09/04/2014

ROBAIR PROJET - RICM4

Tuteur : M. Didier Donsez | BARBIER Jérôme - HUSSON Augustin

Table des matières

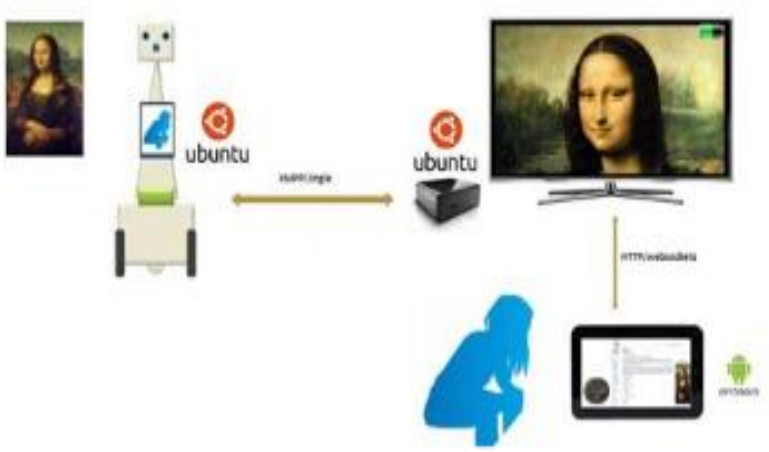
INTRODUCTION 3
CONTEXTE 3
RÉALISATION : 3
WEBRTC : 4
Qu'est-ce que WebRTC ? 4
Réalisation des objectifs : 5
Objectif 1 : La première connexion entre deux navigateurs 5
1°) Mise en place du serveur 5
2°) Côté client 6
Détail du processus de connexion entre deux navigateurs : 6
Objectif 2: L'ajout du streaming 7
Objectif 3: Le multiUser 7
ROS : 8
PERSPECTIVES POSSIBLES A NOTRE DEVELOPPEMENT : 9
Gestion de projet 9
Métriques : 10
CONCLUSION 10

INTRODUCTION

Le projet RobAIR a pour objectif le développement d'une plateforme de robotique de téléprésence ouverte, destiné à la fois à l'enseignement de l'intelligence ambiante et à l'expérimentation à faible coût (ie très en dessous du prix des plateformes du marché) de la robotique de service dans des environnements réels. Cette plateforme se veut extensible et libre source (open software, open hardware, open design, open data) et d'un coût abordable. Ce projet a commencé il y a plusieurs années et chaque année une nouvelle fonctionnalité est ajoutée. Cela a pour but de faire en sorte que d'ici quelques années, Robair soit opérationnel à l'extérieur de nos établissements.

CONTEXTE

Robair est un robot de télé-présence qui sera utilisé dans le contexte de visite de musée. Le rôle principale de ce robot est de permettre à un utilisateur de réaliser une visite virtuelle depuis chez soit.



L'objectif principal de notre projet est de réaliser le système de visio-conférence en utilisant un protocole de "Real Time Communication".

De plus, n'importe quel utilisateur doit pouvoir contrôler le robot via une interface graphique.

Notre projet est susceptible d'être associé à d'autres projets en cour afin d'offrir toute une panoplie de service : comme par exemple le projet COQP qui permettrait de choisir sur quel robot l'on souhaite se connecter.

RÉALISATION :

Durant le temps qui nous été impartie, nous avons tenus nos objectifs et tenté de commencer/finir certaines extensions citées plus bas. En effet, la réalisation concrète comprend la mise en place de la communication webRTC, la communication entre webRTC et ROS afin de contrôler le robot, une connexion multi-client ainsi qu'une page web de connexion.

Les premières étapes de réalisation ont été de reprendre et comprendre les projets des années précédentes sur Robair. Pour cela, les pages wiki ainsi que les quelques explications présentes sur leurs pages wiki nous ont aidé à comprendre leurs codes et leurs réflexions. Cependant, cela fût très long et fastidieux. Nous avons donc décidé avec

nos tuteurs de tout reprendre et d'implémenter directement notre protocole.

En premier lieu, le protocole utilisé par skype avait été choisi, mais très vite, le fait que le code soit entièrement fermé à empêcher de continuer dans cette voie-là. C'est pourquoi l'utilisation de webRTC (utilisé par google hangout) fut notre second choix.

En même temps que webRTC, le nœud ROS doit permettre la communication entre notre application web et le robot.

Une autre étape très importante a été de monter notre robot. Pour cela, nous avons récupéré les pièces déjà découpées et les avons assemblées.

WEBRTC :

Qu'est-ce que WebRTC ?

D'après wikipédia :

*« **WebRTC** (**Web Real-Time Communication**, littéralement communication web en temps réel) est une interface de programmation (API) JavaScript actuellement au stade de brouillon (Draft) développée au sein du W3C et de l'IETF. C'est aussi un canevas logiciel avec des implémentations précoces dans différents navigateurs web pour permettre une communication en temps réel. Le but du WebRTC est de lier des applications comme la voix sur IP, le partage de fichiers en pair à pair en s'affranchissant des plugins propriétaires jusqu'alors nécessaires. »*

Cette API est implémentée dans les navigateurs internet suivant : Google Chrome, Firefox et Opéra.

Après avoir regardé quelques exemples sur internet pour voir comment fonctionnait l'API JavaScript, nous avons alors établit une série d'objectif allant progressivement en complexité :

- Objectif 1 : établir une connexion permanente entre deux navigateurs internet et envoyer via cette connexion des messages écrits. Cet objectif devait être rapidement mis en place afin de pouvoir voir comment nous pourrions commander Rob-Air.
- Objectif 2 : faire passer via cette connexion un flux streaming : Cette étape était déterminante car elle constitue un moment critique pour la réalisation du cahier des charges. En effet l'envoi d'un flux vidéo est le point essentiel d'un robot de vidéo-conférence.
- Objectif 3 : étendre les deux premiers objectifs à plus que deux utilisateurs et donc faire les adaptations qui en résultent

Réalisation des objectifs :

Tout d'abord, nous avons réalisé les 3 objectifs décrits ci-dessus avec plus ou moins de réussites. C'est-à-dire que quelques bugs sont encore présents. Certains peuvent être expliqués car liés à notre implémentation, d'autres ne le sont pas car liés au fonctionnement interne de WebRTC.

Objectif 1 : La première connexion entre deux navigateurs

L'objectif 1 a été assez difficile à mettre en place. D'une part car nous ne connaissions pas du tout ce qu'était WebRTC et qu'il fallait donc rechercher la doc nécessaire. Cette technologie étant en cours de développement il n'existe que très peu de tutorial décent et d'explication complète à ce sujet. D'autre part toutes les documentations étant en anglais, même si ce n'est pas une excuse en école d'ingénieur, cela a tout de même contribué à la difficulté pour comprendre comment fonctionnait l'API et le protocole en lui-même.

Ensuite les exemples proposés sont faits uniquement en local et s'abstraient donc de la nécessité de mettre en place un serveur.

En effet le serveur à mettre en place est essentiel car il permet de faire circuler les paquets nécessaires à la connexion entre deux navigateurs internet.

Les paquets nécessaires sont les suivants :

- SDP (Session Description Protocol) : on créé une requête qui décrit ce que l'on désire faire (établissement d'un data channel, d'un flux streaming…) ainsi que des informations liées à notre navigateur internet ainsi que notre IP.
- Afin d'assurer la continuité de la connexion en cas de translation d'adresse par NAT et éviter le blocage par les pare-feu, on utilise le protocole ICE. Ce dernier détermine grâce à plusieurs paquets de tests les règles de filtrage du pare-feu. Ces paquets sont également échangés entre les navigateurs internet.

Après cette petite introduction, nous allons maintenant passer sur les aspects pratiquent de notre implémentation.

1°) Mise en place du serveur

Nous avons décidé de mettre en place un serveur node JS.

Concrètement ce serveur se contente de faire deux choses :

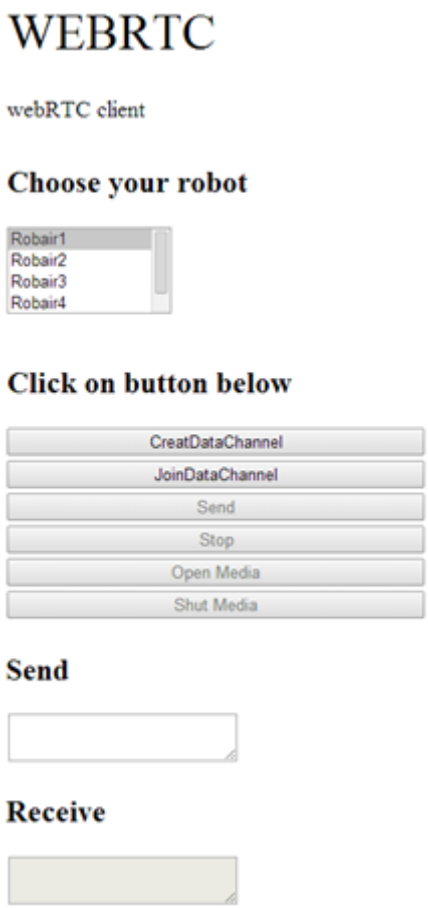
- Sauvegarder les connexions entrantes en les rangeant dans un tableau de connexion suivant une « room » prédéfini. Ainsi pour que plusieurs navigateurs internet communiquent entre eux, il est nécessaire qu'ils soient dans la même « room ». Dans le cas présent nous avons décidé pour simplifier que tous les navigateurs ne connecteraient à la même « room ».
- Faire transiter les paquets vers le navigateur qu'on lui indique. Les paquets échangés ont le format de type JSON. Ce type de donnés permet une plus grande facilité de manipulation.

2°) Côté client

On a décidé de réaliser une classe JavaScript nommée webRTC.

Cette classe permet tout d'abord d'établir la connexion avec le serveur node JS puis de permettre à l'utilisateur de créer/rejoindre une « room ». Une fois que deux utilisateurs sont connectés à la même « room » ils peuvent alors s'envoyer des messages.

Bien entendu toute l'interaction avec l'utilisateur se fait via une page web dont en voici un aperçu :



Détail du processus de connexion entre deux navigateurs :

Il existe un diagramme sur le site web http://www.w3.org/TR/2013/WD-webrtc-20130910/ qui détail très bien ce qu'il se passe pour que deux utilisateurs puissent se connecter entre eux. Il est donc conseillé d'aller jeter un œil pour plus d'approfondissement

On va donc maintenant décrire les différentes étapes qui permettent d'établir une connexion entre deux navigateurs :

- Tout d'abord l'utilisateur (Bob) appui sur « CreatDataChannel » ce qui a juste pour conséquence de s'enregistré auprès du serveur node JS en étant le premier connecté à une « room ».
- Ensuite un deuxième utilisateur (Alice) appui cette fois sur « JoinDataChannel ». Ce qui déclenche les étapes qui viennent ci-après :
- Côté Alice, une connexion webRTC est créée (grâce à un new RTCPeerConnection)

- On ajoute ensuite des paramètres à cette connexion. Typiquement un dataChannel et/ou un flux streaming.
- Ensuite on demande à recevoir notre IP et un port.
- Une fois que c'est chose faite, on envoie une offre de connexion à Bob qui transite bien entendu par le serveur nodeJS.
- A la réception de cette offre, Bob fait la même chose qu'Alice. C'est-à-dire une connexion webRTC est créée auquel on ajoute diverses paramètres. L'offre d'Alice est enregistrée dans la connexion webRTC de Bob. Une fois que l'on a récupéré port et IP, on envoie une réponse à Alice.
- A la réception, cette réponse est enregistrée dans la connexion webRTC d'Alice.
- Ensuite les deux navigateurs s'échangent des paquets du protocole ICE afin de garantir que la connexion soit persistante et traverse et les pare-feu et NAT
- Une fois que c'est fait la connexion est alors établit entre les deux navigateurs et le serveur devient alors inutile.

NB : même si nous nous sommes efforcés de faire en sorte que notre classe webRTC fonctionne quelques soient le navigateur internet, nous n'avons eu aucun bug avec uniquement Firefox. Il réside toutefois encore des problèmes sous Google Chrome que nous n'avons pas eut le temps de corriger.

Objectif 2: L'ajout du streaming

Cet objectif n'a pas posé de problème. Du moment que l'on ne désire pas couper et allumé la caméra, il est très simple d'ajouter un flux streaming avec webRTC lors de l'établissement de la connexion entre deux navigateurs.

Par contre le fait de couper et d'allumer la caméra se révèle étonnamment compliqué et c'est donc pour ça que cette option n'a pas été implémentée à ce moment-là. Ceci afin de garder une version suffisamment stable pour que les tests pour commander Rob-Air via webRTC se passe sans accros.

Objectif 3: Le multiUser

L'idée ici est de reprendre l'objectif 1 et 2 et de l'étendre afin qu'il y ait plus que deux utilisateurs qui communiquent entre eux.

Ce but ne constitue pas une très grande difficulté en soit. Il faut mettre en place des tableaux de connexion de webRTC et arriver à gérer correctement les indices afin de faire correspondre les connexions entre les différents utilisateurs.

L'ajout de la caméra avec l'option de pouvoir la couper/démarrer à volonté nécessite d'établir de nouveaux des connexions webRTC entre les différents utilisateurs. C'est la seule solution que nous avons trouvé.

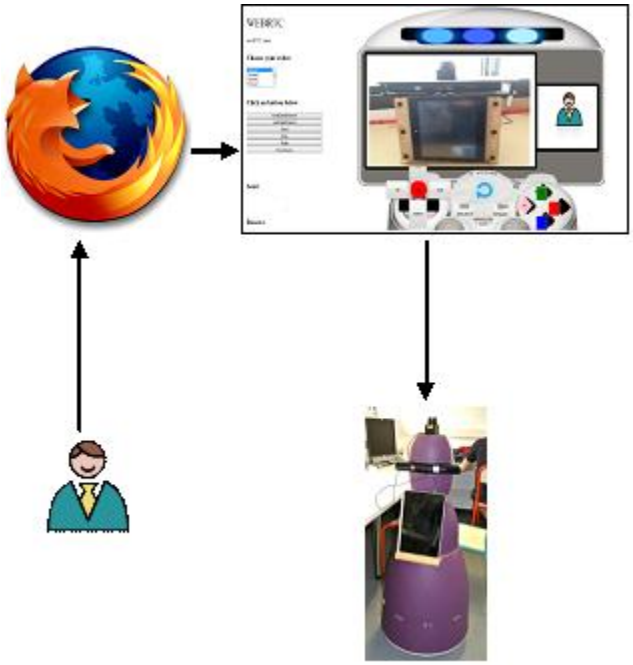
ROS :

Nous allons maintenant nous attaquer à un tout autre domaine. Tout d'abord, nous avons pris rendez-vous avec M. Amr en collaboration avec le second groupe de projet qui travaille sur Robair car le lancement du robot nécessite l'installation de ROS qui n'est pas intuitive.

Avant toutes choses, mettons au clair les différentes technologies utilisées pour la communication entre Robair et l'interface Web. Étant donné que nous reprenons globalement le code et la manière de penser des projets précédents, nous devons utiliser du PYTHON sur Robair. Cependant, notre interface web est constituée en html5, css3 et JavaScript. Or comme nous devons faire communiquer ces deux entités différentes, il à fallu trouver un format de donnée compréhensible des deux parties : JSON.

Avant tout envoie de données, il faut que la communication soit établie entre le client et le robot. Une fois celle-ci effectuée, nous pouvons envoyer les données souhaitées. Lors de l'appuie d'un bouton, une donnée est envoyée au format JSON avec en paramètre l'action à effectuer : par exemple TOP pour faire avancer le robot vers l'avant.

Du côté de Robair, le programme ROS tourne en arrière plan. Ce programme va nous permettre de contrôler Robair et de le faire parler. Comme nous l'avons vu pour webRTC, nous établissons un socket pour la connexion entre notre client, le browser et Robair. Cependant, il existe une connexion particulière entre le browser et Robair et celle-ci s'effectue encore une fois via un socket.



Le browser va donc se connecter à une partie de Robair particulière : server.py. Ce nœud va nous permettre de transcrire les informations javascript envoyées par le browser, en command python qui exécutera les instructions ROS.

Si la donnée est de type "son", notre serveur va vérifier si le paramètre correspond à un son enregistré, si c'est le cas il exécutera la commande ROS adéquat pour lancer le fichier son. Il en est de même pour toutes les commandes de mouvements.

PERSPECTIVES POSSIBLES A NOTRE DEVELOPPEMENT :

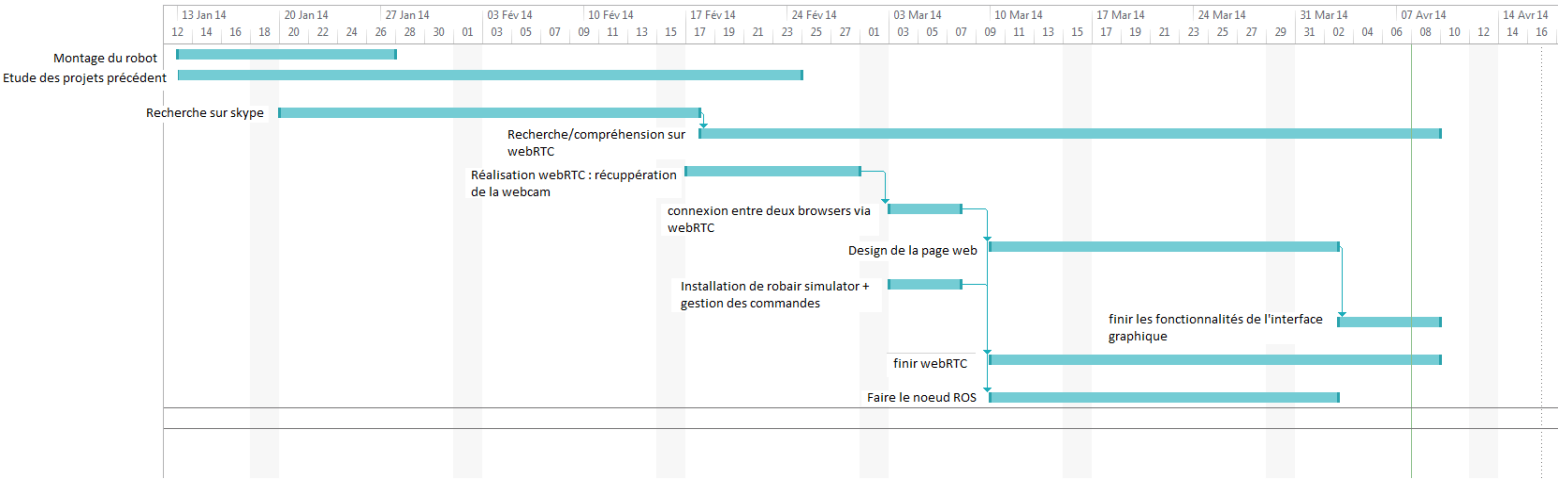
- Ajouter d'autres projets comme COQP.
- Pouvoir communiquer avec plusieurs robots.
- Contrôler le robot avec un appareil neuronal (un casque?).
- La capacité de lire des QRCODES et récupérer les informations sur l'interface graphique.
- Faire parler robair avec notre propre texte plutôt qu'avec un texte préalablement enregistré.

Gestion de projet

Mode Tâche	Nom de la tâche	Durée	Début	Fin	Prédécesseurs
	Montage du robot	11 jours	Lun 13/01/14	Lun 27/01/14	
	Etude des projets précédent	31 jours	Lun 13/01/14	Lun 24/02/14	
	Recherche sur skype	21 jours	Lun 20/01/14	Lun 17/02/14	
	Recherche/compréhension sur webRTC	37 jours	Mar 18/02/14	Mer 09/04/14	3
	Réalisation webRTC : récuppération de la webcam	10 jours	Lun 17/02/14	Ven 28/02/14	
	connexion entre deux browsers via webRTC	5 jours	Lun 03/03/14	Ven 07/03/14	5
	Design de la page web	18 jours	Lun 10/03/14	Mer 02/04/14	6
	Installation de robair simulator + gestion des commandes	5 jours	Lun 03/03/14	Ven 07/03/14	
	finir les fonctionnalités de l'interface graphique	5 jours	Jeu 03/04/14	Mer 09/04/14	7
	finir webRTC	23 jours	Lun 10/03/14	Mer 09/04/14	6
	Faire le noeud ROS	18 jours	Lun 10/03/14	Mer 02/04/14	8

En ce qui concerne notre gestion de projet, nous ne plaçons ici que notre diagramme final car celui-ci a énormément évolué au court de ce projet.

Nous avons commencé à travailler en collaboration pour comprendre le fonctionnement du robot, le monter, le faire marcher, puis nous nous sommes divisés pour mieux régner : Une partie WebRTC, une partie ROS.



Métriques :

En ce qui concerne nos métriques, nous ne comptons pas les lignes de code de ROS déjà implémenté, même si plusieurs modifications on été effectué dans les fichiers déjà existants :
- 1010 lignes de code javascript
- 271 lignes de code en html
- 186 lignes de code en css
- nombre indéterminé de lignes de code de python car plusieurs fichiers déjà existants on été modifiés en plus des fichiers créés.

CONCLUSION

Notre point de vue au niveau de ce projet est mitigé. Nous allons commencer par le point négatif qui est le manque de suivit d'une année à une autre et il a été très compliqué de faire marcher ce qui avait déjà été implémenté : du manque de documentations aux manques de commentaires dans les codes fournis en passant par le manque de spécifications sur les bibliothèques utilisées. C'est pourquoi nous avons du à nouveau réaliser une partie du projet des années précédentes pour pouvoir avancer.

Cependant, il est fort agréable de découvrir une toute nouvelle technologie encore peu utilisée (ici webRTC) et d'apprendre des langages de programmation non étudiés à POLYTECH. Cela maintient la dynamique de découverte et l'enthousiasme tout au long de la réalisation. Cette technologie offre également de nombreuses perspective tel que la réalisation d'une application Android, ou la liaison avec tant d'autres projets en cours de réalisation.