Projet De Stijl 2.0 : Plateforme pour robots mobiles

Programmation et conception de systémes temps réel – 4éme année AE/IR

Institut National des Sciences Appliquées de Toulouse

Cahier des charges fonctionnel

Version 3.0.β (7 février 2018)

Référent pédagogique : P.-E. Hladik (pehladik@insa-toulouse.fr)

Référents plateforme : L. Senaneuch (senaneuc@insa-toulouse.fr), S. Di Mercurio (dimercur@insa-toulouse.fr)

---

1 Travail demandé

1. Le travail sera réalisé dans le cadre d'un groupe de trois ou quatre étudiants sur deux séances de TD et cinq séances de TP,
2. Le projet fera l'objet d'un rapport évalué ainsi qu'une démonstration. Un squelette pour le rapport est disponible sur la page moodle de l'UE,
3. L'évaluation a pour but de juger de vos compétences pour rédiger un compte rendu, réaliser la conception d'une application temps réel et programmer sur un système temps réel ainsi que de vos connaissances sur les systèmes temps réel,
4. Le code réalisé devra être rendu sous forme d'archive ne comprenant que les codes sources modifiés,
5. Le rendu du rapport et du code se fera sous la forme d'une archive à envoyer à votre encadrant de TP,
6. La démonstration aura lieu la semaine qui suit la dernière séance de TP.

2 Vue générale de la plate-forme

Le projet *De Stijl* est une plate-forme de contrôle d'un robot mobile servant pour les enseignements du département Génie Électronique et Informatique. Le travail à réaliser dans le cadre du cours « Conception et programmation de systèmes temps réel » concerne le développement logiciel de la plate-forme de contrôle.

Les éléments constituant la plate-forme sont fournis et ont été testés de manière unitaire, mais nous ne garantissons pas un fonctionnement parfait. Toutes suggestions, corrections et modifications seront appréciées pour faire évoluer ce TP.

3 Équipement constituant la plate-forme

La plate-forme est constituée de quatre éléments (voir figure 1) :

1. **L'arène :** Boîte rectangulaire de couleur grise, l'arène est le terrain dans lequel le robot évolue.
2. **Le robot mobile :** Robot deux roues embarquant un microcontrôleur et un ensemble de composants matériels nécessaires à son déplacement. L'intelligence embarquée est volontairement limitée au contrôle de son déplacement (contrôle des moteurs) et à des fonctionnalités pour connaître son état.
3. **Le superviseur :** Entité principale de la plate-forme, elle est dédiée au contrôle et à la supervision du robot. Elle est couplée à une Webcam fixée au-dessus du terrain afin d'en faire une acquisition visuelle et de localiser le robot.
4. **Le moniteur :** Entité entièrement logicielle et distante, son rôle est d'offrir une interface de contrôle pour l'utilisateur.

La communication entre les équipements est hétérogène et assurée par différents supports :

1. La communication entre le robot et le superviseur est assurée par une liaison sans fils point à point à l'aide de modules XBee. Cette communication est vue comme une liaison série.
2. La communication entre le superviseur et le moniteur est réalisée par un serveur http déployé sur le superviseur. Le moniteur est une page affichées dans un navigateur. La liaison physique est assurée par WiFi.

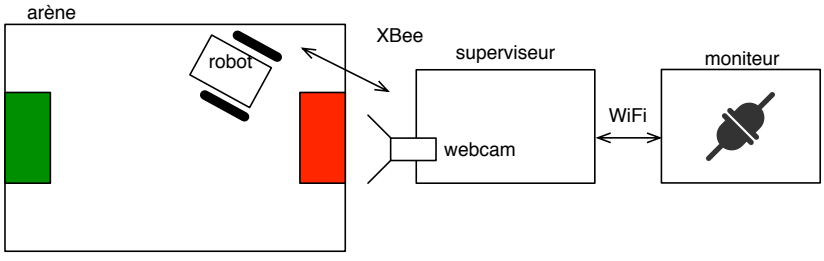

Fig. 1: Vue générale de la plate-forme

3.1 Arène

L'arène (voir figure 2) a été réalisée dans les ateliers du département par J. Perez. C'est une boîte de 60×80 cm en Medium et recouverte d'une peinture uniforme grise. Les bords sont délimités par un trait blanc. Deux zones distinctes sont peintes sur les bords opposés. L'une étant de couleur verte (ou orange) et l'autre de couleur rouge.

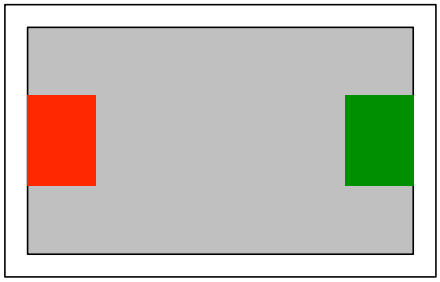

Fig. 2: Vue schématique de l'arène

3.2 Robot

La version 2.1 du robot a été conçue et réalisée par S. Di Mercurio. Schématiquement (figure 3), le robot est constitué de deux moteurs, d'un microcontrôleur STM32F103RB et d'une puce Xbee.

Le robot est mobile sur deux roues avec un point d'appui sur patin. Le contrôle de la trajectoire est assuré par le contrôle en vitesse des moteurs.

Le robot communique avec le superviseur à l'aide d'une puce Xbee assurant une liaison série point à point.

Le robot a une flèche blanche sur le dos afin de le localiser dans l'arène.

Le code déployé sur le robot a été produit par S. Di Mercurio et L. Senaneuch. Il offre tous les services nécessaires pour contrôler le robot.

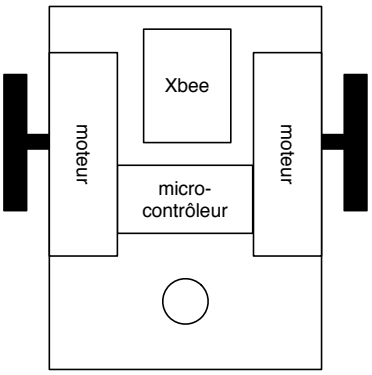

Fig. 3: Vue schématique du robot

3.3 Superviseur

Le superviseur est une Raspberry Pi 3B sur lequel est installé un Ubuntu patché PREEMPT-RT avec l'extension mercury de Xenomai 3.0 afin de lui apporter des fonctionnalités temps réel. Une Raspberry Pi est un ordinateur à processeur ARM de taille réduite. La version 3B possède un processeur Broadcom BCM2837 64 bit à quatre cœurs ARM Cortex-A53 à 1,2 GHz, d'une puce Wifi 802.11n et Bluetooth 4.1 intégrée.

Le rôle du superviseur est d'orchestrer le fonctionnement de la plate-forme en assurant le respect des contraintes temporelles du système.

La webcam est intégrée au bloc du superviseur et est une caméra Raspberry.

> **Remarque :** L'application que vous allez réaliser sera celle du superviseur. Vous ne toucherez pas au code embarqué dans le robot.

3.4 Moniteur

Le moniteur a été développé par L. Senaneuch. Il utilise un navigateur web pour afficher l'interface. Pour répondre aux requêtes, un serveur nodejs tourne sur le superviseur. Les requêtes de l'utilisateur sont transmises via le serveur qui le relaie ensuite au superviseur (voir schéma 4). La communication entre le serveur et le superviseur est réalisée par un socket logiciel.

L'interface permet à l'utilisateur de saisir des ordres à réaliser par le robot et de connaître l'état global du système.

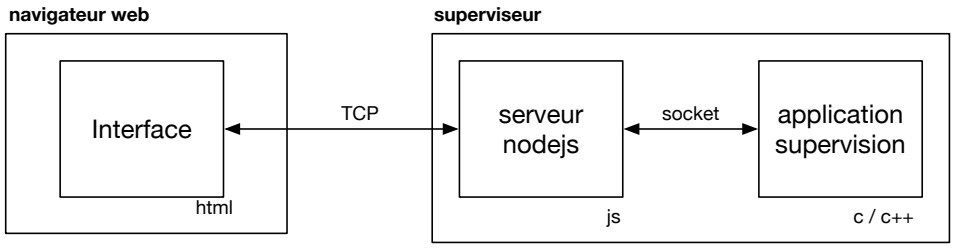

Fig. 4: Interface graphique du moniteur

4 Bibliothèques logicielles

Vous allez concevoir l'architecture logicielle du superviseur. Toutes les fonctions de traitement (communication, vidéo, etc.) **ont déjà été implémentées** par L. Senaneuch. Vous n'aurez pas à modifier ce code, simplement à faire appel aux fonctions. Il n'en reste pas moins un gros travail d'architecte logiciel à faire.

Le code est disponible sur un dépôt git https://github.com/daihitsuji/C-TP-RT. L'annexe B présente sous forme de diagramme de classes les fonctions disponibles dans les bibliothèques.

Les fonctions de traitement sont réparties en trois bibliothèques :
— monitor : services de communication entre le superviseur et le moniteur,
— robot : services de communication entre le superviseur et le robot,

— image : services réalisant tous les traitements vidéos.

> Pour des raisons pédagogiques, toutes les fonctions ont une interface écrite en C. Cependant, la grande majorité des fonctions ont été codées en C++. Cela est complètement transparent pour vous. Vous pouvez considérer que le langage de programmation est du C (aucun support ne sera fourni par les encadrants pour le C++).

5 Expression fonctionnelle du besoin

5.1 Présentation générale

L'objectif est de concevoir et de développer l'architecture logicielle du superviseur afin d'assurer le fonctionnement de la plate-forme. Les fonctions attendues sont :

1. Assurer la communication entre le moniteur et le superviseur.
2. Assurer la communication entre le superviseur et le robot.
3. Superviser l'état du robot.
4. Contrôler le déplacement du robot.
5. Contrôler et diffuser les images de la webcam.
6. Réaliser des missions.

> **Remarque :** Les plateformes matérielle et logicielle ont été refaites cette année. Les bibliothèques fournies ont été réécrites pour faciliter leur prise en main. Bien qu'elles aient été testées, elles comportent certainement des erreurs. Soyez indulgents, nous faisons de notre mieux pour fournir une plate-forme opérationnelle !
> **Si vous remarquez des erreurs, des problèmes ou des optimisations à faire, n'hésitez pas à le communiquer** (avec un email c'est plus simple pour en garder la trace).

5.2 Diagramme de contexte

La figure 5 présente le superviseur dans son contexte et ses interactions avec les autres composants de la plate-forme.

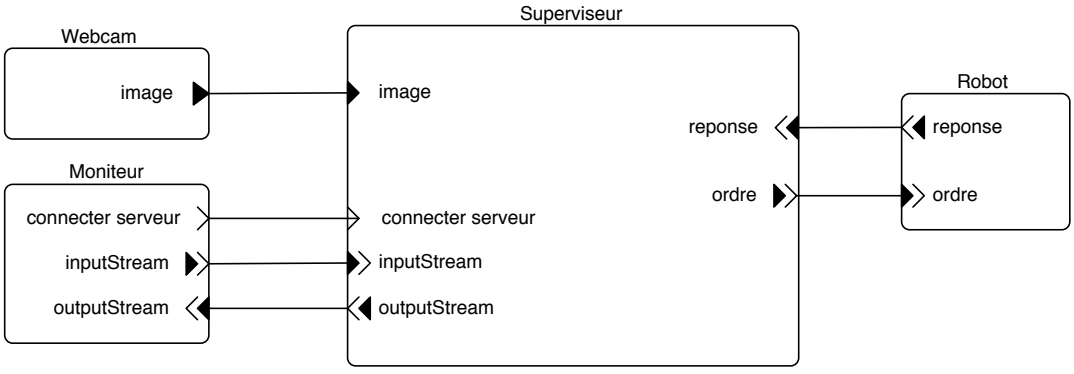

Fig. 5: Diagramme de contexte

La webcam produit des données (`image`) sous la forme d'un tableau d'octets. Le robot reçoit des ordres (`ordre`) sous la forme d'une chaîne de caractéres et retourne une réponse aussi sous la forme d'une chaîne (`reponse`). Le moniteur envoie un événement (`connecter serveur`) pour demander la connexion avec le serveur puis établit une communication bi-directionnelle sous la forme d'une flux d'octets (`inputStream` et `outputStream`).

5.3 Description des fonctionnalités

Vous trouverez dans cette partie la description des différentes fonctionnalités attendues. Les annexes apportent des compléments techniques.

L'expression des fonctionnalités est repérée par des encadrés avec un fond gris. C'est ce que vous devez réaliser.

5.3.1 Fonctionnement du moniteur

Le moniteur sert pour le contrôle du robot et la visualisation d'une scène. La communication entre le moniteur et le superviseur est réalisée par un serveur nodejs. La communication est bi-directionnelle : le moniteur peut envoyer des requêtes au serveur et le serveur peut envoyer des données pour mettre à jour certains éléments de l'interface.

Le serveur nodejs sert de passerelle entre le moniteur et l'application de supervision. La communication entre ce deux éléments est réalisée à travers un socket. Les messages sont prédéfinis et sont présentés dans l'annexe A.2.

5.3.2 Communication entre le superviseur et le moniteur

La communication entre le serveur et le superviseur est réalisée à l'aide d'un socket. Le superviseur joue le rôle de client. Toutes les fonctions pour gérer la communication entre le moniteur et le superviseur dont fournies dans `monitor`.

Les figures 6 et 7 illustrent le mode de fonctionnement nominal attendu pour la communication entre le superviseur et le moniteur.

Lancement de nodejs et mise en attente de la connexion. Le lancement du serveur nodejs est réalisé à l'aide de la fonction `run_nodejs`. L'attente de la connexion entre le superviseur et le serveur nodejs se fait par l'appel à la méthode `open_server`.

Fonctionnalité 1 : Le lancement du serveur nodejs doit être réalisé au démarrage du superviseur. Le superviseur doit ensuite se mettre en attente de la connexion via le socket. En cas d'échec du démarrage du serveur nodejs, un message textuel doit être affiché sur la console de lancement de l'application. Ce message doit signaler le problème et le superviseur doit s'arrêter.

Etablissement du socket. La connexion entre nodejs et le superviseur est réalisée à la demande de l'utilisateur via le moniteur. Lorsque la demande est faite, nodejs demande la création du socket avec le superviseur, il faut donc que celui-ci soit en attente d'une demande de connexion, c'est-à-dire que le superviseur doit avoir exécuter la fonction `open_server`.

Fonctionnalité 2 : La connexion entre nodejs et le superviseur (via le socket) doit être établie suite à la demande de connexion du moniteur.

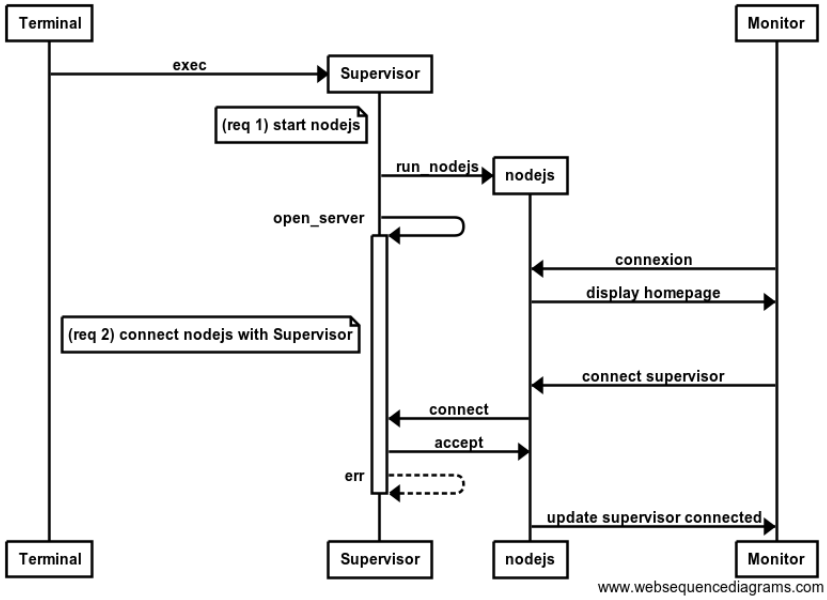

Fig. 6: Diagramme de séquence des fonctionnalités 1 à 2

Réception des messages. La réception des messages du serveur par le superviseur est réalisée par l'appel de la fonction `receive_message_from_monitor`. Cette fonction est bloquante. Les messages du moniteur vers le superviseur sont pré-formatés et définis dans l'annexe A.2.1.

Fonctionnalité 3 : Tous les messages envoyés depuis le moniteur doivent être réceptionnés par le superviseur.

Envoi des messages. L'envoi des messages du superviseur vers le moniteur est réalisé à l'aide de la fonction `send_message_to_monitor`. Les messages du superviseur vers le moniteur sont pré-formatés et définis dans l'annexe A.2.2.

Fonctionnalité 4 : L'application superviseur doit être capable d'envoyer les messages au moniteur (via le serveur) avec un délai d'au plus 10 ms.

Détection de la perte de communication. La perte de communication entre nodejs et le superviseur est détectée lors de la lecture ou de l'écriture sur le socket. Les fonctions `send_message_to_monitor` et `receive_message_from_monitor` retournent une valeur négative ou égale à 0 si la communication est perdue.

Fonctionnalité 5 : Le superviseur doit détecter la perte de communication avec nodejs. En cas de perte de la communication un message doit être affiché sur la console de lancement du superviseur.

Reprise de la communication. Pour fermer la communication entre le moniteur et le serveur nodejs il faut faire appel à la fonction `close_server`. Pour tuer le serveur nodejs, il faut faire

appel à la fonction `kill_nodejs`.

Fonctionnalité 6 : En cas de perte de communication entre le superviseur et nodejs, il faut stopper le robot, la communication avec le robot, le serveur nodejs et la camera afin de revenir dans le même état qu'au démarrage du superviseur.

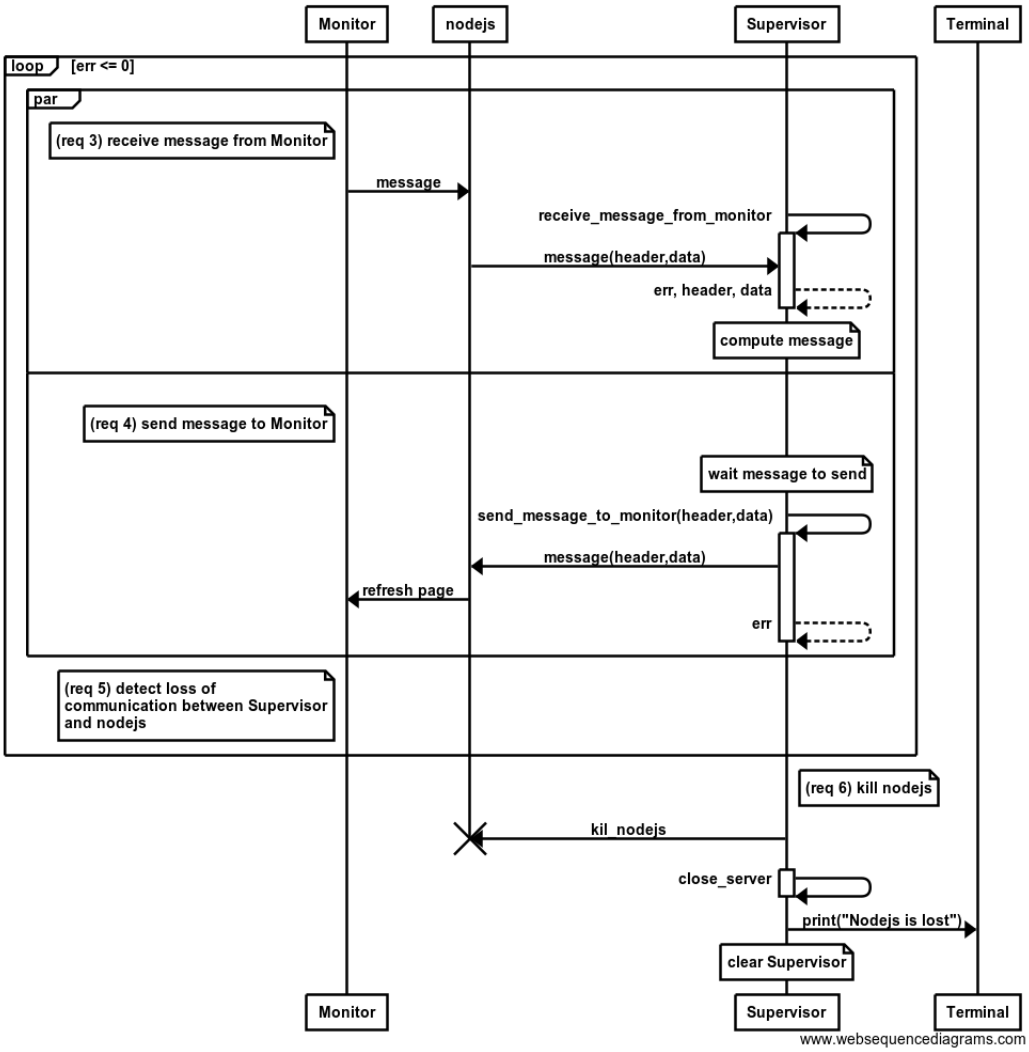

Fig. 7: Diagramme de séquence des fonctionnalités 3 à 6

5.3.3 Communication entre le superviseur et le robot

La communication avec le robot s'effectue par le biais d'émetteur-récepteur Xbee. Pour communiquer, il est nécessaire de commencer par ouvrir la communication entre le superviseur et le boîtier Xbee. Toutes les fonctions de gestion de la communication entre le robot et le superviseur sont fournies dans la bibliothèque `robot`. La figure 8 illustre les fonctionnalités 7 à 9.

Mettre en place la communication avec le robot. L'ouverture de la communication (c'est-à-dire la réservation d'un port série) avec le robot se fait à l'aide de la fonction `open_communication_robot`.

Fonctionnalité 7 : Lorsque l'utilisateur demande, via le moniteur, à connecter le robot, la communication entre le superviseur et le robot doit être mise en place. Si la communication est active, il faut envoyer un message d'acquittement au moniteur. En cas d'échec, un message le signalant est renvoyé au moniteur.

Surveillance de la communication avec le robot. La communication entre le robot et le superviseur peut être perdue. Afin de surveiller cela, il faut mettre en place un mécanisme permettant d'inférer cette perte. L'envoi des messages au robot se fait à l'aide de la fonction `send_command_to_robot` qui retourne un message d'erreur en cas d'échec de communication. Cependant, l'envoi d'un message par Xbee peut retourner un échec même si le médium de communication est encore opérationnel.

De ce fait, le simple retour d'un échec ne suffit pas à déterminer si la communication est définitivement perdue ou bien si c'est une erreur fugace. Afin de conclure que la communication est perdue, il faut donc mettre en place un mécanisme de compteur. Pour cela, il faut incrémenter un compteur à chaque échec sur l'envoi d'un message et de le remettre à zéro à chaque succès. Si le compteur dépasse 3, la communication est alors déclarée perdue.

Fonctionnalité 8 : La communication entre le robot et le superviseur doit être surveillée par un mécanisme de compteur afin de détecter une perte du médium de communication.

Perte de la communication avec le robot. Le medium de communication entre le robot et le superviseur est stoppé par l'appel à la fonction `close_communication_robot`.

Fonctionnalité 9 : Lorsque la communication entre le robot et le superviseur est perdue, un message spécifique doit être envoyé au moniteur. Le système doit fermer la communication entre le robot et le superviseur et se remettre dans un état initial permettant de relancer la communication.

5.3.4 Démarrage du robot

Tous les messages entre le superviseur et le robot sont décrits dans l'annexe A.1. L'envoi d'un message se fait par l'utilisation de la fonction `send_command_to_robot`. La figure 9 représente la séquence pour les fonctionnalités 10 et 11.

Remarque : Aucun mécanisme n'est mis en œuvre dans l'implémentation de `send_command_to_robot` pour se prémunir des appels concurrents.

Démarrage sans watchdog du robot. Le robot a deux modes de démarrage. Un mode simple, dit sans watchdog et un mode évolué exposé ci-après.

Fonctionnalité 10 : Lorsque l'utilisateur demande, via le moniteur, le démarrage sans watchdog, le robot doit démarrer dans ce mode. En cas de succès, un message d'acquittement est retourné à nodejs. En cas d'échec, un message indiquant l'échec est transmis à nodejs.

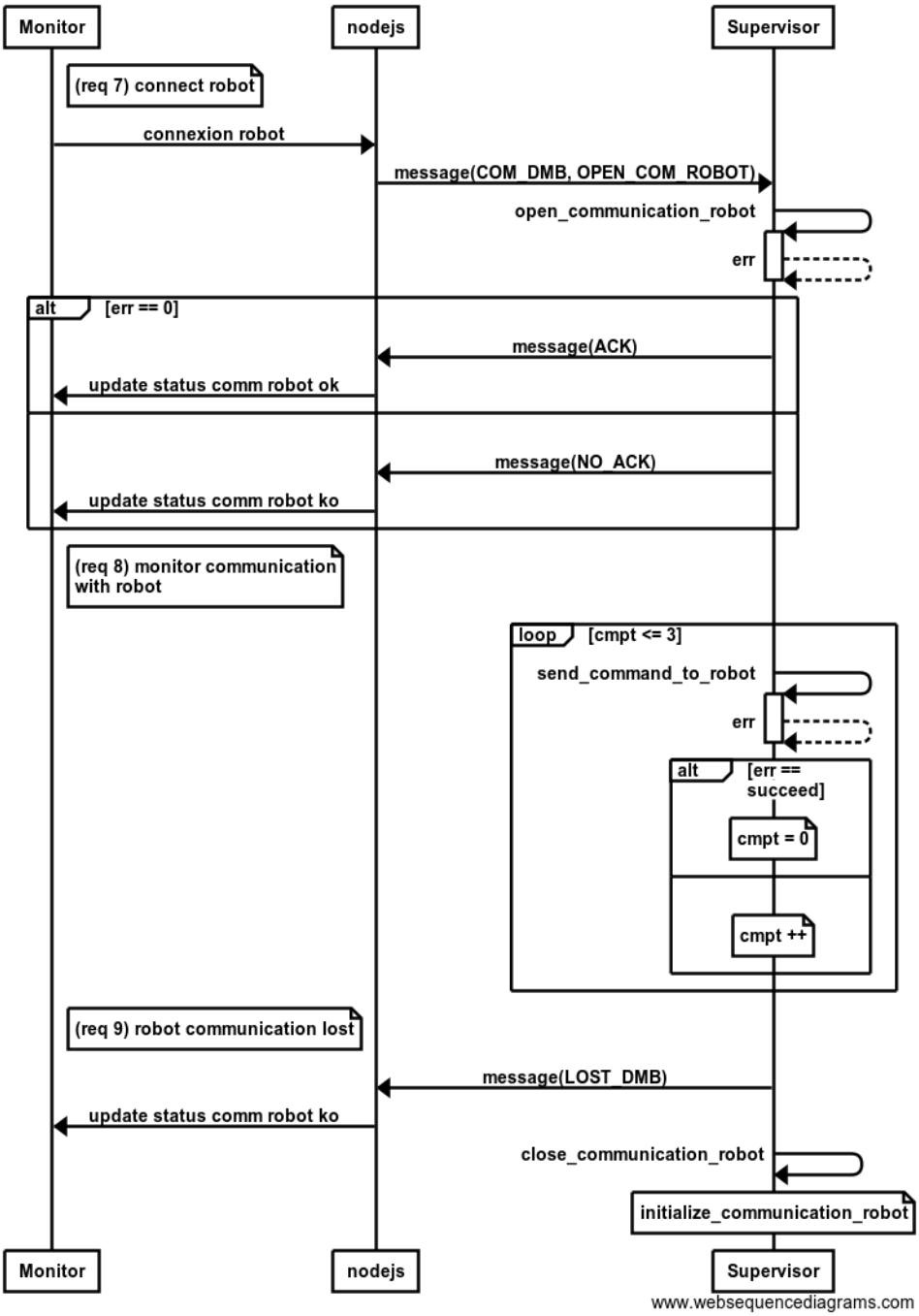

Fig. 8: Diagramme de séquence des fonctionnalités 7 à 9

Démarrage avec watchdog du robot. En cas de perte de la communication entre le moniteur et le robot, ce dernier n'est plus contrôlable et continue à exécuter des ordres. Pour éviter cela, un mécanisme de surveillance à base de watchdog a été mis en place sur le robot.

Le principe est simple : au démarrage du robot (c.-à-d. quand le robot traite l'ordre de démarrage avec watchdog), un watchdog périodique d'une seconde est lancé. À chaque expiration du watchdog un compteur dans le robot est incrémenté. Si le compteur atteint 3, le robot s'arrête et doit être redémarré manuellement. Pour éviter cela, le robot doit recevoir du superviseur un ordre spécifique de rechargement du watchdog (DMB_RELOAD_WD). Si l'ordre est valide, le compteur est décrémenté de 1 (minimum 0). Pour que cet ordre soit valide il faut qu'il arrive au moment où le watchdog expire avec un tolérance de 50 ms.

Fonctionnalité 11 : Lorsque l'utilisateur demande, via le moniteur, le démarrage avec watchdog, le robot doit démarrer dans ce mode. Un message d'acquittement est retourné à nodejs. En cas d'échec, un message indiquant l'échec est transmis à nodejs.
Une fois le démarrage effectué, le robot doit rester vivant en envoyant régulièrement le message de rechargement du watchdog.

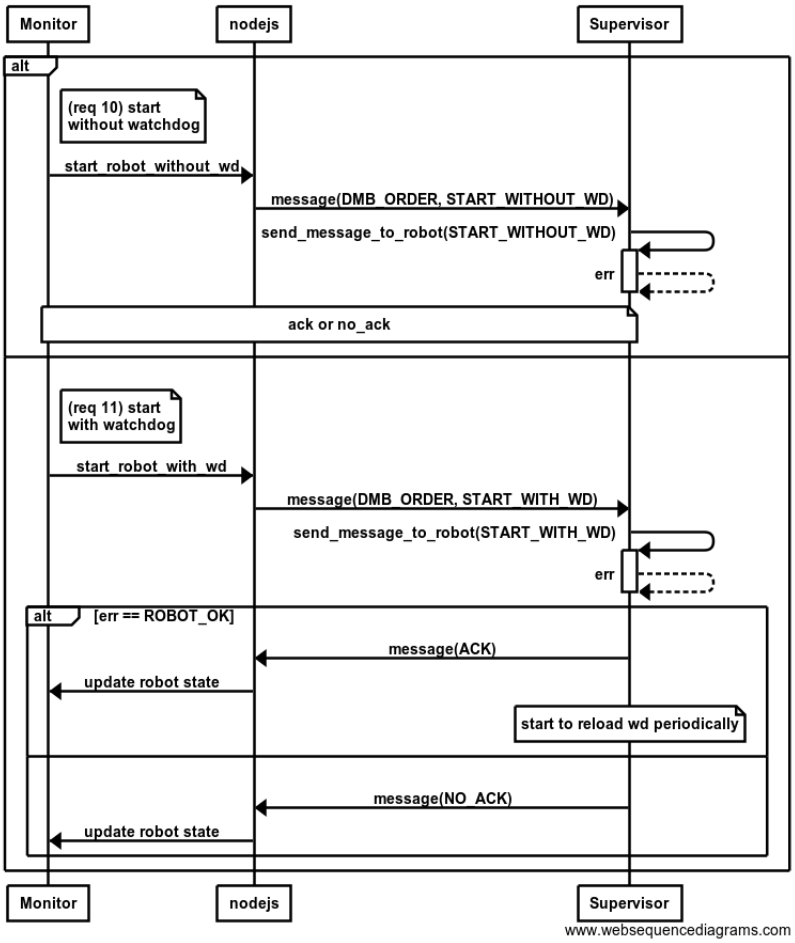

Fig. 9: Diagramme de séquence des fonctionnalités 10 et 11

5.3.5 Déplacement et état du robot

Le robot n'a aucune intelligence et ne réagit qu'aux ordres qu'il reçoit. La figure 10 représente la séquence pour les fonctionnalités 12 et 13.

Déplacement manuel du robot. Le robot peut recevoir cinq ordres de mouvement (avancer, reculer, tourner à droite, tourner à gauche et stopper). Lorsque l'utilisateur presse les flèches du clavier du moniteur un message est envoyé au superviseur avec le mouvement à réaliser. Quand l'utilisateur relâche une touche, un message pour stopper le robot est envoyé.

Fonctionnalité 12 : Lorsque qu'un ordre de mouvement est reçu par le superviseur, le robot doit réaliser ce déplacement en moins 100 ms.

Niveau de batterie du robot. Il est possible de récupérer le niveau de la batterie du robot à l'aide de la fonction `send_command_to_robot` avec l'entête `DMB_GET_VBAT`. La valeur retournée est ensuite à transmettre au moniteur.

Fonctionnalité 13 : Le niveau de la batterie du robot doit être mis à jour tous les 500 ms sur le moniteur.

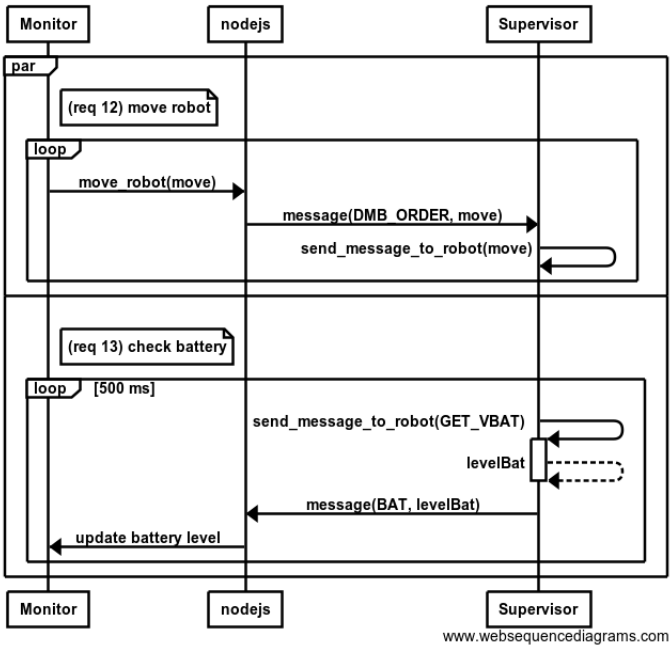

Fig. 10: Diagramme de séquence des fonctionnalités 12 et 13

5.3.6 gestion de la caméra

Les fonctions permettant la manipulation des images sont fournies dans la librairie `image`. L'implémentation des méthodes utilise OPENCV, une libraire libre en C/C++ de traitement d'images.

Ouverture de la caméra. La fonction `open_camera` donne accès à la camera. Il est possible de fermer proprement l'accès à la caméra en faisant appel à `close_camera`.

Fonctionnalité 14 : La camera doit être démarrée suite à une demande provenant du moniteur. Si l'ouverture de la caméra a échoué, il faut envoyer un message au moniteur.

Capture d'une image (mode nominal). La fonction `get_image` permet de capturer une image. La compression de l'image se fait à l'aide de la fonction `compress_image`. L'envoi de l'image au moniteur s'effectue normalement par l'envoi d'un message.

Fonctionnalité 15 : Dès que la camera est ouverte, une image doit être envoyée au moniteur (via nodejs) toutes les 100 ms.

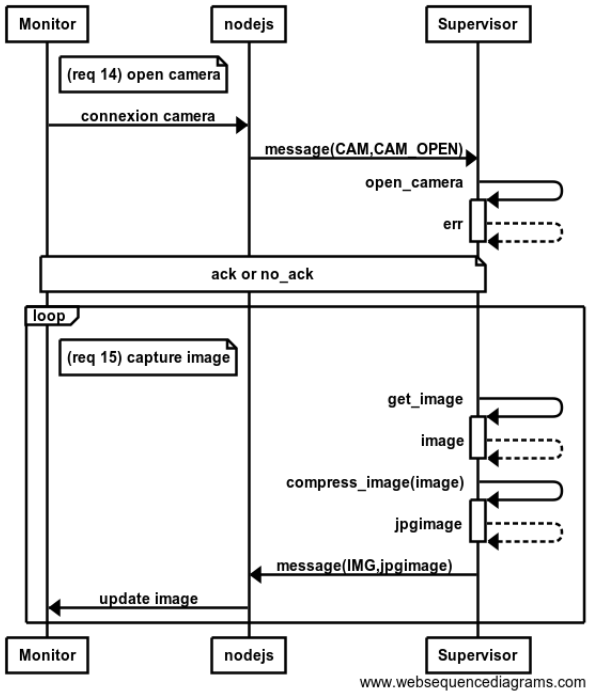

Fig. 11: Diagramme de séquence des fonctionnalités 14 et 15

Calibration de l'arène Pour accélérer le traitement de l'image et le rendre plus stable [1], il est préférable de limiter la zone d'étude à l'arène. Pour cela il est nécessaire de faire un pré-traitement qui recherche l'arène.

Le calcul se fait par l'appel à la méthode `detect_arena`. Il est possible de tracer sur l'image l'arène en faisant appel à la fonction `draw_arena`.

1. Si vous placez un triangle blanc de la forme du robot en dehors de l'arène, il y a de fortes chances pour que cela perturbe le calcul de la position du robot.

Fonctionnalité 16 : Suite à une demande de recherche de l'arène, le superviseur doit stopper l'envoi périodique des images, faire la recherche de l'arène et renvoyer une image sur laquelle est dessinée cette arène. Si aucune arène n'est trouvée un message d'échec est envoyé.

L'utilisateur doit ensuite valider visuellement via le moniteur si l'arène a bien été trouvée. L'utilisateur peut :

- valider l'arène : dans ce cas, le superviseur doit sauvegarder l'arène trouvée (pour l'utiliser ultérieurement) puis retourner dans son mode d'envoi périodique des images en ajoutant à l'image l'arène dessinée.
- annuler la recherche : dans ce cas, le superviseur doit simplement retourner dans son mode d'envoi périodique des images et invalider la recherche.

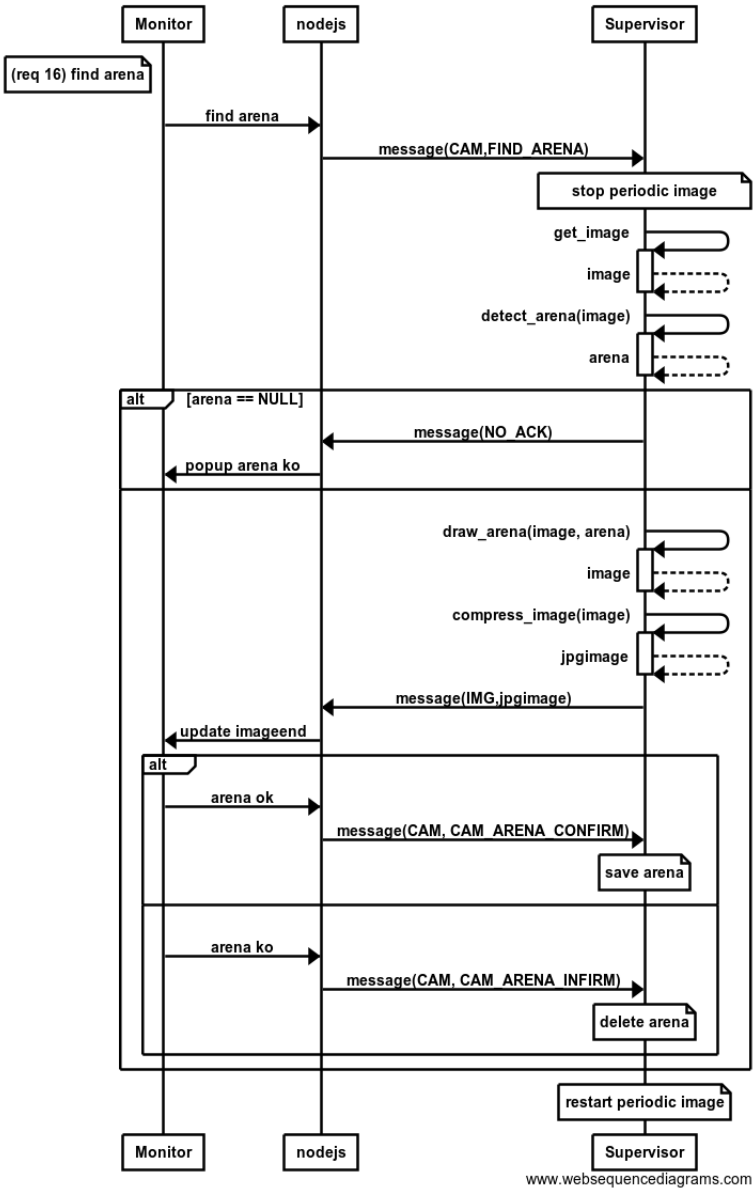

Fig. 12: Diagramme de séquence de la fonctionnalité 16

Calcul de la position du robot. Le traitement d'une image pour trouver la position du robot se fait à l'aide de la méthode `detect_position`. Il est possible de dessiner sur l'image la position trouvée en faisant appel à la fonction `draw_position`. La position est envoyée du superviseur vers le moniteur en utilisant un message avec une entête POS.

Fonctionnalité 17 : Suite à une demande de l'utilisateur de calculer la position du robot, le superviseur doit calculer cette position, dessiner sur l'image le résultat et envoyer un message au moniteur avec la position toutes les 100 ms. Si le robot n'a pas été trouvé, un message de position est envoyé avec une position (-1,-1).

Stopper le calcul de la position du robot. Il est possible pour l'utilisateur de demander l'arrêt du calcul de la position.

Fonctionnalité 18 : Suite à une demande de l'utilisateur de stopper le calcule de la position du robot, le superviseur doit rebasculer dans un mode d'envoi de l'image sans le calcul de la position.

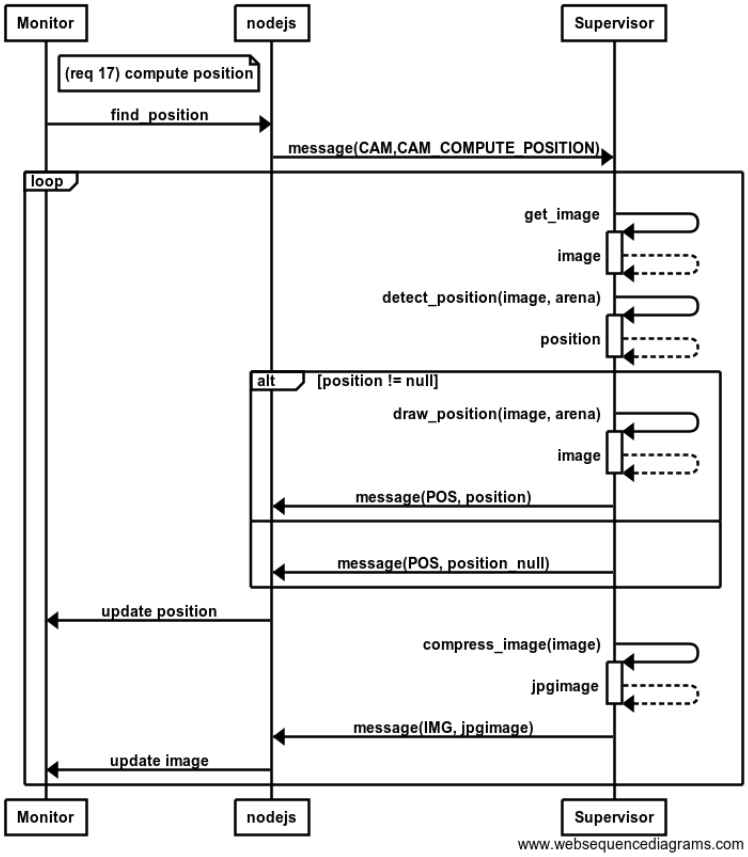

Fig. 13: Diagramme de séquence de la fonctionnalité 17

5.3.7 Réaliser une mission

TBD

Annexes

A Annexes

A.1 Messages robot-superviseur

La communication fonctionne sur le principe d'une communication synchrone de type requête. La communication ne peut être initiée que par le superviseur.

Toutes les communications ont par défaut un retour en cas d'erreur. Si la communication s'est bien déroulée la valeur ROBOT_OK est retrounée, sinon une valeur de retour correspondante à l'un des cas d'erreur suivant est produite :

— ROBOT_TIMED_OUT : la réponse n'est pas arrivée avant 80 ms,
— ROBOT_UNKNOWN_CMD : la commende n'a pas été comprise par le robot,
— ROBOT_ERROR : la commande n'est pas conforme à sa définition,
— ROBOT_CHECKSUM : le contrôle de checksum est erroné.

Les messages envoyés au robot sont composés d'une entête codée sur un octect (`char`) correspondant à l'ordre à réaliser. Certains ordres sont aussi accompagnés d'une donnée codée sur un entier (`int`). Le tableau 1 décrit les entêtes.

Entête	Données	Description	Retour
DMB_PING	–	Teste la disponibilité de la communication	–
DMB_START_WITHOUT_WD	–	Démarre le robot sans le *watchdog*	–
DMB_START_WITH_WD	–	Démarre le robot avec le *watchdog*	–
DMB_RELOAD_WD	–	Recharge le *watchdog*	–
DMB_IDLE	–	Remet le robot dans l'état *idle*	–
DMB_GET_VBAT	–	Retourne le niveau de la batterie	niv. batterie
DMB_IS_BUSY	–	Retourne l'état du robot	état
DMB_GO_FORWARD	–	Déplace le robot en avant	–
DMB_GO_BACK	–	Déplace le robot en arrière	–
DMB_GO_LEFT	–	Fait tourner dans le sens anti-horraire	–
DMB_GO_RIGHT	–	Fait tourner dans le sens horraire	–
DMB_STOP_MOVE	–	Stoppe le mouvement du robot	–
DMB_MOVE	distance	Déplace en ligne droite	–
DMB_TURN	angle	Tourne d'un angle donné	–

Tab. 1: Liste des messages du superviseur vers le robot

Le niveau de la batterie peut prendre comme valeur :
— DMB_BAT_LOW : niveau bas,
— DMB_BAT_MED : niveau moyen,
— DMB_BAT_HIG : niveau haut.

Les états possibles du robot sont :
— DMB_BUSY : le robot est en train de réaliser un mouvement,
— DMB_DO_NOTHING : le robot ne réalise pas de mouvement.

A.2 Messages moniteur-superviseur

A.2.1 Moniteur vers Superviseur

Les message envoyés du moniteur vers le superviseur sont composés d'un entête de trois octets et d'une donnée d'un octet. Le tableau 2 décrit ces entêtes et les données. Certains messages nécessitent une réponse par un acquittement (cf. messages section A.2.2).

Entête	**val.**	**Description**
HEADER_MTS_COM_DMB	COM	Message gérant la communication avec le robot
Données		
OPEN_COM_DMB	o	Demande d'ouverture de la comm. avec le robot
CLOSE_COM_DMB	C	Demande de fermeture de la comm. avec le robot
Acquittement : oui		

Entête	**val.**	**Description**
HEADER_MTS_DMB_ORDER	DMB	Message portant un ordre pour le robot
Données : cf. tableau 1		
Acquittement : les messages DMB_START_WITHOUT_WD et DMB_START_WITH_WD nécessitent un acquitement		

Entête	**val.**	**Description**
HEADER_MTS_CAMERA	CAM	Message d'action sur la caméra
Données		
CAM_OPEN	A	Demande d'ouverture de la caméra
CAM_CLOSE	I	Demande de fermeture de la caméra
CAM_ASK_ARENA	y	Demande de détection de l'arène
CAM_ARENA_CONFIRM	x	L'arène est la bonne
CAM_ARENA_INFIRM	z	L'arène n'est pas la bonne
CAM_COMPUTE_POSITION	p	Calcul de la position du robot
CAM_STOP_COMPUTE_POSITION	s	Calcul de la position du robot
Acquittement : le message CAM_OPEN attend un acquittement.		

Entête	**val.**	**Description**
HEADER_MTS_MSG	MSG	Message textuel
Données : chaînes de caractères		
Acquittement : aucun		

Tab. 2: Description des messages du moniteur vers le superviseur

A.2.2 Superviseur vers Moniteur

Les messages du superviseur vers le moniteur sont composés d'une entête de 3 octets et de données de taille variables. Le tableau 3 décrit ces messages.

Entête	val.	Données	Description
HEADER_STM_ACK	ACK	–	Message d'acquittement en cas de succès
HEADER_STM_NO_ACK	NAK	–	Message d'acquittement en cas d'échec
HEADER_STM_LOST_DMB	LCD	–	Signale la perte de la comm. avec le robot
HEADER_STM_IMAGE	IMG	Image	Envoi d'une image
HEADER_STM_POS	POS	Position	Envoi de la postion du robot
HEADER_STM_MES	MSG	char *	Envoi un message pour la console
HEADER_STM_BAT	BAT	int	Envoi du niveau de la batterie

Tab. 3: Liste des messages du moniteur vers le superviseur

B Diagramme de classe des bibliothèques

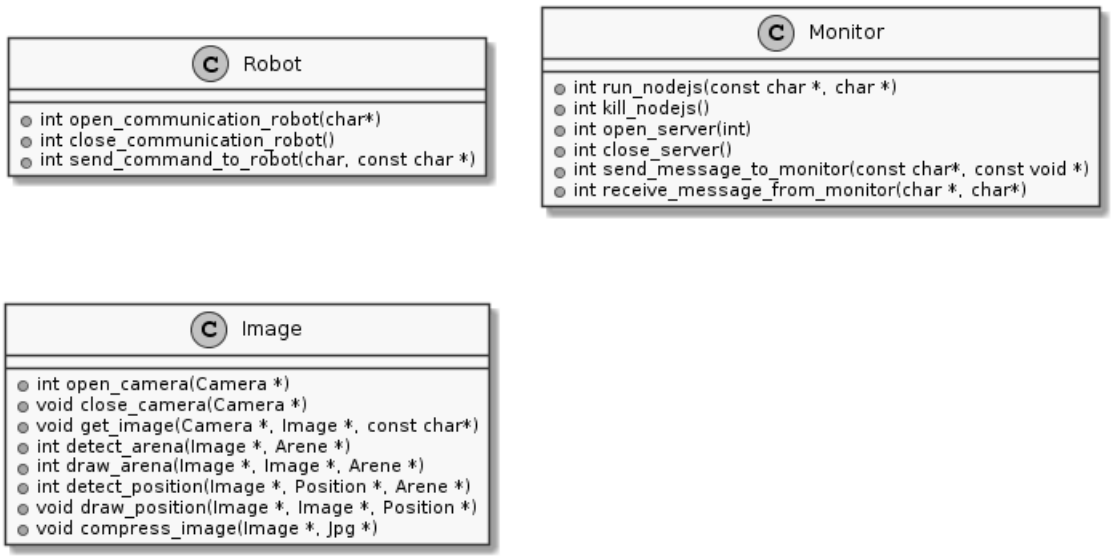

Fig. 14: Diagramme de classes des bibliothèques De Stijl