Investissements d'Avenir
Développement de l'Economie Numérique
« Logiciel embarqué et objets connectés »

Livrables 2.1 & 2.2
Couche internet & logiciels audio/vidéos

Thierry Bergeron, *thierry.bergeron@inria.fr*
Laboratoire I3S, Projet AZKAR, Sophia-Antipolis, France

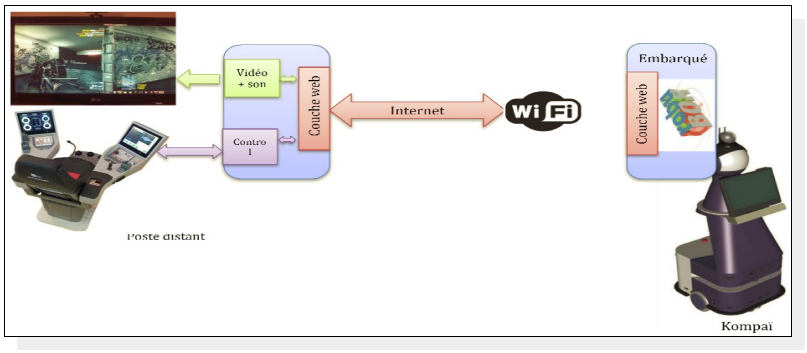
Table des matières

A - Livrable 2.1 : Couche internet....2
B - Livrable 2.2 : Logiciels internet audio/vidéo....4
B.a - Déploiement & configuration....5
B.a.a - Robubox....5
B.a.b - Application 1to1....5
B.a.c - Proxy....5
B.a.d - Serveur Stun/Turn....6
B.a.e - VPN....6
B.b- Lancement & description de l'application....7
B.b.a - Connexion à l'application....7
B.b.b - Page d'accueil....7
B.b.c - Page web du client "Pilote"....8
B.b.d - Mode conférencier....9
B.b.d.a - GamePad....9
B.b.d.b - Plein-écran....10
B.c - Code source....11
B.c.a - Organisation des fichiers....11
B.c.b - Algorithme principal....12
B.c.c - Bugs connus & Optimisations....15

Ce workpackage constitue l'implémentation dans les équipements retenus pour les évaluations des solutions de contrôle-commande par internet. Ce sont essentiellement des développements logiciels, pour des équipements existants. Comme le montre la figure suivante, ces développements concernent :

- La couche web par laquelle transitent aussi bien les ordres de contrôle commande que les images issues du robot.

- L'interface entre la couche web embarquée et RobuBOX en charge de la gestion elle même du contrôle des mouvements du robot et de l'interaction avec l'utilisateur.



Synoptique des couches logicielles dans un schéma classique de contrôle distant de robots

A - Livrable 2.1 : Couche internet

Cette couche internet prend en charge:

- La gestion et l'optimisation des trames de commandes du robot,
- Les informations en provenance du robot (géolocalisation, niveau de batterie, etc),
- Les images et le son vers et en provenance du robot.,
- Les différents protocoles de connexion/déconnexion entre l'utilisateur et le robot.

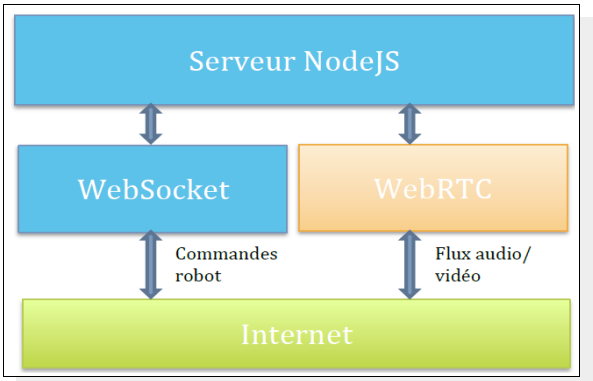


Schéma de principe de la couche internet

Optimisation des commandes : Pour réduire le nombre et la taille des requêtes de contrôle, il est fait appel à la technologie WebSocket. WebSocket est l'un des protocoles de transport les plus polyvalents et souples actuellement disponibles. L'API simple et minimale permet à la couche de livrer des protocoles arbitraires d'application entre client et serveur dans un mode de transmission en continu, où chaque côté peut envoyer des données à tout moment.

Informations techniques (Cartographie, géolocalisation, batterie..) : Là aussi, pour des raisons de fiabilité et de robustesse (reconnexions transparentes), il est fait appel à la technologie webSocket pour faire transiter des informations techniques entre le robot et le poste client. Ainsi en cas déconnexion peer-to-peer, ces informations restent toujours disponibles.

Optimisation des flux audio/vidéo : le streaming de la vidéo et de l'audio nécessite des débits importants de données entre le robot et l'opérateur. La technologie WebRTC permet d'optimiser au maximum l'échange de ces données par une connexion de type pair-à-pair et permet l'utilisation de médias haute définition.

Protocoles de connexion/déconnexion: Compte-tenu de ses qualités (réactivité, transparence), la technologie WebSocket est encore mise à contribution comme canal de signaling pour gérer les protocoles de connexions et de reconnexion WebRTC. Cela permet d'ajouter de la "robustesse" aux communications audio/vidéo en peer-to-peer de WebRTC.

Ainsi, grâce à un serveur NodeJS installé sur un serveur distant, les commandes sont envoyées du poste de pilotage vers le robot via les Web Socket et les flux vidéo et audio bidirectionnels via WebRTC. De plus, la gestion asynchrone des requêtes et l'architecture basée sur les événements de NodeJS permet une meilleure gestion des commandes en boucle fermée.

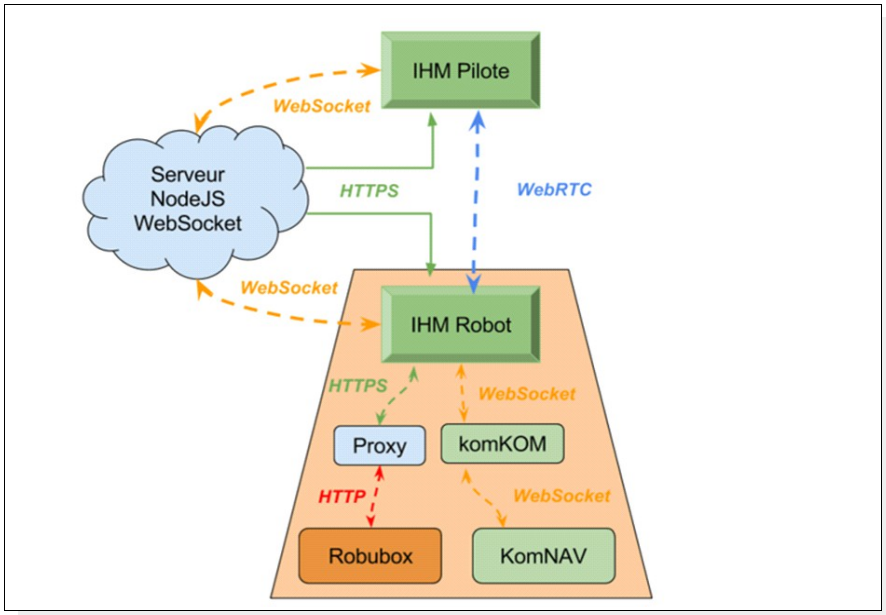


Schéma de déploiement de la couche internet

B - Livrable 2.2 : Logiciels internet audio/vidéo

Le livrable 2.2 est constitué d'un ensemble de logiciels nécessaires à l'écosystème de la solution de contrôle-commande par internet. Il comprend des applications développées au laboratoire I3S à l'aide de technologies open-sources et fournies dans le livrable, mais aussi des applications open-source tierces non fournies qu'il faudra télécharger à part.

Les applications développées au laboratoire I3S, en collaboration avec Robosoft et Anotherworld:

- Le logiciel principal, baptisé "1to1", exposant deux clients web, l'un destiné au robot et fonctionnant en mode passif (aucune intervention de l'utilisateur n'est requise sur cette IHM) , l'autre faisant office de poste de pilotage distant. Ce logiciel assure en arrière-plan les fonctions de Signaling, de configuration des caméras distantes, de "robustesse" du signal, et la commande à distance à l'aide d'un Gamepad ou d'un joystick virtuel, mais aussi la cartographie de géolocalisation et la liaison audio/vidéo. Exposée à l'aide de NodeJS en HTTPS, fonctionnant exclusivement sous Chrome et reposant sur un serveur webSocket, cette application est écrite en javascript et HTML5. Cet ensemble de scripts fait appel à plusieurs bibliothèques open-source dont la plus connue est jQuery.
- Un proxy à installer sur le robot afin d'assurer le dialogue entre le navigateur (client web) du robot et le système embarqué "RobuBOX", mais aussi le dialogue avec un premier prototype du futur système "KomNAV" en cours de développement par Robosoft.

Les applications Open-Source Tierces:

- Serveur nodeJS & WebSocket installé sur un serveur distant.
- Serveur STUN/TURN (RFC-5766) installé sur un serveur distant.
- VPN (solution commerciale ou serveur de type OpenVPN) pour les rares cas où les filtrages réseaux rendraient les relais TURN inopérants.

Concernant les serveur STUN/TURN, les solutions commerciales semblent encore peu nombreuses et seules deux sont proposées sous forme de service (xirsys.com & anyfirewall.com). Toutefois, aucune de ces deux solutions n'as donné satisfaction lors des essais:

- Soit l'API était écrite en php et son portage en javascript (non documenté) n'as rien donné.
- Soit les crédentials éphémères fournis par l'API avaient une durée de vie trop courte pour s'accorder à l'algorithme de signaling WebRTC de l'application 1to1.

La solution OpenSource RFC-5766 qui permet de relayer les connexions peer-to-peer en TURN si le STUN ne passe pas fonctionne dans la majorité des cas. Toutefois, à notre grande surprise, il s'est révélé des situations où les filtrages des réseaux Wifi étaient tels que mêmes ces relais TURN ne passent plus (c'est le cas des réseau Wifi universitaire...). Ce cas de figure n'est malheureusement pas documenté ni même évoqué dans les forums spécialisés. La seule solution de contournement dans ce cas est d'utiliser un VPN afin de "tunelliser" la connexion des clients filtrés.

Des solution VPN open-source existent (la plus recommandée étant OpenVPN), mais sont complexes et laborieuses à mettre en œuvre. Le plus simple et le moins coûteux en terme de ressources temps est de souscrire à une offre commerciale illimitée en débit et de connecter au web le client "filtré" par le biais de ce VPN.

B.a - Déploiement & configuration

B.a.a - Robubox

Au préalable, récupérer manuellement la cartographie du lieu ou évoluera le robot. Pour cela lancer la robubox, (clic droit sur le nom de la carte active > menu contextuel > exporter la carte en png). A chaque modification ou changement de carte active, il faudra veiller à ce que cette version soit exportée en png et copiée dans le répertoire "images" de l'application 1to1.

B.a.b - Application 1to1

1. Sur le serveur choisi (local, distant ou virtuel, avec nodeJS et npm préalablement installé), il suffit de décompresser le contenu du zip dans un répertoire dédié.
2. Ajouter dans le répertoire "images" de l'application la carte courante exportée par la Robubox.
3. Renseigner les paramètres de configuration propre à la branche de travail (js/branch_settings/common_app_branch_settings.js). NOTE IMPORTANTE: Ne pas toucher au fichier de configuration par défaut (js/common_app_settings.js) car il sera automatiquement remplacé à chaque changement de version de la branche principale lors des mises à jour majeures et toutes les configurations personnalisées seront perdues.
    1. Dans la fonction serServers():
        1. L'adresse IP (ou le nom de domaine) du serveur de l'application,
        2. le numéro de port (80 par défaut, idéalement 443 pour du http),
        3. Générer des certificats auto-signés en suivant le tutoriel présent dans le répertoire ssl de l'application, puis renseigner les variables aux certificats générés. Il est toutefois possible de laisser les certificats et clefs proposés par défaut (hacksparow-key et acksparow-cert), mais ceux-ci ne sont pas mémorisables par le navigateur et il faudra à chaque nouvelle instance de Chrome autoriser manuellement la connexion https sur les deux clients.
    2. Dans la fonction setIceServers() Renseigner la liste des serveurs STUN/TURN (). Il est conseillé de ne mettre qu'un seul serveur TURN fonctionnel pour diminuer les échanges de "candidates" superflus et raccourcir les délais de connexion/reconnexion.
    3. Dans la fonction isFakeRobubox(): Si on veut juste tester l'application sur un pc non relié à un robot, il faut émuler une Robubox. Pour cela mettre le flag fakeRobubox à "true". Ce flag permet d'émuler des données de cartographie, de géolocalisation et de batterie et bloque la retransmission de commandes reçues par le client robot vers le robot physique pour éviter de provoquer des erreurs d'exécution.
    4. Renseigner le nom et l'emplacement de la carte courante (fonction getMapSource).
4. Pour lancer l'application, se placer dans son répertoire racine et lancer la commande appropriée (sous Windows: "node server" et sous Linux: "sudo nodejs server.js")

B.a.c - Proxy

Récupérer sur dans le Github du projet ou dans son archive zip le répertoire "PROXY" et le copier quelque part sur le pc qui embarque la robuBox. L'utilisation de ce proxy est inutile si le mode "fakeRobubox" est activé. Pour lancer le proxy, se placer dans son répertoire et faire (sous Windows: "node proxy.js"). Remarque: Ce proxy sera inutile une foi KomNav totalement fonctionnel.

B.a.d - Serveur Stun/Turn

Téléchargement, installation & configuration basique sur un serveur Ubuntu 14.04 desktop:

1. Dans la logithèque ubuntu, entrer dans l'onglet de recherche "rfc 5766" et sélectionner l'archive "TURN and STUN server for Voip" et précisant "rfc5766-turn-server" et procéder à l'installation. Les fichiers seront installés dans le répertoire usr/share/rfc5766-turn-server.
2. Copier les fichiers de configuration turnserver.conf et turnuserdb.conf présent dans usr/share/rfc5766-turn-server/examples/etc
3. Renseigner ces fichiers en suivant les instructions en commentaires et les placer dans etc/
4. Dans un terminal, lancer manuellement le serveur en tapant par exemple la ligne de commande suivante: " sudo turnserver -v -r xx.xxx.xxx.xx -a -b turnuserdb.conf -c turnserver.conf ". Les détails sont les suivants:
   - -v = Petite verbosité (ou -V = grande verbosité)
   - -r = Realm (obligatoire avec le mecanisme "Longs terms credentials")
   - xx.xxx.xxx.xx correspondant a l'adresse IP ou à l'url du serveur
   - -a = Mécanisme "Longs terms credentials" (obligatoire pour WebRTC)
   - -b = Nom du fichier stockant les credentials utilisateurs
   - -c = Nom du fichier de configuration
5. Pour vérifier que le serveur est bien lancé, dans un teminal taper la commande " sudo pgrep turnserver " qui retournera l'ID du processus si celui-ci est actif.
6. Si le processus est lancé, pour l'arrêter taper "sudo pkill turnserver ".

B.a.e - VPN

De nombreuses solutions commerciales existent et sont extrêmement simples à mettre en œuvre. Il suffit de souscrire à l'abonnement désiré auprès du fournisseur de service et de charger le client proposé sur la machine à connecter. Nous utilisons au laboratoire I3S une solution commerciale basée sur OpenVPN, privatetunnel.com, qui nous permet de passer les filtrages des réseaux wifi universitaires. Cette solution nous a donné entière satisfaction en terme de rapidité, de stabilité et de facilité de mise en œuvre.

A titre d'exemple, un abonnement permet de connecter jusqu'à 10 appareils simultanément pour un coût de 25/30 € par ans sans limitation de bande passante et avec un débit très suffisant pour faire transiter un flux audio/video bidirectionnel en haute définition.

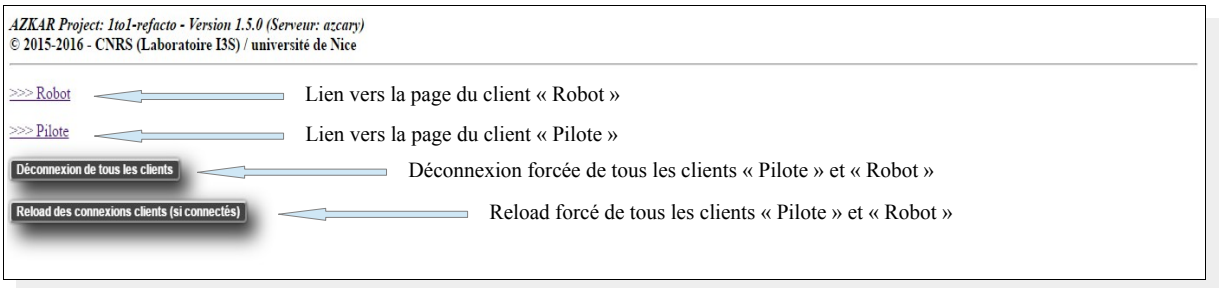
B.b - Lancement & description de l'application

B.b.a - Connexion à l'application

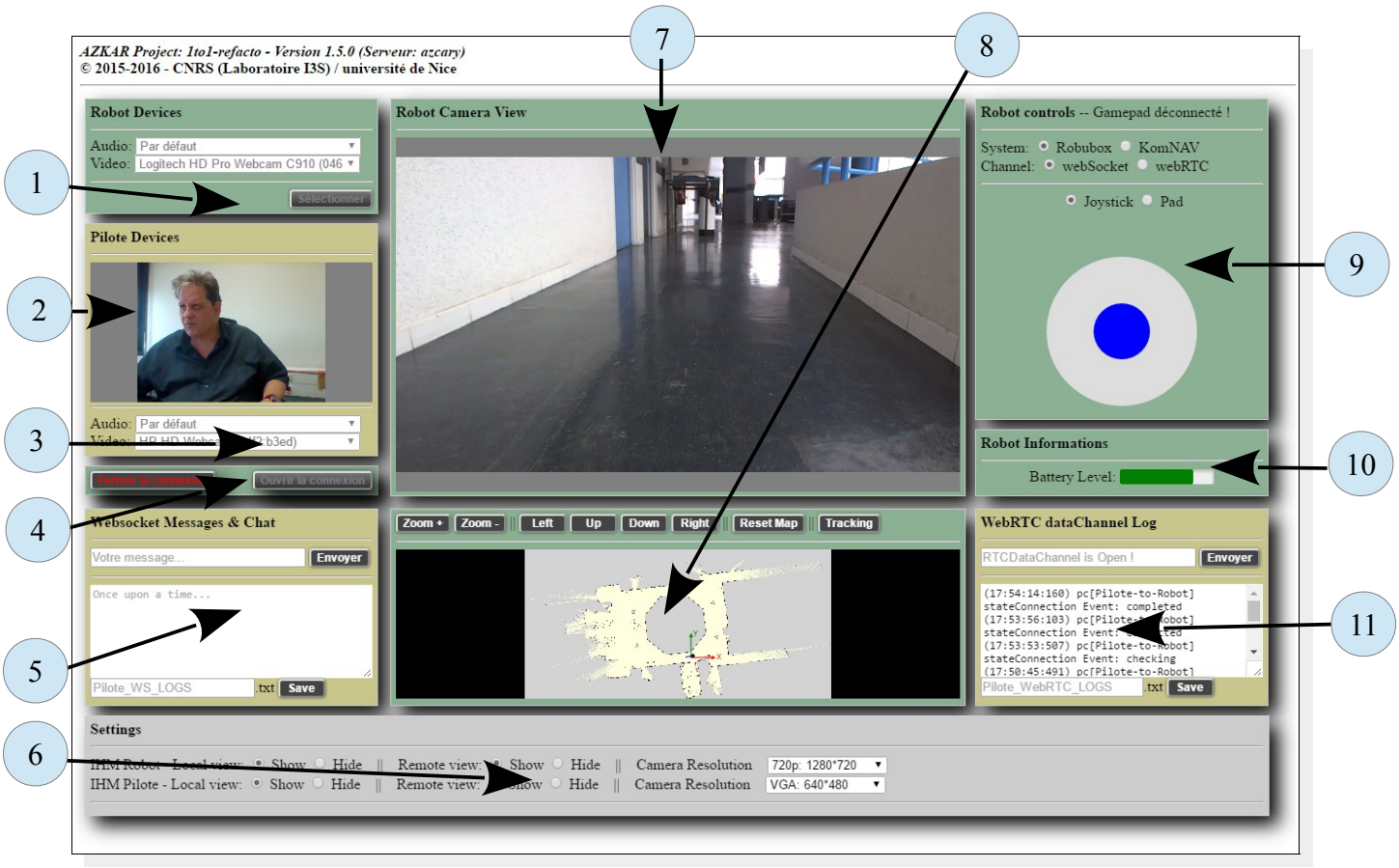
1. Coté Robot:
    1. Activer la Robubox. (Inutile si fakeRobubox est activé).
    2. Activer le proxy. (Inutile si fakeRobubox est activé).
    3. Dans Chrome, ouvrir le proxy et garder l'onglet actif (Inutile si fakeRobubox est activé).
    4. Dans un autre onglet, entrer l'url de l'application puis aller sur la page "Robot."
    5. Entrer un nom d'utilisateur si demandé ou laisser par défaut.
    (Ce pseudo est enregistré dans un cookie)
    6. Attendre qu'un pilote se connecte et laisser faire...

2. Coté pilote:
    1. Dans Chrome, entrer l'url de l'application puis aller sur la page "Pilote."
    2. Entrer un nom d'utilisateur si demandé ou laisser par défaut.
    (Ce pseudo est enregistré dans un cookie)
    3. Sélectionner les caméras locales et distantes, puis, si besoin, sélectionner la résolution voulue pour chaque caméra et configurer pour chaque client les flux vidéos à afficher.
    4. Lancer la connexion et attendre que la caméra distante apparaisse.
    5. Si le message d'erreur suivant "Connection Failed" apparaît, alors l'un des deux clients (ou les deux) sont probablement filtrés par le réseau. Il faudra alors ouvrir une connexion « tunellisée » VPN sur le client concerné et reprendre l'opération.
    6. S'il n'y as aucun message d'erreur mais que l'image distante n'apparaît pas, on peut dans un premier temps "fermer la connexion" et relancer celle-ci. Si ça échoue, alors il faut recharger la page, reprendre tous les réglages (caméras, résolutions, etc..) et lancer une nouvelle connexion.

B.b.b - Page d'accueil



B.b.c - Page web du client "Pilote"



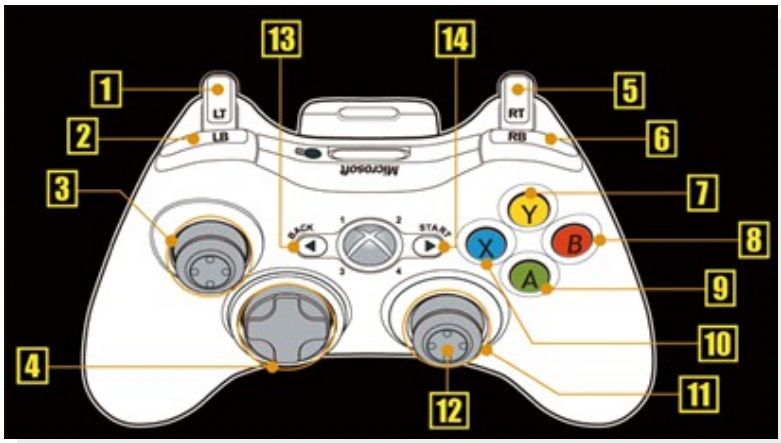
1. Sélection de la caméra et du micro distant (sur le robot)
2. Affichage de la caméra locale (pilote)
3. Sélection de la caméra et du micro local (pilote)
4. Boutons d'ouverture et de fermeture de la connexion audio/vidéo (WebRTC)
5. Console de débuggage, tchat & messages propres au canal webSocket
6. Bloc de configuration des affichages & sélection des résolutions caméras
7. Affichage de la caméra distante (robot)
8. Affichage cartographie & géolocalisation du robot distant
9. Bloc des commandes (Joystick & pad virtuels, détection du Gamepad, etc...)
10. Jauge de la batterie du robot
11. Console de débuggage, tchat & messages propres au canal WebRTC

B.b.d - Mode conférencier

Le mode "conférencier" consiste simplement à utiliser conjointement un Gamepad et le plein écran.

B.b.d.a - GamePad

Il suffit de brancher et/ou d'allumer le Gamepad pour que celui-ci soit reconnu automatiquement par l'application. Celui-ci doit obligatoirement être de type Xbox 360. Tout autre modèle ne sera pas pris en compte. Au premier envoi d'une commande valide, l'affichage du bloc de commande bascule et affiche en temps réel les vitesses avant/arrière et la direction sous forme de jauges. Pour revenir au Joystick/pad virtuel, il suffit d'éteindre le Gamepad et l'affichage bascule automatiquement.



- 1) Marche & vitesse arrière
- 2) Sélection de la caméra distante
- 3) Rotation gauche/droite
- 4) UP: Homme mort pour le joystick gauche
- 5) Marche & vitesse avant
- 6) Sélection de la résolution de la caméra distante
- 7) Ouverture/Fermeture de la connexion Audio/Vidéo (WebRTC)
- 8) Bouton STOP (en cours d'implémentation)
- 9) Bouton "Homme mort" pour la vitesse normale (mode standard)
- 10) Bouton "Homme mort" pour la vitesse réduite (mode précision)
- 12) Joystick gauche (Avant/Arrière & Droite/gauche & Vitesse)
- 13) Fermeture des notifications
- 14) Activation/désactivation du mode plein-écran

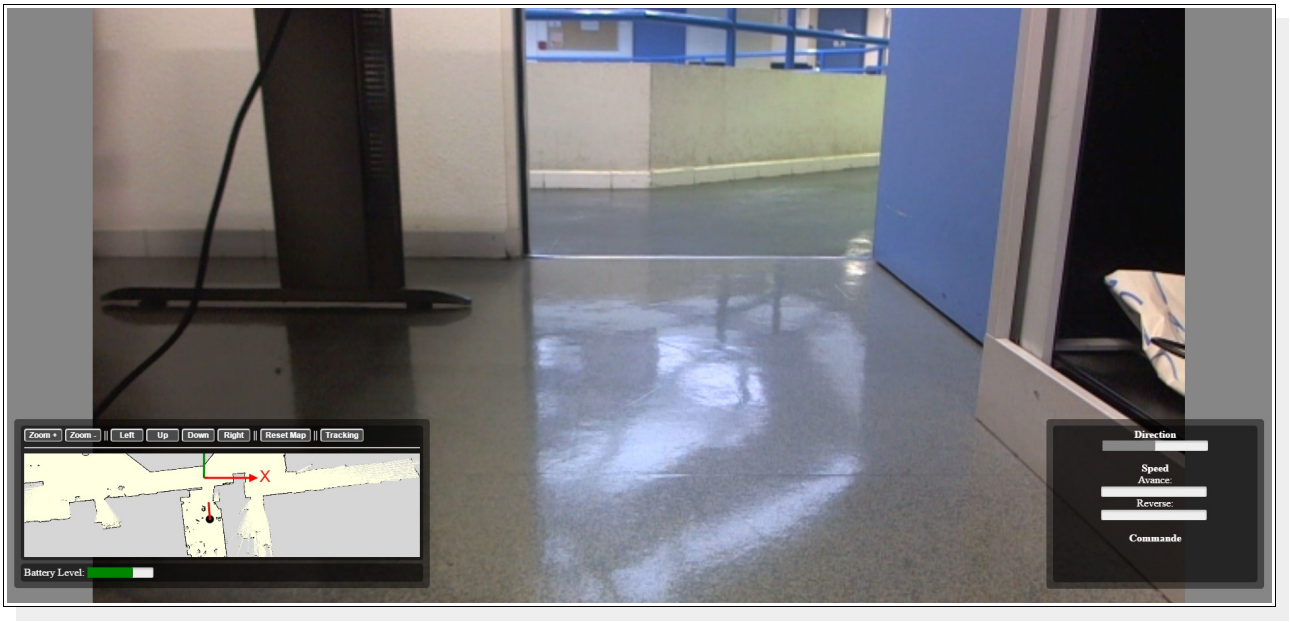
Pour que les déplacements soient actifs, il est impératif de maintenir enfoncé l'un des 3 boutons homme-mort (A ou X ou croix « UP ») en fonction du mode choisi.

B.b.d.b - Plein-écran

Sur le clavier, pour activer/désactiver le plein-écran, appuyer sur la touche de tabulation.

Il ne s'agit pas d'un plein écran réel qui s'affiche sur tout l'écran, mais d'un fullScreen « responsive » qui s'adapte à la largeur de la fenêtre du navigateur. Cette solution à été choisie pour 2 raisons:

- Le vrai mode FullScreen vidéo proposé par HTML5 ne permet pas d'afficher des éléments (tel les bandeaux de notifications ou les éléments HTML comme le Canvas affichant la cartographie) par dessus la vidéo, il faudrait que ces éléments soient inclus dans le flux vidéo lui-même.
- Pour des raisons de sécurité, Chrome bloque le déclenchement de ce mode s'il ne provient pas exclusivement d'un événement clavier (human gesture), ce qui rend très compliqué son activation par l'intermédiaire du Gamepad ou par l'intermédiaire d'un script.



B.c - Code source.

Le code source de l'application est disponible sur GitHub à l'adresse suivante:

https://github.com/AzkarProject/websocket-azkar

La branche principale est « 1to1-master ». les branches « 1to1-cnrs », « 1to1-robosoft » et « 1to1-anotherworld » ont été crées pour chaque partenaire devant adapter le code à ses besoins. A chacun de cloner et travailler sur sa propre branche.

La commande git pour cloner une branche spécifique plutôt que l'ensemble du projet est:

git clone -b **nomdelabranche** --single-branch https://github.com/AzkarProject/websocket-azkar.git

B.c.a - Organisation des fichiers

Pour adapter l'application aux besoins spécifiques de chaque évaluation et à l'évolution des systèmes embarqués par le robot (passage progressif de la Robubox au nouveau système KomNav), le code de l'application à été découpé en différents modules fonctionnels. La convention de nommage adoptée est la suivante:

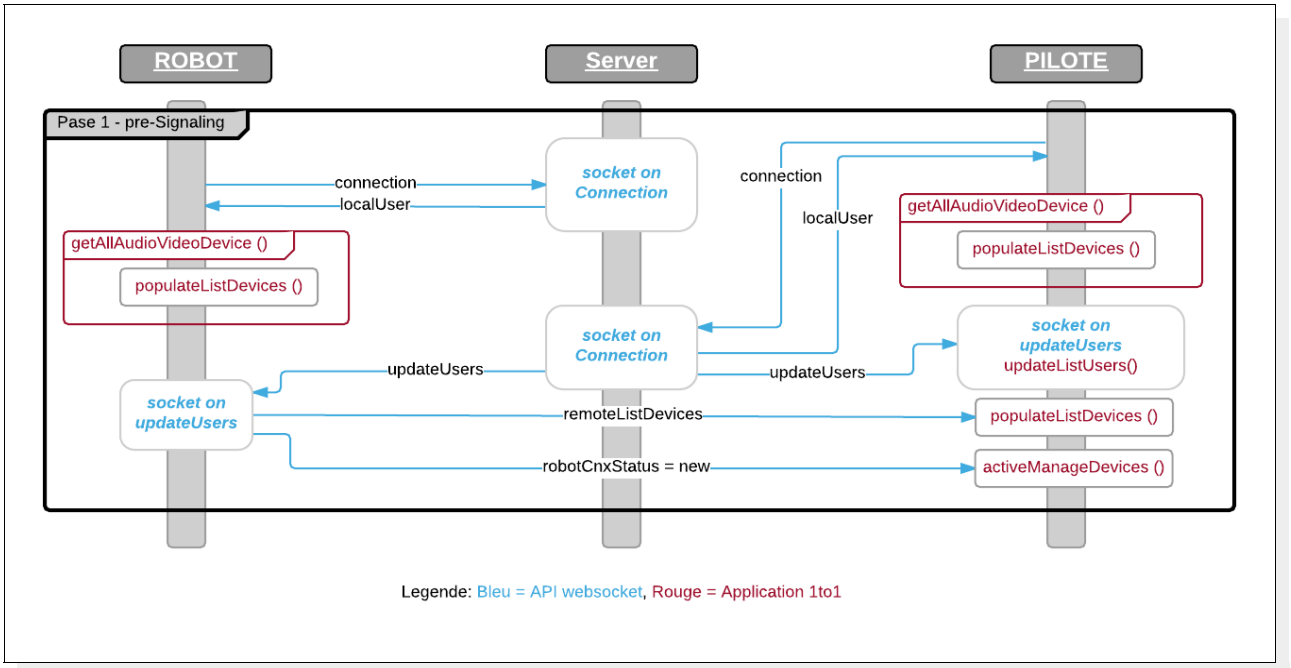
- **common_** pour les fonctions génériques, objets et paramètres communs à la partie cliente et serveur. Ces éléments doivent être chargés en tout premier lieu.
  - common_tools.js
  - common_appsettings.js
  - common_models.js
- **1to1_** pour les scripts procéduraux et l'algorithme principal de l'application. Ces scrips doivent êtres chargés et exécutés dans un ordre précis. Il ne devrais pas avoir lieu a les modifier sauf pour des raisons de débuggage ou de factorisation du code.
  - 1to1_01_webSocket.js
  - 1to1_02_pre_signaling.js
  - 1to1_03_signaling
  - 1to1_04_post_signaling.js
- **module_** pour les sous-ensembles fonctionnels regroupés par usage.
  - module_carto.js
  - module_fake_robubox.js
  - module_forms.js
  - module_gamepad.js
  - module_ihm.js
  - module_navigation_interface.js
  - module_notifications.js
  - module_userconnections.js
  - module_logs.js
  - module_komremote.js*
  - module_komcom.js**

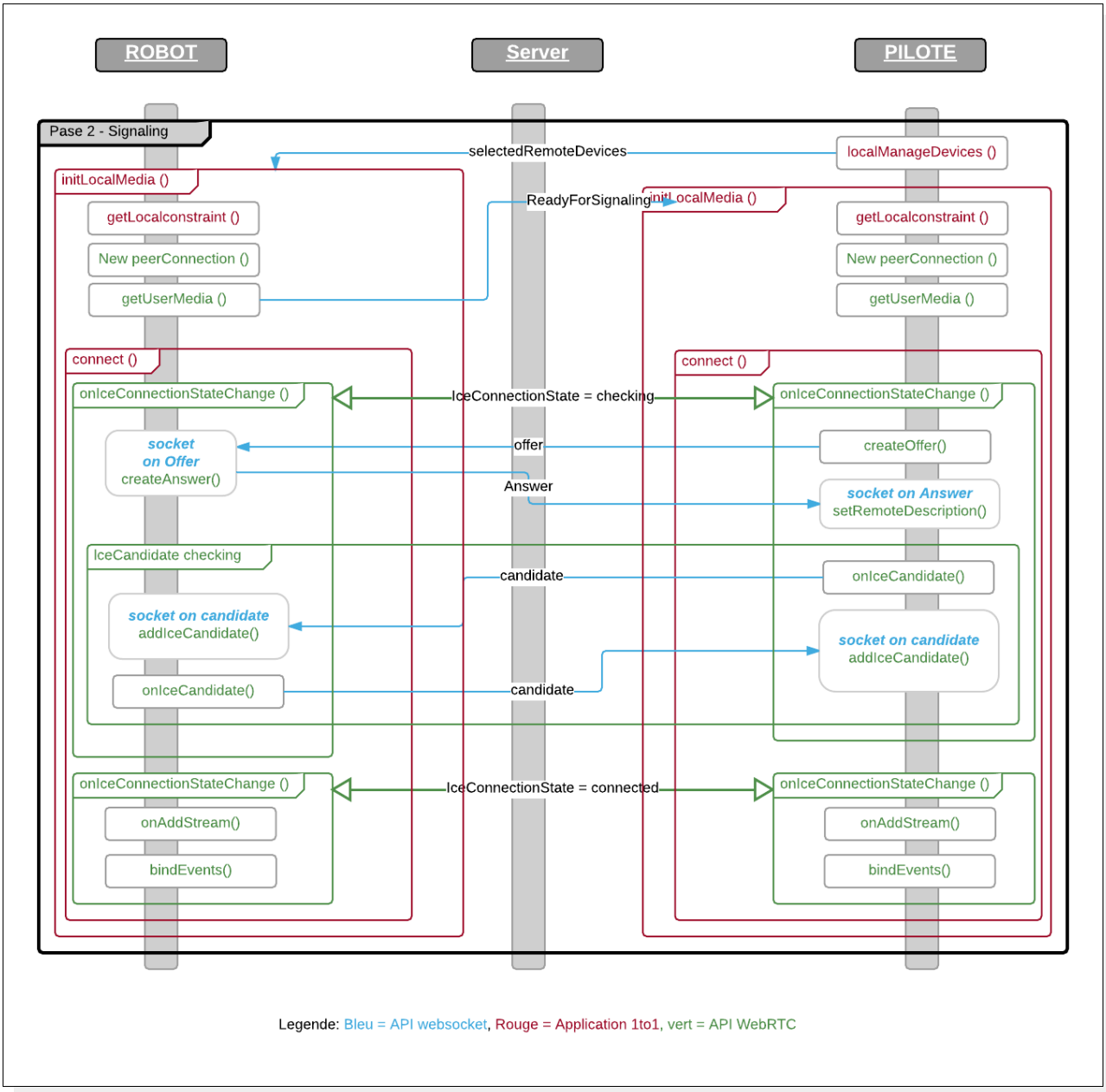
* KomRemote est le module permettant de gérer le joystick virtuel et la croix directionnelle virtuelle
** Komcom est le module faisant interface entre l'application et les versions actuelles de la Robubox et KomNav. C'est ici qu'il faudra intervenir pour interfacer l'existant avec les futures version de KomNav.

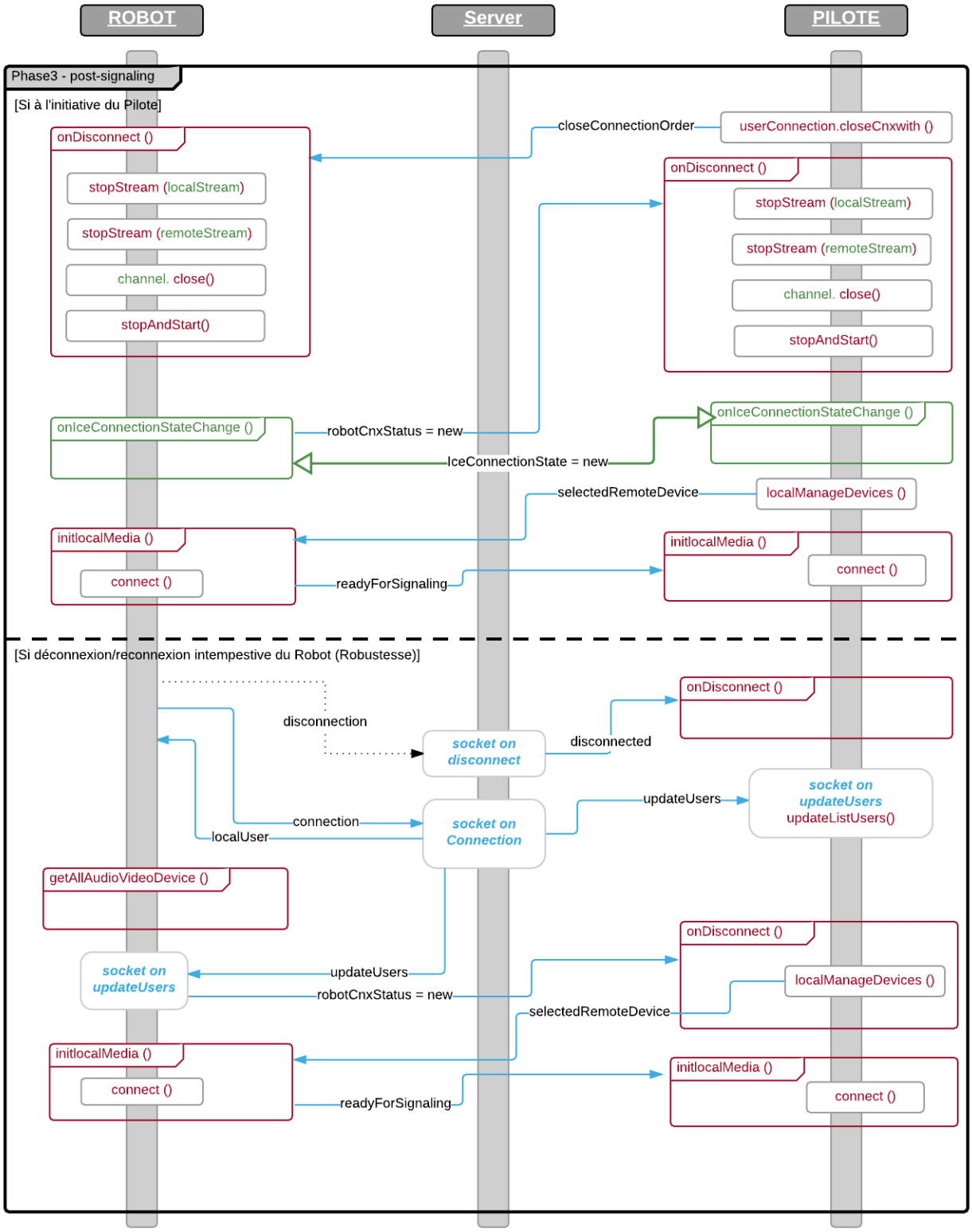
B.c.b - Algorithme principal

L'algorithme procédural de l'application est divisé en 3 grandes phases.

1. **pre-signaling**: Initialisation et gestion de la connexion Websocket, sélection des paramètres d'exécution (caméras, définitions et affichages locaux et distants)
2. **signaling**: Initialisation de la connexion WebRTC. Ouverture des flux audio/vidéos et du canal de donnée (WebRTC data channel).
3. **post-Signaling**: Procédures de déconnexion & reconnexion WebRTC, manuelles et automatiques (robustesse).




ROBOT
Server
PILOTE
Pase 2 - Signaling
selectedRemoteDevices
localManageDevices ()
initLocalMedia ()
ReadyForSignaling
initLocalMedia ()
getLocalconstraint ()
New peerConnection ()
getUserMedia ()
getLocalconstraint ()
New peerConnection ()
getUserMedia ()
connect ()
connect ()
onIceConnectionStateChange ()
IceConnectionState = checking
onIceConnectionStateChange ()
socket on Offer
createAnswer()
offer
createOffer()
Answer
socket on Answer
setRemoteDescription()
IceCandidate checking
candidate
onIceCandidate()
socket on candidate
addIceCandidate()
socket on candidate
addIceCandidate()
onIceCandidate()
candidate
onIceConnectionStateChange ()
IceConnectionState = connected
onIceConnectionStateChange ()
onAddStream()
bindEvents()
onAddStream()
bindEvents()
Legende: Bleu = API websocket, Rouge = Application 1to1, vert = API WebRTC



Legende: Bleu = API websocket, Rouge = Application 1to1, vert = API WebRTC

B.c.c - Limitations & Optimisations

- Limitations:
  - Le poste pilote doit OBLIGATOIREMENT disposer d'une webcam.
  - Le gamepad doit IMPERATIVEMENT être de type XBOX 360
  - Mêmes navigateurs (et mêmes versions) coté pilote et robot: Chrome V 49 minimum
- Optimisations à envisager:
  - Modification de l'interfaçage avec avec KomNav. (voir module_komcom.js)
- Bug aléatoires non bloquants:
  - L'image du pilote se fige coté robot alors que les connexions WebRTC et webSocket ne sont pas interrompue. Il semblerai que le problème soit du à une chute du framerate de l'image reçue. L'implémentation actuelle de WebRTC ne permet pas de détecter et d'intercepter ce type d'événement, pour, par exemple, enclencher un processus de reconnexion automatique (robustesse).
  - les caméras distantes ne sont pas identifiées par un label (mais restent sélectionnables). Le mieux est d'abord un refresh de la page coté robot puis un refresh de la page coté pilote.
- Bug aléatoires bloquants:
  - L'image du robot reste vide et la connexion distante en WebRTC ne se fait pas. C'est souvent un échec pendant la phase d'échange de candidates du signaling (message en console: "Error processing ICE candidate"). La meilleure des solutions reste le refresh de la page pilote et la relance du processus.

En dernier recours: 2 boutons sont présents sur la page d'accueil:

- Le premier permet de forcer la déconnexion de tous les clients (pilotes et robots) et de forcer leur retour sur la page d'accueil afin de vider la liste des utilisateurs connectés. L'inconvénient et qu'il faut manuellement reconnecter le robot à sa page (avec teamviewer par exemple)
- Le second permet de forcer le reload des pages "pilote" et "robot" de tous les clients connectés en dehors de la page d'accueil. Attention, si exécuté dans une même instance du navigateur (dans un autre onglet par exemple), cela forcera le retour du client connecté dans l'autre onglet à la page d'accueil...

Enfin, si vraiment rien ne fonctionne (souvent a cause d'un problème de contrôle d'accès lié à un utilisateur fantôme qui ne s'est pas proprement déconnecté de la session websocket), on peut toujours relancer le serveur nodejs de l'application pour réinitialiser la session websocket.