DOCUMENTATION SEGWAY

INSA Toulouse par HENG Chun Yang et PENIN Dorian

Période de stage : Juin-Septembre 2017

Tuteurs: M. MARTIN José et Mme. BARON Claude

Contents

No table of contents entries found.

# Introduction

Ce document a pour but de décrire le travail de programmation qui a été effectué sur la maquette du Segway en Juin/Juillet 2017. Ce système est destiné à être utilisé pour mettre en œuvre des Travaux Pratiques pour les étudiants de 4 année, en spécialité Ingénierie des Systèmes. En termes de software, les principales fonctions réalisées sont l’acquisition des données des capteurs, l’asservissement des moteurs, la supervision de l’état du Segway, ainsi que l’affichage d’informations destinées à l’utilisateur. Ces fonctions sont partagées entre un contrôleur STM32 et une carte Raspberry Pi 3.

Ce document est donc divisé en deux parties, la programmation du microcontrôleur STM32f3, puis celle de la carte Raspberry Pi 3. Ces deux cartes ont des caractéristiques et des rôles différents, qui seront expliqués dans la suite du document.

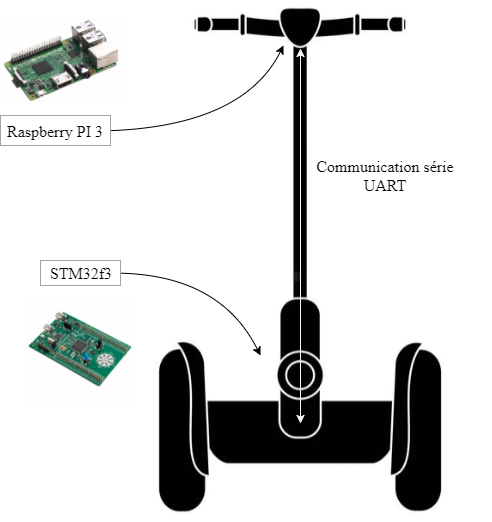


Figure 1 Le modèle Segway

# Chapitre 1

## La programmation du STM32

La carte STM32 est un microcontrôleur qui embarque un processeur Cortex-M3 de la même marque ARM. Nous avons ici utilisé le modèle STM32F303 Discovery kit, qui intègre un accéléromètre et un gyroscope, capteurs qui sont nécessaires pour faire l’asservissement des moteurs du Segway. Dans notre système, c’est la carte qui est au plus près du système mécatronique. C’est elle qui va être chargée de récupérer les informations relatives au fonctionnement du gyropode, de commander les moteurs et d’implanter des fonctions de sécurité.

Le STM32 est donc chargé d’effectuer les fonctions suivantes :

• L’acquisition des données à travers les différents capteurs qui lui sont reliés

• L’asservissement en courant des moteurs

• La réception et l’envoi de données à la carte Raspberry PI

• La gestion d’arrêt d’urgence

La gestion d’arrêt d’urgence est normalement effectuée par la Raspberry PI 3, cependant en cas de rupture de la communication avec cette dernière, le STM32 doit être capable d’assurer les mesures nécessaires à la sécurité de l’utilisateur.

Toutes les fonctions que nous venons de présenter sont écrites en langage C et sont embarquées dans le STM32. Pour le développement du code du STM32, nous avons utilisé l’IDE Keil qui est adapté pour travailler avec cette carte. Le programme est répartit en plusieurs fichiers:

• main.c: contient le programme principal exécuté au démarrage du Segway

• fonctions.c: contient les différentes fonctions utilisées pour l’acquisition des données des capteurs ou les calculs d’asservissement

• it.c : contient la déclaration et le lien des interruptions avec les différentes fonctions

Excepté *main.c*, ces fichiers sont accompagnés de leurs fichiers headers respectifs.

### 1.1 La communication via l’interface UART

#### 1.1.1 Envoi de données du STM32 sur la liaison série

Dans notre programme, l’envoi de données est effectué dans la gestion d’interruption externe du gyroscope, qui déclenche à 94 Hz. Afin de simplifier les envois, toutes les données sont envoyées dans une seule trame, qui mesure 37 octets en incluant les caractères de contrôle.

L’envoi de données de STM32 se fait grâce à l’envoi de trames par la méthode de division du nombre flottant en 4 octets sur le port série. Le décodage demande moins de calcul que pour des trames en ASCII. Pour éviter que les octets puissent prendre n’importe quelle valeur, les octets des données sont entourés par des caractères de contrôle.

Ces trames sont composées des champs suivants:

• *début de trame*: contient le caractère ’R’, permet de reconnaître le début d’une trame lors de la lecture du port série

• *paquet de donnée:*

* *début paquet*: contient le caractère ’<’ qui indique le début d’un paquet
* *label*: contient un caractère qui permet d’identifier la grandeur associée aux informations du champ data
* *data* : contient l’information envoyée
* *fin paquet*: contient le caractère ’\n’ qui indique la fin d’un paquet
* *fin de trame*: contient le caractère ’X’, permet de reconnaître la fin de la trame

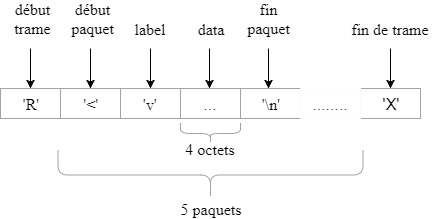


Figure 2 Exemple d’une trame

Voici les informations envoyées par le STM32 à la Raspberry PI:

|  |  |  |  |
| --- | --- | --- | --- |
| Donnée | Type | Unité | Label |
| position angulaire | float | rad | ’p’ |
| vitesse angulaire | float | rad /s | ’s’ |
| niveau batterie | integer | % | ’b’ |
| vitesse linéaire | float | m/s | ’v’ |
| présence utilisateur | integer | 1 si présent, 0 sinon | ’u’ |

#### 1.1.2 Réception de données de Raspberry Pi sur liaison série

La réception de données par le STM32 est gérée par l’interruption de l’UART. La réception est déclenchée si une donnée arrive dans le buffer de réception. Nous allons recevoir des trames sous le format d’un paquet (7 octets), ce qui est différent du cas précédent. Ce choix vient du fait que l’envoi de consignes de Raspberry Pi se fait à la vitesse de 50 Hz, le temps de traitement des messages est donc largement suffisamment. De plus, comme il existe d’autres threads de temps réel qui peuvent envoyer une trame d’urgence pour arrêter le Segway, il est nécessaire de recevoir les informations paquet par paquet.

### 1.2 La boucle de courant

L’asservissement en angle du Segway est effectué par la Raspberry PI. Cet asservissement permet au Segway de rester en équilibre, et d’avancer ou de reculer lorsque l’utilisateur penche le système. Pour cela, comme nous le verrons dans le chapitre dédié à la Raspberry, celle-ci envoie au STM32 des consignes de couple pour les moteurs. Afin que ces couples soient effectivement les couples fournis par les moteurs, le STM32 s’occupe de l’asservissement du courant. Il reçoit les consignes de couples calculés par la Raspberry PI, et les applique sur les moteurs grâce à une boucle de courant.

Le courant des moteurs est mesuré grâce à l’ADC 12 du STM32 (PC0 et PC1). Une interruption est levée à 5 kHz, dans laquelle une moyenne glissante de 64 valeurs d’ADC est calculée. La fonction Trait\_b\_courant () est alors appelée, il s’agit du calcul du correcteur. Ce correcteur est de type PI. Un correcteur P aurait pu être utilisé, mais l’erreur statique qu’il provoque et le meilleur temps de réponse produit par le correcteur PI nous a fait opter pour cette dernière solution.

Nous avons obtenu les coefficients du correcteur grâce à la modélisation mécanique effectuée par le Génie Mécanique, qui est DISPONIBLE EN ANNEXE.

Nous avons donc comme paramètres:

•

•

L’erreur est calculé1e en comparant les consignes envoyées par la Raspberry PI et les courants mesurés par l’ADC. A la sortie du correcteur nous avons alors les nouvelles valeurs de courant à appliquer aux moteurs.

### 1.3 Les mesures de sécurité

Les mesures de sécurités déployées par le STM32 sont présentes en cas de rupture de communication avec la Raspberry PI. En effet la surveillance effectuée par le programme de temps réel de cette carte permet de couvrir la plupart des exigences. Cependant, dans le cas où les deux cartes ne peuvent plus communiquer, le STM32 doit être capable de protéger l’utilisateur et le système.

#### 1.3.1 Présence utilisateur

La présence de l’utilisateur est détectée par le STM32. Les boutons du guidon sont reliés au port 4 du GPIOA. Si les boutons ne sont pas actifs pendant 500 ms, le champ présence user envoyé par le STM32 à la Raspberry PI est mis à 1. C’est ensuite la Raspberry qui se charge de traiter cette information, et de faire des modifications au niveau de l’asservissement afin de remplir des exigences de sécurité.

#### 1.3.2 Calcul de vitesse

N’ayant pas de dispositif de mesure de la vitesse de rotation des roues, le calcul de la vitesse du Segway se fait à partir des valeurs de courant, et du rapport cyclique appliqué suite au calcul de la boucle de courant. Le courant mesuré par l’ADC, lorsqu’il est utilisé par le correcteur, est déjà moyenné. Afin d’avoir une moyenne sur un temps encore plus long, ces valeurs sont une nouvelle fois moyennées (avec 20 valeurs). Il en va de même pour les valeurs de rapport cyclique.

Une fois ces valeurs obtenues, l’équation suivante est utilisée pour calculer la force électromotrice de chaque moteur :

Ensuite, nous pouvons déduire la vitesse angulaire grâce à la formule suivante:

Le rapport de réduction de la transmission par chaîne intervient:

Avec rapport de réduction

Ce qui donne finalement:

La vitesse est alors communiquée à la Raspberry PI 3 via la liaison UART.

#### 1.3.3 Rupture de communication

Lors d’une rupture de communication entre le STM32 et la Raspberry PI, le système doit s’arrêter, car les nouvelles valeurs de consigne de couple envoyées par la Raspberry n’arrivent plus jusqu’au STM32. Il faut alors éviter que le microcontrôleur continue d’appliquer les anciennes consignes de courant aux moteurs, ce serait dangereux pour l’utilisateur.

Pour éviter cela, un système de Watchdog est en place. Un compteur est d’une période de 500 ms est incrémenté continuellement, lorsqu’un message est reçu il est remis à zéro. Ainsi, si aucun message n’est reçu pendant 500 ms le compteur déborde et une interruption est levée. Cette interruption change l’état du booléen arrêt à 1. Lorsque arrêt est à 1, les consignes de courant sont décrémentées progressivement jusqu’à 0, et ceci en 2 secondes.

# Chapitre 2

## La programmation de la Raspberry Pi 3

Le Raspberry Pi est une nano-ordinateur mono-carte à processeur ARM. La carte joue le rôle de superviseur et il effectue les opérations temps réel, en interaction avec la carte STM32. Le système temps réel de Raspberry Pi 3 est réalisé dans Xenomai. Xenomai est un système d’exploitation temps réel qui a Linux pour tâche de fond et son architecture est en Co-noyau de noyau Cobalt et Linux. Ce noyau Cobalt a la priorité plus haute que le noyau Linux, qui permet de faire des opérations temps réel en délivrant une faible latence. Linux reste utilisé pour les services non temps réel qui procure et devient une simple tâche du noyau Cobalt.

Dans la carte Raspberry Pi, nous avons deux types d’opérations. La première opération est sur le système temps réel de Segway, et le deuxième est des tâches de système Linux. Avec son caractéristique de Co-noyau, nous pouvons effectuer ces deux opérations différentes sans problème.

• Partie Temps Réel

La fonction principale de cette carte est de faire l’asservissement du système mécatronique. De plus, le traitement de données et la surveillance du fonctionnement de système sont faits par ce programme. Toutes ces opérations sont inscrites dans l’historique des événements (log) mis à disposition pour le programmeur.

• Partie Linux

La carte est connectée avec un écran tactile, qui sert de moniteur. Il y a une interface graphique qui affiche des données du système et des historiques des événements en temps réel. En plus de cette interface, étant donné que la communication entre la carte Raspberry Pi 3 et la carte STM32 se fait par le protocole série UART (non synchrone), qui n’est pas encore dans la librairie de Xenomai, cette communication est gérée par le noyau Linux.

### 2.1 La communication via l’interface UART

Dans la carte Raspberry Pi 3, il n’y a pas d’interface de liaison série USART. Il est donc obligé d’utiliser l’interface UART, qui est asynchrone.

#### 2.1.1 Envoi de données de Raspberry Pi sur liaison série

Dans notre programme, l’envoi de données (les consignes de courant des deux moteurs pour et le message d’arrêt d’urgence) est effectué par le thread Envoyer. Le thread Asservissement va envoyer tous les 50 Hz des données à la file de message et le thread Envoyer va envoyer ces messages en FIFO. Il y a d’autres threads de surveillance qui peuvent envoyer le message d’arrêt urgence à la file d’attente, comme le thread Arrêt Urgence.

#### 2.1.2 Réception de données de Raspberry Pi sur liaison série

### 2.2 L’interface graphique sur Raspberry Pi

L’interface graphique qui a été conçue sur la carte Raspberry PI a deux objectifs:

• Permettre à l’utilisateur du Segway de consulter certaines informations sur le système, comme sa vitesse ou le niveau de batterie. Les informations sont disponibles avec leur valeur numérique mais aussi grâce à des messages d’information qui préviennent l’utilisateur de certains dangers (batterie faible, vitesse trop élevée...)

• Permettre au programmeur du code de temps réel de vérifier l’état et le comportement du système pendant l’exécution de son programme et suite à des modifications de certaines variables (voir section ?). De plus, l’interface graphique affiche également le log du programme de temps réel, qui contient des informations sur l’exécution des tâches, ainsi que la manipulation des mutex et sémaphores.

Cette interface graphique a été programmée à l’aide de l’API GTK+ et codée en langage C. L’exécution de programme se fait dans un terminal de Linux, qui n’est pas une tâche de temps réel. Cela a pour but de ne pas consommer de ressources temps réel et de laisser au programmeur des libertés quant à l’architecture de son programme.

En effet l’interface graphique est exécutée par le noyau Linux, en même temps que le programme de temps réel. En revanche, toutes les tâches du noyau Xenomai sont prioritaires, ce qui implique que les instructions du programme de l’interface graphique ne s’exécutent que lorsque le CPU n’est pas occupé par les tâches temps réel.

L’interface graphique doit cependant afficher des variables qui viennent du programme de temps réel. La transmission de données du processus temps réel à non-temps réel est faite grâce à l’utilisation d’un socket. Les fonctions suivantes sont disponibles: ajout\_info et send\_to\_socket, elles permettent au programmeur d’envoyer les données qu’il souhaite au socket (et donc de les afficher sur l’interface graphique) à partir de n’importe quelle tâche de son programme de temps réel.

### 2.3 Le système temps réel

Le programme de temps réel est exécuté sur la carte Raspberry Pi 3. Ce programme gère le monitoring du système, ainsi que l’asservissement en angle du Segway. Nous avons distribué les fonctions en 6 threads :

#### 2.3.1 Communication

Le thread de Communication se charge de récupérer périodiquement, à une fréquence de 94Hz, les informations reçues à l’aide de la liaison série du STM32. Il déchiffre les trames et mettre à jour les variables partagées utilisées par les autres threads. Il doit d’abord vérifier que la communication UART est bien établie. Si ce n’est pas le cas, il va réessayer de connecter. Le thread se met ensuite en attente d’une trame sur la liaison série. A la réception de celle-ci, les données sont extraites et les variables partagées sont mises à jour en fonction inclus la variable état communication qui doit être utilisé pour les autres threads.

#### 2.3.2 Asservissement

Le thread Asservissement effectue les calculs nécessaires au contrôle des déplacements du Segway. Si la variable état communication est égale à vrai, il récupère les informations d’angle au niveau, et après son calcul, écrit les résultats dans la variable partagée de consigne courant et envoyer ces valeurs à la file d’attendre d’envoyer . Ce thread fonctionne à une fréquence de 50 Hz.

#### 2.3.3 Surveillance moteur

Le thread Surveillance Moteur se charge de vérifier le bon fonctionnement des moteurs. Il consulte périodiquement, à une fréquence de 10 Hz, les informations contenues dans la variable partagée moteur. Si la variable état communication est égale à faux, la variable moteur n’est pas mise à jour. Si le moteur est en dysfonctionnement, la variable arrêt est mise à jour pour déclencher l’arrêt d’urgence.

#### 2.3.4 Surveillance batterie

Le thread Surveillance Batterie se charge de vérifier le bon fonctionnement des moteurs. Il consulte périodiquement, à une fréquence de 1 Hz, les informations contenues dans la variable partagée batterie, tant que la variable état communication est à l’état vrai. Si le niveau de batterie est faible, la variable battery warning est mise à jour afin que l’utilisateur en soit informé par le thread Affichage. Enfin si le niveau de batterie atteint un niveau critique, la variable arrêt est mise à jour pour déclencher l’arrêt d’urgence.

#### 2.3.5 Affichage

Le thread Affichage se charge de communiquer à l’utilisateur toutes les informations importantes récoltées par le STM32. Pour cela il consulte certaines variables partagées et affiche leur état. Les valeurs sont rafraîchies à 2 Hz pour une bonne lisibilité. L’affichage n’est pas mis à jour si la variable état communication est dans l’état false.

#### 2.3.6 Présence User

Le thread Présence User vérifie de manière périodique à 10 Hz la présence de l’utilisateur en consultant la variable partagé. Si cette variable est à l’état false, un compteur est incrémenté. Lorsque le compteur est égal à 5 (correspond à 500 ms d’absence d’appui sur le bouton), la variable arrêt est mis à l’état true, afin que l’arrêt d’urgence soit déclenché. De la même manière, le thread consulte la variable état communication, et grâce à un autre compteur, il vérifie la durée du problème de communication et demande l’arrêt du système si ce compteur est égal à 2 (200 ms de perte de communication).

# Chapitre 3

## Simulateur du système gyropode

Le simulateur du système Segway est destiné à être utilisé par les étudiants de 4 année Ingénierie des Systèmes. Le code dédié à la communication des données avec la carte Raspberry PI qui est identique au code de la maquette réelle. Deux parties simulation ont été rajoutées. Une partie est dédiée à la simulation physique du Segway, tandis que l’autre sert à simuler quelques paramètres tels que le niveau de batterie, ou la présence de l’utilisateur.

### 3.1 Simulation physique du Segway

Il est réalisé par une carte STM32. La simulation du comportement physique du Segway est effectuée grâce au modèle du système qui nous a été fourni. Il s’agit d’un modèle d’état avec les matrices suivantes:

Ces valeurs ont été calculées pour une fréquence de 50 Hz, le STM32 calcule alors le vecteur x afin de l’envoyer à la Raspberry PI:

Avec:

Ce modèle nous permet en connaissant l’ordre de l’utilisateur, de simuler les valeurs d’angle et de vitesse angulaire. Il est aussi utile de connaître d’autres informations comme le courant moteur ou la vitesse linéaire, celles-ci peuvent être déduites des valeurs que nous avons.

* Le courant du moteur peut être déduit du couple grâce à cette formule. Cette méthode ne prend pas en compte le temps de réponse de la boucle de courant, mais pour une simulation destinée à un usage pédagogique, ce n’est pas pénalisant.
* La vitesse linéaire est calculée de la même manière que le STM32 le fait sur la maquette réelle, c’est à dire en effectuant un moyenne du courant moteur et du rapport cyclique. Le rapport cyclique est calculé par rapport au courant, grâce à la formule du correcteur PI.

### 3.2 Simulation des variables du système

Dans le but de tester le code de temps réel, il peut être utile de modifier des variables du système comme le niveau de batterie ou la présence de l’utilisateur. Pour cela, il est possible de connecter le STM32 à un ordinateur et de modifier ces variables grâce à un terminal. Un menu de dialogue propose les différentes variables modifiables. Après avoir choisi la nouvelle valeur de la variable, celle-ci est mise à jour dans le STM32 et est donc communiquée au système de temps réel comme sur la maquette réelle.