Compte-rendu de projet

version 2015

--

Conception et programmation de systèmes temps réel

--

Projet De Stijl

Date : 03/03/2015

Noms : CANTEGRIL LALOY MAURY PREVOST (conception, robot, vidéo, intégration, rédaction du compte-rendu)

Temps consacré : 50 heures

Exigences

N° exigence	Description
E1 (S) Réinitialisation	Le superviseur doit se remettre dans l'état initial (variables réinitialisées, tâches redémarrées, traitement d'image arrêté) après que la communication avec le moniteur soit perdue.
E2 (S) Échec communication	La communication avec le robot est perdue lorsque l'on a trois échecs successifs quelconques de tentative de communication (tentative de communication via le thread watchdog, batterie ou déplacer). Lors de la perte de communication, les threads doivent être mis en pause en attendant la reconnexion. Le robot sera redémarré manuellement et l'utilisateur devra faire une reconnexion du robot via le moniteur pour relancer les threads et continuer d'utiliser le robot.
E3 (S) Identification arène	Lors de l'identification de l'arène, il faut envoyer un message au superviseur lui indiquant de stopper l'envoi des images de la webcam, de rechercher l'arène et de renvoyer un dessin de l'arène. L'utilisateur doit ensuite valider visuellement le résultat de la recherche. Si elle n'est pas valide il peut relancer la recherche de l'arène. Sinon il peut valider l'arène trouvée. Ainsi, le superviseur enregistre l'image de l'arène et le cycle normal reprend. L'utilisateur sera dans un même temps notifié via le moniteur. Si il n'est pas satisfait de l'arène, une nouvelle lui est proposée.
E4 (T) État batterie	La période de rafraîchissement de la batterie sera de 250 ms, mais comme ce n'est pas une action essentielle, il se pourrait qu'elle soit retardée. Il faut informer l'utilisateur de l'état de la batterie via le moniteur dès que l'état de la batterie change.
E5 (T) Ordres de déplacements	Une fois la communication établie entre le robot et le superviseur, et que le moniteur connaît l'état de cette communication via un message du superviseur, si l'utilisateur clique sur un ordre de mouvement, il faut que le superviseur envoie toutes les 200 ms des ordres de déplacement au robot.
E6 (T) Watchdog	Un watchdog doit être mis en place. Il faudra s'assurer de recharger le watchdog du robot toutes les secondes en respectant la tolérance de 50 ms et éviter que le robot ne s'arrête tout le temps. En effet, un message de recharge du watchdog ne sera valide que si il est reçu par le robot lorsqu'il attend le message, avec la tolérance de 50 ms. Donc, une synchronisation devra être mise en place lors de l'initialisation du système pour être sur que les messages de recharge du watchdog soient bien coordonnés avec le robot.

Tests

N° Exigence	Tests
E1 (S) Réinitialisation	1. Une fois le système mis en fonctionnement (le robot est opérationnel), le moniteur est tué puis relancé. 2. L'opérateur doit vérifier qu'il est possible de : ○ relancer la connexion avec le moniteur. ○ démarrer et piloter le robot. ○ lancer la détection de l'arène, le calcul de la position et l'envoie d'images de façon périodique.
E2 (S) Echec communication	1. Une fois le système en fonctionnement (robot opérationnel) le robot est mis hors de porté du superviseur. a. La perte de communication doit être détectée et le robot arrêté b. Le moniteur et le superviseur ne doivent pas "crasher" (s'éteindre ou se bloquer) c. L'utilisateur doit quand même pouvoir détecter l'arène 2. Le robot est remis à la porté du moniteur et l'utilisateur reconnecte le robot via l'interface du moniteur a. L'utilisateur doit pouvoir utiliser le robot normalement et détecter l'arène.
E3 (S) Identification arène	1. Une fois le système mis en fonctionnement, on vérifie que l'on reçoit bien les images d'un cycle normal de fonctionnement. 2. On demande ensuite l'identification de l'arène. On vérifie que l'on ne reçoit plus les images de la webcam. 3. On attend la réception du dessin de l'arène au niveau du moniteur. En attendant la réception du dessin, l'utilisateur doit pouvoir continuer à utiliser normalement le robot. 4. Lorsque l'on reçoit un dessin de l'arène, l'utilisateur doit pouvoir : ○ annuler la recherche de l'arène ○ relancer la recherche de l'arène ○ valider le dessin de l'arène et continuer à utiliser le robot 5. Si il choisit de valider le dessin ou d'annuler la recherche, le cycle normal reprend. L'utilisateur doit donc pouvoir : ○ utiliser le robot normalement ○ relancer une recherche d'arène ○ recevoir les images de la webcam 6. Si il décide de relancer la recherche, on attend la nouvelle image de l'arène et la nouvelle décision de l'utilisateur
E4 (T) État batterie	1. Après avoir initialisé le système, les messages de rafraîchissement de la batterie seront affichés toutes les 250 ms sur le moniteur et l'état général de la batterie sera mis à jour en conséquent.
E5 (T) Ordres de déplacements	1. Mettre le système en fonctionnement et vérifier que le moniteur soit notifié que la communication est établie entre le robot et le superviseur (message affiché dans le cadre en bas à droite du moniteur) 2. Envoyer un ordre de déplacement 3. L'utilisateur devra pouvoir constater que le robot réagit relativement vite aux ordres de déplacement. En moins d'une seconde étant donné que la

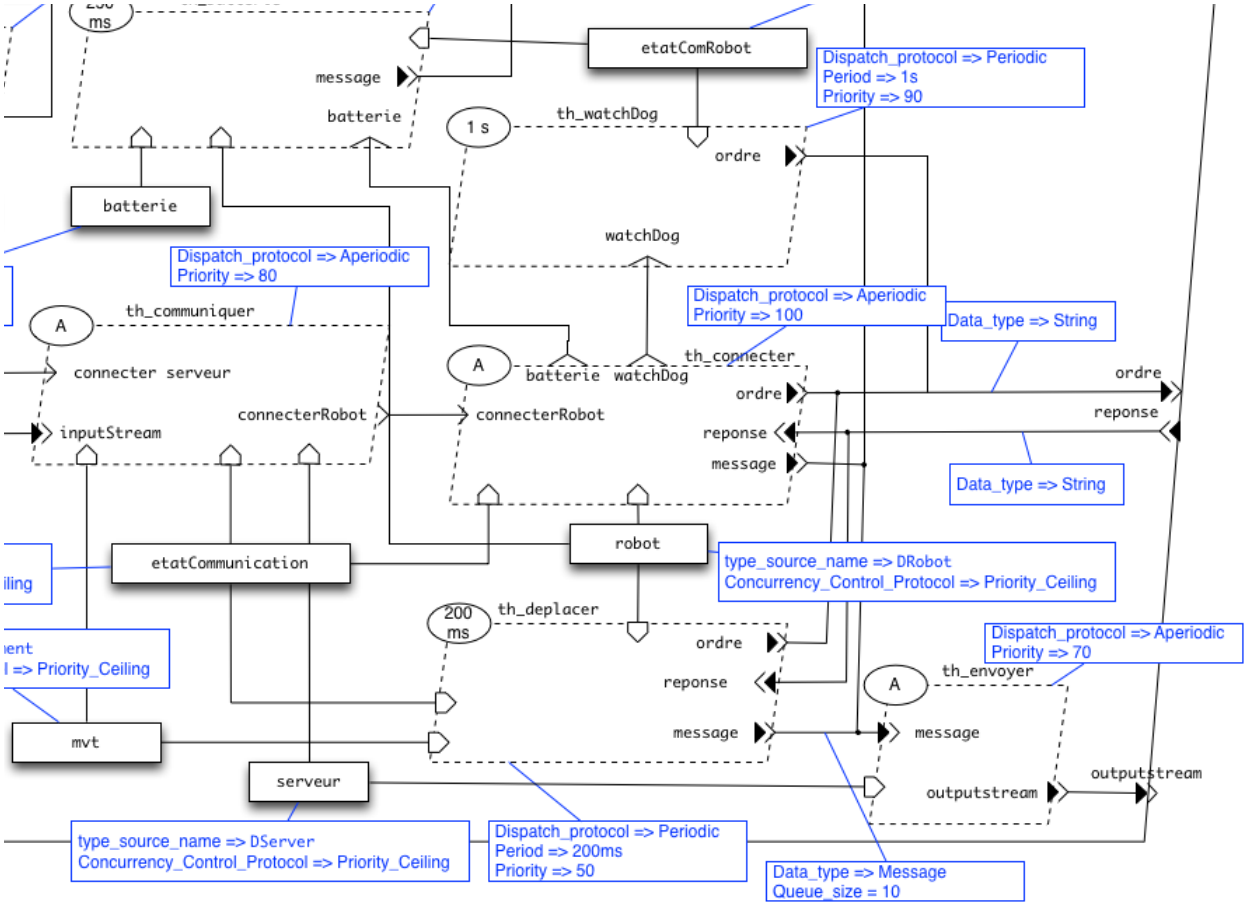
	période est de 200 ms.
E6 (T) Watchdog	1. Une fois le système mis en fonctionnement, on envoi un ordre au robot pour qu'il se mette en mouvement. 2. Ensuit plus aucun ordre n'est envoyé au robot via le moniteur et un message de recharge du watchdog doit être envoyé pour vérifier que le robot ne s'arrête pas. 3. Si il ne s'arrête pas au bout d'une minute on peut considérer que le watchdog est synchronisé avec le superviseur.

Conception

Diagramme fonctionnel

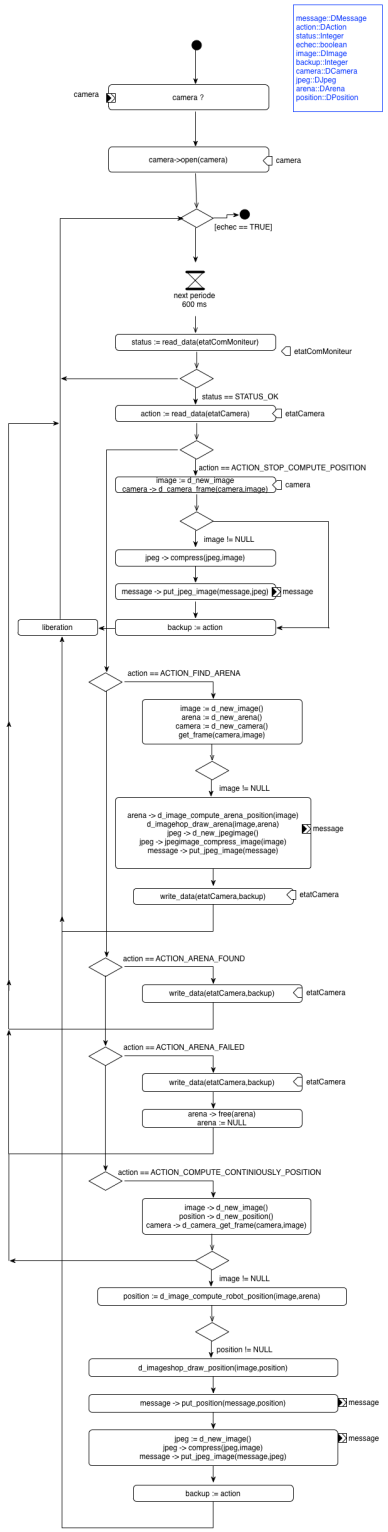
Nom du thread	**Rôle**	**Période**
tServeur	Prend en charge la communication entre le moniteur et le superviseur	Aucune
tEnvoyer	Envoi l'ensemble des messages du superviseur au moniteur	Aucune
tConnecter	Tente de se connecter au robot et met à jour les états des communications	Aucune
tWatchDog	Envois des signaux de rechargement au watchDog du robot	1s
tCamera	Gère la gestion de l'acquisition de la vidéo via la caméra et l'affichage sur le moniteur. Et gère la détection de l'arène quand l'utilisateur le demande. Il est indépendant de la connexion du robot	600 ms
tBatterie	Communique le niveau de la batterie du robot	250 ms
tDeplacer	Contrôle le robot dans la direction souhaitée. Avec deux vitesses possibles	200 ms

Aucune période : le thread ne travaillera qu'à la demande de l'utilisateur. Tant qu'il ne lui est pas demandé d'effectuer une action, il devra être en attente sans consommer de mémoire.

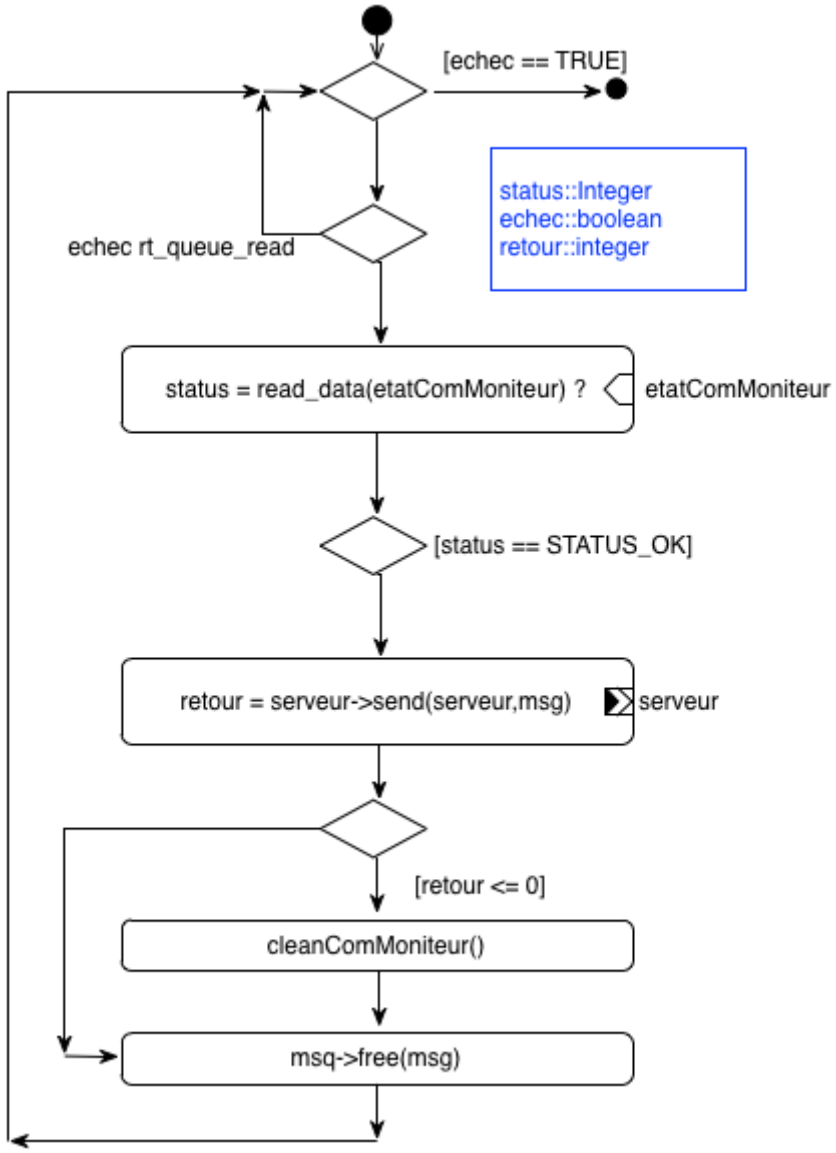

200 ms
etatComRobot
Dispatch_protocol => Periodic
Period => 1s
Priority => 90
message
batterie
1 s
th_watchDog
ordre
batterie
watchDog
Dispatch_protocol => Aperiodic
Priority => 80
Dispatch_protocol => Aperiodic
Priority => 100
Data_type => String
A
th_communiquer
connecter serveur
A
batterie
watchDog
th_connecter
ordre
ordre
connecterRobot
connecterRobot
reponse
reponse
inputStream
message
Data_type => String
etatCommunication
robot
type_source_name => DRobot
Concurrency_Control_Protocol => Priority_Ceiling
200 ms
th_deplacer
ordre
Dispatch_protocol => Aperiodic
Priority => 70
reponse
A
th_envoyer
message
message
mvt
serveur
outputstream
outputstream
type_source_name => DServer
Concurrency_Control_Protocol => Priority_Ceiling
Dispatch_protocol => Periodic
Period => 200ms
Priority => 50
Data_type => Message
Queue_size = 10

Diagrammes d'activité

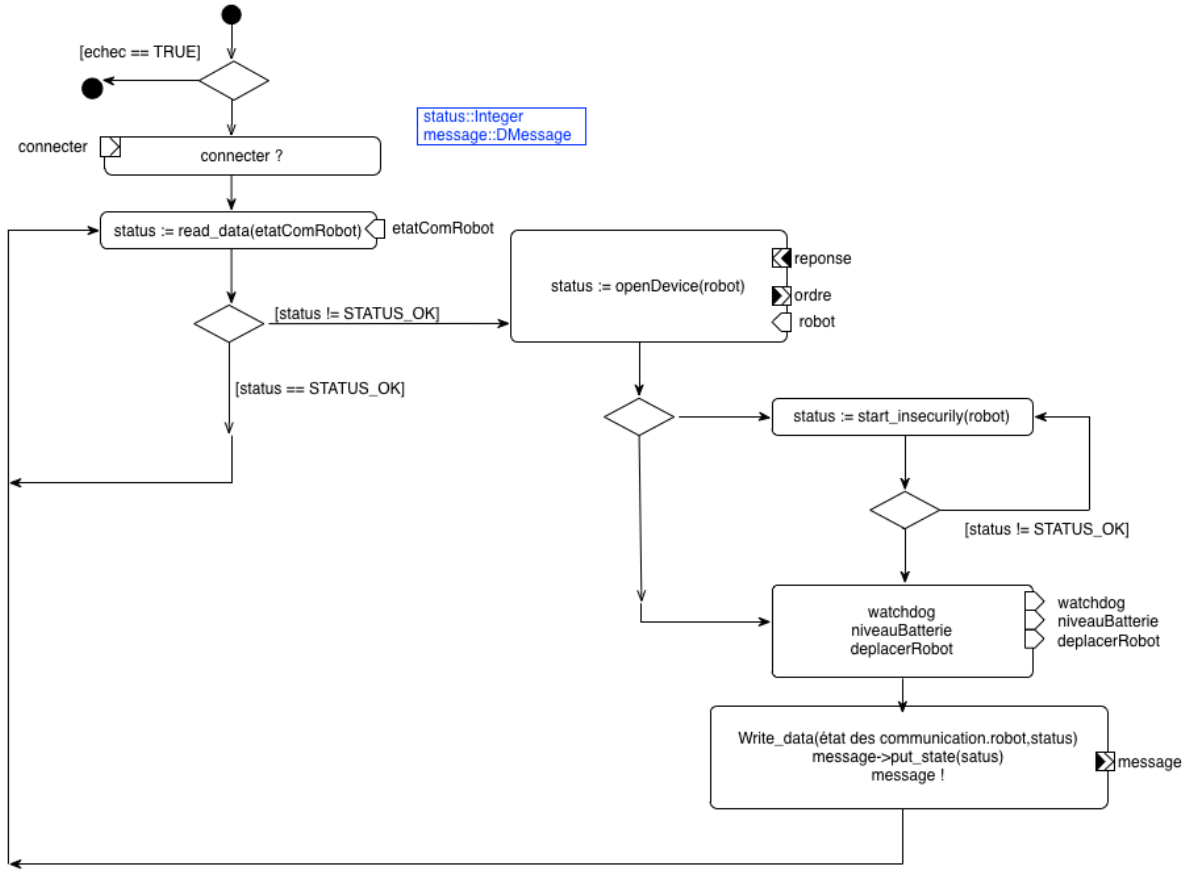
Thread camera



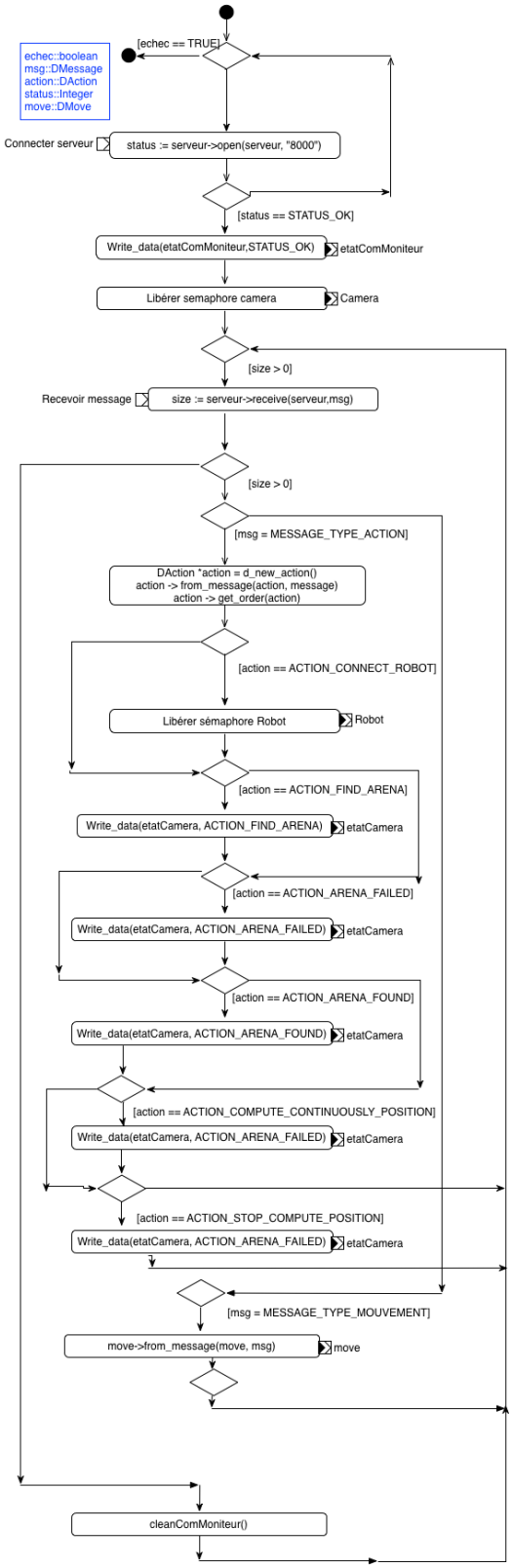
Thread envoyer



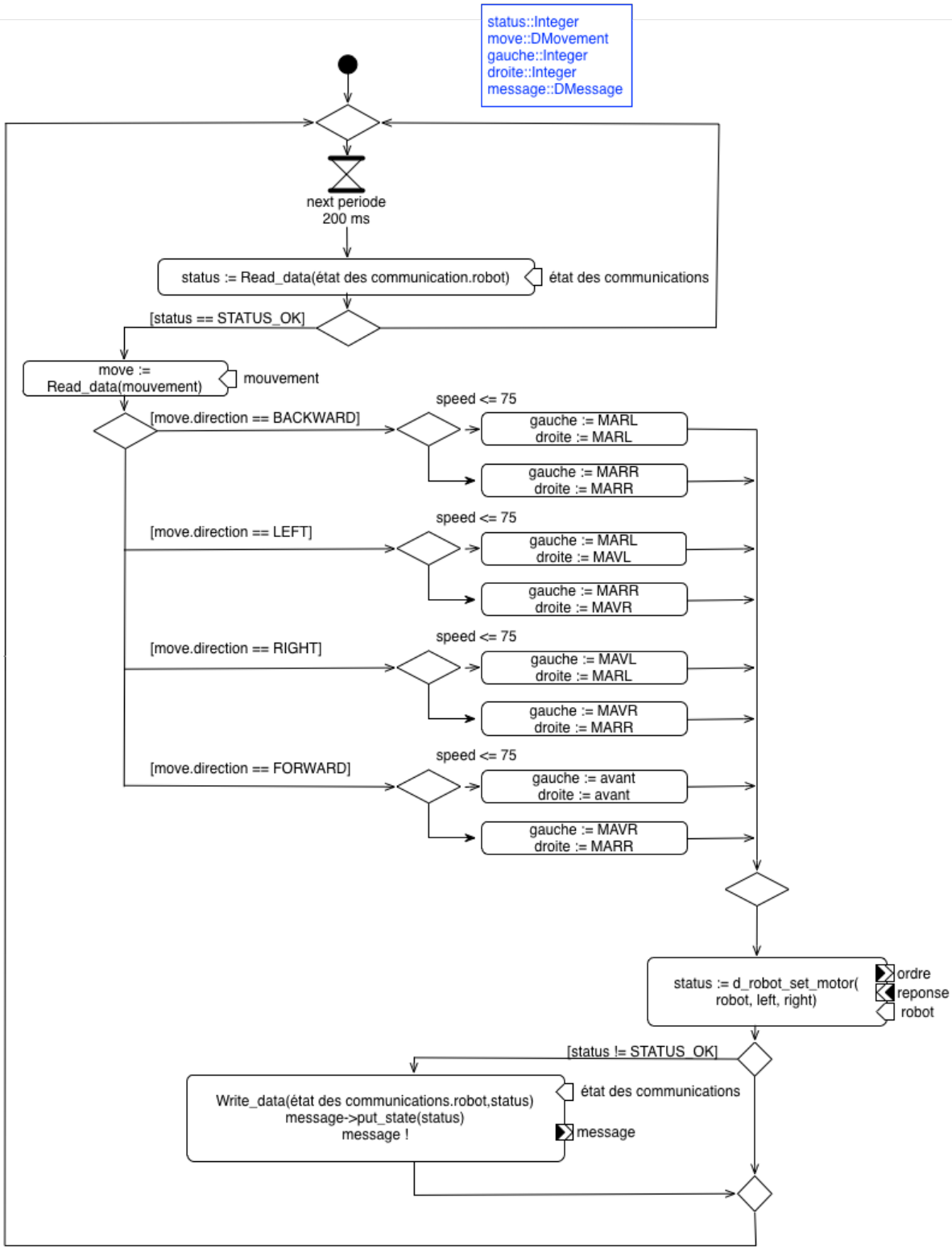
Thread connecter



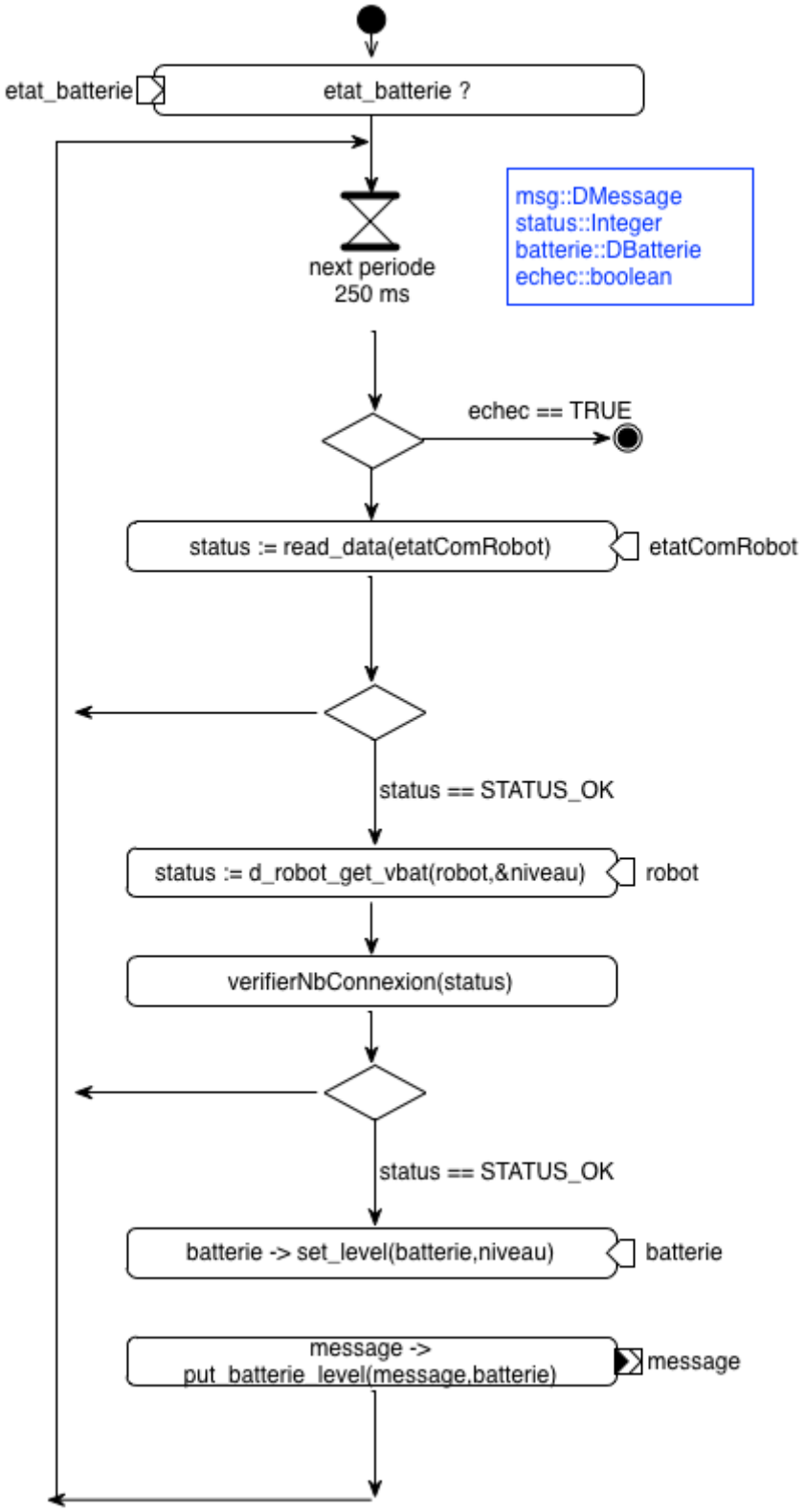
Thread communiquer



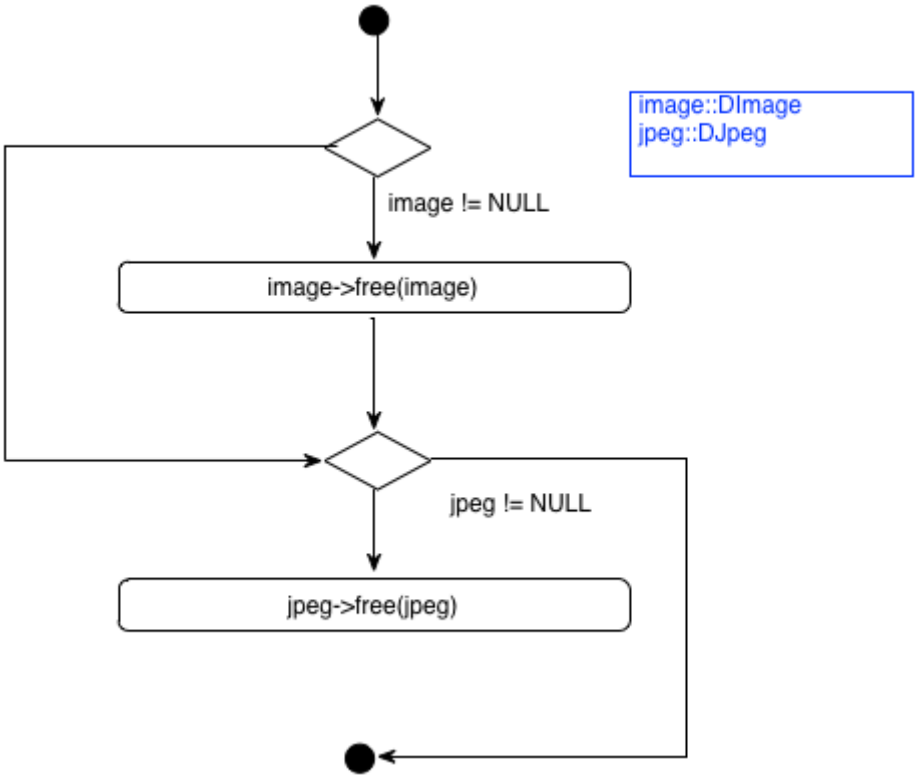
Thread deplacer



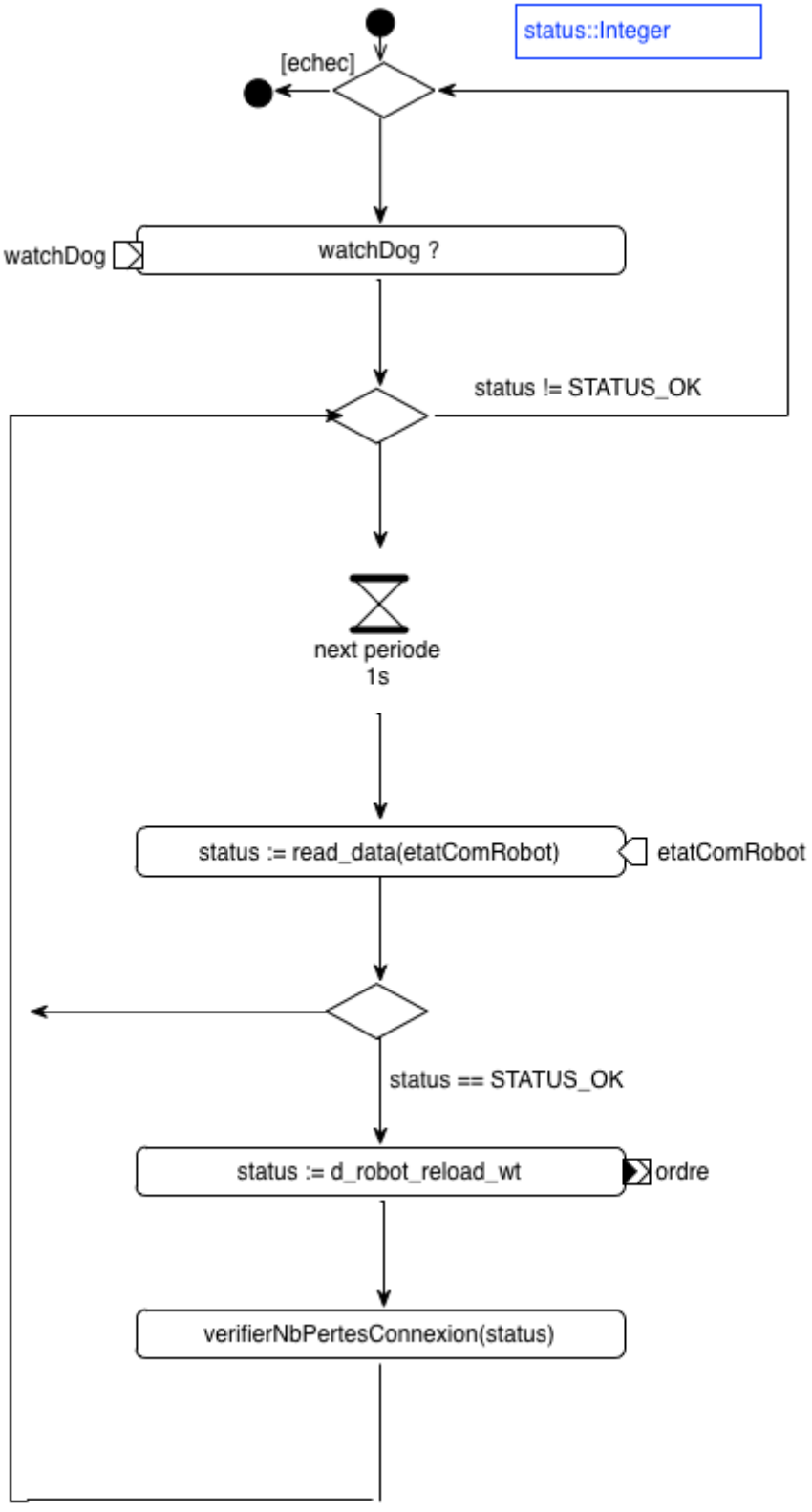
Thread etatBatterie



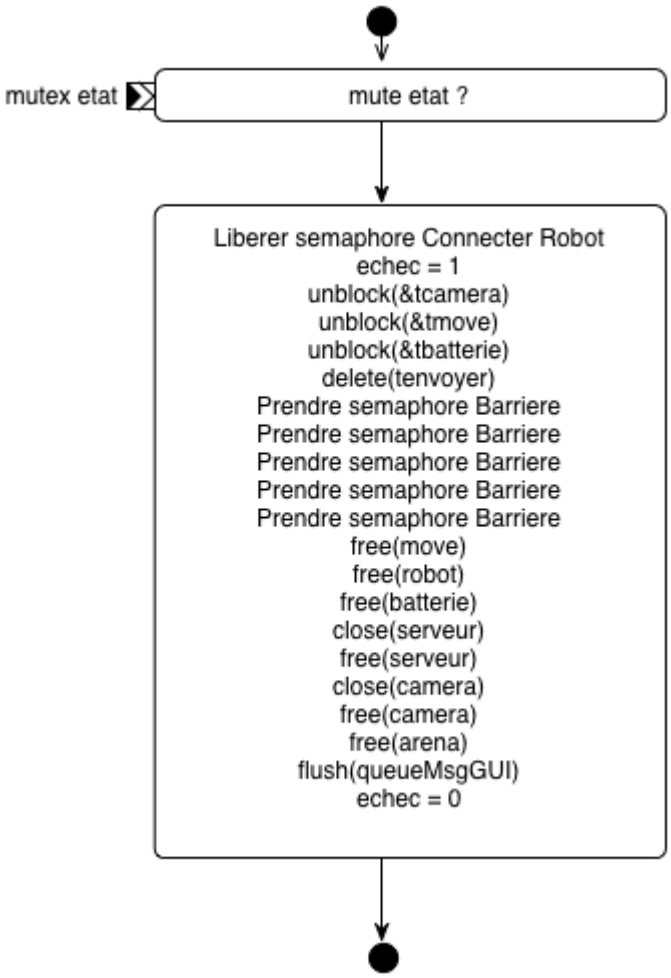
Bloc UML *liberation*



Thread watchDog



Fonction CleanComMoniteur



Affectation des priorités

Nom du thread	Priorité	Justifications
tWatchdog	99	Toutes les recharges du watchDog du robot doivent être traitées en priorité forte pour ne pas que le robot s'arrête à cause d'un retard. De plus, la tolérance est de 50 ms, donc la recharge doit à tous prix être envoyée au plus vite.
tCamera	90	Thread périodique (600ms). Il permet à l'utilisateur de contrôler la position du robot, notion de système critique pour que le robot n'entre pas en collision avec un obstacle.
tConnecter	80	Une fois la connexion entre le superviseur et le moniteur étbie, celle entre le robot et le superviseur doit se faire dès qu'elle est demandée car sans connexion il sera impossible de commander le robot.
tServeur	70	Ce thread assume la fonction de Communiquer, il est donc responsable du comportement du robot (via tDeplacer). Donc

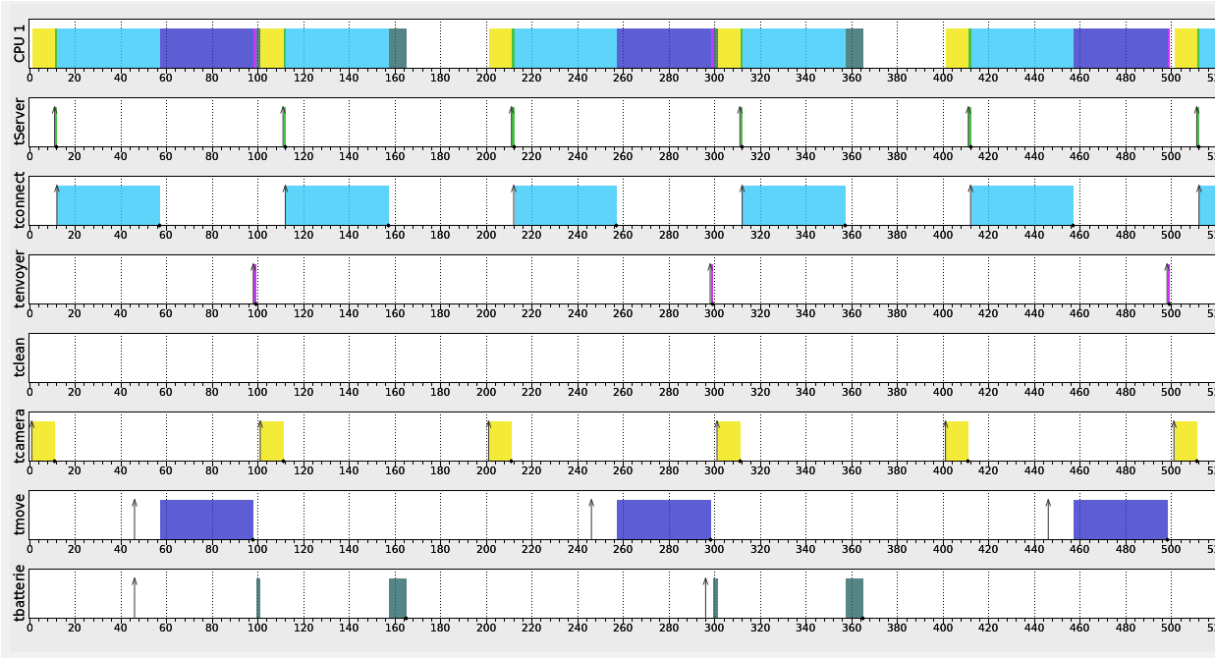
		sa priorité doit être des plus importantes.
tEnvoyer	60	tEnvoyer est générique et succède notamment à tServeur pour la communication donc sa priorité doit être importante mais directement après celle de tServeur.
tDeplacer	40	Thread périodique. Son exécution est modifiée uniquement sur ordre d'un tiers via le moniteur. Il peut donc prendre un peu de retard car dans le pire des cas le robot continuera à exécuter l'action précédente pendant un peu plus longtemps que prévu.
tBatterie	5	Il n'est pas prioritaire de vérifier le niveau de la batterie car le passage d'un niveau à un autre ne se fait pas en quelques secondes : si la période de rafraîchissement de l'état n'est pas totalement respectée, l'utilisateur sera simplement averti un peu plus tard que le niveau est faible, ce qui n'est pas critique en soit car il aura largement le temps d'agir en conséquent avec que le robot atteigne le prochain état de batterie directement inférieur.

Couverture des exigences niveau conception

N° Exigence	Résultat du test
E1 (S) Réinitialisation	La prise en compte de la perte de communication entre le moniteur et le superviseur est réalisée par le thread tServeur. Nous voyons bien dans le schéma de comportement de ce thread que la perte de communication est détectée lorsque *size* est inférieure à 0. Dans ce cas, le robot est arrêté, la communication fermée ainsi que la caméra. Les threads tDeplacer, tCamera, tEnvoyer, tBatterie, tConnecter et tServeur sont tués puis relancés afin de les mettre dans un état similaire au démarrage de l'application. Les variables mvt, etatCommunication, etatCamera et serveur sont remises à leur valeur initiale. Le serveur est ensuite redémarré et mis en attente d'une connexion avec le moniteur.
E2 (S) Échec communication	Lorsque le robot est hors de porté, on voit bien que les threads tmove, twatchdog et tbatterie sont mis en pauses dans le fichier des logs : ils arrêtent de tracer. Alors que les threads camera et envoyer continuent à tourner (ils continuent de tracer). Toujours dans les logs, on observe que le thread connecter est tué avant d'être relancé.
E3 (S) Identification arène	L'identification de l'arène est mise en place dans le même thread que celui de l'acquisition de la vidéo. On voit bien que la demande de l'action à effectuer conditionne bien la détection de l'arène. Concrètement, on met en place l'écriture des logs pour le thread tCamera qui affichera si il est en train de recevoir les images de la camera pour informer l'utilisateur de la position du robot ou de si il est en train de détecter l'arène. Le reste de la séquence peut être testé grâce au moniteur qui demande à l'utilisateur de valider ou non le dessin de l'arène. Comme seuls trois choix sont possibles, nous pouvons les tester "à la main" pour confirmer le bon déroulement de la séquence.
E4 (T) État batterie	Le **chronogramme** suivant montre bien que toutes les 250ms un relevé et donc une mise à jour du niveau de la batterie est réalisée. Nous remarquons sur ce chronogramme que le thread batterie ne peut s'éxécuter en une seule fois sans être interrompu car il est de priorité la plus faible, ce qui ne pose pas de problème pour la contrainte souhaitée.
E5 (T) Ordres de déplacements	Une fois la communication établie entre le robot et le superviseur, et que l'état des communications est mis à jour, suite à un clic de l'utilisateur le superviseur doit envoyer toutes les 200 ms des ordres de déplacement au robot. Le contrôle du robot se fait grâce au thread tDeplacer qui possède une période de 200 ms. Le thread envoie le dernier ordre reçu au robot de manière continue : c'est vérifié lorsque l'on envoie un message déplacé, il ne cesse d'être envoyé au robot qui continue de se déplacer dans le même sens. Le **chronogramme** ci-dessous montre bien que la **période de l'envoie des messages est respectée**.

E6 (T) Watchdog	Le thread tWatchdog s'occupe du rechargement du watchdog du robot. Le robot s'arrête si on stoppe l'envoi des messages pendant 7 secondes au robot. Aussi, le robot ne s'arrête pas prématurément si l'on s'en sert de manière normale : le watchdog est bien rechargé. Le message de recharge respecte la tolérance de 50 ms. Il est bien synchronisé avec le robot.

Chronogramme illustrant les exigences sur tBatterie et tmove



Couverture des exigences niveau test

N° Exigence	Résultat du test
E1 (S) Réinitialisation	Après avoir relancé le moniteur, l'opérateur à : • relancer la connexion avec le moniteur, • démarrer le robot, • calibrer la caméra, • lancer le calcul de la position.
E2 (S) Échec communication	Après avoir perdu la communication avec le robot, l'utilisateur à : • reconnecter le superviseur avec le robot • utiliser le robot normalement • recevoir les images de la webcam
E3 (S) Identification	Après avoir lancé l'identification de l'arène, l'utilisateur à : • validé le dessin de l'arène

arène	• relancé la détection de l'arène • annulé la détection de l'arène • utilisé le robot normalement pendant la détection et après
E4 (T) État batterie	Les dates d'arrivée des messages d'état de la batterie sont : « 0:08:208 », « 0:08:427 », « 0:08:696 », « 0:08:927 » soit des écarts autour de 250ms.
E5 (T) Ordres de déplacements	Les dates d'arrivée des messages de déplacement sont : « 1:03:358 », « 1:03:555 », « 1:03:756 », « 1:03:958 » soit des écarts autour de 200ms ce qui correspond à notre exigence.
E6 (T) Watchdog	Non utilisé.

Règles de codage

Élément AADL	Transformation en C
Thread	
Création	Une tâche est crée et démarrée dans le *main*. Cela se fait à l'aide des services *rt_task_create* et *rt_task_start*. Tous les threads, sauf le thread tCommuniquer, possédent un sémaphore pour n'exécuter sa tâche ou démarrer sa périodicité que lorsque le thread tCommuniquer partagera la sémaphore.
Affectation de la priorité	Utilisation du service *rt_task_create* permettant d'affecter une priorité au thread.
Activation périodique	Utilisation du service *rt_task_set_periodic* pour rendre la tâche périodique.
Activation événementielle	Le thread est créé dès le lancement du système. Un thread événementiel attend d'acquérir un sémaphore (*rt_sem_p*) avant d'effectuer la tâche qui lui est attribuée. Une fois la tâche effectuée, il se remet en état d'attente du sémaphore avant d'effectuer à nouveau sa tâche.
Port d'événement	
Création	Création de l'événement avec le service *rt_sem_create* qui permet de créer un sémaphore lié à un thread particulier.
Envoi d'un événement	L'envoi de cet événement se fait avec *rt_sem_v* qui libère le sémaphore et permet le déclenchement du thread.
Réception d'un événement	Lorsque le sémaphore est libéré, le thread correspondant peut prendre le sémaphore via le service *rt_sem_p* et donc s'éxécuter.
Données partagées	
création	La donnée partagée sera crée de manière à ce qu'elle soit commune à l'ensemble des threads. Elles sont **toutes** déclarées dans le fichier global.c pour que tous les threads puissent y accéder.

Protection	Utilisation du service *rt_mutex_acquire* pour protéger les données partagées et éviter que deux threads ne l'utilisent en même temps. Chaque donnée est protégée par son propre mutex. Utilisation du service *rt_mutex_release* pour libérer la donnée partagée et autoriser un autre thread de l'utiliser. Lors de la demande d'accès à une ressource partagée, le thread en question est bloqué à l'infini jusqu'à qu'il ait accès à la ressource. On utilise pour cela un flag lors de l'utilisation permettant l'acquisition du mutex. Avec l'utilisation d'un autre flag, la demande d'accès à une ressource pourrait être limitée à un certain temps avec que le thread continue son exécution sans avoir eu accès à la ressource.
Accès en lecture	Pour accéder en lecture à la variable partagée, il faut s'en servir comme ci-dessous : *variable_non_partagee = variable_partagee*
Accès en écriture	Pour accéder en écriture à la variable partagée, il faut s'en servir comme ci-dessous : *variable_partagee = variable_non_partagee*
Port d'événement-données	
Création	Utilisation du service *rt_mutex_create* pour créer nos structures permettant de gérer l'accès aux ressources partagées.
Envoi d'une donnée	Le service *rt_mutex_release* permet de relâcher le mutex protégeant une ressource ce qui permet à n'importe quel autre thread demandant l'accès à la ressource de l'avoir. Et ainsi d'avoir accès à la nouvelle valeur de la ressource partagée si celle-ci a été modifiée avec l'utilisation du service.
Attente d'une donnée	Le service *rt_mutex_acquire* demande l'accès à une ressource partagée et attente jusqu'à avoir acquis le mutex protégeant la ressource en question.