CAN** ne prennent en compte **que des entiers non signés (du type uint16_t)**.

Ces trois défauts constituent des axes d'amélioration du programme. Néanmoins, il s'avère fonctionner sans problème.

Il est à noter que **seules les modifications de la dernière version sont précisées**. Ainsi, l'ensemble des modifications faites entre la version v1C et la version v2c dont décrites **en annexes en rouge**.

1. Version v2a : Permutation entre les deux bus CAN

Pour réaliser cette modification, nous avons édité les fichiers suivants :

- « sendFloat.cpp » et « sendFloat.h » (présents dans les deux dossiers)

- « Simulateur_Platform_v1C/fils2.cpp »

- « Superviseur_Platform_v1C/fils1.cpp »

Concernant les fichiers « **sendFloat.cpp » et « sendFloat.h »**, nous avons modifié le prototype de la fonction afin qu'il prenne en argument le bus CAN sur lequel écrire. L'ancien prototype était :

*float *sendFloat(int identificateur, float nombre)*

Le **nouveau prototype** est :

float *sendFloat(int identificateur, float nombre, char *port)

Les fichiers « Simulateur_Platform_v1C/fils2.cpp » et « Superviseur_Platform_v1C/fils1.cpp » ont alors été modifiés en conséquence.

Ainsi, cette version donne la **possibilité de choisir quel bus CAN utiliser** en modifiant les fichiers suivants :

- ./platform_v1C/pere.cpp ligne 25

- ./platform_v1C/fils2.cpp ligne 19

- ./superviseur_v1C/fils1.cpp ligne 11

- ./superviseur_v1C/fils2.cpp ligne 11

2. Version v2b : Envoie des données en virgule fixe

Nous avons développé une **classe *FixedPoint*** permettant d'envoyer les données sur 16 bits, signé ou non signé. Les deux fichiers qui décrivent cette classe sont « **fixedPoint.cpp » et « fixedPoint.hpp »**. Cette nouvelle classe dispose des **quatre méthodes** suivantes :

- ***uint16_t to_can_u16(float val, float max, float min)*** : prend en argument la valeur à transmettre, sa borne supérieure et sa borne inférieure, et retourne le non signé 16 bits envoyé sur le bus CAN

- ***int16_t to_can_s16(float val, float max, float min)*** : prend en argument la valeur à transmettre, sa borne supérieure et sa borne inférieure, et retourne le signé 16 bits envoyé sur le bus CAN

- ***float from_can_u16(uint16_t n, float max, float min)*** : prend en argument le non signé 16 bits reçu sur le bus CAN, sa borne supérieure et sa borne inférieure, et retourne le flottant correspondant

- ***float from_can_s16(int16_t n, float max, float min)*** : prend en argument le signé 16 bits reçu sur le bus CAN, sa borne supérieure et sa borne inférieure, et retourne le flottant correspondant

Un **programme de test** est également disponible, afin de pouvoir vérifier le bon fonctionnement de cette classe. Ce programme de test est décrit par le fichier « **main.cpp »**.

Nous avons donc intégré cette classe à la version v2a décrite ci-dessus, ce qui nous à amené à modifier les fichiers « **sendFloat.cpp » et « recvFloat.cpp** ». En effet, afin d'être en accord avec la classe FixedPoint, le prototype de sendFloat devait prendre comme argument un 16 bits non signé au lieu d'un flottant, et la fonction recvFloat devait traiter un 16 bits non signé au lieu d'un flottant. Nous avons ensuite modifier l'ensemble des fichiers afin d'assurer le bon fonctionnement.

3. Version v2c : Envoie des données en virgule fixe (nettoyé)

Cette version a **exactement le même fonctionnement que la version v2b**, **modulo des printf** qui apparaissent ou disparaissent. Le code a été **brièvement commenté**, et les différents **fichiers restructurés**. Les **fichiers** « sendFloat.cpp », « sendFloat.h », « recvFloat.cpp » et « recvFloat.h » ont respectivement été **renommés** en « sendInt16.cpp », « sendInt16.h », « recvInt16.cpp » et « recvInt16.h ». L'ensemble des modifications est décrit ci-après.

supervisor_v2c/main.cpp

```
/*************************************************************
***************    Main supervisor     ***************
*************************************************************/

#include <stdio.h>

#include "pere.h"
#include "fils1.h"
#include "fils2.h"

#define TAILLE_MESSAGE 256 /* Correspond à la taille de la chaîne à lire et à écrire */

/* Définition du pipe en dehors du main pour qu'il ne dépende pas du main.
 * Si on le crée dans le main, il ne pourra pas être vu par les fils ! */
int pipe_1[2];

int main(void)
{
    pid_t pid_fils1; // Déclaration du processus fils1
    pid_t pid_fils2; // Déclaration du processus fils2

    char messageLire[TAILLE_MESSAGE], messageEcrire[TAILLE_MESSAGE];

    pipe(pipe_1); //Pour communication entre les processus

    pid_fils1 = fork(); // Création du fils1
    if(pid_fils1 == -1)
    {
        fprintf(stderr, "Erreur de création du processus1.\n");
        return 1;
    }
    if(pid_fils2 == -1)
    {
        fprintf(stderr, "Erreur de création du processus2.\n");
        return 1;
    }

    if(pid_fils1 == 0) {
        //On est dans le fils1
        //Envoie la consigne dans le bus CAN
        fils1();
    }
    else{
        pid_fils2 = fork(); // Création du fils2
        if(pid_fils2 == -1)
        {
            fprintf(stderr, "Erreur de création du processus2.\n");
            return 1;
        }
```

```
        if(pid_fils2 == 0){
            // On est dans le fils 2
            // Réception et stockage de la sortie
            fils2();
        }
        else{
            //On est dans le pere
            //Génération de la consigne
            pere();
        }
    }

    return 0;
}
```

supervisor_v2c/pere.cpp (instead of consigne.cpp)

```
/*************************************************
 * ******************   Père  ****************
 *************************************************/

#include "pere.h"

#define TAILLE_MESSAGE 256

int pere() {

   char messageLire[TAILLE_MESSAGE], messageEcrire[TAILLE_MESSAGE];

   extern int pipe_1[2];          //Pour communication entre les processus

   //printf("\nFermeture de la sortie tube dans le père (pid = %d).\n",getpid());
   close(pipe_1[0]);

   float fa_tq;

   int ii=0; //initialisation pour la sauvegarde des points (ii=0 pour effacer ii=1 pour ecrire à la suite)

   for (int i=0;i<1000;i++){

      printf("Début de l'envoi des consignes de couples au fils1 par le père (pid = %d)\n",getpid());

      //entrée fa_tq
      if (i<350){
         sprintf(messageEcrire, "2500");
      }
      else{
         sprintf(messageEcrire, "100");
      }
      fa_tq=atof(messageEcrire);
      saveData(fa_tq,"fa_tq.txt",ii);

      printf("Le père envoie au fils1 : \"%s\".\n",messageEcrire);
      write(pipe_1[1], messageEcrire, TAILLE_MESSAGE);

      //entrée ra_tq
      if (i<350){
         sprintf(messageEcrire, "500");
      }
      else{
         sprintf(messageEcrire, "50");
      }
      fa_tq=atof(messageEcrire);
      saveData(fa_tq,"ra_tq.txt",ii);

      printf("Le père envoie au fils1 : \"%s\".\n",messageEcrire);
      write(pipe_1[1], messageEcrire, TAILLE_MESSAGE);
```

```
        printf("Fin de l'envoi des consignes de couples au fils1 par le père (pid = %d)\n\n",getpid());

        usleep(10000); //Horloge de 10 ms
        ii=1;
    }
    wait(NULL) ;
    return 0;
}
```

supervisor_v2c/pere.h (instead of consigne.h)

```
#ifndef PERE_H
#define PERE_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

#include "saveData.h"

using namespace std;

int pere();

#endif
```

supervisor_v2c/fils1.cpp

```
/*************************************************
 * ******************  Fils 1      ****************
 *************************************************/

#include "fils1.h"

#define TAILLE_MESSAGE 256

int fils1()
{
   char can[15] = "canA";

   char messageLire[TAILLE_MESSAGE], messageEcrire[TAILLE_MESSAGE];
   extern int pipe_1[2];

   //printf("Fermeture de l'entrée tube dans le fils1 (pid = %d).\n",getpid());
   close(pipe_1[1]);

   uint16_t consi;

   FixedPoint fixed_point;

   while(1) {
      printf("Début de l'envoi des consignes à la plateforme par le fils1 (pid = %d)\n",getpid());

      //entrée fa_tq
      read(pipe_1[0], messageLire, TAILLE_MESSAGE);
      consi = fixed_point.to_can_u16(atof(messageLire), 5000.0, 0.0);
      sendInt16(1,consi,can);

      //entrée ra_tq
      read(pipe_1[0], messageLire, TAILLE_MESSAGE);
      consi = fixed_point.to_can_u16(atof(messageLire), 1000.0, 0.0);
      sendInt16(2,consi,can);

      printf("Fin de l'envoi des consignes à la plateforme par le fils1 (pid = %d)\n\n",getpid());
   }

   return 0;
}
```

supervisor_v2c/fils1.h

```
#ifndef FILS1_H
#define FILS1_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
```

```
#include "sendInt16.h"
#include "fixedPoint.hpp"

using namespace std;

int fils1();

#endif
```

supervisor_v2c/fils2.cpp

```
/*************************************************
 * ******************  Fils 2      **************
 *************************************************/

#include "fils2.h"

int fils2()
{
   char can[15] = "canA";

   int ii=0; // initialise le fichier de sauvegarde pour ii=0

   float sortie_modele_pla_speed,sortie_modele_pla_N ;
   uint16_t pla_speed_u16, pla_N_u16;

   FixedPoint fixed_point;

   while(1){
      printf("Début de la réception des vitesses par le fils2 (pid = %d).\n",getpid());

      //reception de la vitesse plateforme
      pla_speed_u16 = recvInt16(1,can);
      sortie_modele_pla_speed = fixed_point.from_can_u16(pla_speed_u16,130.0,0.0);

      //reception de la vitesse des roues
      pla_N_u16 = recvInt16(2,can);
      sortie_modele_pla_N = fixed_point.from_can_u16(pla_N_u16,1100.0,0.0);

      printf("Fin de la réception des vitesses par le fils2 (pid = %d).\n",getpid());

      saveData(sortie_modele_pla_speed,"pla_speed.txt",ii);
      saveData(sortie_modele_pla_N,"pla_N.txt",ii);

      printf("Écriture dans le fichier texte terminée par le fils 2 (pid = %d) .\n",getpid());
      ii=1;
   }

   return 0;
}
```

supervisor_v2c/fils2.h

```
#ifndef FILS2_H
#define FILS2_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

#include "recvInt16.h"
```

```
#include "saveData.h"
#include "fixedPoint.hpp"

using namespace std;

int fils2();

#endif
```

platform_v2c/main.cpp

```
/*************************************************
 * ******************  Main    *****************
 *************************************************/

#include <stdio.h>

#include "pere.h"
#include "fils1.h"
#include "fils2.h"

#define TAILLE_MESSAGE 256 /* Correspond à la taille de la chaîne à lire et à écrire */

int pipe_p_1[2]; // Déclaration du pipe entre le père et le fils1
int pipe_1_2[2]; // Déclaration du pipe entre le fils1 et le fils2

int main(void)
{
   pid_t pid_fils1; // Déclaration du processus fils1
   pid_t pid_fils2; // Déclaration du processus fils2

   pipe(pipe_p_1); // Création du pipe entre le père et le fils1
   pipe(pipe_1_2); // Création du pipe entre le fils1 et le fils2

   char messageLire[TAILLE_MESSAGE], messageEcrire[TAILLE_MESSAGE];

   pid_fils1 = fork(); // Création du fils1
   if(pid_fils1 == -1)
   {
      fprintf(stderr, "Erreur de création du processus fils1.\n");
      return 1;
   }
   if(pid_fils2 == -1)
   {
      fprintf(stderr, "Erreur de création du processus fils2.\n");
      return 1;
   }

   if(pid_fils1 == 0)
   {
      // On est dans le fils1
      fils1();
   }
   else
   {
      pid_fils2 = fork(); // Création du fils2
      if(pid_fils2 == -1)
      {
         fprintf(stderr, "Erreur de création du processus fils2.\n");
         return 1;
      }
```

```
        if (pid_fils2 == 0)
        {
            // On est dans le fils2
            fils2();
        }
        else {
            // On est dans le père
            pere();
        }
    }

    return 0;
}
```

platform_v2c/pere.cpp

```
/************************************************
 * ******************    Père       ***************
 ************************************************/

/* Père, qui récupère les nouvelles consignes de couple envoyées par le superviseur sur le bus CAN désiré */

#include "pere.h"

#define TAILLE_MESSAGE 256 /* Correspond à la taille de la chaîne à lire et à écrire */

int pere(){

   char can[15]="canA";

   char messageLire[TAILLE_MESSAGE], messageEcrire[TAILLE_MESSAGE];
   extern int pipe_p_1[2];

   //printf("\nFermeture de la sortie tube p_1 dans le père (pid = %d).\n",getpid());
   close(pipe_p_1[0]);

   float consigne;

   FixedPoint fixed_point;

   while(1) {
      printf("Début de l'envoi des consignes de couples au fils1 par le père (pid = %d)\n",getpid());

      //entree fa_tq
      consigne = fixed_point.from_can_u16(recvInt16(1,can), 5000.0, 0.0);
      sprintf(messageEcrire, "%f",consigne);
      printf("Le père envoie le message suivant au fils1 : \"%s\".\n",messageEcrire);
      write(pipe_p_1[1], messageEcrire, TAILLE_MESSAGE);

      //entree ra_tq
      consigne = fixed_point.from_can_u16(recvInt16(2,can), 1000.0, 0.0);
      sprintf(messageEcrire, "%f",consigne);
      printf("Le père envoie le message suivant au fils1 : \"%s\".\n",messageEcrire);
      write(pipe_p_1[1], messageEcrire, TAILLE_MESSAGE);

      printf("Fin de l'envoi des consignes de couples au fils1 par le père (pid = %d)\n\n",getpid());
   }

   return 0;
}
```

platform_v2c/pere.h

```
#ifndef PERE_H
#define PERE_H

#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

#include "recvInt16.h"
#include "fixedPoint.hpp"

using namespace std;

int pere();

#endif
```

platform_v2c/fils1.cpp

```
/*************************************************
 * ******************   Fils 1      **************
 *************************************************/

/* Fils 1, qui effectue le pas de simulation de la plateforme après avoir reçu les nouvelles consignes du père */

#include "fils1.h"

#define TAILLE_MESSAGE 256 /* Correspond à la taille de la chaîne à lire et à écrire */

int fils1() {

   char messageLire[TAILLE_MESSAGE], messageEcrire[TAILLE_MESSAGE];
   extern int pipe_p_1[2];
   extern int pipe_1_2[2];

   //printf("Fermeture de l'entrée tube p_1 dans le fils1 (pid = %d).\n",getpid());
   close(pipe_p_1[1]);
   //printf("Fermeture de la sortie tube 1_2 dans le fils1 (pid = %d).\n",getpid());
   close(pipe_1_2[0]);

   platform_initialize();

   while(1) {
      printf("Début de l'exécution d'un pas de simulation par le fils1 (pid = %d)\n",getpid());

      read(pipe_p_1[0], messageLire, TAILLE_MESSAGE);
      platform_U.FA_TQ=atof(messageLire);
      //printf("Le fils1 a reçu la consigne FA_TQ suivante du père : \"%s\".\n", getpid(),
messageLire);

      read(pipe_p_1[0], messageLire, TAILLE_MESSAGE);
      platform_U.RA_TQ=atof(messageLire);
      //printf("Le fils1 a reçu lla consigne RA_TQ suivante du père : \"%s\".\n", getpid(),
messageLire);

      rt_OneStep();

      sprintf(messageEcrire, "%f",platform_Y.PLA_SPEED);
      write(pipe_1_2[1], messageEcrire, TAILLE_MESSAGE);

      sprintf(messageEcrire, "%f",platform_Y.PLA_N);
      write(pipe_1_2[1], messageEcrire, TAILLE_MESSAGE);

      printf("Fin de l'exécution d'un pas de simulation par le fils1 (pid = %d)\n\n",getpid());
   }

   platform_terminate();
```

```
    return 0;
}
```

platform_v2c/fils1.h

```
#ifndef FILS1_H
#define FILS1_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

#include "platform.h"
#include "rtwtypes.h"
#include "ert_main.h"

using namespace std;

int fils1();

#endif
```

platform_v2c/fils2.cpp

```
/************************************************
 * ******************  Fils 2      **************
 ************************************************/

/* Fils 2, qui envoie les nouvelles vitesses (vitesse du véhicule et vitesse de la roue) au superviseur par le bus CAN souhaité */

#include "fils2.h"

#define TAILLE_MESSAGE 256 /* Correspond à la taille de la chaîne à lire et à écrire */

int fils2() {

   char can[15] = "canA";

   char messageLire[TAILLE_MESSAGE], messageEcrire[TAILLE_MESSAGE];
   extern int pipe_1_2[2];

   //printf("Fermeture de l'entree tube 1_2 dans le fils2 (pid = %d) .\n",getpid());
   close(pipe_1_2[1]);

   float pla_speed;
   float pla_n;

   FixedPoint fixed_point;

   while(1) {
      printf("Début de l'envoi des vitesses au superviseur par le fils2 (pid = %d)\n",getpid());

      // Vitesse du véhicule
      read(pipe_1_2[0], messageLire, TAILLE_MESSAGE);
      pla_speed = atof(messageLire);
      printf("Le fils2 envoie la vitesse du véhicule km/h= %f au superviseur\n",pla_speed);
      sendInt16(1,fixed_point.to_can_u16(pla_speed, 130.0, 0.0),can);

      // Vitesse des roues
      read(pipe_1_2[0], messageLire, TAILLE_MESSAGE);
      pla_n = atof(messageLire);
      printf("Le fils2 envoie la vitesse de la roue tr/min = %f au superviseur\n",pla_n);
      sendInt16(2,fixed_point.to_can_u16(pla_n, 1100.0, 0.0),can);

      printf("Fin de l'envoi des vitesses au superviseur par le fils2\n (pid = %d)\n\n",getpid());
   }

   return 0;
}
```

platform_v2c/fils2.h

```
#ifndef FILS2_H
#define FILS2_H
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

#include "sendInt16.h"
#include "fixedPoint.hpp"

using namespace std;

int fils2();

#endif
```

sendInt16.cpp (instead of sendFloat.cpp)

```
/*************************************************
 * ******************   sendInt16   **************
 *************************************************/

/*/
Fonctionnement de sendInt16 :

Ouvre le port canA ou canB (argument en entrée de la fonction)
Récupère le uint16_t (ce nombre entier doit avoir au maximum 8 caractères, ce qui est bien le cas puisqu'il va jusqu'à 66535)
Le transforme en char
Envoie sur le CAN
/*/

#include "sendInt16.h"

extern int optind, opterr, optopt;

float *sendInt16(int identificateur,uint16_t nombre, char *port) {

    char appDeviceName[15];
    sprintf(appDeviceName,"/dev/%s",port);

    struct can_frame frame = {
        //can_id = 1,
    };
    int loopcount = 1, infinite = 0;
    int s, opt, ret, i, dlc = 0, rtr = 0, extended = 0;
    int verbose = 0,pattern = 0;

    short appRetVal, appWRetVal;
    char *optout = NULL;
    //char *optout = TRUE;
    FILE *TX;
    int appDevHandle;
    char ch;
    int count = 0, id = 1;
    unsigned long fcount = 1;

    void handler_fin (int unused);

    struct can_btr btr;
    struct sigaction sa_fin;                //Declaration pour terminaison (ctrl-C)
    pid_t pid;

    /*************************************************
     * ***********   Conversion int16_t ASCII     **********
     *************************************************/
    ostringstream os;
    os << nombre;
```

```
    string st=os.str();
    char CHAINE[st.size()];
    sprintf(CHAINE , "%d",nombre);

    //cout << "Size = " << s.size() <<endl;

    /*
     *    for (int i1=0 ; i1<s.size() ; i1++){
     *    int x = CHAINE[i1];
     *
     *    cout << "The character '" << CHAINE[i1] << "' has an ASCII code of " << std::hex << x <<
endl;
}
*/

    /*****************************************************************************
     * ***********   Mise en mémoire dans data  **********
     *****************************************************************************/

    /*
     * for (int i2=0 ; i2<s.size() ; i2+=2){
     *    int x1 = CHAINE[i2];
     *    int x2 = CHAINE[i2+1];
     *
     *    ostringstream os2;
     *    os2 << std::hex << x1 << std::hex << x2;
     *    string caracteres_s = os2.str();
     *    char const *caracteres_c=caracteres_s.c_str();
     *    cout << "caracteres_c = " << caracteres_c << endl;
     *    frame.data[dlc] = strtoul(caracteres_c, NULL, 0);
     *    cout << "frame.data[dlc] = " << frame.data[dlc] << endl;
     *    dlc++;
     *    if (dlc == 8)
     *        break;
     *
}
*/

    for (int i2=0 ; i2<st.size() ; i2++){
        int x1 = CHAINE[i2];
        //int x2 =CHAINE[i2+1];

        ostringstream os2;
        os2 << x1;
        //os2 << std::hex << CHAINE[i2] << CHAINE[i2+1];
        string caracteres_s = os2.str();
        char const *caracteres_c=caracteres_s.c_str();
        //cout << "caracteres_c = " << caracteres_c << endl;
        frame.data[dlc] = strtoul(caracteres_c, NULL, 0);
        //cout << "frame.data[dlc] = " << frame.data[dlc] << endl;
        dlc++;
        //cout << "dlc = " << dlc << endl;
```

```
    if (dlc == 8) break;
}
frame.can_dlc = dlc;
frame.can_id=identificateur;

/*****************************************************
 * ***********   Ouverture du port can      **********
 *****************************************************/
appDevHandle = open(appDeviceName , O_RDWR);
if(appDevHandle < 0) {
    printf("Device Open Error (%s)\n", appDeviceName, appDevHandle);
    exit(0);
} else {
    //printf("Device Opened par sendFloat (%s)\n", appDeviceName, appDevHandle);

    /*****************************************************
     * ***********   Envoie direct des données  **********
     *****************************************************/

    while (infinite || loopcount--) {
        if(pattern) {
            for (i = 0; i < frame.can_dlc; i++)
                frame.data[i] = frame.data[i] + 1;
            appWRetVal = ioctl(appDevHandle, SJA1000_IOCTRANS, (unsigned long)&frame);
        } else {

            appWRetVal = ioctl(appDevHandle, SJA1000_IOCTRANS, (unsigned long)&frame);
        }
        if (frame.can_id & CAN_EFF_FLAG) {
            printf("<0x%08x> ", frame.can_id & CAN_EFF_MASK);

        }
        else
            printf("<0x%03x> ", frame.can_id & CAN_SFF_MASK);
        dlc = frame.can_dlc;
        printf("[%d] ", dlc);

        for (i = 0; i < frame.can_dlc; i++) {
            printf("%02x ", frame.data[i]);
        }
        if (frame.can_id & CAN_RTR_FLAG)
            printf("remote request");
        printf("\n");
        usleep(2000);
    }
}
appRetVal = close(appDevHandle);
if(appRetVal == 0) {
    //printf("Device Closed par sendFloat \n", appRetVal);
} else {
    printf("Device Close Error\n", appRetVal);
}
```

```
    return 0;
}
```

sendInt16.h (instead of sendFloat.h)

```
#ifndef SENDINT16_H
#define SENDINT16_H

#include <string>
#include <iostream>   //Pour afficher les messages (cout)
#include <fstream>              //Pour la lecture des fichier en C++
#include <getopt.h>
#include <libgen.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <limits.h>
#include <stdint.h>
#include <sys/uio.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <sstream>              //Pour la conversion float  Ascii

#include "can.h"
#include "sja1000_ioctl.h"

using namespace std;

float *sendInt16(int identificateur,uint16_t nombre, char *port);

#endif
```

recvInt16.cpp (instead of recvFloat.cpp)

```
/*************************************************
 * ******************   recvInt16   ***************
 *************************************************/

/*/
Fonctionnement de recvInt16 :

Ouvre le port canA ou canB (argument en entrée de la fonction)
Rentre dans la boucle while(running)
Attend que quelqu'un parle sur le bus can
Dès qu'une trame est reçue, sauvegarde la trame dans data avec ioctl
Converti la trame en uint16_t
Renvoie le uint16_t
/*/

#include "recvInt16.h"

#define BUF_SIZ (255)

extern int optind, opterr, optopt;

static int         s = -1;
static int         running = 1;

static void sigterm(int signo) {
    running = 0;
}

uint16_t recvInt16(int identificateur,char *port){

    FILE *out = stdout;
    char *optout_recv = NULL;
    char *ptr;
    char buf[BUF_SIZ];
    int n = 0, err;
    int nbytes, i_recv, dlc_recv;
    int opt_recv, optdaemon = 0;
    uint32_t id_recv, mask;

    short appRetVal_recv;
    int appWRetVal_recv;
    char appDeviceName_recv[5];
    int appDevHandle_recv;
    unsigned long data[10];
    int fileflag = 0;
    FILE *RX;

    unsigned long fcount_recv = 1;

    struct can_frame *frame_recv;
```

```
sprintf(appDeviceName_recv,"/dev/%s",port);
//printf("appDeviceName_recv : %s\n",appDeviceName_recv);

signal(SIGPIPE, SIG_IGN);

struct option       long_options[] = {
    { "help", no_argument, 0, 'h' },
    { 0, 0, 0, 0},
};

if (optdaemon)
    daemon(1, 0);
else {
    signal(SIGTERM, sigterm);
    signal(SIGHUP, sigterm);
}

appDevHandle_recv = open(appDeviceName_recv , O_RDWR);
if(appDevHandle_recv < 0) {
    printf("Device Open Error (%s)\n", appDeviceName_recv, appDevHandle_recv);
    exit (0);
} else {
    //printf("Device Opened (%s) par recvInt16\n", appDeviceName_recv, appDevHandle_recv);

    /*************************************************
     * *********  Récupération des données   **********
     *************************************************/
    running = 1;
    while (running) {
        appWRetVal_recv = ioctl(appDevHandle_recv, SJA1000_IOCRECV, (unsigned long) data);
        if(appWRetVal_recv != 0)
        {
            if(data[0] == identificateur)
            {
                //Permet d'écrire dans la console ce qui est dans la trame reçue
                i_recv=0;
                if (data[i_recv] & CAN_EFF_FLAG){
                    printf("<0x%08x> ", data[i_recv++] & CAN_EFF_MASK);
                }
                else
                {
                    printf("<0x%03x> ", data[i_recv++] & CAN_SFF_MASK);
                }
                //(unsigned long) DATA_RECV = data[i_recv]
                dlc_recv = data[i_recv++];
                printf("[%d] ", dlc_recv);
                running=0;
                while(i_recv <= dlc_recv + 1) {
                    printf("%02x ", data[i_recv++]);
                }
                if (data[0] & CAN_RTR_FLAG)
```

```
                printf("remote request");
            printf("\n");

            running=0;
          }
          else{printf("L'identificateur ne correspond pas à la trame reçue\n");
            //while(1){}
          }
        }
    }
}

/*
for (int indice = 0 ; indice<10 ; indice++){
    // printf("<0x%03x> ", data[indice] & CAN_EFF_MASK);
    cout << "data[indice] = " << data[indice] << endl ;
}
*/

appRetVal_recv = close(appDevHandle_recv);
if(appRetVal_recv == 0) {
    //printf("Device Closed par recvFloat\n", appRetVal_recv);
} else {
    printf("Device Close Error - %d\n", appRetVal_recv);
}

/*************************************************
 **********  Conversion ASCII int16_t   **********
 *************************************************/

int taille=dlc_recv;
int i[taille-1];

char b[taille];

for (int ind=0 ; ind<taille ; ind++)
{
    i[ind]=data[ind+2]; //valeur en ascii
    b[ind]=(char)i[ind];//valeur du caractere
}
b[taille]='\0'; //caractère qui indique la fin de la chaine
char chaine[taille];

sprintf(chaine,"%s",b);
cout << "l'identificateur est : " << data[0] << endl;
cout << "la chaine vaut : " << chaine << endl;

uint16_t nombre = (uint16_t) atoi(chaine);

//     cout << "le float vaut avec cout : " << nombre << endl;
//    printf("Le float vaut avec printf : %f\n",nombre);
```

```
    return nombre;
}
```

recvInt16.h (instead of recvFloat.h)

```
#ifndef RECVINT16_H
#define RECVINT16_H

#include <string>
#include <errno.h>
#include <iostream>   //Pour afficher les messages (cout)
#include <fstream>              //Pour la lecture des fichier en C++
#include <getopt.h>
#include <libgen.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <limits.h>
#include <stdint.h>
#include <sys/uio.h>
#include <sys/ioctl.h>
#include <fcntl.h>

#include "can.h"
#include "sja1000_ioctl.h"

using namespace std;

static void sigterm(int signo);
uint16_t recvInt16(int identificateur,char *port);

#endif
```

fixedPoint.cpp (new file)

```
/* Source code to ensure the traffic on CAN bus is with 16 bits type.
 * Designed by Nicolas Berling
 * January 2018
*/

#include "fixedPoint.hpp"

#define NB_BITS 16

FixedPoint::FixedPoint() {
}

FixedPoint::~FixedPoint() {
}

uint16_t FixedPoint::to_can_u16(float val, float max, float min) {
    uint32_t n;

    q = max/(pow(2, NB_BITS) - 1);
    //printf("q = %f\n", q);
    n = (uint32_t) floor(val/q);
    //printf("n = %" PRIu32 "\n", n);

    if (n > pow(2, NB_BITS)) {
        n = (uint16_t) pow(2, NB_BITS);
    }
    else if (n < 0) {
        n = (uint16_t) 0;
    }
    else {
        n = (uint16_t) n;
    }

    return n;
}

int16_t FixedPoint::to_can_s16(float val, float max, float min) {
    float max_abs;
    int32_t n;

    max_abs = abs(max);
    if (abs(min) > max_abs) {
        max_abs = abs(min);
    }

    q = max_abs/(pow(2, NB_BITS-1) - 1);
    //printf("q = %f\n", q);
    n = (int32_t) floor(val/q);
    //printf("n = %" PRId32 "\n", n);
```

```
    if (n > pow(2, NB_BITS - 1) - 1) {
        n = (int16_t) pow(2, NB_BITS - 1) - 1;
    }
    else if (n < -pow(2, NB_BITS - 1)) {
        n = (int16_t) -pow(2, NB_BITS - 1);
    }
    else {
        n = (int16_t) n;
    }

    return n;
}

float FixedPoint::from_can_u16(uint16_t n, float max, float min) {
    float val;

    q = max/(pow(2, NB_BITS) - 1);
    //printf("q = %f\n", q);
    val = (float) n*q;

    return val;
}

float FixedPoint::from_can_s16(int16_t n, float max, float min) {
    float max_abs;
    float val;

    max_abs = abs(max);
    if (abs(min) > max_abs) {
        max_abs = abs(min);
    }

    q = max_abs/(pow(2, NB_BITS-1) - 1);
    //printf("q = %f\n", q);
    val = (float) n*q;

    return val;
}
```

fixedPoint.hpp (new file)

```
/* Source code to ensure the traffic on CAN bus is with 16 bits type.
 * Designed by Nicolas Berling
 * January 2018
*/

#ifndef FIXEDPOINT_H
#define FIXEDPOINT_H

#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include <math.h>
```

```
class FixedPoint {
public:
    FixedPoint();
    ~FixedPoint();

    uint16_t to_can_u16(float val, float max, float min);
    int16_t to_can_s16(float val, float max, float min);
    float from_can_u16(uint16_t n, float max, float min);
    float from_can_s16(int16_t n, float max, float min);

private:
    float q;
};

#endif
```