TP Temps Réel Gyropode

INSA Toulouse par HENG Chun Yang et PENIN Dorian

Table de matières

[Partie 1. Présentation du gyropode 3](#_Toc495586849)

[1.1 Mission et objectifs du gyropode  3](#_Toc495586850)

[1.2 Exigences  6](#_Toc495586851)

[1.3 Architecture physique 6](#_Toc495586852)

[1.4 Commande du gyropode 8](#_Toc495586853)

[1.5 Architecture logique 9](#_Toc495586854)

[Partie 2. Commande en temps réel du gyropode 11](#_Toc495586855)

[2.1 Utilisation du noyau temps réel Xenomai 11](#_Toc495586856)

[2.2 Présentation Xenomai-Cobalt 12](#_Toc495586857)

[2.3 Présentation du logiciel de supervision 13](#_Toc495586858)

[2.4 Correction : AADL complet 15](#_Toc495586859)

[Partie 3. Prise en main du TP 16](#_Toc495586860)

[3.1 Présentation du matériel (Simulateur) 17](#_Toc495586861)

[3.2 Présentation de l’interface graphique 18](#_Toc495586862)

[3.3 Présentation du log du programme de supervision 19](#_Toc495586863)

[3.4 Présentation de Netbeans 20](#_Toc495586864)

[Annexes 22](#_Toc495586865)

# Partie 1. Présentation du gyropode

### 1.1 Mission et objectifs du gyropode

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. La direction du gyropode est contrôlée par l'angle que forme le guidon avec le plateau. Cet angle est mesuré par une jauge de contrainte et est compris en -20° et 20°.

Le système de notre maquette est destiné à déplacer une personne seule (charge utile max 100 kg), sur un terrain plat 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.

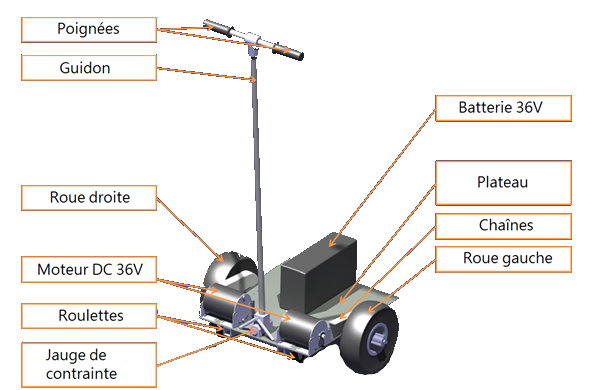


Figure 1 Modèle de la maquette de gyropode de TP (pour disposition des éléments dans le plateau, voir figure suivante)

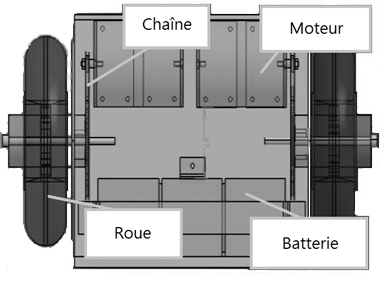


Figure 2 Vue de dessus de l'intérieur du plateau de la maquette du gyropode

La commande de la vitesse du gyropode est effectuée 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.

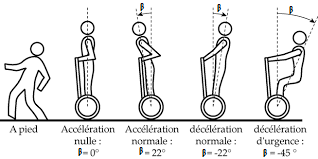


Figure 3 Cas d'utilisation et représentation de l'angle β

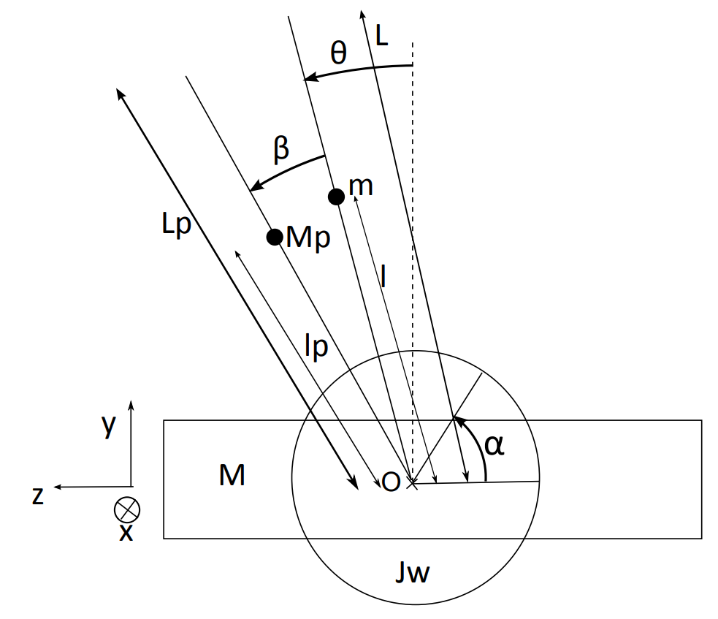


Figure 4 Schéma mécanique du gyropode

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 qui est d'habitude égale à θ = 0, est modifiée afin que le système se rétablisse de manière forcée. Si la vitesse dépasse 7 km/h, la consigne évolue progressivement de manière négative. 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. La détection de l’absence de l’utilisateur sur le plateau doit être effectuée dans un temps suffisamment court pour prévenir tout accident. 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 supplémentaires peuvent être envisagées, selon le modèle et l'utilisation du système, comme un mécanisme de géolocalisation et de verrouillage/déverrouillage à distance par exemple, si le gyropode est utilisé dans le cadre d’un système de location similaire au Vélib (cf. projet de l’UF « Processus d’Ingénierie Système »).

### 1.2 Exigences

Le système doit donc satisfaire plusieurs exigences. Certaines sont fonctionnelles (E1, E2) et d'autres non fonctionnelles (EF7, EF8) :

E1 : Le gyropode doit permettre le déplacement de l'utilisateur (accélération, freinage, arrêt, direction) à une vitesse < 7 km/h.

E2\* : Le gyropode doit pouvoir limiter la vitesse de l'utilisateur si celle-ci dépasse les 7 km/h.

E3\* : Le gyropode surveille le niveau de la batterie régulièrement (toutes les secondes). Il affiche une alarme 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.

E4\* : Le gyropode doit s'arrêter en moins de 500 ms lorsque l'utilisateur n'est plus présent sur le système.

E5 : Le gyropode doit permettre à l'utilisateur de connaître sa vitesse, le niveau de batterie et l'état général du système à travers l'affichage de ces valeurs ainsi que l'affichage d'informations textuelles.

EF6 : Le gyropode doit protéger physiquement l'utilisateur en cas de choc frontal à 10 km/h.

EF7 : Le gyropode doit permettre de le conduire la nuit.

EF8 : Le gyropode doit être protégé de la pluie et de l'humidité.

EF9 : Le gyropode doit être facilement transportable

Les exigences marquées d'une \* sont les exigences qui doivent être remplies par le programme de supervision.

### 1.3 Architecture physique

#### 1.3.1 Architecture physique générale

Le gyropode est constitué d'un châssis, dans lequel se trouvent deux batteries, deux moteurs ainsi que la carte STM32. Les moteurs sont reliés aux roues grâce à un système de réduction à chaînes qui se trouve à l'intérieur du châssis. Le gyropode est composé d'un sous-système de motorisation (moteur, chaînes, roues), piloté par la carte STM32 grâce à un pont en H et une régulation par PWM (voir section 4.2.). A l'avant du châssis est fixé le guidon, formant une liaison pivot. L'angle de ce pivot est mesuré grâce à la jauge de contrainte. En haut du guidon, sur le manche, se trouvent deux boutons, servant à détecter la présence de l'utilisateur, ainsi qu'un écran destiné à l'affichage de l'interface graphique.

On peut donc structurer le système en deux sous-systèmes: un sous-système de motorisation, comprenant le moteur, les chaînes, la carte électronique du pont en H, et les batteries; et un sous-système de commande avec la carte STM32, la Raspberry Pi 3 ainsi que tous les capteurs du système. Ce dernier sous système de commande peut être éclaté en deux sous-systèmes : un sous-système bas niveau comprenant les capteurs et la carte STM32, et un sous-système haut niveau, ou de *supervision*, avec la carte Raspberry Pi 3, qui exécute le programme de supervision que vous allez concevoir, ainsi que l'interface graphique.

#### 1.3.2 Architecture physique du sous-système de commande

L'architecture physique du sous-système de commande est composée de deux cartes ayant des rôles différents, une carte STM32 et une carte Raspberry Pi 3. Le microcontrôleur STM32, situé physiquement au niveau du châssis, s'occupe de la partie bas niveau de la programmation, c’est-à-dire la récolte des informations des capteurs et leur envoi au sous-système de supérieure via le protocole UART. La carte Raspberry Pi 3, située au niveau du guidon, s'occupe donc de la partie supervision et affichage d'informations.

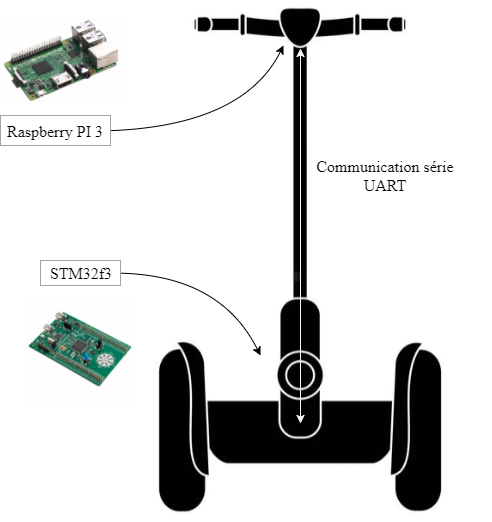


Figure 5 Schéma de positionnement des cartes électroniques sur le châssis

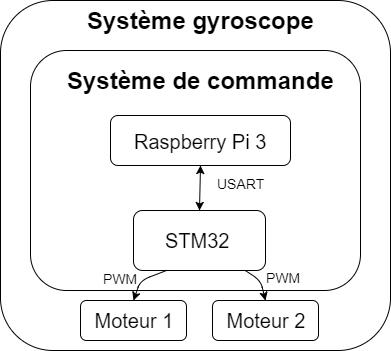


Figure 6 Architecture physique du sous-système de commande

### 1.4 Commande du gyropode

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 ; elle 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.

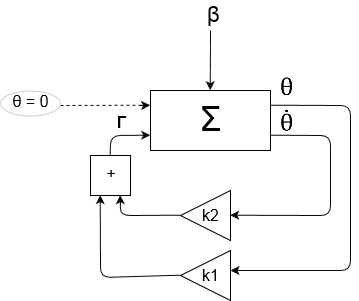


Figure 7 Schéma bloc de la boucle d'asservissement principale

#### 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.

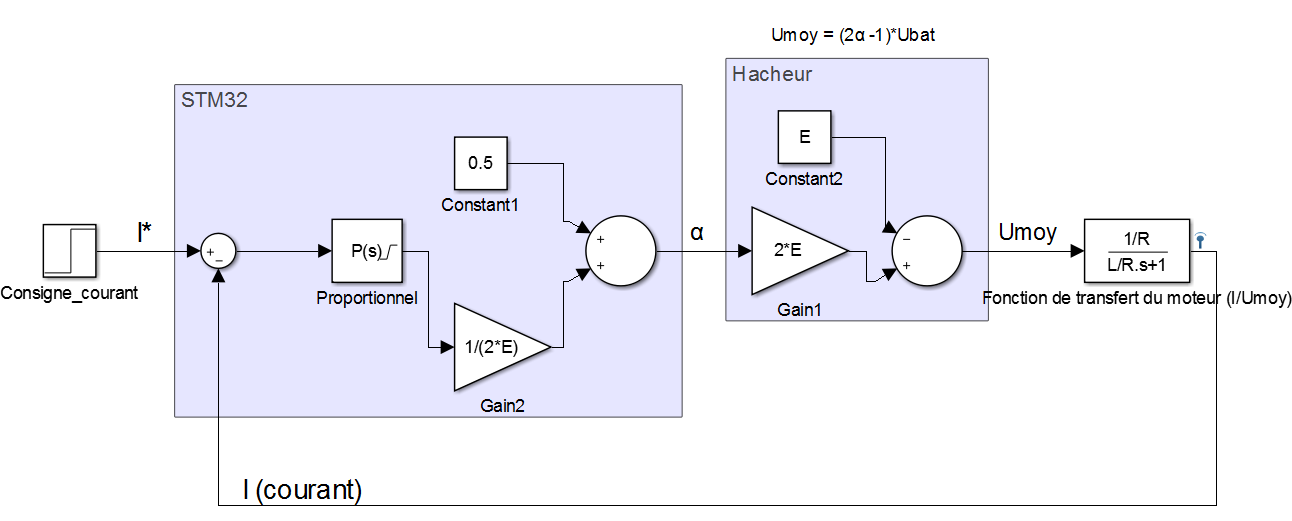


Figure 8 Asservissement de courant pour la commande des moteurs

### 1.5 Architecture logique

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

* Le STM32 collecte 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 (voir figure 5). De plus, c'est cette carte qui est chargée d'effectuer la boucle de courant (voir section 1.4). 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 présence utilisateur est détectée lors de l'appui sur l'un des boutons du guidon. Cette information est envoyée à la Raspberry Pi 3 au même titre que les autres informations des capteurs citées précédemment.
* La carte Raspberry Pi s'occupe de la supervision du système. Elle traite les informations du STM32 pour assurer les fonctions du gyropode telles que la surveillance de la batterie et la gestion de l'arrêt d'urgence grâce à l’exécution d’un logiciel sous contraintes de 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 (ex: 'c' pour consigne, 'b' pour batterie), 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.

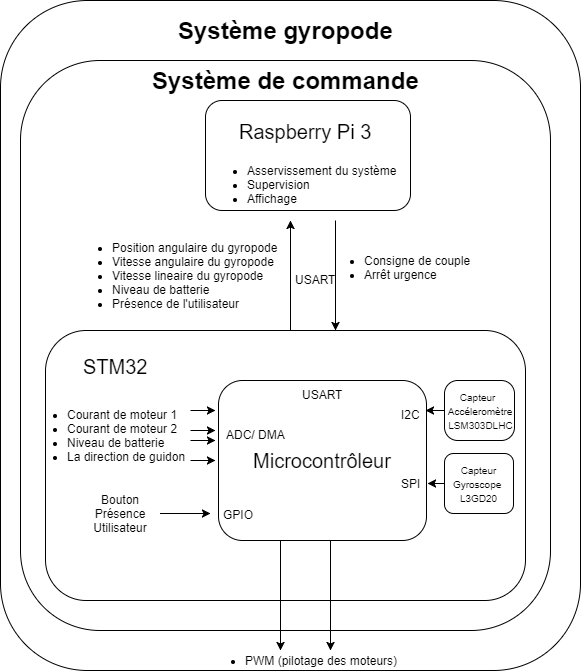


Figure 9 Flux de données pour le sous-système de « commande »

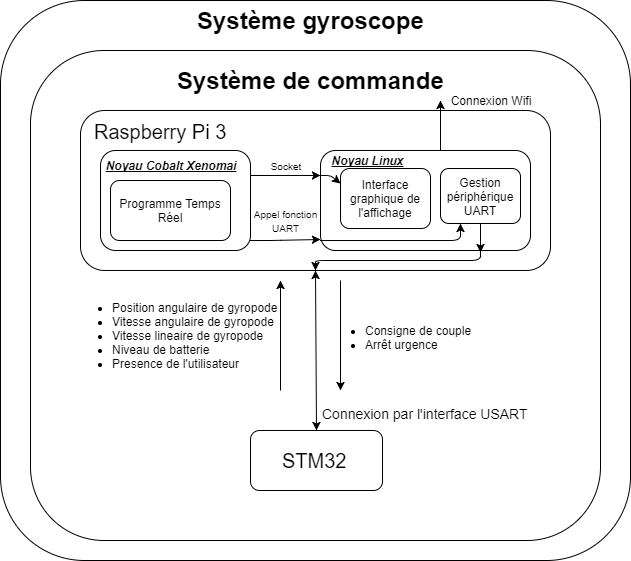


Figure 10 Diagramme de contexte

# Partie 2. Commande en temps réel du gyropode

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

Le Raspberry Pi est un nano-ordinateur mono-carte à processeur ARM. La carte embarque le programme de supervision du gyropode, qui effectue les opérations de temps réel, en interaction avec la carte STM32.

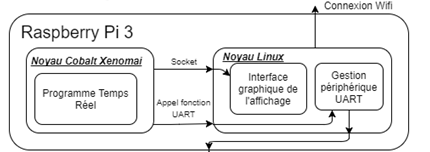


Figure 11 Diagramme de contexte de Raspberry Pi 3

### 2.1 Utilisation du noyau temps réel Xenomai

• Partie Linux

Un programme d'interface graphique est disponible, qui affiche des données du système et l'historiques des événements en temps réel (à faire apparaître sur la figure). 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.

### 2.2 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.

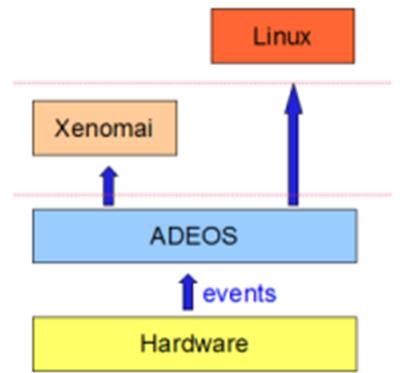


Figure 12 Schéma de couche de Co-noyau dans Xenomai 3

### 2.3 Présentation du logiciel de supervision

Le logiciel de supervision est le programme de temps réel qui s'exécute en utilisant le noyau de cobalt. Mais dans ce logiciel, des fonctions non temps réel (qui s'exécute sous linux) sont appelées pour permettre l'affichage de données (socket) et également la communication avec la partie de commande des moteurs (STM32).

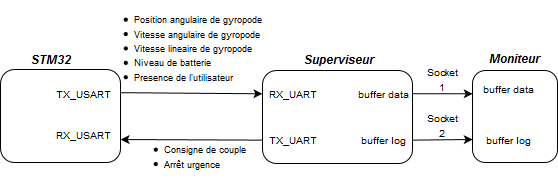


Figure 13 Diagramme de contexte avec Entrée-Sortie

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

Vous devez, à partir des 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.

##### 2.3.1 Conception du programme

Des moyens de synchronisation et de protection fournis par Xenomaï 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.

##### 2.3.2 Validation de la conception

A partir des exigences fonctionnelles, il faut prévoir dès le début la façon dont vous allez vous assurer que le fonctionnement de votre application est conforme au comportement attendu. Vous allez donc élaborer et décrire par écrit des scénarios de tests, que vous allez dériver des exigences, fonctionnelles et temporelles, qui couvrent le plus exhaustivement ces exigences, et que vous exécuterez ultérieurement 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.

##### 2.3.3 Programme initial

Un programme incomplet et mis à votre disposition. Deux tâches sont déjà fonctionnelles: *Envoyer* et *Affichage*. Ce programme compile et est exécutable, vous pourrez le lancer pour prendre en main le matériel.

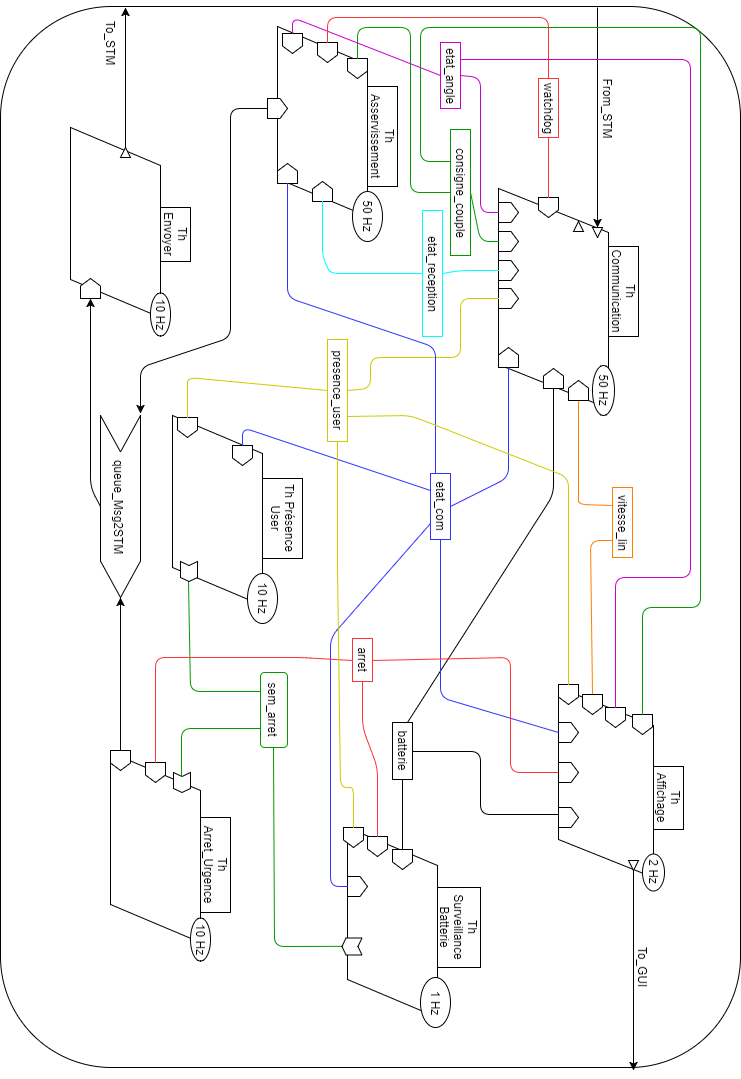
* Communication

Le thread de Communication se charge de récupérer périodiquement les informations reçues à l’aide de la liaison série du 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.

* 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 Correction : AADL complet



# Partie 3. Prise en main du TP

Cette partie présente tout ce que vous devez savoir sur le matériel disponible et les manipulations à faire lors des séances de TP. Lors des TP vous travaillerez au sein d’un groupe de trois ou quatre. Chaque groupe s’auto organise et dispose de deux PC pendant les créneaux réservés.

Vous pouvez travailler dans toutes les salles informatiques du GEI pour le développement.

Pour des raisons de sécurité le code de temps réel sera d'abord programmé et testé sur un simulateur qui reproduit le comportement physique du gyropode avant l'embarquement du code directement sur la maquette.

Le matériel disponible est une carte Raspberry Pi (carte sur laquelle s'exécutera votre programme superviseur) et une carte de STM32 (qui exécute le programme qui simule le comportement du gyropode.

Les deux ordinateurs sont lancés dans une session Linux, l'un sert à la programmation du code de temps réel et le deuxième est pour l’observation des données de programme temps réel.

Le développement du code temps réel sera effectué dans les ordinateurs de bureau des salles TP sous Linux. Comme la compilation de code doit se faire dans la carte Raspberry Pi, afin de faciliter la programmation à distance, vous utiliserez l’IDE de programmation Netbeans. Il vous permettra de connecter la Raspberry Pi à distance avec protocole SSH et compiler à distance.

Les procédures de configuration de l’environnement Netbeans sont décrites dans la section 3.4.1

Pour pouvoir observer des variables internes de simulateurs, une interface graphique est disponible. Pour lancer ce programme, il faut démarrer une session Linux. Pour que la carte Raspberry Pi puisse communiquer avec la session Linux via Wifi, 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 (il faudra renseigner cette adresse IP dans le programme de temps réel)
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*

A l’exécution de ce programme d'affichage, un socket est créé, qui attend la connexion de la carte Raspberry Pi qui s'effectuera à l'intérieur du thread Affichage. Il faut ensuite lancer le programme de temps réel pour le connecter.

(ps. Nous n’avons pas de photo de poste de travail ni des détails sur le téléchargement de code et la compilation sous IDE Netbeans car ce ce n’est pas nous de s’occuper. Ici il manque encore des choses à faire)

### 3.1 Présentation du matériel (Simulateur)

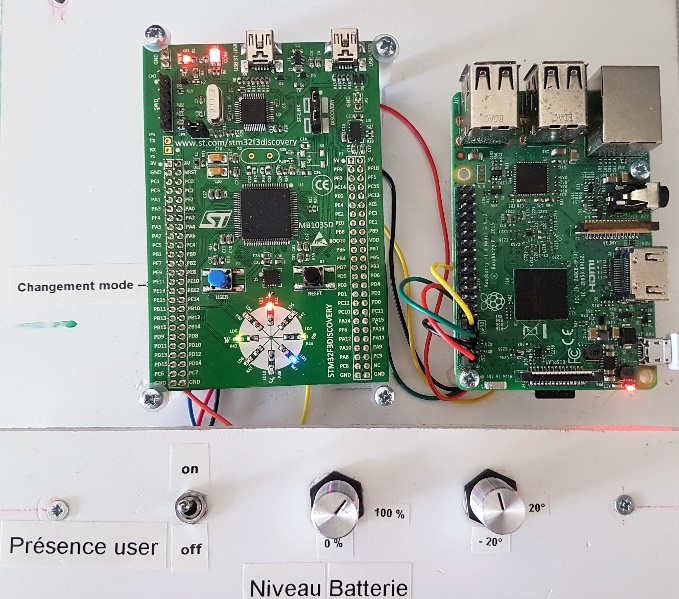
Le simulateur a pour but de simuler le comportement physique de la maquette du gyropode. Pour pouvoir reproduire le comportement de la maquette, nous utilisons une carte STM32 qui est le même modèle que celle présente dans la maquette pour assurer un fonctionnement et une communication identique.

La carte STM32 peut simuler le niveau de batterie, la présence d’utilisateur et également l’angle relatif entre l’utilisateur et le guidon. Les données utilisées par le simulateur sont renseignées grâce à un bouton et deux potentiomètres. L’angle peut également être réglé en inclinant le plateau du simulateur en basculant de mode grâce au bouton "user".

La simulation 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°.

(Partie du superviseur)  
Raspberry Pi

(Partie du simulateur)  
STM 32



Potentiomètre 1

Switch de présence d’user

Potentiomètre 2

Bouton Reset

Bouton Bleu

Figure 14 Photo de Simulateur et Raspberry Pi

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 °
2. *Bouton Reset :*   
   Le bouton pour réinitialiser le programme de simulation.

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

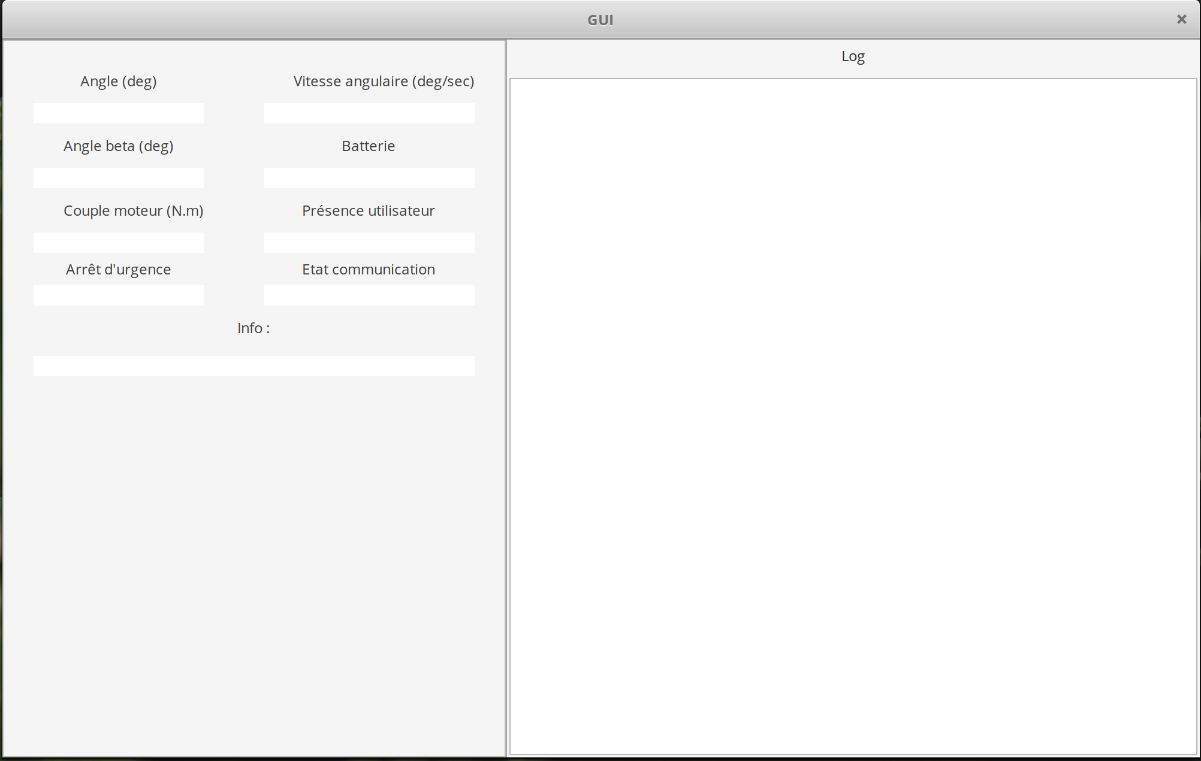
|  |  |
| --- | --- |
| Numéro de LED | Action |
| LED 3 | Clignote à la réception de la trame de consigne |
| LED 4 | Mode d’entrée de l’angle relatif de l’utilisateur au guidon  => ON : Mode d’accéléromètre => OFF: Mode de potentiomètre |
| LED 6 | Etat de présence de l’utilisateur  => ON : Présence => 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. L’interface graphique est déjà codée, et pour pouvoir afficher des données de temps réel, il faut utiliser la librairie de lib\_msg\_GUI dans le thread Affichage. L’envoi de données ne se fait que dans un sens, du programme de supervision vers le programme d'affichage, s'exécutant dans le noyau de Linux.

Les logs de temps réel seront affichés également dans cette interface graphique.



Afficher des logs en temps réel

Afficher des informations sur le simulateur de gyropode

Afficher des valeurs de variables partagées

Figure 15 Capture d'écran de GUI

### 3.3 Présentation du log du programme de supervision

Pour mieux comprendre l'exécution des tâches dans le programme de supervision, une librairie *lib\_monitor* permet d'écrire des informations d'exécution dans le document *log.xenolog*. Il s’agit d'une tâche de priorité faible qui tourne en fond, en même temps que le programme de supervision. Une file de message permet de stocker les évènements afin de les traiter lorsque le programme de temps réel ne consomme pas de ressources du processeur. Pour écrire des évènements dans le fichier de log, il faut utiliser des fonctions de la librairie dédiée. Voici des fonctions à utiliser dans le programme de temps réel avec ses descriptions:

|  |  |
| --- | --- |
| Noms de fonctions | Description de fonction |
| int init\_recording(); | Initialisation de l'enregistrement des évènements par le log (A lancer au début du programme principal) |
| int stop\_recording(); | Arrêt de l'enregistrement des évènements par le log (A lancer à la fin du programme principal) |
| void log\_wait\_for\_mutex(RT\_MUTEX \* mut); | Ecrire le log lorsque la tâche attend un mutex, cette fonction doit être écrire avant la fonction de *rt\_mutex\_acquire()* |
| void log\_mutex\_acquired(RT\_MUTEX \* mut); | Ecrire le log lorsque la tâche acquiert un mutex, cette fonction doit être écrire après la fonction de *rt\_mutex\_acquire*() |
| void log\_mutex\_released(RT\_MUTEX \* mut); | Ecrire le log lorsque la tâche relache un mutex, cette fonction doit être écrire après la fonction de *rt\_mutex\_release()* |
| void log\_sem\_waiting(RT\_SEM \* sem); | Ecrire le log lorsque la tâche attend un sémaphore, cette fonction doit être écrire avant la fonction de *rt\_sem\_p()* |
| void log\_sem\_entered(RT\_SEM \* sem); | Ecrire le log lorsque la tâche attend un sémaphore, cette fonction doit être écrire après la fonction de *rt\_sem\_p()* |
| void log\_sem\_signaled(RT\_SEM \* sem); | Ecrire le log lorsque la tâche signale un sémaphore, cette fonction doit être écrire avant la fonction de *rt\_sem\_v()* |
| void log\_task\_entered(); | Ecrire le log lorsque la tâche commence son exécution, cette fonction doit être écrire au début de la fonction |
| void log\_task\_new\_iteration(); | Ecrire le log lorsque la tâche commence une nouvelle itération, cette fonction doit être écrire au début de la boucle de fonction |
| void log\_task\_ended(); | Ecrire le log lorsque la tâche se termine, cette fonction doit être écrire à la fin de la fonction |
| void log\_task\_deleted(RT\_TASK \* task); | Ecrire le log lorsque une tâche est supprimée, cette fonction doit être écrire à la fin de la fonction de *deleteTasks()* de programme principale |

Exemple de logs de temps réel

|  |
| --- |
|  |
|  | [9745 ms] [MUTEX] Communication: mutex: mutex\_etat\_com => acquired |
|  | [9745 ms] [MUTEX] Communication: mutex: mutex\_etat\_com => released |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### 3.4 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.

#### 3.4.1 Configuration de l’IDE Netbeans

A faire

#### 3.4.2 Premier lancement de code

Un code à trous vous a été mis à disposition. Les tâches ont déjà été créées 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

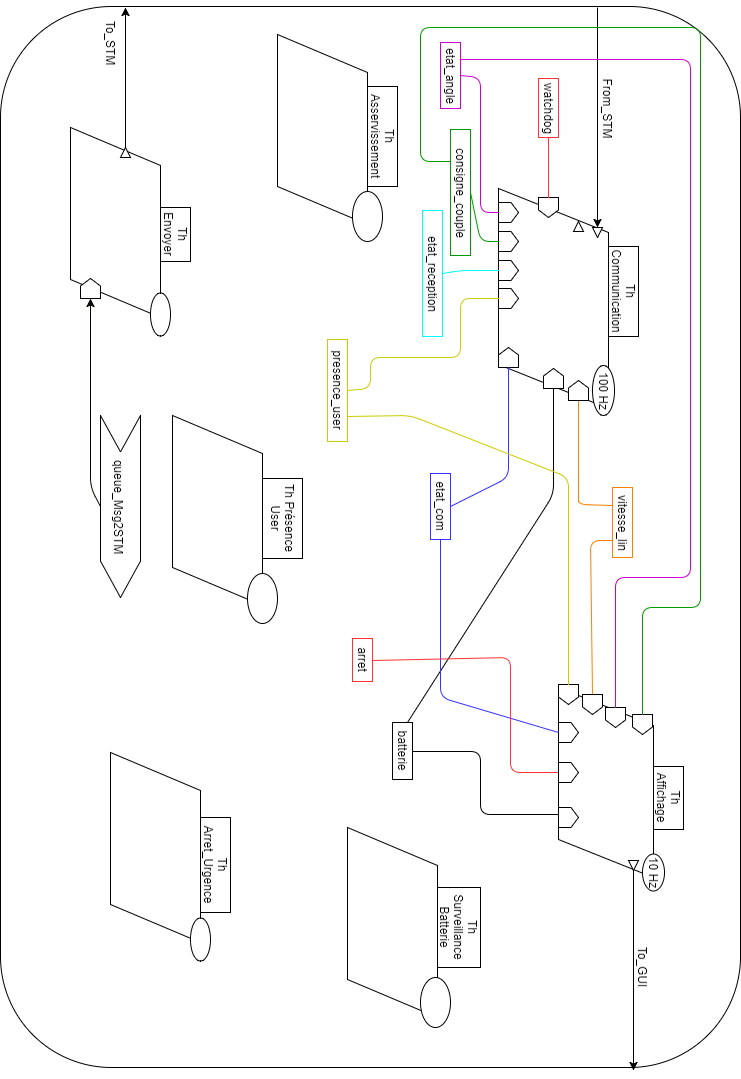
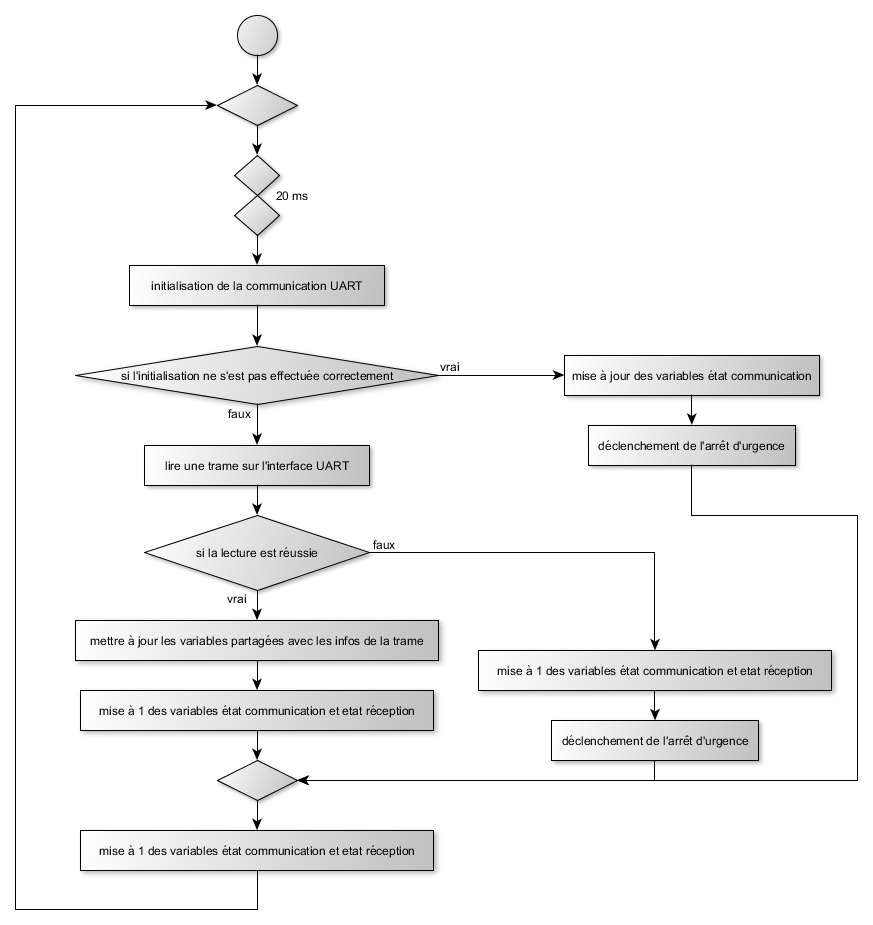
AADL:

Diagramme d'activités

Communication



Affichage

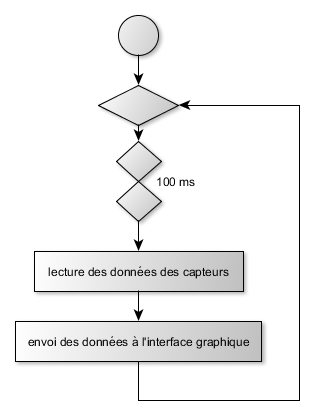
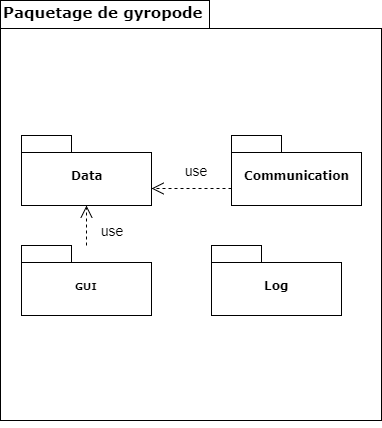


Diagramme de paquetage:



Diagrammes de classe

