DOCUMENTATION GYROPODE

INSA Toulouse par HENG Chun Yang et PENIN Dorian

Période de stage : Juin-Septembre 2017

Tuteurs: M. MARTIN José et Mme. BARON Claude

Table de matières

[Introduction 1](#_Toc493703870)

[Chapitre 1 2](#_Toc493703871)

[Présentation du système 2](#_Toc493703872)

[1.1 Introduction du gyropode 2](#_Toc493703873)

[Chapitre 2 7](#_Toc493703874)

[Partie du STM32 7](#_Toc493703875)

[2.1 La communication via l’interface USART 8](#_Toc493703876)

[2.2 La boucle de courant 9](#_Toc493703877)

[2.3 Les mesures de sécurité dans STM32 10](#_Toc493703878)

[2.4 La commande de direction du gyropode 11](#_Toc493703879)

[Chapitre 3 12](#_Toc493703880)

[Partie Superviseur : Raspberry Pi 3 12](#_Toc493703881)

[3.1 Présentation Xenomai-Cobalt 13](#_Toc493703882)

[3.2 Présentation du superviseur 14](#_Toc493703883)

[3.3 La communication via l’interface UART 15](#_Toc493703884)

[3.4 L’interface graphique sur Raspberry Pi 16](#_Toc493703885)

[3.5 Le système temps réel 16](#_Toc493703886)

[Chapitre 4 19](#_Toc493703887)

[Simulateur du système gyropode 19](#_Toc493703888)

[4.1 Simulation physique du gyropode 19](#_Toc493703889)

[4.2 Simulation des variables du système 20](#_Toc493703890)

[Chapitre 5 22](#_Toc493703891)

[Remarques 22](#_Toc493703892)

[5.1 Ajustements 22](#_Toc493703893)

[5.2 Reste à faire 22](#_Toc493703894)

[Annexe 23](#_Toc493703895)

# Introduction

Ce document a pour but de décrire le travail de programmation qui a été effectué sur la maquette du gyropode en Juin/Juillet 2017. Ce système est destiné à être utilisé pour mettre en œuvre des Travaux Pratiques pour les étudiants de 4-ème 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 gyropode, 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.

# 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 des moteurs est de 500W x2. La capacité de batterie est 12 Ah et d’après des mesures de courants de batterie en moyenne, il fournit 3A. Donc, des batteries permettent une autonomie de 4 heures

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.

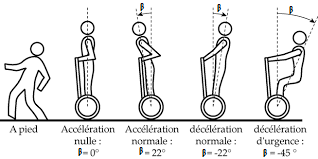


Figure 1 Cas d'utilisation et représentation de l'angle β

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.

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 qui doit être exercé par les roues pour rétablir l'angle θ.

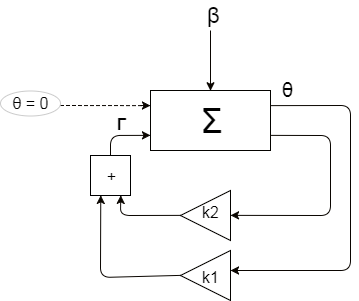


Figure 2 Schéma bloc de la boucle d'asservissement principale

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.

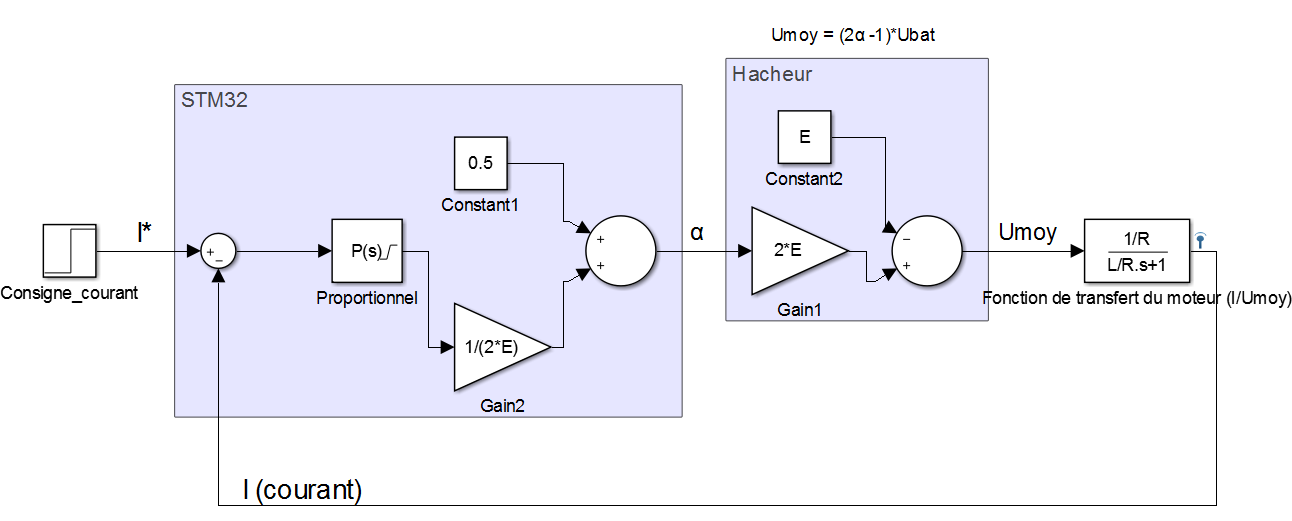


Figure 3 Asservissement de courant pour la commande des moteurs

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.

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.

La partie logicielle 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.

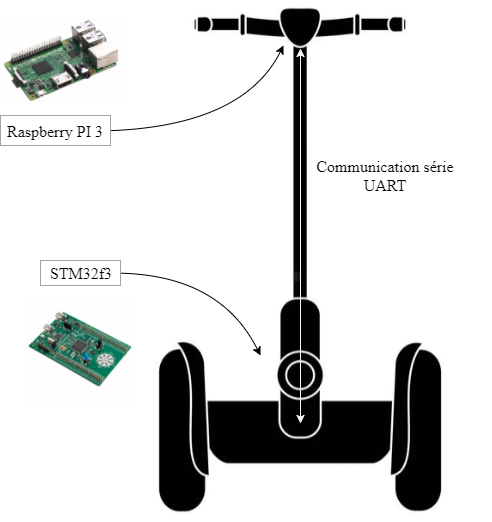


Figure 4 Le modèle d’un gyropode

* 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.

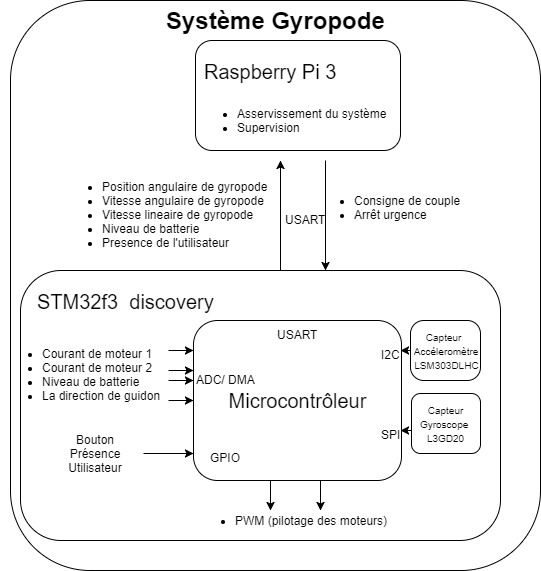


Figure 5 Acquisition et Communication des données dans le système

# Chapitre 2

## Partie du STM32

La carte STM32 est un microcontrôleur qui embarque un processeur Cortex-M4 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 gyropode. 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 et reprendre du système

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 gyropode

• *stm32f30x\_it.c* : contient la déclaration et le lien des interruptions avec les différentes fonctions

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

• *init.c*: contient les différentes fonctions utilisées pour les initialisations et configurations des ports de STM32

• *gyroscope.c*: le pilote de capteur gyroscope. Il contient de fonction de configuration de gyroscope et la fonction de l’acquisition de valeurs de gyroscope

• *accelerometre.c*: le pilote de capteur accéléromètre. Il contient de fonction de configuration d’accéléromètre et la fonction de l’acquisition de valeurs d’accéléromètre

### 2.1 La communication via l’interface USART

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

Dans notre programme, l’envoi de données est effectué dans la gestion d’interruption externe de l’accéléromètre, 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. Le trame est remplie et envoyé dans la fonction *Trait\_Gyro\_Acc* de *fonction.c.*

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

• *fin de trame*: contient le caractère ’X’, permet de reconnaître le fin d’une trame

• *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

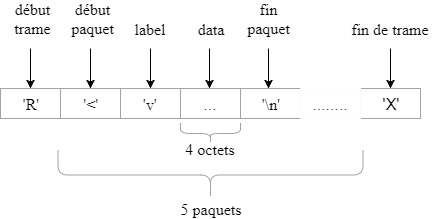


Figure 6 Exemple d’une trame

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’ |

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

La réception de données par le STM32 est gérée par l’interruption de l’USART (dans *stm32f30x\_it.c*). La réception est déclenchée si une donnée arrive dans le buffer de réception. Les données sont reçues sous le format d’un paquet (7 octets), ce qui diffère du cas précédent. La détection d’un paquet complet se fait au début et à la fin de paquet (caractères ‘<’ et ‘\n’). Ce choix vient du fait que l’envoi de consignes de Raspberry Pi se fait à la vitesse de 100 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 gyropode, il est nécessaire de recevoir les informations paquet par paquet. Au niveau du STM32, la consigne de couple reçue du Raspberry Pi est convertie directement en courant (A). La conversion est faite en divisant par la valeur 0.80435, qui est le produit de (le constant de couple de moteur DC()) et le rapport de réduction de moteur à la roue.

Voici les informations reçues par le STM32 de la part de la Raspberry Pi:

|  |  |  |  |
| --- | --- | --- | --- |
| Donnée | Type | Unité | Label |
| Consigne de couple | float | N.m | ’c’ |
| Arrêt | int | 1 si arrêt d’urgence, 0 sinon | ’a’ |

### 2.2 La boucle de courant

L’asservissement en angle du gyropode est effectué par la Raspberry PI. Cet asservissement permet au gyropode 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 du STM32 (PC0 et PC1). Une interruption de type DMA de l'ADC est levée à 5 kHz, dans laquelle une moyenne glissante de 64 valeurs d’ADC est calculée. La fonction *Trait\_b\_courant ()* dans le fichier *fonction.c* est alors appelée, il s’agit du calcul du correcteur. La raison pour laquelle nous avons mis la boucle de courant dans cette interruption est pour prendre moyenner les valeurs des courants le plus rapidement possible. Le 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 dans le document Projet-5GM-Segway.pdf de page 22.

Nous avons donc comme paramètres:

•

•

L’erreur est calculée 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. Dans le traitement de boucle de courant, il existe des limitations en courant (10 A) et rapport cyclique de PWM de moteur (0.1) pour éviter la saturation de commande, puis protéger les hacheurs 4 quadrants et les autres composants électroniques.

### 2.3 Les mesures de sécurité dans STM32

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.

#### 2.3.1 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 200 ms, le champ *présence\_user* de trame envoyée par le STM32 à la Raspberry PI est mis à 1. C’est ensuite la Raspberry qui se charge de traiter cette information, pour déclencher l’arrêt urgence du système afin de remplir des exigences de sécurité. La surveillance de bouton présence user est faite par un compteur dans la boucle de 94 Hz pour réaliser la détection de délai de 200 ms.

#### 2.3.2 Calcul de vitesse

N’ayant pas de dispositif de mesure de la vitesse de rotation des roues, le calcul de la vitesse du gyropode 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 moyenne de vitesse est faite dans la boucle de courant car dans la boucle de courant, nous avons les plus récentes valeurs de courants de moteurs. La vitesse est alors communiquée à la Raspberry PI 3 via la liaison USART. C’est ensuite la Raspberry Pi qui se charge de traiter cette information, et de faire des modifications au niveau de l’asservissement afin de remplir des exigences de sécurité.

#### 2.3.3 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. La détection doit réaliser dans la réception de donnée car c’est la seule méthode que nous pouvons observer la rupture de communication.

Pour éviter cela, un système de Watchdog est en place. Un compteur est incrémenté continuellement dans la boucle de 94 Hz, lorsqu’un message est reçu il est remis à zéro. Ainsi, si aucun message n’est reçu pendant 500 ms, les moteurs vont s’arrêter.

#### 2.3.4 Réinitialisation de l’état d’arrêt urgence

Lorsque l’arrêt urgence est déclenché par la Raspberry Pi, le système s’arrête. Mais pour pouvoir redémarrer le système, la carte STM32 doit être capable de réinitialiser l’état d’arrêt urgence. Pour l’instant l’arrêt urgence du système peut être déclenché par la Raspberry Pi pour des raisons de sécurité comme la présence de l’utilisateur et le niveau de batterie. Le superviseur de Raspberry Pi va toujours envoyer des trames d’arrêt d’urgence si une des deux conditions n’est pas remplie. Donc, nous avons un compteur dans la partie réception de données pour vérifier si STM32 reçoit toujours des trames de l’arrêt d’urgence. Une fois que STM32 reçoit une trame d’arrêt, le compteur est remis à zéro, et si le STM32 reçoit des trames différentes de l’arrêt, le compteur est incrémenté. Si le compteur est incrémenté jusqu’à 300 (environ 3 secondes si la fréquence est 100 Hz dans l’envoi de données de Raspberry Pi), il indique les conditions de sécurité sont remplies et réinitialise l’état d'arrêt urgence.

### 2.4 La commande de direction du gyropode

La direction du gyropode est contrôlée par le guidon du système. Le guidon est relié au gyropode à l'aide d'une jauge de contrainte. Le fonctionnement est basé sur le changement de résistances dans la jauge de contrainte. Nous avons noté que les valeurs d’angle maximum du guidon sont +10° et -10° puis avons entré dans le logiciel la valeur de la jauge lorsque le guidon est centré. Ensuite, en fonction de la valeur lue par l'ADC connecté à la jauge, le courant d'un des moteurs est augmenté par rapport à l'autre, ce qui produit le virage souhaité. Nous avons pu vérifier le bon fonctionnement de cette fonction, mais la jauge se brisa ensuite, nous n'avons donc pas pu améliorer la nature du virage.

# Chapitre 3

## Partie Superviseur : 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.

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

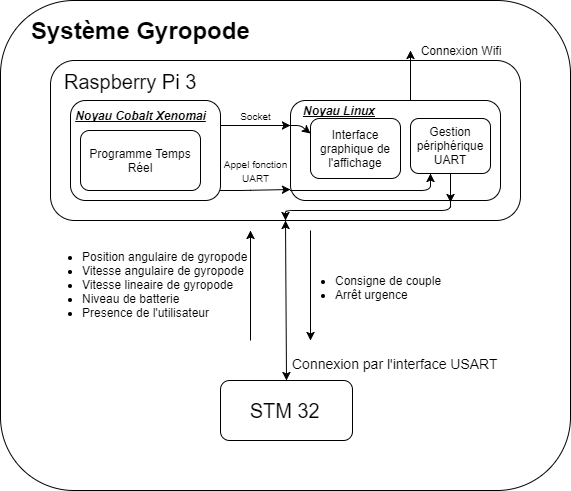


Figure 7 Diagramme de contexte

### 3.1 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. Cette architecture est largement décrite dans la suite de cette section car c’est l’architecture retenue pour le projet. 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 trait de toutes les activités de temps critiques, telles que la gestion des interruptions et la planification des threads en temps réel cobalt.

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.

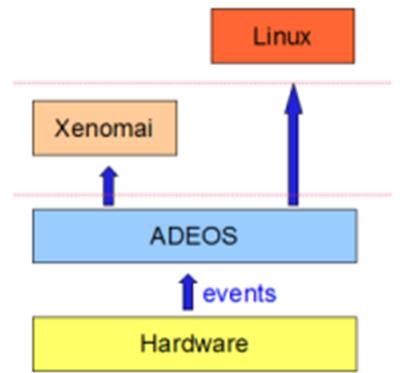


Figure 8 Schéma de couche de Co-noyau dans Xenomai 3

#### Le partage des ressources

Lorsque nous utilisons *Cobalt*, nous avons deux OS, chacun possédant son propre scheduler. Les tâches Xenomai sont donc dans le scheduler de Xenomai, les tâches Linux dans le scheduler Linux. Cependant, les tâches Xenomai peuvent être amenées à accéder à une ressource appartenant au noyau de Linux. Il est impossible d’aller la chercher directement depuis Xenomai car seul Linux peut vérifier sa validité. Il faut donc passer par Linux via un appel système Linux. Pour traiter ces appels, les tâches sont déplacées d’un scheduler à un autre en fonction des ressources accédées. Le mécanisme dit de Shadow threading permet l’exécution des tâches Xenomai sous Linux.

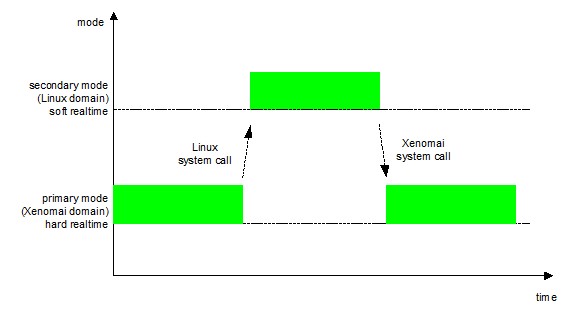


Figure 9 Gestion de l'interruption Co-noyau Cobalt

### 3.2 Présentation du superviseur

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

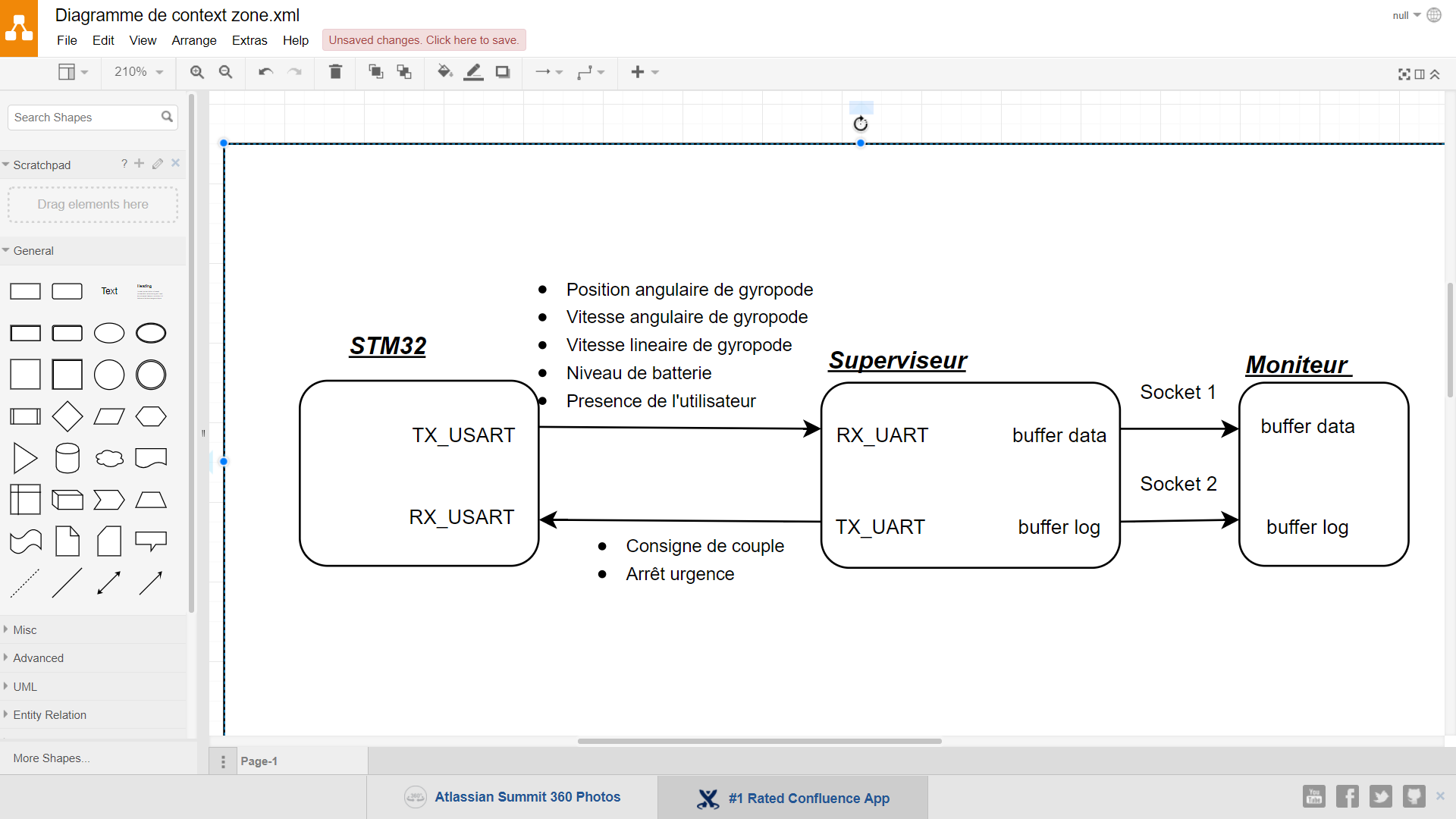


Figure 10 Diagramme de contexte avec Entrée-Sortie

• 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 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. Ce programme est exécuté dans le noyau Linux et 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.

### 3.3 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 le protocole UART, qui est asynchrone.

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

Dans notre programme, 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 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.

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

Fonctions logicielles

Non temps réel :

* Acquisition des mesures (angle, vitesse angulaire, niveau de batterie, direction, présence de l'utilisateur)
* Estimation de la vitesse linéaire (à partir du courant des moteurs, approximation couple constant)
* Boucle de courant
* Transmission des données (UART côté STM32)

Temps réel :

* Asservissement en angle
* Surveillance du niveau de batterie
* Surveillance de la présence de l'utilisateur
* Gestion d'arrêt d'urgence

### 3.4 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 gyropode 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. 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: *add\_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.

### 3.5 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 gyropode. Nous avons distribué les fonctions en 7 threads :

2.4.1 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. La fréquence a été choisie pour correspondre à celle de l'envoi des données par le STM32. La tâche déchiffre les trames et met à jour les variables partagées utilisées par les autres threads. Elle doit d’abord vérifier que la communication UART est bien établie. Si ce n’est pas le cas, elle 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. Ce thread fonctionne à une fréquence de 50 Hz.

2.4.2 Asservissement

Le thread Asservissement effectue les calculs nécessaires au contrôle des déplacements du gyropode. Si la variable état communication est égale à *vrai*, il récupère les informations d’angle, et après son calcul, écrit les résultats dans la variable partagée de consigne courant et ajoute ces valeurs à la file de message. La fréquence de 50 Hz correspond à la fréquence d'échantillonnage calculée lors de la conception des lois d'automatique. La sauvegarde de la valeur de consigne de couple sert au thread Affichage qui relaie cette information au programme GUI.

2.4.3 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 *batterie* 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. La fréquence de ce thread n'a pas besoin d'être plus élevée, le niveau des batteries évoluant de manière relativement lente. Afin de pouvoir notifier la remise en marche du système après un arrêt d'urgence, il est nécessaire de remettre la variable partagée *arret.* C'est cette tâche qui a été choisie pour cette fonction car sa fréquence car il n'est pas nécessaire de faire la vérification du bon fonctionnement du système à une fréquence supérieure à 1 Hz. Cette tâche consulte alors régulièrement la valeur de la variable *presence\_user* et remet à 0 la valeur de la variable *arret* si *presence\_user* est à l'état *true*.

2.4.4 Affichage

Le thread Affichage se charge de communiquer à l’utilisateur toutes les informations importantes récoltées par le STM32 ainsi que les variables du programme de temps réel comme l'état de la communication ou le déclenchement de l'arrêt d'urgence. Lors de sa première exécution, il initialise le socket qui sera utilisé pour communiquer les informations au processus de l'interface graphique. Ensuite à chaque exécution, il consulte certaines variables partagées et envoie des trames sur un socket pour que le programme GUI puisse les afficher. Pour bien pouvoir notifier la perte de communication, la priorité de ce thread doit être supérieur au thread Communication.

2.4.5 Présence User

Le thread Présence User vérifie de manière périodique à 10 Hz la présence de l’utilisateur en consultant la variable partagée *presence\_user*. Si cette variable est à l’état false, la variable arrêt est mis à l’état *true*, afin que l’arrêt d’urgence soit déclenché. Puisque le STM32 effectue lui-même un comptage de 500 ms, il n'est pas nécessaire de le faire dans le système de temps réel. *Cependant dans la version simulateur, utilisée par l'étudiant, il n'y a pas de compteur de 500 ms au niveau du STM32, ce qui signifie que l'étudiant devra gérer cette contrainte de temps (qui vise à éviter qu'une absence de l'utilisateur soit détectée suite à un faux contact).* 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).

2.4.6 Arrêt d'urgence

Le thread Arrêt Urgence est déclenché par le sémaphore *arret* (qui peut être déclenché par le thread Surveillance Batterie ou Présence User). Nous avons choisi un fonctionnement apériodique car cette tâche a seulement besoin d'être exécutée lorsqu'il est nécessaire d'arrêter le système, un fonctionnement périodique aurait fait consommer des ressources inutilement. Lorsque la tâche est déclenchée, la variable partagée arrêt est mise à *true*,et un message de type arrêt (de label "a"), avec la valeur égale à "1" est envoyée à la file de message.

2.4.7 Envoyer

Le thread Envoyer est chargé d’envoyé au STM32 les messages qui sont stockés dans la file de messages. Etant donné que le thread Asservissement écrit un message dans la file à 50 Hz, et que le thread Arrêt peut également ajouter un message dans la file, cette tâche doit être périodique et deux fois plus rapide que la tâche Asservissement, elle fonctionne donc à 100 Hz.

# Chapitre 4

## Simulateur du système gyropode

Le simulateur du système gyropode est destiné à être utilisé par les étudiants de 4-ème année Ingénierie des Systèmes. Il est la même carte de STM32 que nous avons utilisée dans la réelle maquette, et toutes les interruptions et les fréquences des tâches sont identiques que le code de la réelle maquette pour simuler le plus proche possible.

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 gyropode, tandis que l’autre sert à simuler quelques paramètres tels que le niveau de batterie, ou la présence de l’utilisateur.

### 4.1 Simulation physique du gyropode

La simulation du comportement physique du gyropode est effectuée grâce au modèle du système qui nous a été fourni. En effet, il s’agit le modèle 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°. Ce modèle est donc basé autours de deux matrices A et B.

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. Mais pour calculer la vitesse linéaire, il nous faut également la tension fourni aux moteurs, qui n’est pas disponible dans le simulateur. C’est pourquoi dans le simulateur, il n’y a pas de variable de vitesse linéaire comme la vraie maquette

### 4.2 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. Nous avons réalisé avec deux façons suivantes :

1. La modification de variables peut être faite à travers une communication sur un Virtual Com Port (VCP) du STM32. La carte est connectée avec un ordinateur via USB et des modifications de variables du système sont transmises grâce à un terminal RS232 de côté d’ordinateur. Après avoir choisi des nouvelles valeurs des variables, ces valeurs sont mises dans une trame et envoyées au STM32. Elles sont donc communiquées au système de temps réel comme sur la maquette réelle. La carte STM32 reçoit la trame et il interprète des valeurs de variables dans la trame. Mais nous n’avons pas maitrisé la configuration de VCP (USB communications device class), il nous faut désactiver l’interruption USART de réception pour éviter des conflits. Pour le réaliser, le bouton de carte STM32 est configuré comme une interruption externe pour désactiver USART et puis le réactiver. Le simulateur peut marcher mais le comportement est système n’est pas le même comme la réalité. De plus, les modifications des variables ne sont pas instantanées et nous ne pouvons pas modifier les valeurs (comme l'angle β) de manière continue. Ce n'est donc pas la méthode que nous avons retenue.
2. La carte du simulateur est connectée à deux potentiomètres et un bouton switch. La modification des variables se fait directement grâce à ces périphériques. Les configurations sont alors identiques à la réelle maquette et les variations sont instantanées et peuvent être continues, ce qui se rapproche plus de la réalité.
   1. Potentiomètres :

Toutes les lectures des valeurs de potentiomètres sont effectuées par l’ADC du STM32 qui utilise la DMA pour effectuer une moyenne glissante sur les valeurs mesurées.

* + 1. Un potentiomètre sert à simuler le niveau de batterie
    2. Un autre potentiomètre sert à simuler la valeur de l'angle utilisateur () entre -20 degrés et +20 degrés.
  1. Boutons :
     1. Le switch sert à simuler le bouton de présence utilisateur, présent sur le guidon de la maquette.
     2. Le bouton bleu sur la carte de simulateur (STM32) sert à changer le mode de modification de l'angle utilisateur (), soit ce dernier est indiqué par le potentiomètre, soit il est simulé par des capteurs intégrés dans la carte STM32 (fusion d’angle de gyroscope et accéléromètre).

Nous avons donc choisi la deuxième méthode car elle est plus intuitive pour l'étudiant et plus simple pour nous à développer.

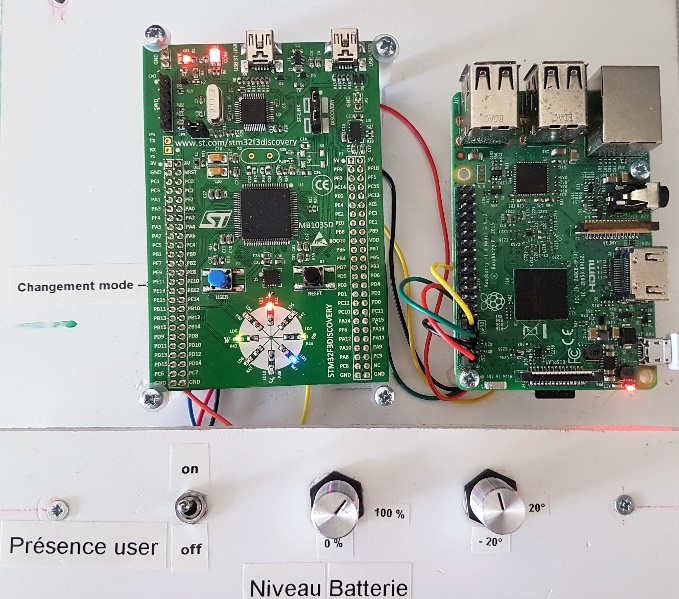


Figure 11 Photo de Simulateur

Pour communiquer avec la Raspberry Pi, nous avons connecté le pin 10 (RX) de la Raspberry au pin PA2 (TX) du STM32. Le pin 8 (TX) de la Raspberry est connecté au pin PA3 (RX) du STM32. Enfin on connecte le pin GND de STM32 au pin 6 du Raspberry pour avoir la masse commune.

Les périphériques de STM32 sont configurés et connectés comme suivant :

|  |  |  |
| --- | --- | --- |
| Numéro de pin | Mode de pin | Fonction |
| PC2 | ADC | Angle de -20 ° à 20 ° |
| PC3 | ADC | Pourcentage de niveau de batterie (%) |
| PA3 | USART | Réception de données de la Raspberry Pi |
| PA2 | USART | Envoi de données à la Raspberry Pi |
| PA4 | GPIO | Bouton de présence de l’utilisateur |

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 |

# Chapitre 5

## Remarques

### 5.1 Ajustements

Log\_queue erreur

Au bout d'un certain moment, la file de messages qui sert à l'envoi des chaînes de caractères de log à l'intention du GUI par le socket est pleine. (*rt\_queue\_write* du fichier lib\_monitor.c renvoie une erreur)

Simulation à revoir

D’après la vérification de M. Hazyuk, nous avons aperçu notre résultat de simulation n’est pas correspondant comme prévu. Le modèle que nous avons utilisé (tiré du rapport de projet tutoré) pour la simulation mécanique du système n'est pas correcte, l'angle θ devrait se stabiliser autour de 0, ce n'est pas le cas. A voir avec le GM pour corriger le modèle ?

Bug démarrage temps réel

Il arrive que le programme de temps réel ne démarre pas correctement, dans ce cas des printf le signalent : "*Problème de communication, redémarrer les deux programmes*". Cela doit venir d'un problème lors de l'initialisation de l'UART (tâche Communication)

### 5.2 Reste à faire

Direction guidon (code STM32)

Ayant eu la jauge de contrainte en mauvais état à la fin du stage, nous n'avons pas pu mettre en place la gestion de la direction. Nous l'avions fait à un certain moment, le code est commenté, au niveau de la fonction *Trait\_courant* (partie STM32).

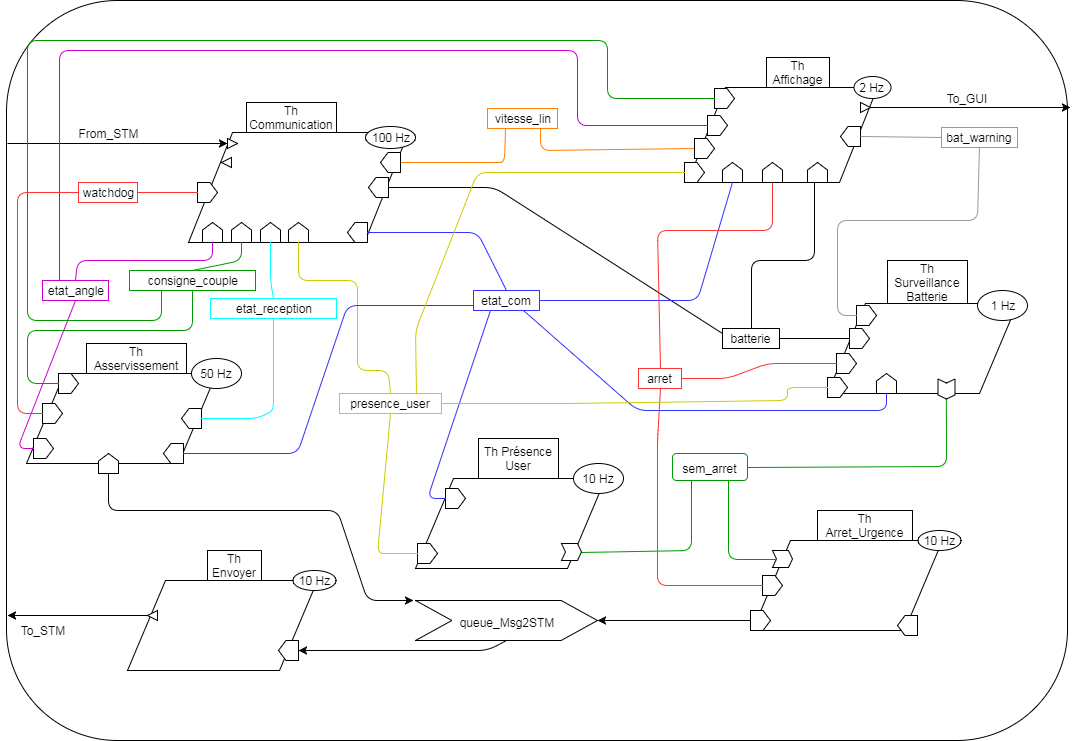
Limitation vitesse (code Temps Réel)

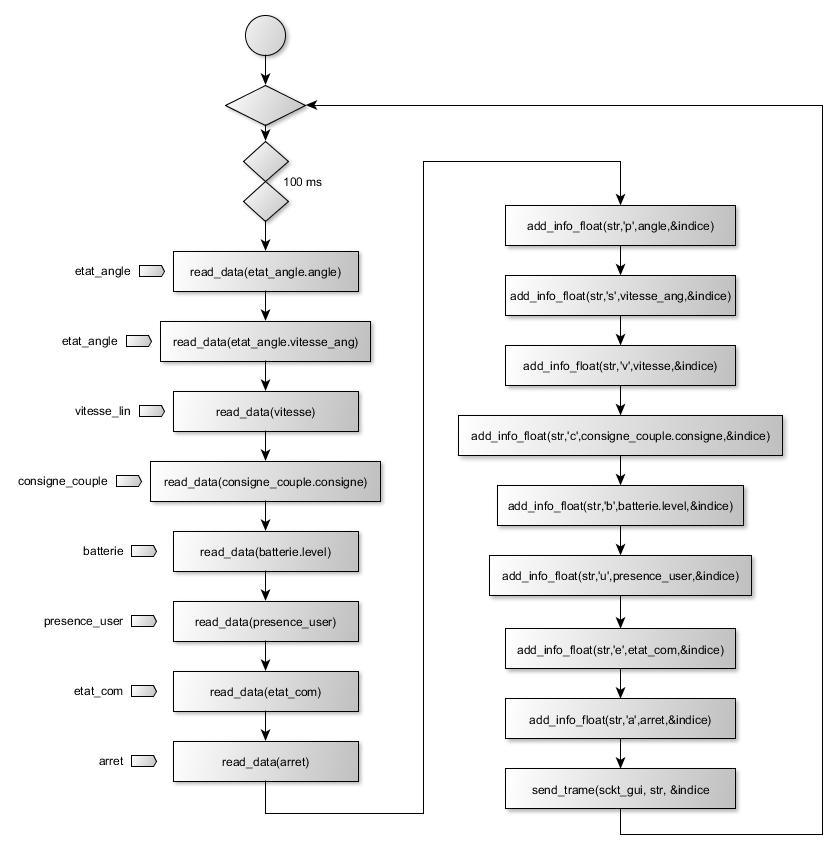
Nous n'avons pas mis en place la limitation de vitesse linéaire, elle est à coder dans le thread Asservissement. Lorsqu’une certaine vitesse est dépassée, la consigne d'angle doit réagir par rapport à la vitesse linéaire du gyropode pour que le système se redresse et freine.

Configuration Netbeans

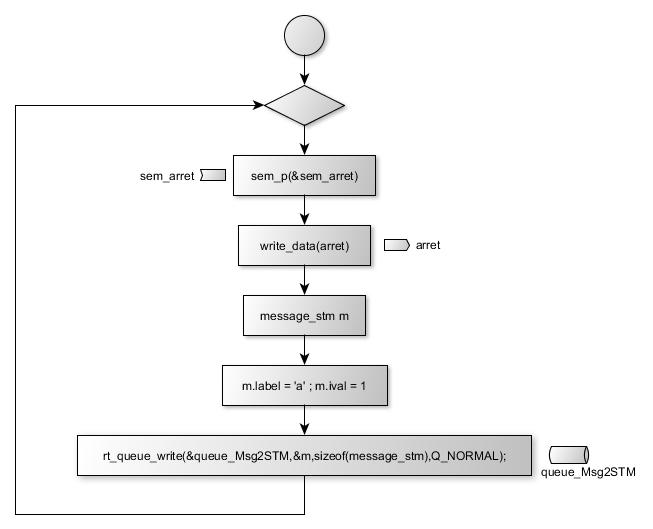
Configurer Netbeans pour pouvoir développer le code de temps réel à distance (SSH) et également compiler et exécuter le programme.

# Annexe

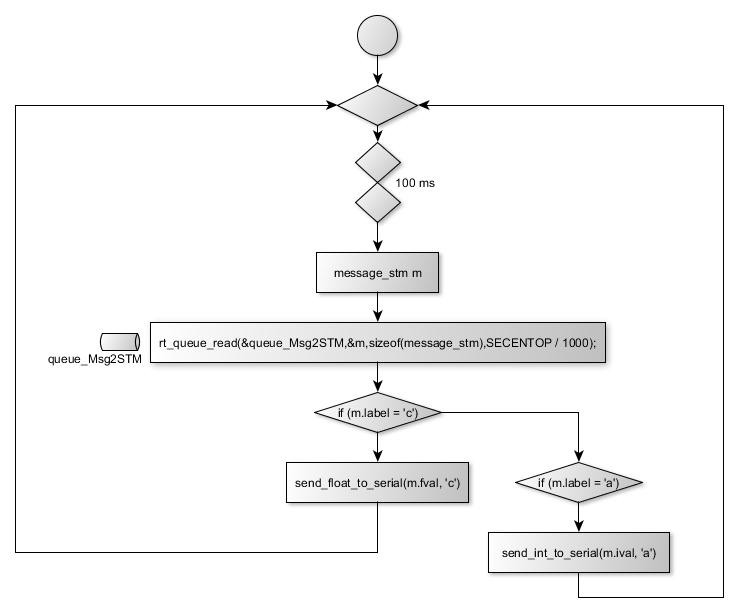
1. Diagramme AADL
2. Diagramme Activité Affichage



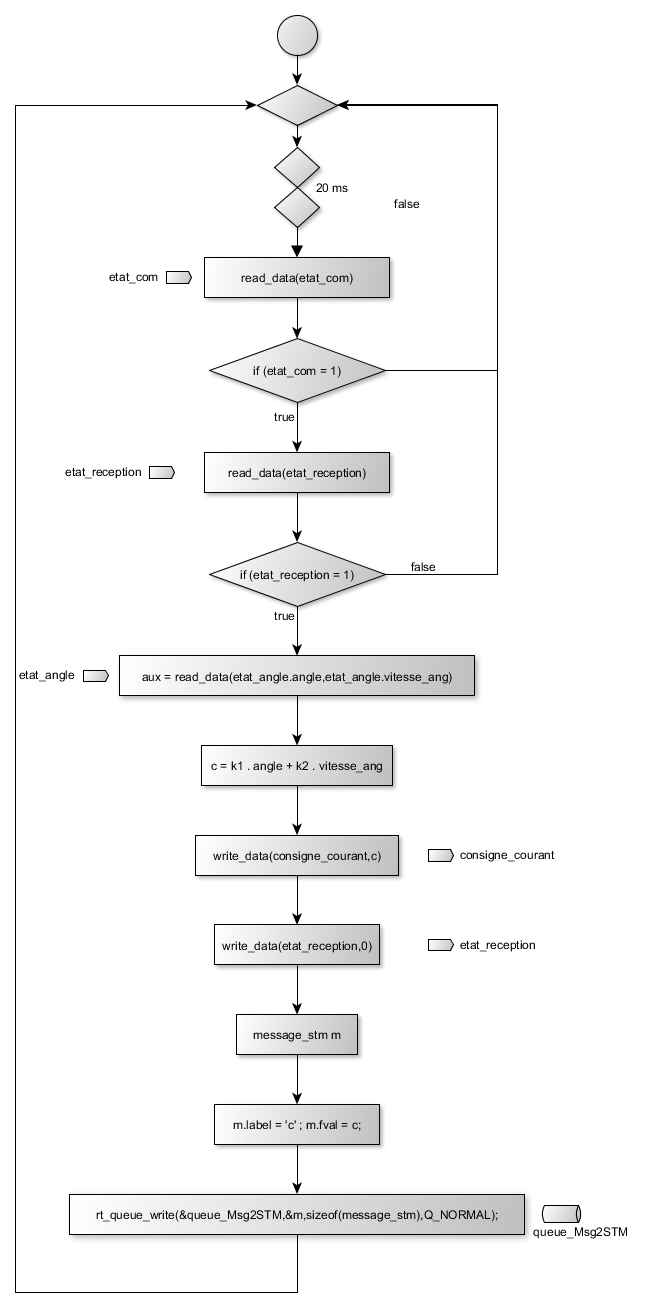
1. Diagramme Activité Arrêt Urgence



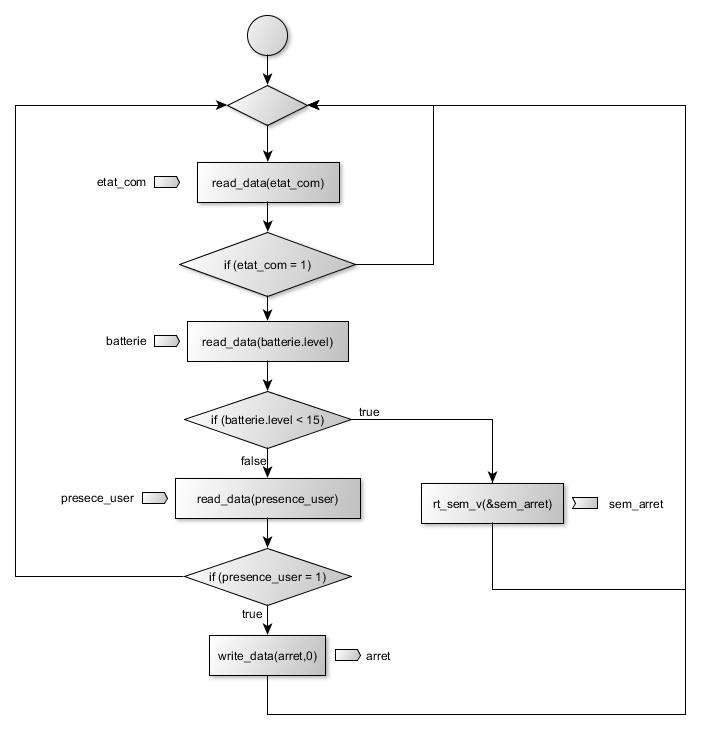
1. Diagramme Activité Envoyer



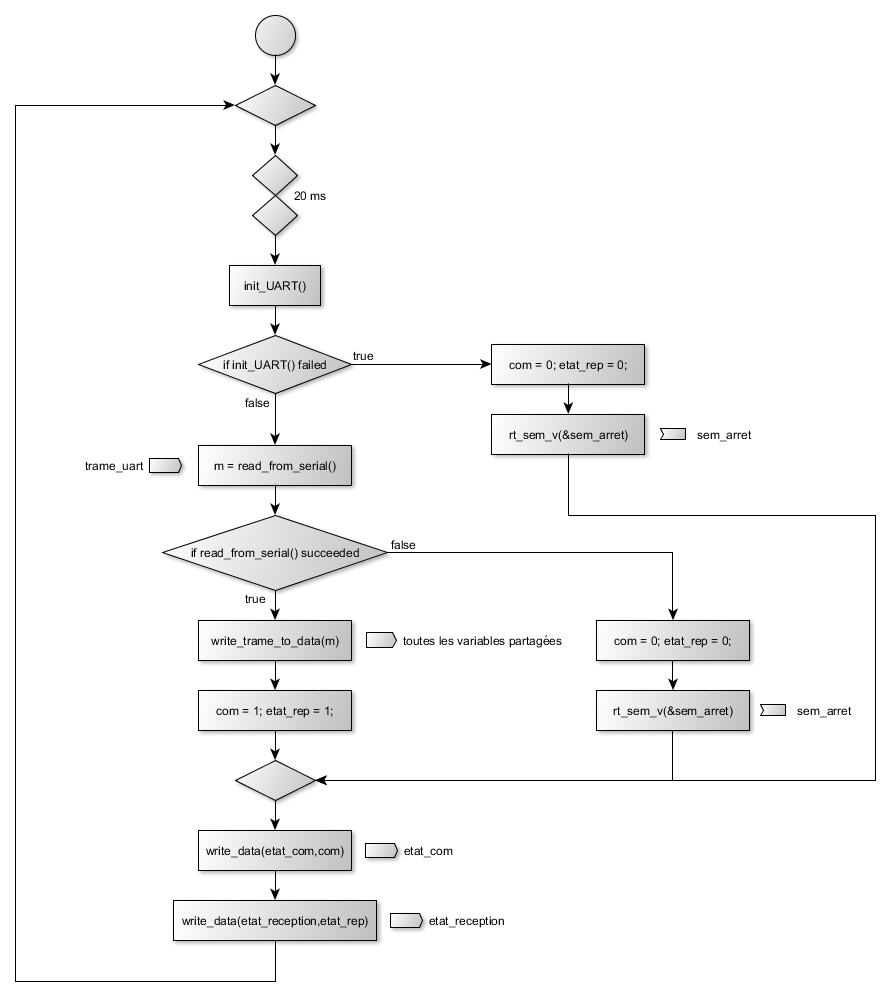
1. Diagramme Activité Asservissement



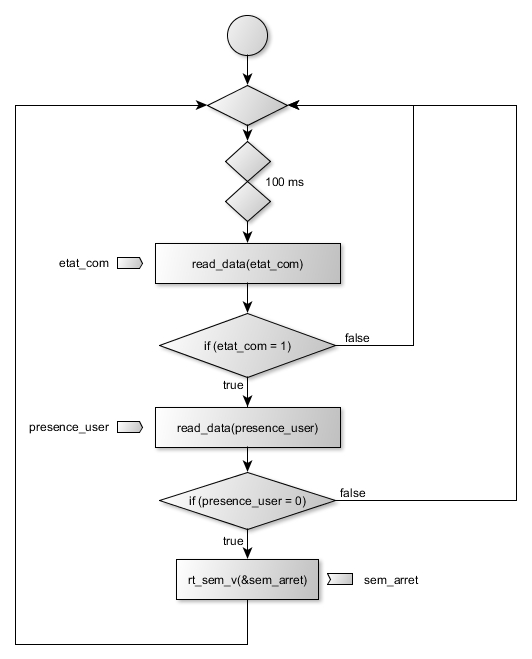
1. Diagramme Activité Surveillance\_batterie



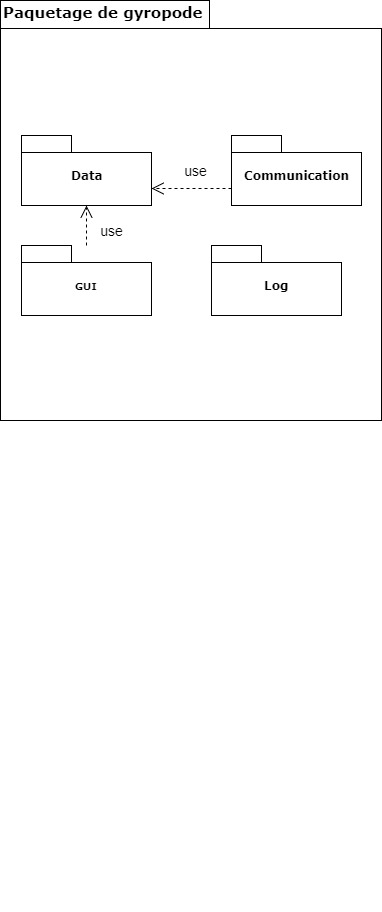
1. Diagramme Activité Communication



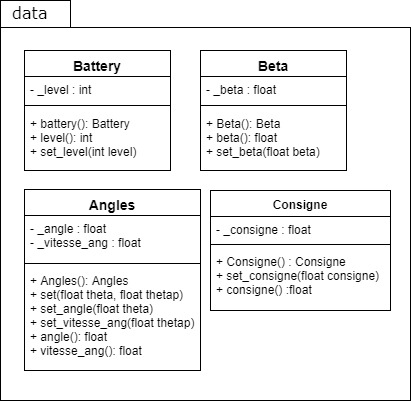
1. Diagramme Activité Presence\_User



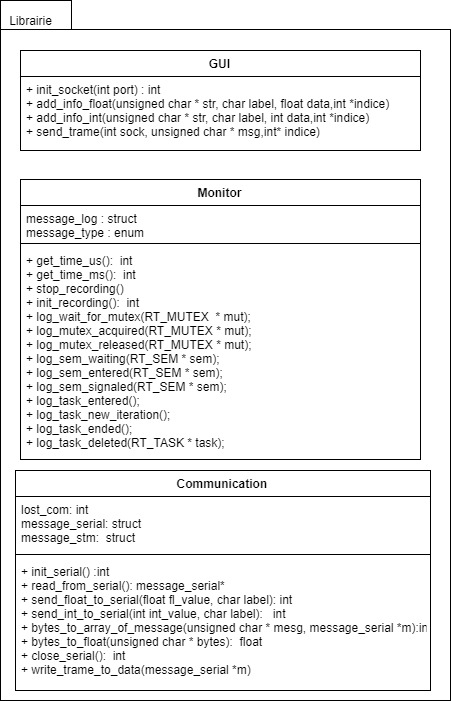
1. Diagramme de paquetage



1. Diagramme de classe du paquetage données



1. Diagramme de classe du paquetage librairie



1. Capture écran de GUI

