TP Temps Réel Gyropode

INSA Toulouse par HENG Chun Yang et PENIN Dorian

Table de matières

[Chapitre 1 3](#_Toc493703900)

[Présentation du système 3](#_Toc493703901)

[1.1 Introduction du gyropode 3](#_Toc493703902)

[1.2 Exigences fonctionnelles 5](#_Toc493703903)

[1.3 Architecture physique 5](#_Toc493703904)

[Chapitre 2 9](#_Toc493703905)

[Contrôle en temps réel du gyroscope 9](#_Toc493703906)

[2.1 Présentation Xenomai-Cobalt 10](#_Toc493703907)

[2.2 Présentation du superviseur 11](#_Toc493703908)

[2.3 La communication via l’interface UART 12](#_Toc493703909)

[2.4 Conception de l'architecture logicielle du superviseur 12](#_Toc493703910)

[Vous devez, à partir des fonctions décrites par les exigences, définir un ensemble de threads (tâches) permettant la réalisation de l'ensemble de ces fonctions. Il est utile de noter l'indépendance des fonctions ou leur synchronisation, la périodicité ou non de certaines actions etc. 12](#_Toc493703911)

[2.4.1 Conception du programme 12](#_Toc493703912)

[Des moyens de synchronisation et de protection sont utiles au bon fonctionnement des programmes de temps réel: 12](#_Toc493703913)

[Chapitre 3 14](#_Toc493703914)

[Prise en main du TP 14](#_Toc493703915)

[3.1 Présentation du matériel (Simulateur) 14](#_Toc493703916)

[3.2 Présentation de l’interface graphique 15](#_Toc493703917)

[3.3 Présentation de Netbeans 16](#_Toc493703918)

# Chapitre 1

## Présentation du système

### 1.1 Introduction du gyropode

Sa mission et ses objectifs :

Un gyropode est un véhicule monoplace à moteur électrique, constitué d’une plateforme munie de deux roues sur laquelle l’utilisateur se tient debout. Grâce à un système de stabilisation gyroscopique et d’un guidon, l'utilisateur peut se déplacer en se penchant sur le guidon pour accélérer, ou en se reculant pour ralentir.

Le système est destiné à déplacer une personne seule (charge utile max 100 kg), sur un terrain de type route à une vitesse max de 10 km/h (utilisation principalement citadine) en sécurité. La puissance de chaque moteur est de 500W et l'autonomie des batteries est de 4 heures.

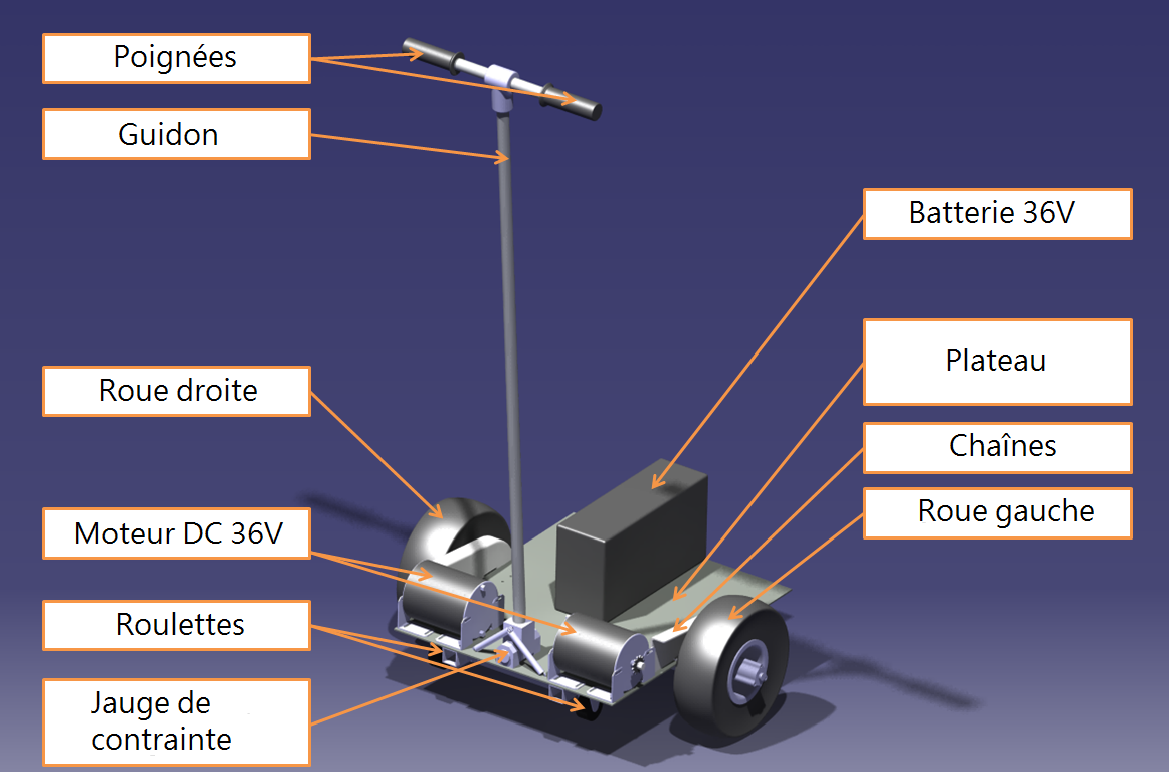


Figure 1 Modèle du gyropode

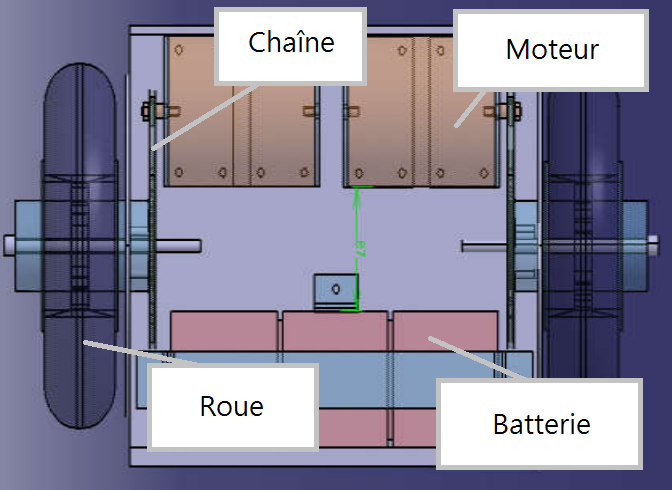


Figure 2 La vue en haut

Le contrôle du gyropode est effectué par l'utilisateur grâce à l'angle β qu'il forme entre son corps et le guidon. Pour accélérer, l'utilisateur se penche en avant, et donc réduit cet angle β, ce qui a pour conséquence d'augmenter θ, angle entre le guidon et la normale du sol. Pour décélérer, l'utilisateur se penche en arrière, l'angle β devient alors négatif, ce qui fait diminuer l'angle θ du système.

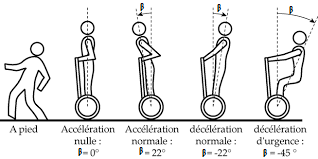


Figure 3 Cas d'utilisation et représentation de l'angle β

Le but de la commande du système est de stabiliser le gyropode à un angle θ = 0. Pour cela, lorsque cet angle est positif (gyropode penché en avant), les moteurs fournissent un couple proportionnel à cet angle afin d'accélérer et de le rétablir à zéro. Le même principe s'applique lorsque l'utilisateur le penche en arrière, le couple est alors inversé. C'est ainsi que le système avance et recule.

Afin de garantir la sécurité de l'utilisateur, certaines fonctionnalités sont mises en place. Pour limiter la vitesse du gyropode par exemple, au-delà d'un certain seuil la consigne (θ = 0) est modifiée afin que le système se rétablisse de manière forcée. Cela est possible en modifiant la consigne de θ. Cette modification de la consigne doit intervenir assez rapidement, car dans le cas contraire le système pourrait avoir atteint une vitesse trop importante, et les moteurs n'auraient pas assez de puissance pour rétablir le système en équilibre, ce qui provoquerait une chute dangereuse de l'utilisateur.

Une autre mesure de sécurité consiste à arrêter le système lorsque l'utilisateur ne se trouve plus dessus, afin d'éviter un choc avec le gyropode si l'utilisateur est au sol par exemple. Cette détection doit être effectuée dans un temps suffisamment court pour prévenir ce genre de choc. Il doit également surveiller le niveau de la batterie afin de prévenir l'utilisateur lorsqu'elle est faible et arrêter le système lorsque le niveau de batterie est critique.

Certaines fonctionnalités peuvent être ajoutées selon l'utilisation du système, comme un système de géolocalisation et de verrouillage/déverrouillage à distance si le gyropode est utilisé grâce à un système de location similaire au Vélib.

### 1.2 Exigences fonctionnelles

Ce sont les exigences que doivent remplir le système gyropode. Certaines (notées \*) sont des exigences qui font intervenir le programme de temps réel à développer.

EF1\* : Le gyropode doit permettre le déplacement de l'utilisateur (accélération, freinage, arrêt, direction) à une vitesse < 10 km/h.

EF2\* : Le gyropode doit pouvoir limiter la vitesse de l'utilisateur si celle-ci dépasse les 7 km/h.

EF3\* : Le gyropode surveille l'état de la batterie, il consulte la valeur de son niveau. Il affiche un avertissement lorsque le niveau de batterie devient inférieur à 25% et déclenche l'arrêt du système lorsque le niveau de batterie est inférieur à 15% en moins de 2 secondes.

EF4\* : Le gyropode doit s'arrêter en moins de 500 ms lorsque l'utilisateur n'est plus présent sur le système.

EF5 : Le gyropode doit protéger physiquement l'utilisateur en cas de choc frontal à 10 km/h.

EF6 : Le gyropode doit permettre de le conduire la nuit.

EF7 : Le gyropode doit être protégé de la pluie et de l'humidité.

EF8 : Le gyropode doit être facilement transportable

### 1.3 Architecture physique

L'architecture physique du système est composée de deux cartes avec des rôles différents. Le microcontrôleur STM32 est situé dans la partie basse du gyropode et s'occupe de la partie bas niveau de la programmation. La Raspberry Pi 3, située au niveau du guidon, s'occupe de la partie supervision et affichage d'informations, des tâches de haut niveau. La communication entre les deux s'effectue grâce à une liaison série de type UART.

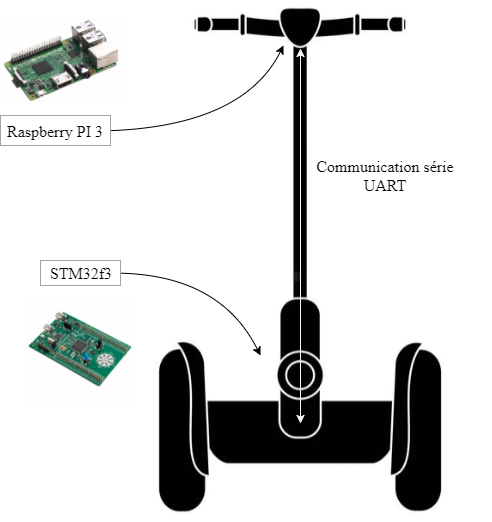


Figure 4 Le modèle de notre gyropode

* Le STM32 (partie bas niveau) récolte les informations des capteurs (angle, vitesse angulaire, niveau de batterie, courant moteurs, bouton de présence utilisateur...) les traite et les envoie à la Raspberry Pi. De plus, c'est cette carte qui est chargée d'effectuer la boucle de courant. Elle reçoit la consigne de couple de la Raspberry Pi, et pilote les moteurs par PWM grâce à un correcteur PI. C'est donc elle qui s'assure que le couple demandé par la carte superviseur est bien appliqué par les moteurs.
* La carte Raspberry Pi (haut niveau) s'occupe de la supervision du système. Elle traite les informations du STM32 pour assurer les fonctions du gyropode comme la surveillance de la batterie et la gestion de l'arrêt d'urgence grâce au programme temps réel. Elle effectue également la boucle d'asservissement en angle, en calculant la valeur de consigne de couple à partir de l'angle θ et de la vitesse angulaire reçus du STM32.
* Les deux cartes sont reliées grâce à une liaison série de type UART. Les données envoyées par le STM32 sont encapsulées dans des trames de 35 bytes, dans lesquelles chaque donnée numérique est associée à un label (char, ex: 'a'), afin de faciliter le traitement des informations par la Raspberry Pi. Le même principe est appliqué dans l'autre sens, dans lequel seulement la consigne de couple et d'éventuelles trames d'arrêt d'urgence sont envoyées par la Raspberry Pi.

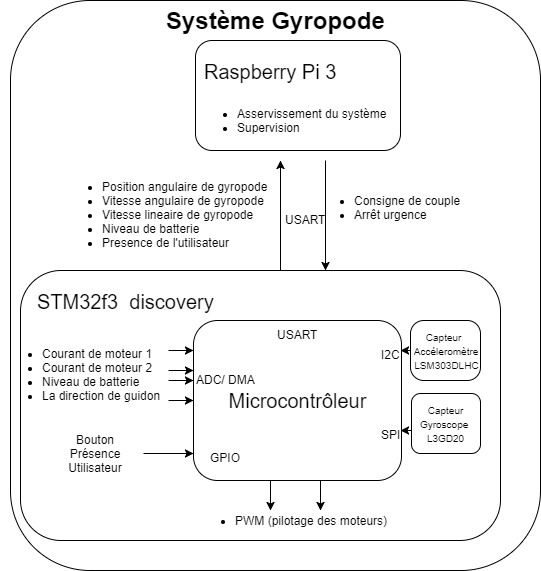


Figure 5 Acquisition et Communication des données dans le système

1.4 Commande du système

La commande du système est effectuée par les deux cartes, elle est composée de deux boucles. La boucle principale s'occupe de la régulation de l'angle θ du gyropode. C'est cette boucle qui calcule le couple Γ nécessaire pour rétablir l'équilibre du système. Cette boucle est effectuée par la Raspberry Pi. La deuxième est la boucle de courant, calculée par le STM32, qui a pour but de s'assurer que le couple à fournir est bien restitué par les moteurs.

#### 1.4.1 La boucle d'asservissement d'angle

La consigne donnée au système est donc θ = 0. L'angle β intervient alors comme une perturbation du système. Il n'est ni mesuré ni pris en compte dans la boucle d'asservissement car il influe sur l'angle θ, qui est lui une variable d'état. On mesure également la vitesse angulaire du gyropode qui est la seconde variable d'état. Ces deux variables sont multipliées par des gains respectifs, puis ajoutées pour connaître le couple Γ qui doit être exercé par les roues pour rétablir l'angle θ.

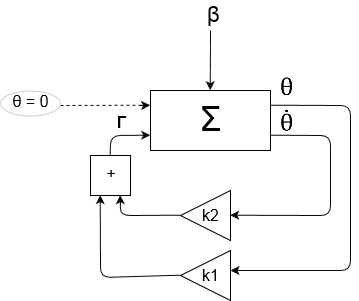


Figure 2 Schéma bloc de la boucle d'asservissement principale

#### 1.4.2 La boucle de courant

Afin de s'assurer que le couple à fournir est bien généré par les roues, il est nécessaire de mettre en place une boucle de courant. La consigne de couple peut être convertie en consigne de courant, grâce à la formule :

Les moteurs étant pilotés par PWM, il est possible de modifier le rapport cyclique α afin de jouer sur le courant moyen circulant dans les moteurs, et donc modifier le couple qu'ils fournissent. La variable mesurée est alors le courant circulant dans les moteurs, qui doit être égal à la consigne de courant.

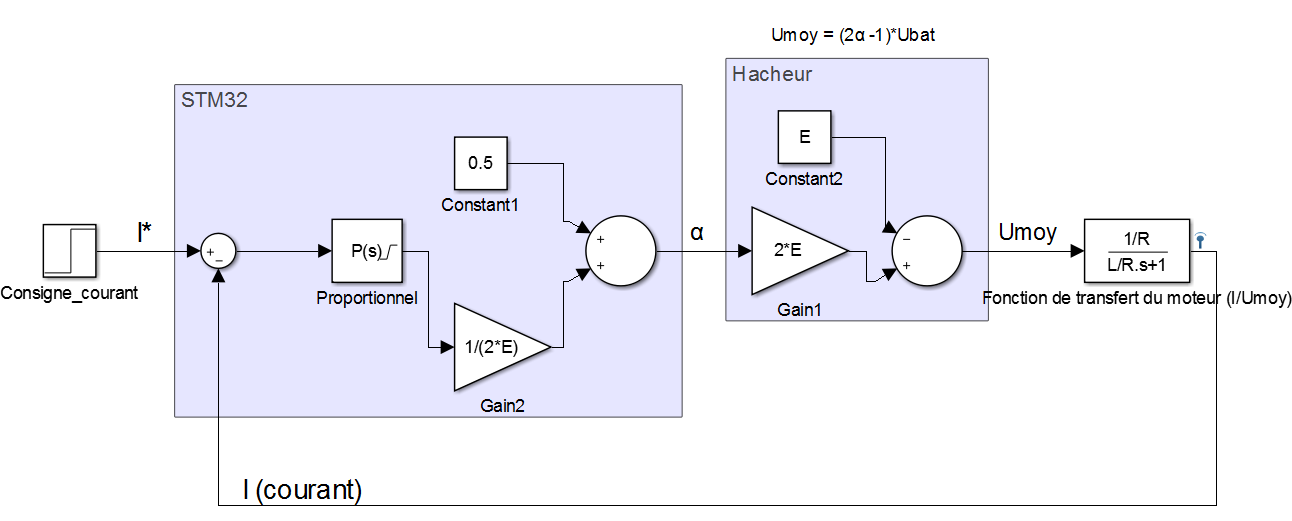


Figure 3 Asservissement de courant pour la commande des moteurs

# Chapitre 2

## Contrôle en temps réel du gyroscope

Vous être chargés de concevoir, d'implanter et de mettre au point un logiciel de contrôle qui va réaliser les fonctions de haut niveau décrites précédemment. Ce logiciel est embarqué dans la carte Raspberry Pi et utilise l'API Xenomai.

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. Dans la carte Raspberry Pi se déroulent deux types d’opérations. Des opérations se déroulent dans le noyau temps réel de gyropode, et d'autres sont des tâches du système Linux. Avec sa caractéristique de Co-noyau, nous pouvons effectuer ces deux opérations différentes sans problème.

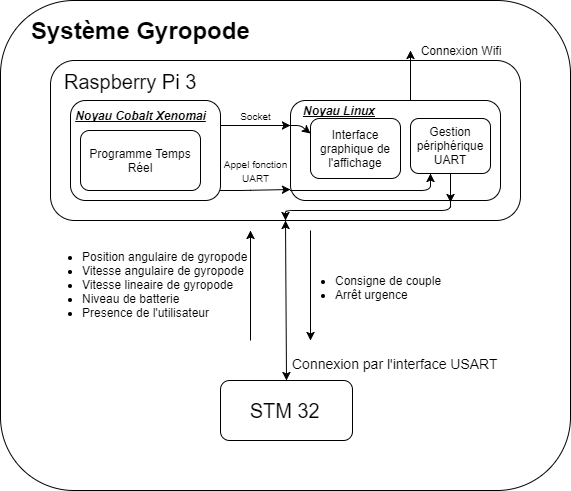


Figure 6 Diagramme de contexte

• Partie Linux

Un programme d'interface graphique est disponible, qui affiche des données du système et des historiques des événements en temps réel. Ce programme reçoit les données du programme de temps réel grâce à deux sockets : un pour la réception de données et l'autre pour recevoir les chaînes de caractères du log à afficher. 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.

• Partie Temps Réel

La fonction principale de ce programme est de faire l’asservissement du système mécatronique ainsi que le traitement de données et la surveillance du fonctionnement de système. Les fonctions à remplir correspondent aux exigences fonctionnelles marquées d’un astérisque (ex : EF3\*)

* Asservissement en angle : le programme superviseur doit envoyer périodiquement au STM32 à une fréquence de 50 Hz une consigne de couple égale à
* Surveillance du niveau de batterie : Le programme surveille périodiquement le niveau de la batterie. Il affiche un avertissement lorsque le niveau de batterie devient inférieur à 25% (exigence déjà remplie par le programme interface graphique (GUI)) et déclenche l'arrêt du système lorsque le niveau de batterie est inférieur à 15% en moins de 2 secondes.
* Surveillance de la présence de l'utilisateur : le programme doit envoyer un message d'arrêt d'urgence au STM32 lorsque l'utilisateur n'est plus sur le gyropode (presence\_user = 0) en moins de 500 ms.
* Communication avec le STM32 : le programme doit pouvoir recevoir les données envoyées par le STM32.
* Communication avec le programme d'affichage (GUI) : le programme doit envoyer par socket à 100 les valeurs de ses variables partagées.

### 2.1 Présentation Xenomai-Cobalt

Xenomai est une extension libre du noyau Linux lui apportant des fonctionnalités temps réel dur. Il apporte une approche symétrique entre programmation noyau et programmation système au niveau utilisateur sous Linux. Xenomai passe par la couche de virtualisation Adeos (Adaptive Domain Environment for Operating Systems) qui est un patch logiciel à installer sur le noyau Linux.

Dans Xenomai, Linux peut être préempté comme une simple tâche. Xenomai offre donc une garantie d’exécution en temps réel dur pour les tâches qu’il gère. Il existe deux variantes de noyau de Xenomai: *Mercury* et *Cobalt*.

La version *Cobalt* prend l’architecture Co-noyau de Xenomai. Le système est donc constitué de deux noyaux, un noyau Linux et un noyau Cobalt. Elle propose elle aussi un temps réel “dur” mais grâce à son routage d’interruption effectué par le noyau parallèle au noyau Linux on obtient des latences plus faibles qu’avec un routage natif. Le noyau cobalt traite toutes les activités de temps critiques, telles que la gestion des interruptions et la planification des threads en temps réel.

Ce noyau Cobalt a la priorité plus haute que le noyau Linux, ce 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.

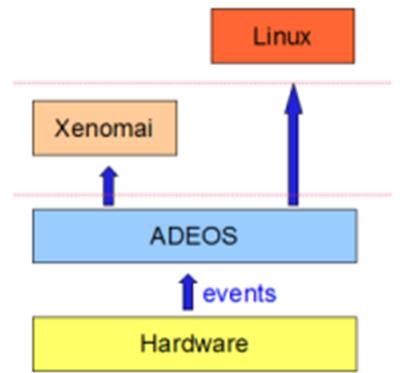


Figure 7 Schéma de couche de Co-noyau dans Xenomai 3

### 2.2 Présentation du superviseur

Le superviseur est le programme de temps réel qui tourne dans le noyau de cobalt. Mais dans ce programme, des fonctions non temps réel (côté linux) sont appelées pour permettre l'affichage de données (socket) et également la communication avec la partie de commande des moteurs (STM32).

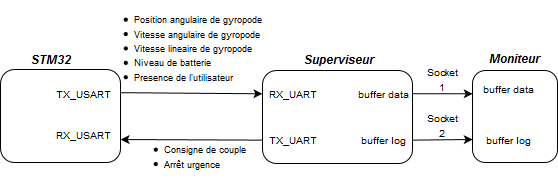


Figure 8 Diagramme de contexte avec Entrée-Sortie

### 2.3 La communication via l’interface UART

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

L’envoi de données (les consignes de courant 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 lire ces messages puis les envoyer en FIFO grâce à la fonction *send\_float\_to\_serial*.

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

La réception des données de Raspberry Pi est déjà effectuée par le thread Communication grâce à la fonction *read\_from\_serial*, les variables globales sont ensuite mises à jour en utilisant la fonction

### 2.4 Conception de l'architecture logicielle du superviseur

Vous devez, à partir des fonctions décrites par les exigences, définir un ensemble de threads (tâches) permettant la réalisation de l'ensemble de ces fonctions. Il est utile de noter l'indépendance des fonctions ou leur synchronisation, la périodicité ou non de certaines actions etc.

#### 2.4.1 Conception du programme

Des moyens de synchronisation et de protection sont utiles au bon fonctionnement des programmes de temps réel:

* Les sémaphores permettent à un thread d'attendre le déclenchement d'un évènement avant de s'exécuter. L'attente d'un sémaphore se fait à l'aide de la fonction *rt\_sem\_p,* tandis que le déclenchement de l'évènement est réalisé grâce à *rt\_sem\_v.*
* Les mutex (pour mutual exclusion), permettent de protéger des ressources qui sont consultées où utilisées par différentes tâches pour communiquer. Lorsqu'une tâche essaie d'accéder à une ressource déjà en cours d'utilisation, elle se met en pause jusqu'à ce que la ressource soit disponible. La fonction *rt\_mutex\_acquire* permet de protéger une donnée avant de l'utiliser, après avoir attendu qu'elle soit disponible. *rt\_mutex\_release* permet de libérer la protection d'une ressource après son utilisation.

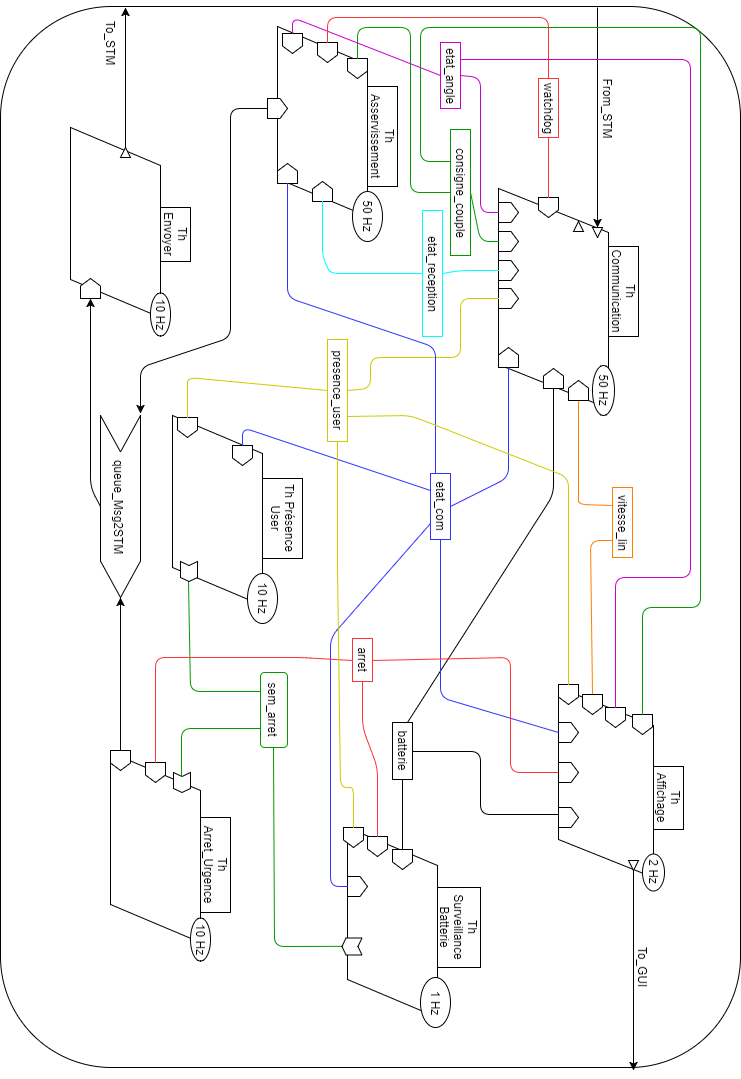
#### 2.4.2 Validation de la conception

A partir des exigences fonctionnelles, dériver quelques scénarios de test pour valider votre programme.

Exemple: après démarrage du simulateur, vérifier que les valeurs affichées par l'interface utilisateur évoluent de manière logique est un moyen de vérifier que les données transitent bien entre le STM32 et la Raspberry Pi, et également du programme de temps réel vers le programme d'affichage.

Vérifier que le résultat obtenu correspond à celui attendu, l'exigence pouvant être fonctionnelle ou temporelle Poursuivre afin de couvrir tous les cas d'utilisation.

2.5 Correction : AADL complet



# Chapitre 3

## Prise en main du TP

### 3.1 Présentation du matériel (Simulateur)

La simulation du comportement physique du gyropode est effectuée grâce au modèle du système physique linéarisé du gyropode qui n’est valable qu’entre une inclinaison de -20° à +20°. Mais cela est suffisant pour simuler le comportement de système car sur la vraie maquette, la variation est dans la zone de -20° à +20°.

La carte de STM32 va réagir comme une vraie maquette d’un gyropode. Les entrées de simulateur sont simulées par un bouton et deux potentiomètres. En réalité, l’entrée de, l’angle de l’utilisateur par rapport au guidon est une perturbation du système et il ne s’agit pas une entrée pour la vraie maquette. Mais dans le cas de modèle linéarisé, il est une entrée.

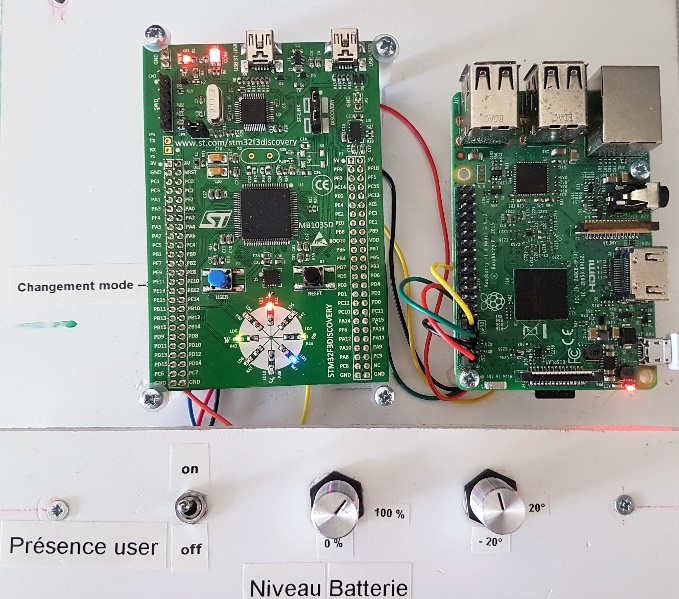


Figure 9 Photo de Simulateur et Raspberry Pi

Fonctionnement des périphéries de STM32- Simulateur

1. *Bouton Bleu :*Le bouton bleu du STM32 (user) sert à changer le mode de l’entrée pour, l’angle de l’utilisateur par rapport au guidon. Les choix possibles sont le mode accéléromètre et potentiomètre. Initialement, il est dans le mode accéléromètre.
2. *Switch de présence d’user :*   
   Le switch sert à simuler le bouton de présence d’utilisateur de la vraie maquette.
3. *Potentiomètre 1 :*

Le potentiomètre sert à simuler le niveau de batterie en pourcent

1. *Potentiomètre 2 :*   
   Le potentiomètre sert à simuler une valeur d’angle entre de -20 ° à 20 °

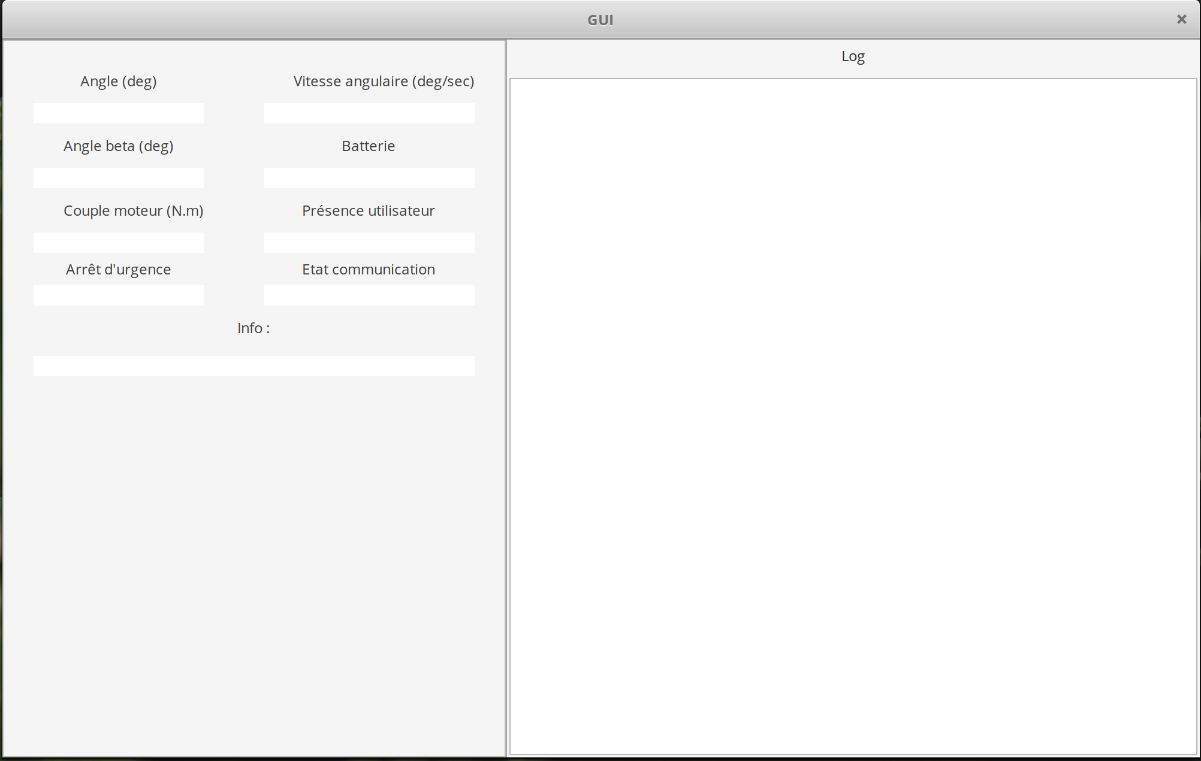
Pour faciliter la compréhension des opérations dans le STM32, les LEDs servent à indiquer des états de la carte.

|  |  |
| --- | --- |
| Numéro de LED | Action |
| LED 3 | Clignote à la réception de la trame de consigne |
| LED 4 | Mode d’entrée de l’angle relatif de l’utilisateur au guidon  => ON : Mode d’accéléromètre => OFF: Mode de potentiomètre |
| LED 6 | Etat de présence de l’utilisateur  => ON : Présence => OFF: Absence |
| LED 8 | Etat d’arrêt d’urgence  => ON : arrêt déclenché => OFF: sinon |
| LED 5 | Clignote à la réception de données de la Raspberry via USART |
| LED 7 | Clignote quand pas de connexion de USART avec Raspberry |
| LED 9 | Clignote à l’envoi de données via USART à Raspberry |

### 3.2 Présentation de l’interface graphique

Cette interface graphique ou GUI (Graphical User Interface) a été programmée à l’aide de l’API GTK+ et codée en langage C. Il permet d'afficher toutes les valeurs de variables partagées du système en temps réel.

L’exécution de programme se fait dans un terminal de Linux, qui n’est pas une tâche de temps réel. Il fonctionne à travers de deux sockets de système Linux. Il est le serveur de ses sockets de communications et il faut donc lancer ce programme avant de lancer le programme de temps réel.



Afficher des logs en temps réel

Afficher des informations sur le simulateur de gyropode

Afficher des valeurs de variables partagées

Figure 10 Capture d'écran de GUI

Pour lancer ce programme, il faut démarrer une session Linux. Pour que la carte Raspberry Pi puisse communiquer avec la session Linux, il faut que le programme de temps réel connaisse l’adresse IP de l’ordinateur. Les étapes sont suivantes :

1. Taper dans console "ipconfig" -> noter l'adresse IP du pc à droite de wlan
2. Ouvrir dossier "GUI"
3. Compiler le programme par la commande

*cc -Wall -g -o GUI \*.c $(pkg-config --cflags --libs glib-2.0 gtk+-3.0) -lm –rdynamic*

1. Exécuter le programme grâce à la commande

*sudo ./GUI*

### 3.3 Présentation de Netbeans

Nous allons programmer le programme de temps réel dans une Raspberry Pi 3 qui a le Co-noyau de Xenomai- Cobalt et Linux. Pour faciliter la programmation à distance, nous utilisons l’IDE de programmation Netbeans. Il nous permettre de connecter le Raspberry Pi en distance avec protocole SSH et compiler en distance.

#### 3.3.1 Configuration de l’IDE Netbeans

A faire

#### 3.4 Premier lancement de code

Un code à trous vous a été mis à disposition. Les tâches ont déjà été crées et sont lancées dans le programme *main.c*. Cependant la plupart de ces tâches sont vides, c'est donc à vous de les compléter. En en-tête de chaque fonction les commentaires précisent le rôle de la tâche, les fonctions qu'elle utilise, les sémaphores ou mutex avec lesquels elle peut interagir etc. Vous pouvez tout de même compiler et lancer le programme pour observer son comportement :

Comme la communication de programme temps réel et l’interface graphique se fait à travers des sockets, il nous faut renseigner l’adresse IP pour l’initialisation des sockets.

Les étapes sont suivantes :

1. Ouvrir le fichier "includes.h"
2. Noter l'adresse IP notée précédemment dans le fichier

#define ADRESSE\_IP "XXX.XXX.XXX.XXX"

1. Compiler

Pour la compilation de code, un fichier Makefile est disponible. Il suffit donc de taper la commande *make* pour compiler l'ensemble du programme.

La première exécution va permettre de se rendre compte que le comportement du système n'est pas celui qui est voulu. Le système n'est pas asservi, et les mesures de sécurité ne sont pas implantées.

Penser à bien nommer les éléments du code : les mutex sont notés *var\_mutex\_xxxx* et les sémaphores *var\_sem\_xxxx,* les tâches ont souvent des noms qui décrivent leur fonction principale.

La programmation orientée objet a été utilisée pour certaines variables partagées, comme c'est le cas pour *consigne\_couple*. Les diagrammes de classe ainsi que la documentation des librairies disponibles sont en annexe.

Le fichier de log peut être utilisé pour vérifier le respect de certaines contraintes de temps fixées par les exigences, il est disponible dans le dossier principal du programme de temps réel et est nommé *log.xenolog.* Les exigences fonctionnelles peuvent être testées grâce à l'interface d'affichage.

ANNEXES

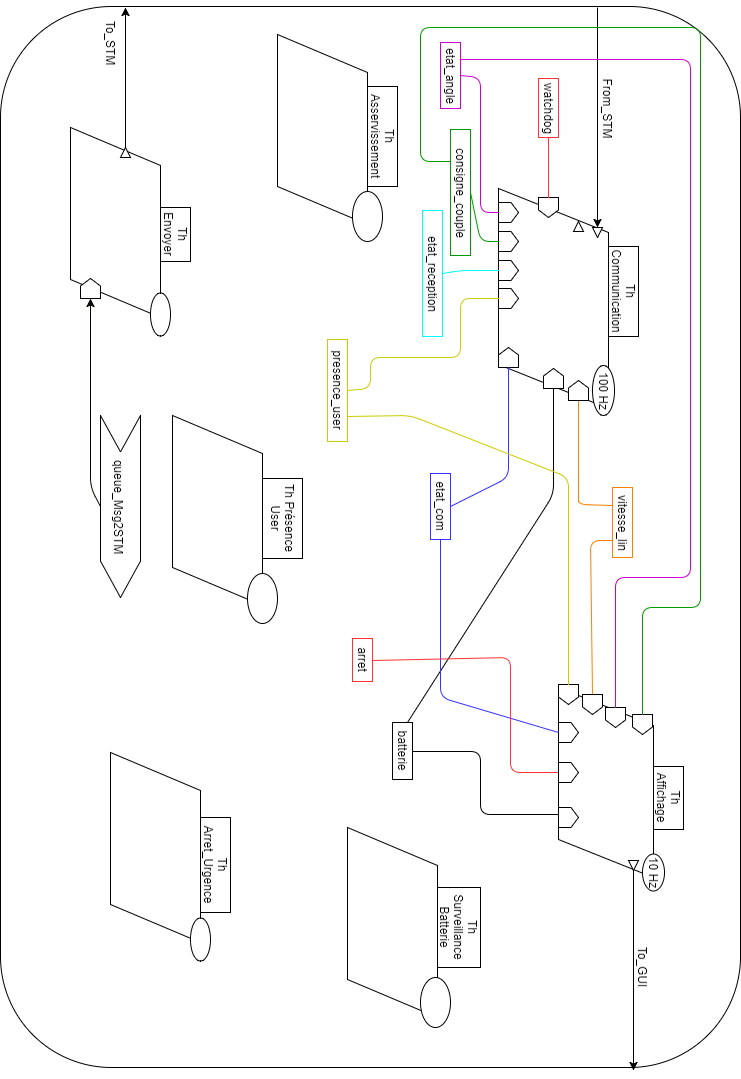
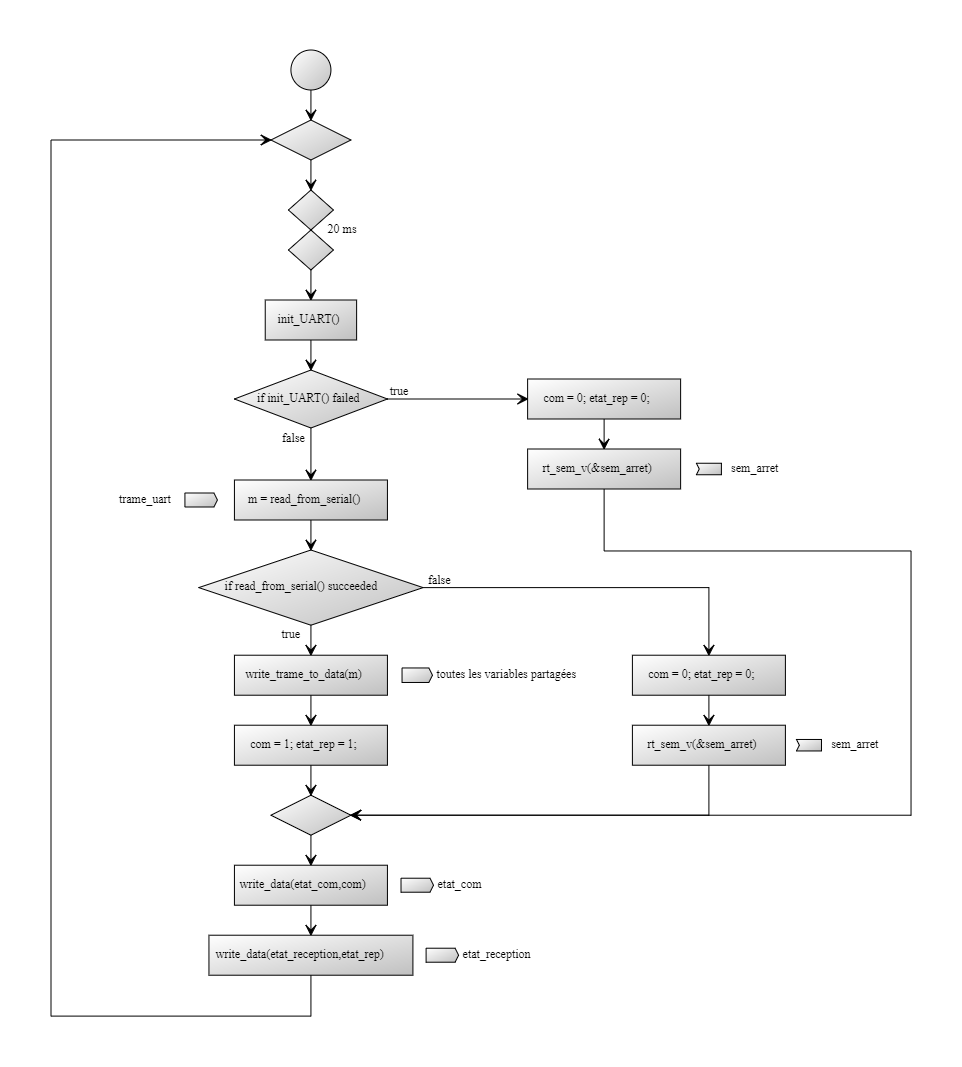
AADL:

Diagramme d'activités

Communication



Affichage

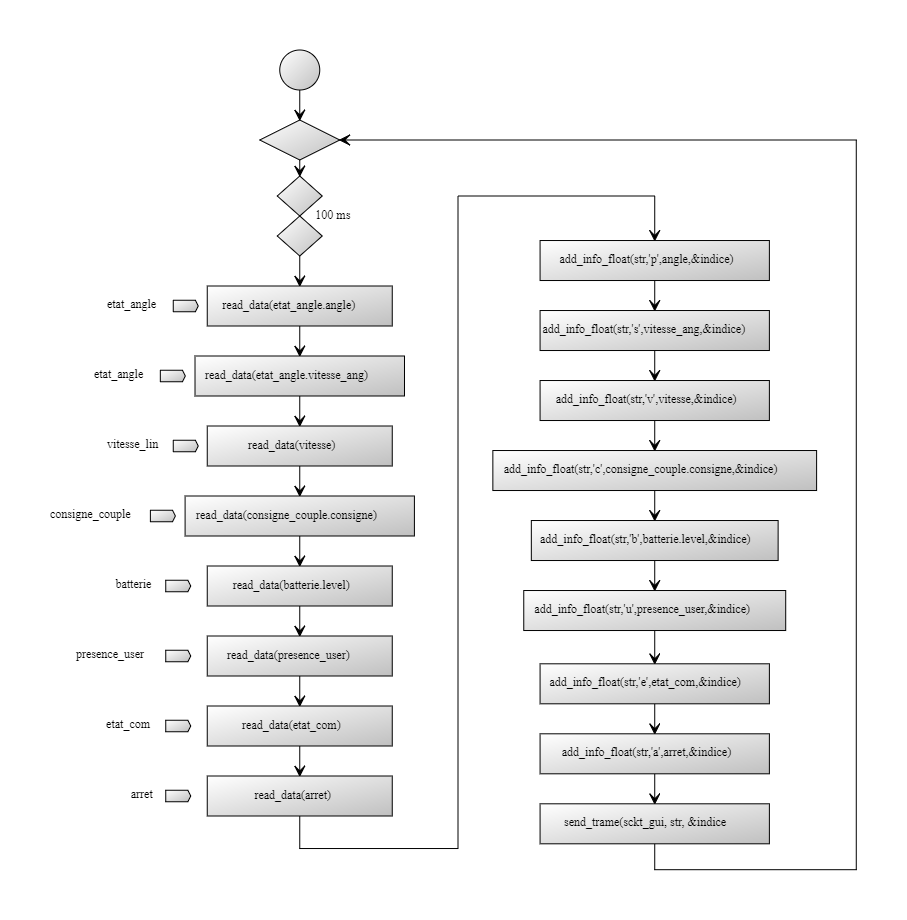
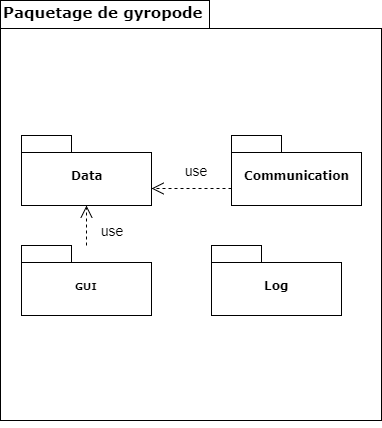


Diagramme de paquetage:



Diagrammes de classe

