

Travaux pratiques de programmation

Polytech Clermont-Ferrand GE4a - Module INFO6

v33 - J. Laffont – M. James – J. Sérot

1. Introduction

On étudie dans cette série de TP une application de contrôle et commande, en temps réel, d'un véhicule, implanté sur un microcontrôleur à l'aide d'un exécutif temps réel. Pour des raisons de facilité de mise en œuvre, ce véhicule sera un véhicule virtuel, simulé sur ordinateur. La vue générale du système est donnée figure 1.

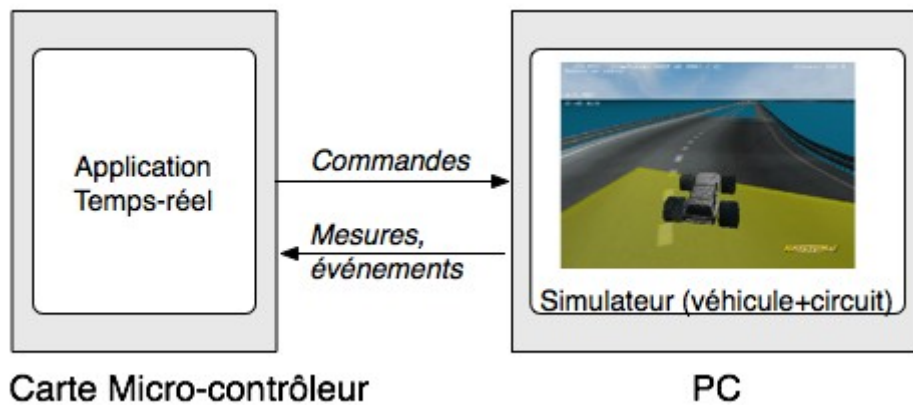


Figure 1 – Vue générale de l'application

L'objectif final du TP est de réaliser un programme permettant au véhicule d'effectuer trois tours de chaque circuit en un temps minimum. Pour cela, l'application devra assurer le guidage du véhicule dans son environnement, en produisant des commandes (vitesse et l'angle des roues) en fonction de mesures (vitesse effective, distance au mur, zone du circuit ...). Le fonctionnement de l'application devant être intégralement autonome. Plusieurs versions seront étudiées, par ordre croissant de complexité.

2. Description du matériel

La délimitation physique du système étudié est donnée sur la Figure 2.

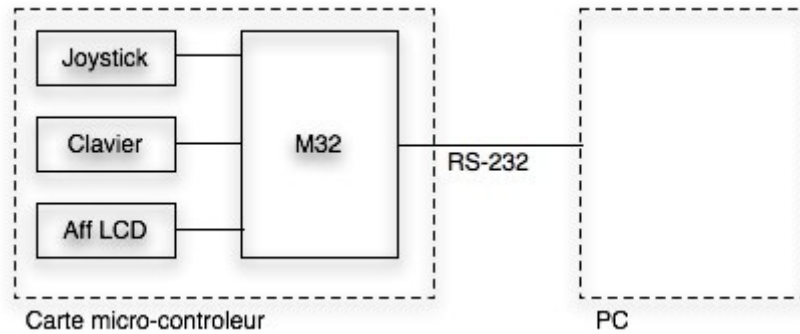


Figure 2 – Délimitation physique du système étudié

Le simulateur (véhicule+circuit) s'exécute sur un PC. Il est fourni.

L'application s'exécutera sur un microcontrôleur M32C (Renesas) avec le support du noyau M308/4 (utilisé en TD).

Le clavier, le joystick et l'afficheur LCD sont intégrés à la plateforme de développement.

Le simulateur et le microcontrôleur dialoguent via une liaison série RS-232.

Les routines d'accès bas niveau aux périphériques (afficheur LCD, clavier, Joystick) vous sont fournies (certaines ont été étudiées par ailleurs en GE3a).

La couche logicielle de communication avec le simulateur est intégrée au projet.

3. Périphériques présents sur la carte microcontrôleur

○ Clavier matricé

Il doit être connecté sur le port 10.

La fonction `clavier_init()` permet d'initialiser le clavier.

Le fonctionnement repose sur les interruptions : la routine d'interruption, `void itouche(void)` du fichier `clavier.c`, est activée lors de l'appui sur une touche. La routine dépose le code ASCII de la touche appuyée dans une queue de message (`QdmTouche`). Le code de la dernière touche appuyée est donc obtenu en lisant cette queue de message :

```
short code_touche;
vrcv_dtq(QdmTouche, &code_touche);
```

○ Afficheur LCD

Il fonctionne en mode 4 bits sur le port 3.

Vous pouvez utiliser les primitives suivantes (cf fichier `lcd.c`) :

- `void lcd_init()` : pour initialiser l'afficheur.

- void lcd_com(char c) : permet d'envoyer une commande à l'afficheur (sous la forme d'un caractère de contrôle)
- void lcd_putc(char c) : permet d'afficher un caractère à la position du curseur.
- void lcd_str(char *s) : permet d'afficher une chaîne de caractères.

○ Joystick

Une carte d'interface branchée sur le port 0 met à disposition 2 potentiomètres, 3 boutons poussoirs et 3 LEDs (Rouge, Jaune, Verte).

Il faut initialiser la direction du port 0 avec la valeur : 0b11100000 (ceci est réalisé par la fonction can_init()).

Les LEDs peuvent être allumées ou éteintes en utilisant les macros suivantes:

LED_R=1 ou LED_R=0 (idem avec LED_J et LED_V).

Pour lire la valeur d'un bouton poussoir, utiliser les macros: Bp_G, Bp_M, Bp_D.

Les valeurs analogiques des potentiomètres (sur 10 bits, entre 0 et 1023) peuvent être lues dans les registres ad00 et ad01.

4. Interface application-simulateur

Du point de vue de l'application, le dialogue avec le simulateur s'effectue via deux queues de messages¹ (cf fig.3) :

- l'application peut émettre des messages à destination du simulateur en les postant dans une queue de message d'identifiant CanTx;
- le simulateur, en réponse aux requêtes de l'application, ou spontanément, émet des messages à destination de l'application en les postant dans une queue de messages d'identifiant CanRx.
 - Le délai entre la réception de la demande et la réponse du simulateur n'est pas déterministe.

Ce mode de communication se rapproche autant que faire se peut de celui que l'on utiliserait pour dialoguer avec une partie opérative réelle via un bus de type CAN, par exemple.

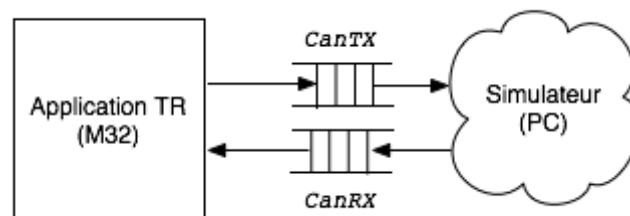


Figure 3 – Communication entre l'application et le simulateur

Le format des messages est inspiré de celui des trames circulant sur un bus de type CAN. Un

¹L'interface entre ces queues de messages et la liaison série est gérée par des tâches qui sont fournies; vous n'avez donc pas à vous en occuper.

message comprend trois champs :

- un champ id qui identifie le type de message;
- un champ rtr qui, s'il est positionné à 1, indique que le message est une requête destinée à déclencher une réponse (Request to Respond);
- un champ val qui contient, éventuellement, un paramètre pour le message.

Définition de type incluse dans « periph.h »

```
typedef union {  
    struct {  
        unsigned char id; /* 8 bits */  
        unsigned char rtr; /* 8 bits */  
        unsigned short val; /* 16 bits */  
    } data;  
    VP_INT msg /* 32 bits */  
} CanFrame;
```

La taille des messages est fixe (32 bits). L'encapsulation dans une union permet d'accéder aux différents champs d'un message tout en gardant la possibilité de le traiter comme un simple entier 32 bits (VP_INT) afin de se conformer aux prototypes des fonctions d'écriture et de lecture des queues de message (cf descriptif des primitives snd_dtq et rcv_dtq dans le manuel de référence du noyau).

Le champ id désigne l'organe concerné (tourelle, roues, direction ...) du véhicule. Par exemple, l'identificateur de la tourelle est 84 (ou 0x54, ou encore 'T'). La liste des identificateurs et leur signification est donnée en annexe 2.

Exemple d'utilisation

Pour commander la vitesse de rotation de la tourelle (12°/s), on écrira :

```
CanFrame comm;  
  
comm.data.id = 'T';  
comm.data.rtr = 0; // Indique une écriture  
comm.data.val = 120;  
snd_dtq(CanTx, comm.msg);
```

Pour lire la position angulaire de la tourelle (en 1/10 de degré) et placer la valeur lue dans la variable alpha on écrira :

```
CanFrame requete, reponse;  
  
requete.data.id = 'R';  
requete.data.rtr = 1; // Indique une requête de lecture  
snd_dtq(CanTx, requete.msg);  
  
rcv_dtq(CanRx, &reponse.msg);  
alpha=reponse.data.val;
```

5. Mise en œuvre de la manipulation

Démarrer le simulateur « simulateurV3.cbpd.exe », celui-ci ouvre une fenêtre contenant le rendu de la scène simulée. Une deuxième fenêtre contient les informations de communication, en particulier le nombre de messages émis et reçus par seconde par le simulateur.

Vous pouvez interagir avec le simulateur en utilisant les touches suivantes :

- 'P' → permet de changer de piste (verte, bleue, rouge, noire),
- '<espace>' → change le mode de course,
 - essais en cours : Le feu tricolore change toutes les secondes,
 - Attente Départ : le feu tricolore est rouge, les voitures doivent être à l'arrêt,
 - course → le feu passe au vert, la course peut débuter.
- 'I' → affiche l'état des capteurs de la voiture (ainsi que d'autres informations utiles à la mise au point de l'application),
- 'A','Z','E' → permettent de modifier la vitesse de simulation en, respectivement, arrêtant le temps, le ralentissant à 10 %, la touche 'E' permet un retour à la normale. Ces touches sont utiles pour déboguer certaines parties critiques du programme. Attention elles n'ont aucune influence sur votre application.
- 'I' Mode libre permet de placer des capteurs sur la piste ou de se déplacer librement.
- 's' mode suivi, la vue suit votre véhicule.

Le temps de course est comptabilisé à partir du passage sur la ligne de départ matérialisée par un capteur vert. Un éventuel retard pris par le véhicule lors du passage au vert du feu tricolore n'aura donc aucun effet sur le temps de course.

6. Personnalisation du simulateur

Il vous est possible de personnaliser votre circuit. Pour cela, vous pourrez ajouter les zones de couleurs et leur affecter une valeur.

En mode 'libre' touche 'I' vous pouvez ajouter un capteur en cliquant sur un endroit de la piste. Vous pourrez sélectionner la forme et la couleur du capteur et lui associer une valeur.

Si vous sélectionnez un capteur, vous pourrez le déplacer ou le faire tourner en utilisant le bouton droit de la souris. Si vous cliquez sur un capteur avec le bouton droit de la souris, vous pourrez le modifier ou le supprimer.

Pour quitter le mode de personnalisation, repasser en mode suivi touche 's'.

La configuration est sauvée dans un fichier dont le nom correspond au nom de votre véhicule. Le fichier n'est pris en compte que si le nom contient une '*' finale (non affichée).

Vous pouvez dès lors placer des capteurs de couleurs aux endroits opportuns et leur affecter des valeurs que vous traiterez dans votre programme. Les capteurs une fois détectés sont signalés dans les périphériques en lettre minuscule 'v','j','r','b','c' (vert,jaune,rouge, bleu, cyan). Le passage sur un capteur de couleur déclenche un événement immédiatement dont la valeur correspond à la couleur.

7. Déroulement des séances

Le TP se déroule sur 6 séances. L'objectif final est de réaliser une application permettant au véhicule d'effectuer trois tours de circuit en un temps minimum, et ce avec un taux d'occupation minimum pour le processeur. Afin d'atteindre cet objectif dans le temps imparti, il est essentiel d'adopter une démarche incrémentale, c.-à-d.. d'implanter et valider une à

une, séparément et progressivement les fonctionnalités. La technique consistant à écrire la totalité du code en croisant les doigts pour que « ça marche du premier coup » a très peu de chance de donner des résultats ici, ne serait-ce que parce qu'elle conduit en général à une explosion des temps de mise en point (debug). Un plan de travail vous est donc proposé. Ce plan de travail comprend à la fois un travail de préparation, à faire avant la séance, et un travail d'implantation et de validation pendant les séances.

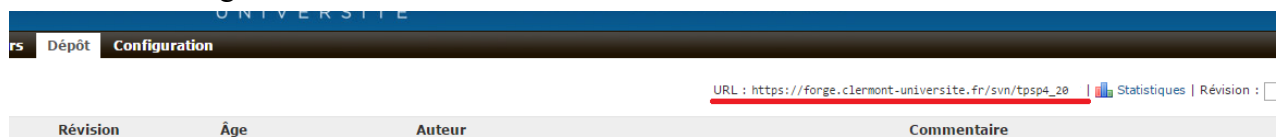
À chaque séance sera associée un ensemble d'objectifs ("ce qui doit marcher à la fin de la séance"). Le respect de ces objectifs entrera aussi en compte dans la notation finale.

8. Gestion des programmes :

Les codes sur lesquels vous allez travailler seront versionnés.

Veillez suivre seul et à votre rythme la procédure suivante pas à pas. La procédure est simple et détaillée, il vous est demandé de la suivre scrupuleusement et intelligemment.

1. Dans la suite, remplacer les X par l'information pertinente.
2. En opération préalable, veuillez vous connecter à la forge de l'université : <https://forge.clermont-universite.fr/> → projet → Polytech Ge Info6 20XX
3. S'assurer que vous avez été ajouté au dépôt du TP :
 - Vous devriez apparaître dans la liste des membres.
 - Vous pouvez sélectionner le projet dans la liste déroulante en haut à droite.
4. Sur le disque « travail » de l'ordinateur de Travaux Pratiques, créer sous la racine un répertoire `tp_info6_20XX_votre_nom` sans majuscules ni caractères accentués.
5. Ouvrez ce répertoire
6. Récupérez le projet type dans ce répertoire, en utilisant le menu contextuel (bouton droit de la souris),
 - Svn Checkout → Entrer l'adresse du projet type :
 - L'adresse du dépôt se trouve en haut à gauche de l'onglet dépôt du projet de la forge :



- elle est du type URL : `https://forge.clermont-universite.fr/svn/polytech-ge-info6-20XX/trunk/`
 - Copier et coller l'adresse du dépôt dans le champ url of repository
 - Utiliser le bouton parcourir à droite du champ d'adresse symbole (...) et sélectionner le répertoire `trunk/tp_info6`
 - Bien vérifier que `tp_info6` est sélectionné
 - Le répertoire de destination (checkout directory) doit être `tp_info6_20XX_votre_nom\tp_info6`
 - Valider le checkout (faire ok).
7. Vous allez développer votre programme dans une branche spécifique et personnelle :
 - Menu contextuel -> tortoise svn → branch/tag
 - Remplacer le champ « to path » avec `/branch/votre_nom`.
 - Comme message, mettez « création branche votre nom ».
 - Sélectionner « HEAD revision in repository »
 - Sélectionnez « create intermediate folders »
 - Sélectionnez « switch to new branch »
 - « ok ».

Après chaque question, mettez à jour votre programme sur le serveur :

- menu contextuel et SVN Commit,

- Mettez OBLIGATOIREMENT un message correspondant à l'état du programme ou aux améliorations apportées,
- Ne jamais faire de commit sans message explicatif (« réponse question2 », « correction fonction xxx »,...),
- Le contenu du commit doit absolument être vérifié avant de faire ok :
 - Le commit a-t-il bien au bon endroit ?
 - bon repository, bonne branche
 - Les fichiers concernés sont-ils les bons ?
 - Vérifiez qu'ils m'appartiennent et que j'ai effectivement modifié
 - Ne jamais commiter des fichiers pouvant être générés par l'outil
 - Ne jamais commiter de fichiers objets ou exécutables
 - Ne jamais commiter de fichiers de backup
 - Des fichiers sont-ils manquants ?
 - Les ajouter si nécessaire
 - La taille du commit est-elle raisonnable ?
 - Corresponds à une modification unitaire ?
 - Sinon sélectionner SI POSSIBLE un sous-ensemble et commiter en plusieurs étapes
- Faire une deuxième vérification du contenu du commit
 - Une fois effectué un commit restera « à vie » dans les journaux
 - Le contenu du commit sera sauvegardé à vie
 - Vérifier la taille des documents commités
 - En cas de doute demander à un enseignant de valider
- Si vous êtes sûr de vous, vous pouvez effectuer le commit.

Vous pouvez travailler sur vos programmes depuis un ordinateur personnel :

- Pour accéder à vos programmes versionnés :
 - Installer le logiciel libre TortoiseSvn
- Pour récupérer une copie de vos programmes sur une machine personnelle
 - Utiliser Svn Checkout
 - Utiliser l'adresse suivante :
<https://forge.clermont-universite.fr/svn/polytech-ge-info6-20XX>https://forge.clermont-universite.fr/svn/polytech-ge-info6-20XXtre_nom
- Vous pouvez mettre à jour vos programmes après correction sur votre machine :
 - SVN Commit.

9. Configuration du poste de travail

Lors de la première séance, si vous vous êtes connecté avec votre compte de l'école veuillez ajouter l'imprimante de la salle de TP.

Pour cela, ouvrir un Explorateur Windows et entrer l'adresse suivante \\172.16.23.205 puis entrée.

Cliquer deux fois sur l'imprimante LaserJet 6MP et/ou Hp Laserjet 1022 pour l'ajouter à votre configuration.

Sur le même ordinateur du réseau dans le répertoire tpinfo6 vous trouverez le simulateur à recopier sur votre disque travail, s'il n'est pas déjà présent.

10.Déroulement des courses

Lors de la dernière séance, une course sera organisée sur chacune des pistes. En fonction de l'ordre d'arrivée un bonus sera affecté aux étudiants ayant les meilleures performances.

Le protocole suivant sera appliqué pour chaque course. Tout étudiant n'étant pas en mesure de respecter ce protocole ne pourra participer à la course.

Les items en Gras sont réalisés par l'enseignant.

1. Les étudiants font leur dernier commit avant le début de la première course.
2. **Positionnement Piste,**
3. **Mise en attente départ**
4. Démarrage des programmes
 - Si le véhicule avance -> Utilisation Bouton Gauche pour le démarrage
 - Si Avance (bouton gauche non géré)
5. **Lancement course**
6. Interdiction de toucher au programme et à la maquette
7. Autorisation de relancer le programme en cas de défaillance, mais vous aurez une pénalité.
 - Temps maximum : 5 minutes pour faire les 3 tours.
8. Après trois tours arrêt du véhicule avant la ligne de départ (D:3 A:3)
9. Après 15s disparition du véhicule.
10. Arrêt du programme.
11. Sur Piste noire :
 - Vous devez être en mesure de déplacer manuellement le véhicule **dans une position définie**
12. Aucune validation du fonctionnement ne sera possible après le début des courses.

11.Hall of Fame

Version 1

- ...

Version 2

- Piste Verte
 - 13.42s M. Euphrasie (2019)
 - 13.58s M. Schiano (2019)
 - 14.20s M. Coccalotto (2019)
- Piste Bleue
 - 15.42s M. Euphrasie (2019)
 - 16.55s M. Schiano (2019)
 - 17.16s M. Coccalotto (2019)
- Piste Rouge
 - 20.14s M. Euphrasie (2019)
 - 20.65s M. Vilain (2019)
 - 22.33s M. Schiano (2019)
- Piste Noire
 - 34.37s M. Vilain (2019)
 - 37.42s M. Beaussant (2019)
 - 42.37s M. Schiano (2019),

Version 3

- Piste Verte (3 tours)
 - 44.23s M. Lucotte (2020)
 - 46.43s M. Abdourahamane (2020)
 - 47.05s M. Etaix (2020)
- Piste Bleue (3 tours)
 - 50,69s M. Vialard (2020)
 - 51,06s M. Chambon (2020)
 - 52,80s M. Etaix (2020)
- Piste Rouge (3 tours)
 - 60,22s M. Lucotte (2020)
 - 66,66s M. Abdourahamane (2020)
 - 67,65s M. Vialard (2020)
- Piste Noire (3 tours)
 - 72,60s M. Chambon (2020)
 - 84,95s M. Chargueraud (2020)
 - 98,49s M. Roche (2020)

Séance 1

Le but de la première séance est de prendre en main les périphériques de la carte et de maîtriser la communication application-simulateur. Pour cela, on va réaliser un programme permettant de contrôler la position angulaire (azimut) de la tourelle portant le télémètre. On ne s'intéresse donc pas encore, à ce niveau, au contrôle du véhicule (qui restera à l'arrêt). La tourelle est commandée en vitesse par un moteur (Périphérique 'T') et est équipée d'un capteur angulaire permettant de lire son azimut (périphérique 'R').

Pour cette partie, trois types de messages vont transiter entre l'application et le simulateur :

1. Des requêtes de lecture de la position angulaire (de l'application vers le simulateur),
2. des positions angulaires (du simulateur vers l'application, en réponse à une requête),
3. des commandes en vitesse de la tourelle qui porte le télémètre (de l'application vers le simulateur).

Pour le 2e et 3e type, le champ val des messages contiendra respectivement la position courante de la tourelle (en 10e de degré) et la vitesse à imposer (en 10e de degré par seconde).

Le système aura une structure de boucle d'asservissement.

Travail préparatoire (à faire avant le début de la séance)

Structure du programme:

Le système à réguler peut-être considéré comme un système du premier ordre.

1. Rappelez le principe d'un correcteur de type proportionnel, permettant de calculer la commande à appliquer à la tourelle en fonction de la position observée et de la consigne (la consigne est vue comme une constante à ce niveau).
2. Donner l'allure de la réponse à une consigne indicielle si le gain du système (correcteur + actionneur tourelle) en boucle ouverte, est petit.
3. Même question dans le cas d'un gain grand.
4. Répondre aux questions 2 et 3 dans le cas d'un système du second ordre.
5. Proposez le pseudo-code d'une tâche Asserv0 permettant d'implanter un asservissement en position en utilisant un tel correcteur. Cette tâche devra effectuer en boucle les opérations suivantes :
 1. envoyer une requête de lecture de la position de la tourelle au simulateur sur la queue de messages CanTx
 2. obtenir la réponse sur la queue de messages CanRx
 3. calculer la commande à appliquer
 4. envoyer la commande sur la queue de messages CanTx
6. Proposer un code en C pour cette fonction dans le contexte d'exécution d'une tâche.

7. À partir de la documentation de référence du noyau MR308, étudier le fonctionnement des primitives associées aux queues de messages (snd_dtq, rcv_dtq, ...).
8. Quelle devra être la priorité de cette tâche (élevée ou faible).

Aspect temporel:

9. Proposer une solution pour qu'une itération de la tâche Asserv0 ne s'exécute que 10 fois par seconde (c.-à-d.. fixer le nombre de lecture de position et d'envoi de commande qu'elle réalise par seconde) en ne modifiant que le code de la tâche ?

Nous supposons que chaque ligne de C dans la tâche d'asservissement hormis les lectures dans la queue de message CanRx s'exécute en 0,1ms et que le temps de réponse du simulateur à une requête de lecture varie (de manière imprévisible) entre 2 et 5ms.

10. Faites un diagramme temporel montrant l'état de la tâche d'asservissement si sa période d'activation est fixée à 100ms (fréquence d'activation = 10 Hz) sur une période de 300ms
11. Proposer une nouvelle structure de programme permettant de garantir la période à 2% près.

Structure optimisée:

12. Rappeler les spécificités d'un handler cyclique.
13. Quelle(s) partie(s) du programme ne sont pas compatibles avec la structure d'un handler cyclique.
14. Proposer une solution permettant à la boucle de régulation de s'exécuter dans le contexte d'un handler cyclique.
15. Quel est l'intérêt de déclarer la fonction comme cyclique handler ? Quelle est la précision de la période d'échantillonnage obtenue ?

Travail en séance

1. Implanter la première version de votre boucle de régulation. Donner comme consigne la valeur de 45°.
2. Ajustez le cas échéant expérimentalement le gain du correcteur proportionnel afin de réaliser un asservissement satisfaisant.
3. Valider son fonctionnement.
4. Combien de messages sont générés par seconde (cette information est affichée par le simulateur – dans la console). Justifiez les valeurs obtenues.
5. Implanter la version optimisée dans une tâche de votre fonction. Valider son fonctionnement.
6. Modifier votre programme pour que la régulation soit exécutée dans un handler cyclique.
7. Modifier le programme pour que la consigne en angle soit fournie par le potentiomètre 0 dans la carte de commande. Le télémètre alternera entre l'angle lu sur le potentiomètre 0 et son opposé.

En fin de séance: faire valider un programme assurant le contrôle de la tourelle du télémètre de telle sorte que l'azimut de celle-ci alterne avec une période de 4s.

Séance 2

Dans cette deuxième séance, on va autoriser l'opérateur à contrôler la position de la tourelle portant le télémètre depuis le clavier (le véhicule restant pour l'instant à l'arrêt) et afficher des informations importantes sur l'afficheur LCD. Lors de la mise au point du programme final, il vous faudra modifier les valeurs de différents paramètres. Il est possible de régler certains d'entre eux en utilisant les potentiomètres, le clavier pourra aussi servir lors de phases de mise au point. En fin de séance, vous ferez rouler le véhicule.

Travail préparatoire (à faire avant le début de la séance)

La gestion du clavier se fait via une routine d'interruption (ISR) itouche, appelée à chaque fois qu'une touche du clavier est appuyée. Le code cette routine, fournie dans le projet type, est le suivant :

```
void itouche(void )
{
    char t;
    if (t=clavier_scan())
        vipsnd_dtq(QdmTouche,t);
}
```

La fonction `clavier_scan()` calcule et retourne le code ASCII de la touche pressée (c'est ce code qui est déposé dans la queue de messages `QdmTouche` évoquée en introduction).

1. Expliquez pourquoi l'ISR `itouche` utilise la primitive `vipsnd_dtq()` et non pas simplement `snd_dtq()`. Détaillez les différences d'utilisations de ces deux primitives.
2. Écrivez une tâche clavier qui lit, un à un, les caractères frappés au clavier et les affiche séparés par des caractères '.' sur l'afficheur LCD (utilisez la fonction `lcd_putc()` du module `lcd.c`).
3. En utilisant une fonction `decode_int` (écrite en première année) qui rappelle le, prends une chaîne de caractère au format suivant : `#nn...n*` où `n` désigne un chiffre (0...9) et retourne l'entier représenté par la sous-chaîne `nn..n` (ou -1 en cas de format erroné). Détailler la structure d'un programme qui commande la position de la tourelle à partir du clavier pour les angles positifs uniquement.
4. Proposer une fonction qui traduise une valeur sur l'intervalle `[0,1024[` vers le nouvel intervalle `]-180,180[`.
5. Tester cette fonction en utilisant un programme minimal développé sous CodeBlocks.
6. Êtes-vous capable de déterminer le principal défaut d'une telle fonction ?
7. Quelle contrainte faut-il assouplir pour obtenir une fonction plus efficace ?

Travail en séance

1. Implantez et validez la tâche `clavier` qui – rappelons-le - lit, un à un, les caractères frappés au clavier et les affiche sur l'écran LCD.
2. Validez la fonction `decode_int`.
3. Modifiez la tâche `clavier` de telle sorte que dès qu'une séquence de caractères lus au clavier est reconnue par la fonction `decode_int`, elle mette à jour une variable globale représentant la consigne courante en position du télémètre. Implantez et validez cette deuxième version de la tâche `clavier`. La validation peut se faire en plaçant un point d'arrêt matériel (break-point) sur l'instruction écrivant la variable de consigne.
4. Écrivez et validez une tâche `affiche_consigne` qui, périodiquement (10 fois par seconde par ex.), va lire la variable globale contenant la consigne courante en azimuth et l'affiche sur l'afficheur LCD (utilisez la fonction `sprintf2` et `lcd_str()` du module `lcd.c`. Pour repositionner le curseur sur la première ligne, utilisez `lcd_com(0x80)` (0xC0 deuxième ligne, 0x90 troisième ligne et 0xD0 quatrième ligne).
5. Intégrez à votre code la tâche d'asservissement en position de la tourelle développée à la séance précédente. Donnez la structure fonctionnelle complète de l'application résultante, dans laquelle on doit notamment retrouver :
 - l'ISR `itouche`,
 - la tâche `clavier`,
 - la tâche d'asservissement en position de la tourelle,
 - les queues de messages `CanTx`, `CanRx` et touches,
 - la variable globale contenant la consigne en position
 - la tâche `affiche_consigne`

Attention en cas de dysfonctionnement inexplicé, veuillez vérifier que les zones réservées au debugger sont correctes. Fermer la connexion avec le debugger, puis la rouvrir. Dans l'onglet Firmware, la zone réservée en ROM doit être en F00000 et celle en RAM à l'adresse C300.

Fin de séance: Faire valider une application qui permet de commander directement la direction du télémètre en entrant des valeurs au clavier et voir la valeur correspondante affichée sur le LCD et commander la direction des roues avant du véhicule à partir du potentiomètre 0 et sa vitesse avec le potentiomètre 1.

2 Attention, la fonction *sprintf* est assez gourmande en mémoire. Vous devrez augmenter la taille de la pile de la tâche qui l'appelle de 512 octets au moins.

Séance 3

L'objectif est désormais de mettre le véhicule en mouvement et de se servir de la tourelle télémétrique pour mesurer la distance du véhicule aux murs de la piste afin de réaliser un asservissement en direction du véhicule c.-à-d. d'agir sur la direction des roues afin de conserver une distance constante au mur.

La distance au mur est obtenue en interrogeant le télémètre porté par la tourelle. La distance est retournée en cm. L'identificateur du télémètre est 'U' (soit 85 ou 0x55).

Travail préparatoire (à rendre en début de séance)

1. Donnez la relation reliant
 - la distance d du véhicule au mur bordant la piste, mesurée perpendiculairement à ce mur,
 - la distance l du véhicule au mur retourné par le télémètre,
 - l'angle de position (azimut) du télémètre (cet angle est nul lorsque la télémètre dans l'axe de la voiture).
2. Donnez une méthode simple pour déterminer la largeur de la route.
3. Combien de tâches constituent votre programme.
4. Essayez de donner une priorité aux différentes tâches de votre programme.

Jusque là, notre application ne comporte une seule tâche d'asservissement (position de la tourelle). La communication de cette tâche avec le simulateur se fait via les queues de messages CanRx et CanTx.

Ce mécanisme simple ne convient pas lorsque l'on va vouloir faire coexister plusieurs tâches d'asservissement concurrentes (une tâche dédiée à l'asservissement de la tourelle, une autre à celle de la direction des roues par exemple). En effet, à cause de l'imprédictibilité des temps de réponse du simulateur, la tâche s'occupant de la tourelle, par exemple, ne peut alors plus faire l'hypothèse que le message prélevé dans la queue de messages CanRx est forcément la position demandée (cela peut être la réponse à une requête émise par une autre tâche !).

On ne peut pas par ailleurs installer un couple de queues de messages par tâche d'asservissement, car il faudrait alors que l'interface des fonctions assurant le couplage avec le simulateur soit dynamique ce qui est irréaliste.

La solution retenue pour ce TP consiste à installer une tâche chargée de trier les messages en provenance du simulateur et de mettre à jour un ensemble de variables globales représentant les grandeurs observées. Le code de cette tâche chargée du "tri" des messages en provenance du simulateur ressemblerait alors à :

```
// Variables globales utilisées par les asservissements
int pos_tourelle; // Position de la tourelle
int vitesse;      // Vitesse du véhicule
...              // etc..

void tri(void)
{
    CanFrame m;
    while ( 1 ) {
        rcv dtq(CanRx, &m.msg); // Attente et lecture de la réponse
```

```

switch ( m.data.id ) {
case 'T' : // Tourelle
pos_tourelle = m.data.val ;
break;
case 'V' : // Tourelle
vitesse = m.data.val ;
break;
... // Autre cas ...
} ...

```

Un problème se pose toutefois avec cette approche : une fois qu'une tâche d'asservissement a fait la requête de lecture au simulateur, comment sait-elle qu'elle peut lire la variable correspondante ? En effet cette variable sera mise à jour, comme on l'a déjà signalé, après un temps en général imprédictible. Il est donc nécessaire de prévoir un mécanisme de synchronisation afin de s'assurer que la tâche d'asservissement n'accède à la valeur mesurée qu'une fois que celle-ci a été mise à jour par la tâche de tri. Il suffit pour cela que la tâche de tri, à chaque fois qu'elle met à jour une variable globale, suite à la réception d'un message du simulateur, le signale explicitement. Il y a deux manières d'effectuer cette signalisation :

- La première consiste à associer à chaque variable observée un compteur (maj) qui sera incrémenté par la tâche de tri à chaque fois qu'elle écrira une nouvelle valeur en réponse à une requête de lecture. De son côté, la tâche d'asservissement qui utilise cette variable, une fois sa requête de lecture émise, scrute périodiquement ce compteur. Dès que le compteur passe à 1, elle le repositionne à 0 et lit la valeur de la variable. C'est la technique dite de polling (scrutation active). Cette solution est illustrée sur le listing suivant :

```

typedef struct {
int val;
unsigned char maj;
} Position; // Par exemple

Position periph[MAX_PERIPHS];
CanFrame req;
int pos;

req.data.id = 'R'; // Par exemple
req.data.rtr = 1; // Requete de lecture
periph[ADDR('R')].maj = 0;
snd_dtq(CanTx, req.msg);
while ( periph[ADDR('R')].maj == 0 )
    dly_tsk(//TEMPO//); // Attente de la réponse
pos = periph[ADDR('R')].val;

```

Le principal problème avec cette approche est que la période de scrutation (TEMPO) doit être ajustée avec soin : trop importante, elle conduit à diminuer les performances de l'asservissement (échantillonnage trop large), trop faible elle conduit à un gâchis de temps CPU.

- une deuxième solution consiste à associer à chaque variable observée un événement qui sera émis par la tâche de tri à chaque fois qu'elle écrira une nouvelle valeur en réponse à une requête de lecture. Cette fois, la tâche d'asservissement, une fois sa requête de lecture postée, n'a plus qu'à se mettre en attente de l'événement correspondant avant de lire la variable observée. Ce deuxième mécanisme est illustré sur le listing suivant :

```

CanFrame req;
Position periph[MAX_PERIPHS];

```

```

int pos;
UINT flag;

req.data.id='R';
req.data.rtr = 1; // Requete de lecture
periph[ADDR('R')].ev=0x01;
snd_dtq(CanTx, req.msg);
// Attente de la réponse
wai_flg(ev_periph, 0x01, TWF_ANDW, &flag);
pos=periph[ADDR('R')].val; // Lecture de la valeur
...
clr_flg(ev_periph, ~(flag & 0x01)) ; // Effacement de l'évènement après réception.
...

```

Ici l'évènement signalant la mise à jour de la variable P est matérialisé par le bit 0 (0x01) d'un registre d'évènements nommé Events.

En pratique, le code qui vous est fourni déclare une variable globale *periph* contenant les structures de données nécessaires pour ce mécanisme ainsi que la tâche de tri (appelée *ID_periph_rx*) assurant la gestion de cette variable :

```

typedef struct {
    unsigned short val; // Valeur observée
    unsigned char maj; // Incrémenté à chaque maj
    unsigned char ev; // Emis à chaque maj si non nul
} Tperiph;

Tperiph periph[NB_PERIPHS];

```

Exemples :

Le champ *periph[ADDR('R')].val* donne la dernière valeur renvoyée par le simulateur en réponse à une requête de lecture du périphérique 'R' (position de la tourelle)³

Le champ *periph[ADDR('U')].maj* est incrémenté à chaque fois que la valeur associée au périphérique 'U' (distance mesurée par le télémètre) est mise à jour suite à une requête de lecture de ce périphérique.

Pour que l'évènement 0x01 soit déclenché à chaque fois que la valeur associée au périphérique 'U' est mise à jour il suffit d'écrire (une seule fois suffit) :

```
periph[ADDR('U')].ev=0x01;
```

La tâche dévolue à cette réception sera fournie lors des Travaux Pratiques, vous n'avez qu'à comprendre le fonctionnement exposé ci-dessus et à adapter vos programmes en conséquence.

5. Réécrivez la fonction d'asservissement en position de la tourelle développée dans les deux séances précédentes afin qu'elle utilise le mécanisme de communication décrit ci-dessus (vous pouvez argumenter sur le choix d'une signalisation par le compteur maj ou par événement).
6. Comment peut-on connaître la couleur de la piste en cours ?
7. Comment sait-on que le feu est passé au vert ?

3 La macro *ADDR* permet de convertir le code du périphérique en index pour le tableau *periph*.

8. Certaines informations peuvent-elles être envoyées de façon spontanée par le simulateur ?

Travail en séance

Remarque: Attention si vous utilisez la fonction sin (sinus) ou cos (cosinus), ainsi que les fonctions de type sprintf, ces fonctions ont besoin d'une place importante dans la pile. Il faut donc que les tâches correspondantes aient une pile d'au moins 512 octets.

1. Validez la nouvelle version de la tâche d'asservissement de la tourelle utilisant le tableau periph. On s'assurera que la tâche de tri et redistribution des messages (ID_periph_rx) est bien démarrée dans le programme principal, avec la ligne

```
sta_tsk(ID_periph_rx);
```

Bien entendu, on n'utilisera plus directement la queue de messages CanRx à partir de maintenant.

2. Modifiez le programme validé en fin de deuxième séance afin qu'il utilise cette nouvelle structure logicielle.
3. Mettre à jour la fonction d'affichage sur le LCD afin qu'elle affiche, toujours périodiquement, en plus de l'azimut de la tourelle aussi la distance actuelle au mur (mesurée par le télémètre).
4. Donnez la structure fonctionnelle complète de votre application (ensemble des tâches, ISRs, cyclic handlers et des mécanismes de communication utilisés)
5. Dès que possible, faites valider un programme remplissant les fonctionnalités suivantes:
 - Asservissement en position de la tourelle du télémètre.
 - Définition de l'azimut du télémètre à partir du clavier.
 - Commande de la direction des roues avant à partir du potentiomètre 0.
 - Affichage de l'angle de consigne du télémètre sur l'afficheur LCD
 - Affichage de la mesure du télémètre sur l'afficheur LCD
 - Commande de la vitesse du véhicule à partir du potentiomètre 1.
 - Régulation automatique de l'angle des roues à partir de la mesure télémètre.
6. Votre programme doit être en mesure d'effectuer en complète autonomie un tour de la piste verte.

Dès lors la partie préparatoire du T.P. Est finie. Vous pouvez maintenant concevoir, programmer, tester et valider votre programme permettant la commande autonome du véhicule ! (Voir séance suivante).

Séances 4,5 et 6

L'objectif, dans ces quatre dernières séances est d'arriver à l'application finale : le véhicule capable de faire le tour de toutes les pistes proposées en un temps minimum et à l'aide d'une application générant un taux d'occupation minimum pour le CPU.

Le plan de travail est ici délibérément moins détaillé. On demande néanmoins de respecter les étapes suivantes (chaque étape devra être spécifiée proprement sous la forme d'une structure fonctionnelle et validée par l'enseignant).

Les points suivants seront validés lors de la dernière séance de travaux pratiques:

1. Le véhicule est forcé à l'arrêt si le bouton poussoir de gauche est appuyé.
2. Le véhicule est forcé à l'arrêt en mode 'course' si le Feu Tricolore n'est pas Vert.
3. Le véhicule démarre normalement lorsque le feu tricolore est vert.
4. L'écran LCD affiche le circuit actif (Vert,Bleu,Rouge,Noir).
5. L'écran LCD affiche la zone du circuit dans laquelle se trouve le véhicule. (Vert,Jaune,Rouge,Bleu,Cyan).
6. L'appui sur une touche du clavier permet de définir un paramètre du programme:
 - La valeur du paramètre sera saisie sous la forme: #nnnn* à la suite du premier chiffre. (1#23* définit la valeur (23) appliquée au paramètre 1).
 - 1,2,3,4 : vitesse zone: départ et cyan, jaune, rouge, bleue respectivement.
 - 5,6,7,8 : azimuth télémètre zone: départ et cyan, jaune, rouge, bleue respectivement.
 - 9 : règle l'azimut télémètre immédiatement
 - La dernière touche appuyée est affichée sur l'afficheur LCD.
7. L'écran LCD affiche le temps actuel de course du véhicule.
8. Lorsque le bouton poussoir du milieu est appuyé, la vitesse du véhicule et l'azimut du télémètre sont commandés par les potentiomètres. Les paramètres appliqués sont ceux « de sécurité » c'est à dire des valeurs assurant que le véhicule peut faire un tour de la piste verte avec une vitesse de 10 et un azimuth de 45°.
9. L'afficheur LCD affiche la charge en % du processeur.
10. Toute mesure aberrante du télémètre (0 ou >20m) allumera la LED jaune pendant 1s.
11. En cas d'arrêt d'urgence (bouton poussoir de gauche) ou de collision grave du véhicule, la LED rouge sera allumée.
12. La Led verte clignote avec une fréquence de 2Hz et un rapport cyclique de 10% pour signaler l'activité du programme, le rapport cyclique est de 90% pendant 2 secondes lors du passage sur une zone de couleur.
13. Le véhicule effectue au moins trois tours du circuit vert de façon complètement automatique.
14. Le véhicule effectue au moins trois tours du circuit bleu de façon complètement automatique.
15. Le véhicule effectue au moins trois tours du circuit rouge de façon complètement automatique.
16. Le véhicule effectue au moins trois tours du circuit noir de façon complètement automatique.
17. Lors de l'arrêt d'urgence du véhicule (bouton gauche appuyé) la charge processeur doit diminuer.
18. À l'issue des trois tours, le véhicule doit s'arrêter et cesser toute communication avec le simulateur (il disparaît donc du simulateur).
19. L'appui sur le bouton poussoir de droite gèle l'affichage LCD (sauf charge Cpu) et

diminue la charge processeur.

20. Le nombre de messages envoyés au simulateur par seconde ne doit pas dépasser 1000.
Évaluation

Elle se fera sur les bases suivantes :

- Travail préparatoire à rendre en début de séance (pour les séances 1, 2 et 3, le travail consiste à répondre précisément et clairement aux questions notées dans ce texte; pour les séances suivantes, le travail consiste à proposer des structures fonctionnelles complètes pour chaque étape / version de l'application)
- Validation des solutions en fin de séance et respect des objectifs fixés par le texte (en particulier pour les séances 1, 2 et 3). Lors des séances 4,5,6 un schéma structurel (décomposition en tâches et relations entre ces tâches) de l'application sera préparé avant la séance il sera discuté avec l'enseignant et servira de base de travail en cas de question sur l'application.
- Un examen de TP de 2h (individuel) destiné à vérifier la maîtrise des outils de développement et l'assimilation des concepts de la programmation temps-réel
- Rapport de conception final, dans lequel figureront notamment la structure fonctionnelle et le code commenté de toutes les solutions validées en TP
- Performances de l'application finale (circuits supportés, temps au tour, mais aussi taux d'occupation du processeur et charge sur le "bus" CAN).

Annexe 1

Identificateurs des périphériques pour la communication application-simulateur

Les codes suivants sont supportés (caractère/code en décimal/code en hexadécimal) :

- 'V'/86/0x56 : Commande en vitesse des roues motrices du véhicule (en radian /secondes).
- 'D'/68/0x44 : Commande de l'angle des roues directrices (en 1/10 de degré).
- 'T'/84/0x54 : Commande en vitesse de la tourelle portant le télémètre (en 1/10 de degré /secondes).
- 'R'/82/0x52 : Lecture de l'angle effectif de la tourelle portant le télémètre (en 1/10 de degré).
- 'U'/85/0x55 : Distance mesurée par le télémètre (1/100 de mètre)
- 'X'/88/0x58 : Position absolue X en cm (debug uniquement)
- 'Y'/89/0x59 : Position absolue Y en cm (debug uniquement)
- 'Z'/90/0x5A : Position absolue Z en cm (debug uniquement)
- 'N'/78/0x4E : Numéro de la voiture (en fonction de l'ordre de connexion)
- 'E'/69/0x45 : Lecture des événements,
- 'H'/72/0x48 : Donne le temps de course actuel
- 'S'/83/0x53 : Temps du tour précédent
- 'I'/73/0x49 : définit le nom du véhicule lettre par lettre. Le caractère '#' désigne le début du nom. Le caractère '*' implique le chargement de la configuration du circuit associée au nom du véhicule.
- 'M'/77/0x7D : Mode de course :
 - Bit 15 : État feu tricolore (1 → Vert, 0 → Orange ou Rouge),
 - Bits 14-8 : 1 Attente, 2 course, 3 essais libres)
 - Bits 7-0 : numéro de la piste (1 à 4)
- 'C'/67/0x43 : Informations sur le dernier capteur touché :
 - 8 bits de poids faible : numéro du capteur
 - 8 bits de poids fort : couleur ('C', 'B', 'R', 'J' ou 'V')
- 'K'/75/0x4B : Téléportation de la voiture sur le tronçon de piste N (correspondant au capteur vert numéro N). Attention : à n'utiliser que pour des tests, car les scores sont invalidés !
- 'J'/74/0x4A : Proposition d'un code composé de 4 couleurs
16 12 8 4 0
|COUL1|COUL2|COUL3|COUL4|
- 'j'/106/0x6A : Retourne le nombre de couleurs bien placées et présentes dans le code proposé. Si la valeur est négative, l'analyse du code n'est pas terminée. Si aucune proposition n'a été faite la valeur renvoyée est 0x77.
16 8 0
| N Bien placées| N Présentes et mal placées|

Annexe 2

Code et Signification des événements envoyés par le simulateur.

Si le simulateur envoie des événements sur 16 bits.

Pour récupérer ces événements, vous pouvez utiliser la primitive suivante (par exemple):
`wai_flg(event,(FLGPTN) 0x0007,TWF_ANDW,&flag);`

Il sera peut-être nécessaire d'effacer l'événement après sa prise en compte :

`clr_flg(event, ~(flag & 0x0007)) ; // Seul les événements attendus sont masqués.`

Chaque événement est codé sur un bit d'un registre en contenant 16.

Bit	Information associée	Remarque
0	Capteur Vert ,	remis à zéro lors de la lecture du périphérique 'C'
1	Capteur Jaune ,	remis à zéro lors de la lecture du périphérique 'C'
2	Capteur Rouge ,	remis à zéro lors de la lecture du périphérique 'C'
3	Capteur Bleu ,	remis à zéro lors de la lecture du périphérique 'C'
4	Capteur Cyan ,	remis à zéro lors de la lecture du périphérique 'C'
5		
6	Collision avec le sol,	Remise à zéro au changement de piste.
7	Fin de course (capteur vert),	remis à zéro lors de la lecture du périphérique 'C'
8	La piste a changé ,	remis à zéro lors de la lecture du périphérique 'M'
9	Le mode de course a changé ,	remis à zéro lors de la lecture du périphérique 'M'
10		
11	Le véhicule a terminé un tour.	Remis à zéro au passage du capteur de départ.
12	La voiture est sortie de la piste,	Rien à faire, le véhicule va être détruit.
13	La fonction de téléportation a été utilisée,	Classement invalidé. Remis à zéro au changement de piste ou du mode de course.
14	Faux départ	Destruction de la voiture , remise à zéro au changement du mode de course.
15		

Annexe 3

Easter Egg

Grâce aux travaux de M. Gaucher et de M. Guo, il est possible d'activer une fonctionnalité cachée de votre voiture.

Pour activer cette fonctionnalité, il faut envoyer deux fois de suite le bon code de quatre couleurs sur le périphérique 'J'(Majuscule).

Le quartet haut des bits forts doit contenir la valeur de la première couleur, le quartet bas des bits de poids doit contenir la valeur de la deuxième couleur, le quartet haut des bits de poids faible la valeur de la troisième couleur et le quartet bas des bits de poids faible la valeur de la quatrième couleur.

Un fois le code proposé, le périphérique 'j'(minuscule), après un temps d'analyse de 10 secondes, donnera sur l'octet de poids fort le nombre de couleurs bien placées (0 à 4) et sur l'octet de poids faible le nombre de couleurs présentes dans le code secret mais mal placées dans la proposition. Si aucun code n'a été proposé la valeur 0x77 est renvoyée.

Si le périphérique est interrogé avant la fin de la période d'analyse il renvoie une valeur négative correspondant au temps à attendre avant la disponibilité du résultat.

Si le code donnant comme résultats : 4 couleurs bien placées est envoyé deux fois. La fonctionnalité est débloquée.

Les couleurs sont encodées comme suit :

- 0 -> Jaune,
- 1 -> Bleue,
- 2 -> Rouge,
- 3 -> Vert,
- 4 -> Blanc,
- 5 -> Noir.