POLYTECH LILLE

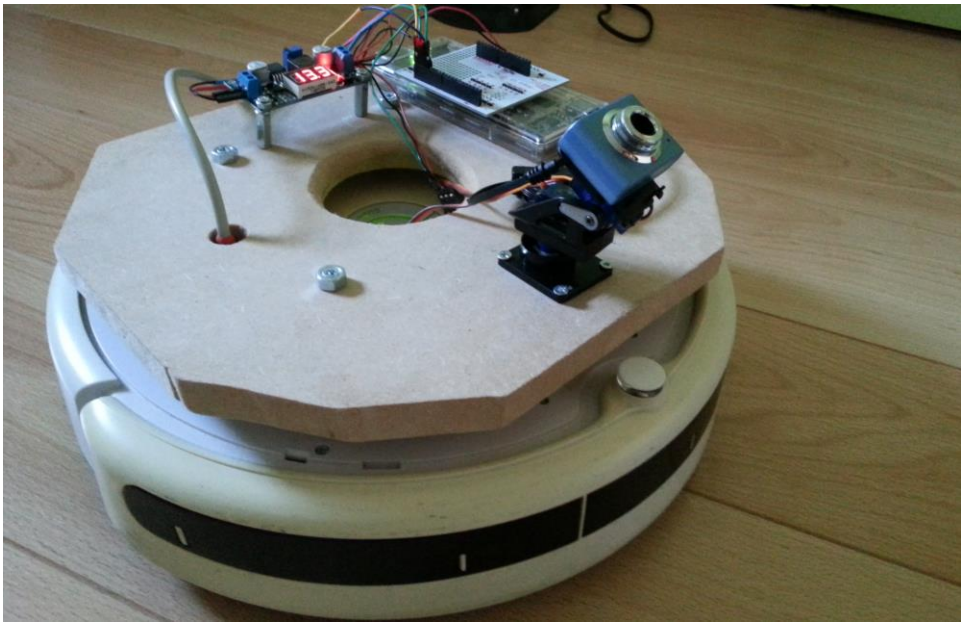
Université Lille1 Sciences et Technologies

ROBOT DE SURVEILLANCE DOMESTIQUE

Rapport de projet de fin d'étude

Cinquième année département

Informatique, Microélectronique et Automatique



Auteurs :

Sébastien DELTOMBE

Nom de la matière :

Projet de fin d'étude

A l'attention de :

Mr. Xavier REDON

Mr. Thomas VANTROYS

Année :

2014/2015

Remerciements

Je tiens tout d'abord à remercier, toute l'équipe pédagogique de Polytech'Lille et les responsables de la formation Informatique, Microélectronique et Automatique de m'avoir enseigné les bases pour réaliser ce projet dans de bonnes conditions.

Je tiens à remercier et à témoigner toute ma reconnaissance à Monsieur Xavier REDON, mon responsable de projet, pour son soutien permanent et sa disponibilité tout au long du projet.

Introduction

La robotique est au service de l'homme dans de nombreux domaines y compris dans celui de la surveillance de nos maisons. En effet de nouveaux robots voient le jour permettant de voir, d'entendre, de parler et de vous déplacer chez vous (ou ailleurs) depuis n'importe quel point de la planète. Le but du projet est la reconversion d'un robot en système de surveillance domestique à l'aide d'un système embarqué. Dans un premier temps, nous verrons les objectifs fixés lors de la rédaction du cahier des charges, puis dans un second temps, le déroulement du projet et enfin les impressions sur celui-ci.

1. Contexte

Figure 1 : *Roomba*

Le sujet concerne la reconversion d'un Roomba en robot de surveillance domestique. Un roomba est un robot aspirateur de la société IRobot. Les Roombas contiennent une interface électronique et logicielle permettant la modification du firmware et ainsi, le contrôle, la modification du comportement et la lecture des données recueillies par les capteurs du robot. Cette fonction, appelée ROI (Roomba Open Interface) par le constructeur, a été introduite afin d'encourager les programmeurs et ingénieurs en robotique à apporter leurs propres améliorations au Roomba.

Figure 2 : *Rovio*

Des entreprises commencent à commercialiser des robots de surveillance. C'est le cas de la société WoWee qui propose un robot espion dénommé « Rovio ». Celui-ci peut être contrôlé à distance par un utilisateur qui dispose d'une simple connexion internet. Il est alors possible de faire bouger le robot ainsi que la caméra embarquée afin de visualiser ce qu'il se passe dans notre habitat. Il est également possible de parler avec des interlocuteurs se trouvant sur place par le biais d'un microphone également embarqué. Ce modèle ne présente cependant pas de fonctions permettant de surveiller automatiquement une pièce afin de détecter une éventuelle intrusion.

2. Cahier des charges

L'objectif de mon projet est de transformer un vieux Roomba dont le bloc de ramassage est HS, en une sorte de Rovio, tout en ajoutant également l'aspect surveillance avec notamment la détection d'intrusion. Il y a plusieurs avantages dans la reconversion d'un Roomba pour ce projet. Tout d'abord il sera possible de réutiliser le dock de recharge ce qui permettra de rendre le robot autonome. De plus le Roomba via la plateforme ROI est pilotable directement par le biais

d’un port série. Et enfin il dispose de nombreux capteurs embarqués qui pourront être utiles pour la suite du projet.

Les principales fonctions seront les suivantes :

- Pilotage à distance du robot (par le biais d’une interface web ou application Android)
- Visualisation du flux vidéo de la webcam embarquée
- Détection d’intrus en utilisant des mécanismes de traitement d’image
- Déclenchement d’une alarme sonore
- Envoi de notification en cas de détection d’intrusion
- Programmation de ronde

3. Description du projet

Le projet s’articule autour de trois périphériques :

- Le Roomba qui sera en attente de commandes envoyées par le système embarqué.
- Le système embarqué qui permettra, par le bais du Wifi, à un utilisateur de se connecter au système. C’est également lui qui s’occupera de commander le Roomba et de gérer les traitements d’images de la webcam.
- La webcam qui permettra à l’utilisateur de visualiser les lieux. Elle servira également à la détection d’intrusion lorsque le Roomba sera en mode surveillance.

Le schéma suivant illustre comment s'effectuent les échanges entre les différents périphériques :

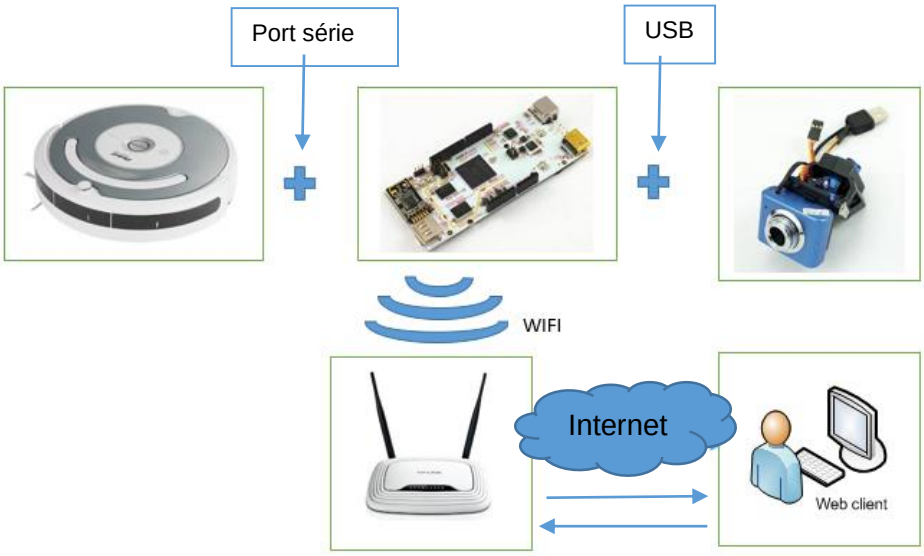


Figure 3 : *Illustration des échanges entre les différents systèmes*

4. Matériel nécessaire

Pour mener à bien ce projet, un Roomba version 520 de la société IRobot sera utilisé. Il dispose d'une batterie de 4500mAh ainsi que d'une base permettant de se recharger de façon autonome. Il a également à sa disposition de nombreux capteurs mécaniques et infrarouges. Le Roomba utilise pour se déplacer deux roues disposées de chaque côté de celui-ci.

J’ai choisi pour le système embarqué un PcDuinoV2 de la société Linksprite.

Figure 4 : *PcDuino V2*

Le PcDuino est un miniPC sous la forme d’une carte électronique embarquant des connectivités d’entrées/sorties analogiques et numériques. Il dispose notamment d’un processeur de type ARM (1ghz Cortex A8), d’1GB de mémoire vive et de 4GB de mémoire NAND embarquée. Au niveau connectique, il est possible d’utiliser un port USB, un port HDMI ainsi qu’un port Ethernet. Ces spécifications me permettront de pouvoir faire le traitement vidéo nécessaire pour la reconnaissance de visage et la détection de mouvements. De plus la version 2 dispose d’un module wifi embarqué ce qui permettra de connecter le robot au réseau domestique pour pouvoir le rendre accessible de l’extérieur.

Figure 5 : *kit webcam*

J’ai également choisi pour la caméra, un kit webcam 3 axes qui dispose d’une structure mécanique de type "Pan & Tilt". Equipé de deux servomoteurs, le kit me permettra de disposer d’une webcam mobile.

5. Etapes du projet

Afin de mener à bien le projet, différents objectifs ont été posés :

- Prise en main du Roomba
- Etude et réalisation de la partie alimentation du PcDuino
- Prise en main du kit webcam
- Réalisation du contrôle à distance (prévue pour fin octobre)
- Détection d'intrus, alarme et notification
- Programmation de rondes

a. Prise en main du roomba

L'objectif de cette partie est de créer un programme simple sur le PcDuino permettant de communiquer avec le Roomba. Principalement cela consiste à envoyer des ordres et récupérer les informations des capteurs.

b. Etude et réalisation de la partie alimentation du PcDuino

La batterie du Roomba délivre une tension qui varie en fonction de la charge de celle-ci (de 12V jusque 20V sur la base de recharge). Il est donc nécessaire de prévoir une régulation de cette tension afin d'obtenir une alimentation stable nécessaire au bon fonctionnement du PcDuino.

c. Prise en main du kit webcam

Dans cette partie il est question de piloter les servomoteurs du kit et d'installer un logiciel permettant la récupération du flux vidéo de la webcam.

d. Réalisation du contrôle à distance

Il s'agit ici de disposer pour fin octobre d'une première version du système permettant de piloter à distance le Roomba via un site web. Il faudra être capable depuis le site de :

- Faire bouger le Roomba
- visualiser le flux vidéo de la webcam
- visualiser les informations des capteurs

e. Détection d’intrus, alarme et notification

Une fois la première version du système finalisée, il faudra étudier et mettre en place un dispositif de détection d'intrusion. Avec la possibilité de déclencher une alarme sonore et d'envoyer une notification au propriétaire. Afin de pouvoir envoyer la meilleure photo possible au propriétaire il serait intéressant que le robot sache se positionner pour suivre l’éventuel intrus.

f. Programmation de rondes

Dans un dernier temps, si possible, réaliser un système de ronde permettant au Roomba d'inspecter une pièce tous les N temps.

1. Préambule

Travaillant sur un système embarqué, il a fallu mettre en place un environnement de développement spécifique. En effet, le PcDuino dispose d'une sortie HDMI et d'un port USB. Pour pouvoir travailler sur la carte il est donc nécessaire de disposer :

- d'un écran avec entrée hdmi (ou utilisation d'un adaptateur VGA/HDMI)
- d'un switch usb
- d'un clavier et d'une souris usb

L'utilisation de ces périphériques risque de poser des problèmes lorsque la carte sera alimentée directement par la batterie du robot. J'ai donc décidé de travailler sur la carte à distance via le wifi par le biais d'un logiciel de type VNC (virtual network computing). VNC est un système de visualisation et de contrôle de l'environnement de bureau d'un ordinateur distant.

Lors de mes différents tests, je me suis servi de mon téléphone en tant que routeur wifi afin de pouvoir être libre sur le choix de mon lieu travail. Je n'étais ainsi pas dépendant de l'environnement wifi sur place.

Principe de fonctionnement de l'environnement de développement :

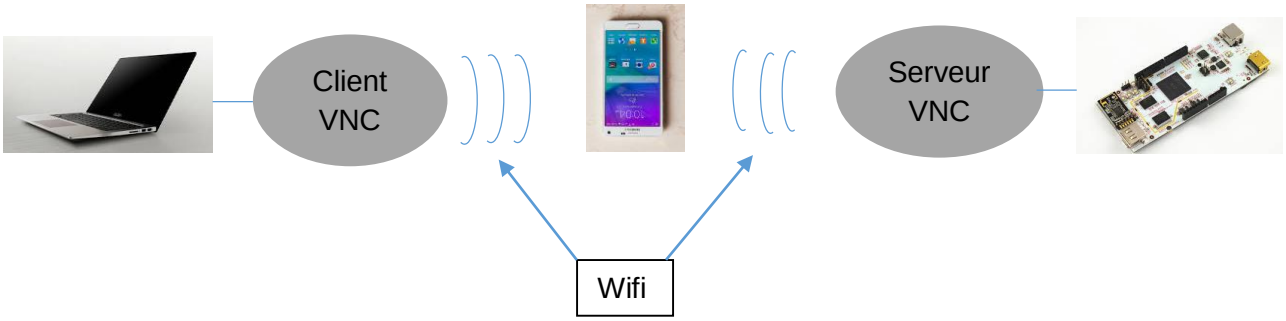


Figure 6 : *Environnement de développement*

Le PcDuino fonctionne avec un système d'exploitation léger « Lubuntu ». Cependant il peut difficilement faire fonctionner un IDE complexe style Eclipse. J'ai donc décidé d'utiliser un éditeur léger capable de supporter la coloration syntaxique de plusieurs langages ainsi que l'IntelliSense dénommé « Geany ».

Le langage C++ a été choisi afin de pouvoir accéder facilement aux fonctionnalités d'entrées sorties de la carte tout en bénéficiant des possibilités de la programmation orientée objet.

J'ai choisi de travailler sur plusieurs parties en parallèle afin de permettre une avancée globale du projet et d'éviter de me retrouver bloqué trop longtemps sur une même partie.

2. Gestion du Roomba

a. Connectique

Afin de pouvoir utiliser le ROI (Roomba Open Interface), le Roomba dispose d'une connectique MiniDin avec 8 pins dont voici la représentation :

Figure 7 : *Port série MiniDin 8 Pins*

Pour pouvoir relier le PcDuino au Roomba, il a donc fallu faire l'acquisition d'un câble ayant une fiche mâle du même type. J'ai ensuite étudié la documentation technique du Roomba afin de déterminer le rôle de chacun des pins.

Voici un diagramme représentant le rôle des différents pins :

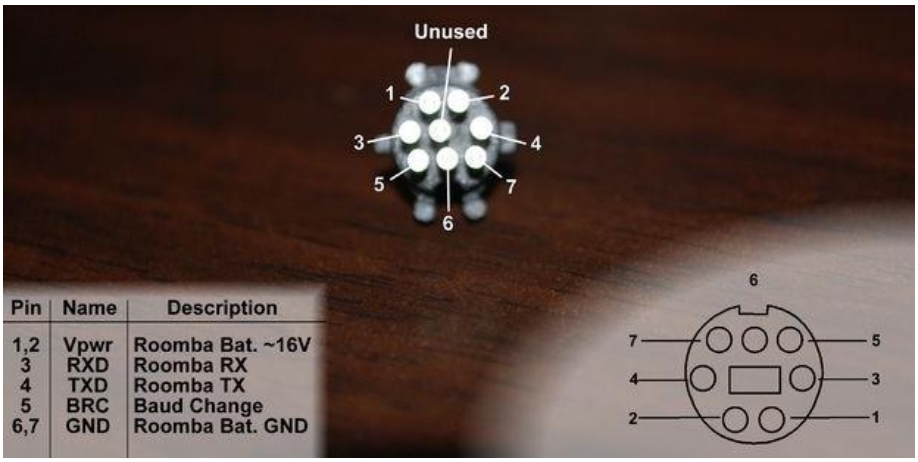


Figure 8 : *Description des Pins*

Remarques :

- ⇨ Les pins 3 et 4 correspondent au port série, ceux sont eux qui seront utilisés par le système embarqué pour piloter le Roomba.
- ⇨ Les pins 6 et 7 représentent la masse du Roomba, ils seront reliés directement sur la masse du PcDuino
- ⇨ Les pins 1 et 2 fournissent le courant de la batterie
- ⇨ Le pin 5 correspond au BRC (Baudrate change) il permet notamment de définir le Baudrate de fonctionnement pour communiquer avec le Roomba (le Baudrate correspond à la vitesse de transmission des informations, en baud, par le biais du port série)

b. Commandes de mouvements

Dans cette partie, il est question d'envoyer des commandes comprises par le Roomba. La description du protocole ROI (Roomba Open Interface) est disponible dans la documentation technique du constructeur. Il s'agit d'envoyer un certain nombre d'instructions en fonction du besoin. Lors de cette partie, j'ai conçu un petit programme en C afin de tester l'envoi basique de certaines instructions. Des commandes simples telles que le déclenchement d'un beep sonore, l'avancement du robot et l'arrêt de celui-ci ont ainsi était réalisées. Cependant dans certains cas,

il était nécessaire d'appuyer sur le bouton Power du Roomba afin de le « réveiller » et de le faire passer dans un mode pilotable.

En effet, il existe différents modes de fonctionnement interne du Roomba dont voici le principe de passage (d'un mode à l'autre):

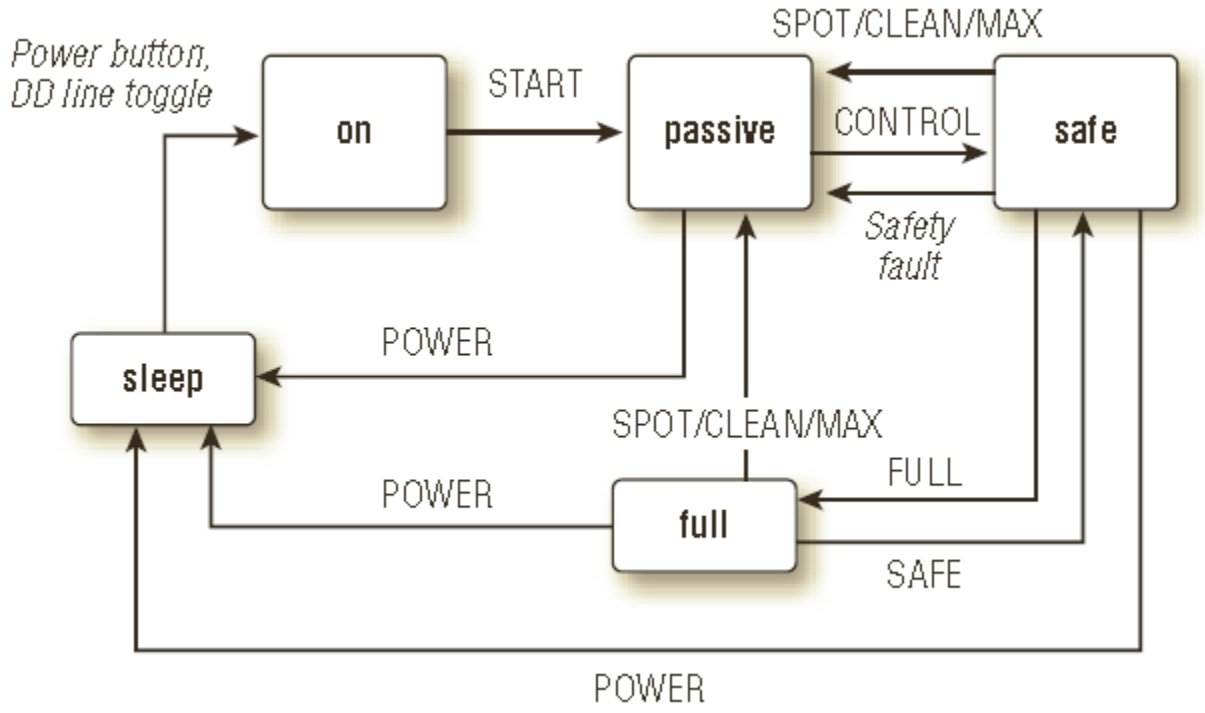


Figure 9 : *Diagramme d'état des modes de marche du robot*

Remarques :

- ⇨ Le mode « on » est un mode temporaire c'est une étape de passage lors de l'appui sur le bouton power du robot
- ⇨ Le mode « passif » est le mode de fonctionnement normal du robot lorsque celui fonctionne en tant qu'aspirateur. Il interagit à la demande lors de l'appui sur les différents boutons de commande présents sur le robot.

- ⇨ Le mode « safe » permet de contrôler le Roomba. Une fois dans ce mode, les instructions de mouvement sont prises en compte. Cependant le système interne du Roomba continue d'accéder aux informations des capteurs. Ainsi en cas de détection d'un escalier ou lorsque les roues ne sont plus aux sols, le robot arrête ce qu'il fait et passe automatiquement en mode passif.
- ⇨ Le mode « full » donne les pleins pouvoirs sur le robot. Les effets sont similaires au mode « safe » à l'exception près que le système interne ne contrôle plus si le robot est en sécurité.
- ⇨ Le mode « sleep » intervient lors de l'appui sur le bouton power alors que le Roomba est dans un autre mode et également après une période d'inactivité du robot.

Dans un premier temps pour la gestion des déplacements seul le mode « safe » a été utilisé. Cependant, le mode sleep survient automatiquement après une période d'inactivité, il est alors nécessaire d'appuyer sur le bouton power du robot pour permettre un passage dans un autre mode. Dans le cas d'un pilotage du robot à distance, il est impossible pour l'utilisateur d'appuyer sur le bouton. Une solution[1], non décrite dans la documentation du constructeur, a été trouvée consistant à envoyer un changement d'état sur le pin du BRC. Dans ce cas, l'effet est similaire à l'appui sur le bouton POWER.

Une fois que les différents tests de pilotage du robot ont été réalisés et validés, j'ai mutualisé les différentes actions possibles du robot dans un objet du programme principal écrit en C++.

1. Solution trouvée dans le livre HackingRoomba de Tod E.Kurt

c. Récupération des données capteurs du Roomba

Dans cette partie, il est question de récupérer les informations des capteurs du Roomba. Afin notamment de déterminer un obstacle, le mode en cours ou même l'état de la batterie. Le ROI (Roomba Open Interface) dispose de deux modes de récupération des données capteurs :

- Un mode synchrone

C'est un mode de type question/réponse, il s'agit de demander au Roomba les informations capteurs dont on a besoin. Celui-ci répond à la suite de la demande avec les informations demandées à l'instant T.

- Un mode asynchrone

Une fois le mode activé, le Roomba envoi périodiquement les informations des capteurs. Il est possible de spécifier les informations voulues avec la demande d'activation du mode.

Les premiers tests de réception se sont avérés problématiques. En effet, le robot ne répondait pas aux différentes sollicitations que ce soit dans un mode d'échange ou dans l'autre (synchrone ou asynchrone). Après recherche, il s'est avéré que ces problèmes étaient dus au mode de travail choisis. En effet, j'avais choisis précédemment le mode « safe » pour effectuer les tests de contrôle du Roomba. Or ce mode ne permet pas la lecture des données capteurs car ceux-ci sont toujours lus par le système interne du robot afin de détecter si celui-ci est en danger. L'utilisation du mode « full » a permis d'éviter le problème, le système interne ne cherchant plus à lire les informations des capteurs.

Les deux modes d'échanges ont ensuite été testés. Le mode asynchrone présentait de nombreux avantages sur le papier. On active le mode une fois, puis il suffit de réceptionner les demandes. Cependant durant les tests, il s'est avéré que l'activation du mode asynchrone empêche, par la suite, toute demande de contrôle du robot. Il était alors impossible de bouger le robot une fois le mode activé. Le mode synchrone quant à lui, permet la récupération des données sans perturbation des commandes de mouvement. C'est donc naturellement le mode synchrone qui

été choisi pour la suite du projet. Il a fallu cependant implémenter l'envoi périodique des demandes d'informations.

Par la suite la réception et le décodage des données n'ont pas occasionné de problèmes particuliers.

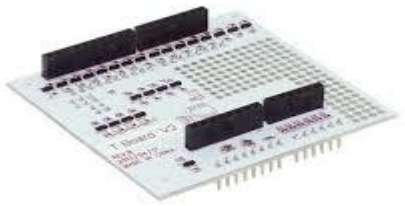


Figure 10 : *Shield d'adaptation*

Une incompatibilité des entrées/sorties a été détectée entre les entrées/sorties du PcDuino qui sont à 3,3V et ceux du Roomba qui fonctionnent en 5V. Même si cela semble marcher, un shield de type TBoard V2 d'adaptation 3V vers 5V a été commandé afin d'éviter d'endommager le PcDuino. Un Shield est une carte qui peut se fixer sur les entrées/sorties d'un système embarqué afin de lui ajouter de nouvelles fonctionnalités.

3. Gestion de l'alimentation du PcDuino

Pour que le robot puisse être autonome, il faut que le système embarqué soit directement alimenté par la batterie du Roomba. Cependant, celle-ci dispose d'une tension qui peut varier de 12V à 20V. Or le PcDuino ne supporte que 5V pour son alimentation. Il a donc été nécessaire d'étudier différentes solutions afin de pouvoir réguler la tension de la batterie pour la rendre utilisable par le système embarqué.

a. Régulateur de tension linéaire

Figure 11 : *Régulateur linéaire*

Dans un premier temps, un régulateur de tension linéaire a été envisagé afin de diminuer la tension de la batterie. Ce type de régulateur dissipe le surplus de tension de façon thermique. Des tests ont été réalisés avec un régulateur L78S05 qui permet de fournir en sortie 5V en tension et 2A ampère en courant. Ce type de régulateur supporte jusque 35V en entrée, supérieur au 20V max de la batterie du robot. Les tests d'alimentation n'ont pas été concluants. En effet, non seulement le PcDuino n'arrive pas à démarrer (seul les leds sont allumés) mais en plus le régulateur chauffe énormément même avec un dissipateur.

b. Régulateur de tension à découpage

Figure 12 : *Régulateur à découpage*

Suite aux échecs rencontrés avec le régulateur précédent et après étude, j'ai choisi d'orienter mon choix sur un autre type de régulateur dit « à découpage ». En plus de proposer un meilleur rendement ce type de régulateur chauffe beaucoup moins qu'un régulateur linéaire. Les régulateurs à découpage utilisent le rapport cyclique entre l'état passant et l'état bloqué du transistor de découpage pour réguler la tension et le courant de sortie.

J'ai choisi pour le projet un module régulateur avec pour base le composant LM2596 qui supporte une tension d'entrée comprise entre 4 et 40V. La tension de sortie est quant à elle ajustable via un potentiomètre (de 1.25V à 37V). L'avantage de ce type de module est qu'il est prêt à l'emploi et bon marché (inférieur à 5 euros).

Les tests, avec ce type de régulateur, ont été très vite concluants, en effet le PcDuino a démarré dès la première tentative et il a fonctionné sans problème pendant plus d'une heure. De plus la

chaleur dissipée est beaucoup moins importante que pour le régulateur linéaire. Il est possible notamment de toucher le composant durant son fonctionnement sans se brûler.

4. Prise en main du kit webcam

La webcam a pour objectif, à terme, de pouvoir réaliser divers type de détection que ce soit de visage ou de mouvement. Pour que celle-ci puisse être dans les meilleures conditions possibles il s'est vite avéré important qu'elle soit mobile.
En effet, le robot se trouvant au sol l'angle d'inclinaison de la caméra devient un paramètre variable en fonction de la distance de l'individu.
Voici un exemple représentant les différents angles d'inclinaison qui seront nécessaires:

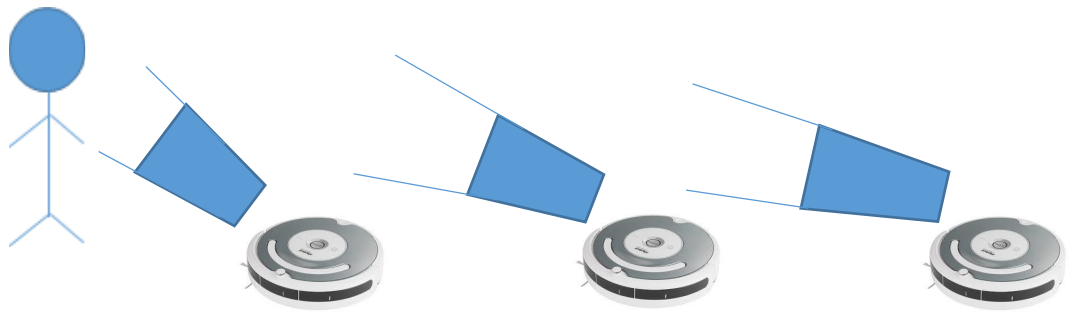


Figure 13 : *Adaptation de l'angle d'inclinaison de la webcam du robot*

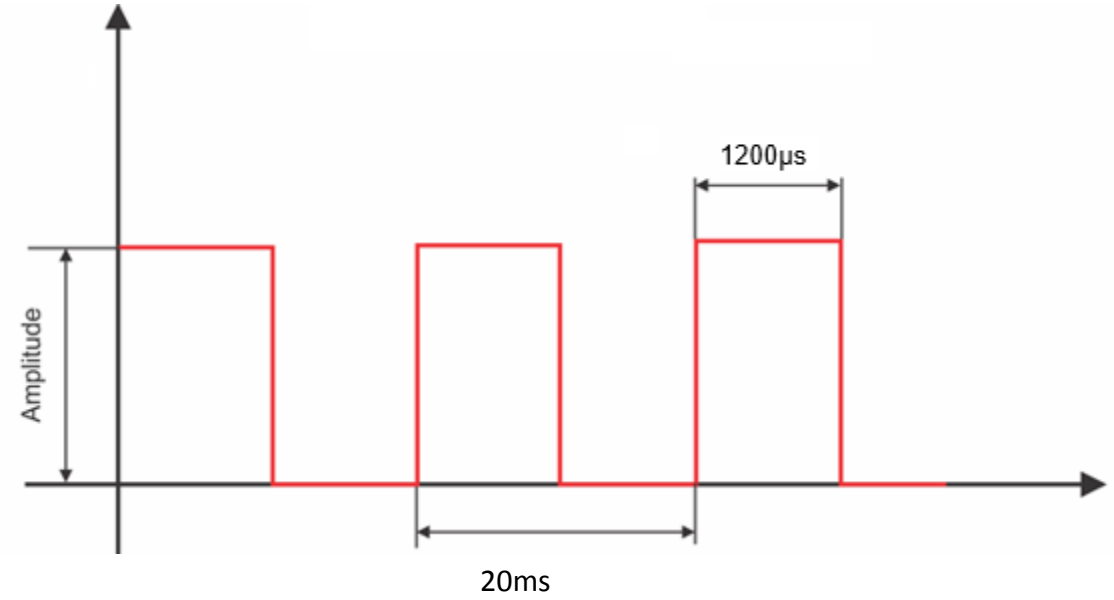
On voit bien avec cette représentation que plus la personne se trouve loin du robot au plus la caméra devra être incliné vers le bas et inversement plus le robot se rapproche de la personne plus la caméra devra être incliné vers le haut.

J'ai donc choisi de travailler avec un kit webcam monté sur plusieurs axes pilotable par le biais de servomoteurs. Ainsi le système embarqué pourra ajuster l'angle de la caméra en fonction de la position de l'individu détecté. La webcam a également été choisie en fonction de sa compatibilité avec le système embarqué. L'alimentation des servomoteurs pourra être fournie par le même régulateur que celui utilisé pour alimenter le PcDuino.

a. Pilotage des servomoteurs

Une fois le kit monté, j'ai réalisé un programme de test en C afin de tester le pilotage des servomoteurs. Le principe utilisé est connu sous le nom de Pulse Width Modulation (PWM) ou Modulation de Largeur d'Impulsions; il consiste à moyenner un signal logique (0 et 1) de fréquence fixe mais de rapport cyclique contrôlé. Dans le cas des servomoteurs utilisés, il faut envoyer une impulsion dans un cycle de 20ms. La durée de l'impulsion permet de faire varier l'angle de rotation du servomoteur. Il a donc fallu dans un premier temps déterminé les différentes largeurs d'impulsion permettant de réaliser des rotations de 0 à 180°. Après différents tests, il s'est avéré que 90° correspond à une largeur d'impulsion de 1200 µs. Cette valeur a servi de référence pour l'élaboration d'un algorithme permettant de déterminer les autres angles par une simple règle de trois. Un objet en C++ a ensuite était implémenté dans le programme principal permettant la gestion des différents servomoteurs.

Exemple pour une rotation à 90° :



Lors des essais, je me suis aperçu que l'alimentation des deux servomoteurs rendaient instable l'alimentation du système embarqué. En effet, suite à une utilisation prolongée des servomoteurs, le PcDuino avait tendance à s'éteindre sans raison apparente. Suite à ce problème, j'ai décidé de débrancher le servomoteur permettant de pivoter la caméra de gauche à droite.
Ce servomoteur n'était pas important pour la suite du projet étant donné que la caméra pouvait déjà bouger de gauche à droite par le biais du robot.

b. Gestion du flux vidéo

Afin de capturer le flux vidéo de la caméra pour pouvoir le rendre accessible depuis un client web, j'ai choisi d'utiliser le programme mjpeg-streamer. En effet, celui-ci permet de pouvoir afficher le flux vidéo dans une page HTML par le simple biais d'une balise image en spécifiant le port de l'application. Le logiciel a fonctionné sans difficulté sur le système embarqué. Cependant afin de bénéficier d'une vidéo fluide sans délai apparent, il a été nécessaire d'imposer un certain nombre d'image par seconde (framerate). Dans mon cas la meilleure fluidité a été constatée aux environs de 6 images par seconde.

5. Réalisation de la prise en main à distance

Afin de réaliser la prise en main à distance et pour des raisons de portabilité, j'ai choisi de réaliser une application web. Le gros avantage de ce type d'application est qu'elle sera utilisable aussi bien depuis le navigateur d'un ordinateur que depuis celui d'un smartphone. L'application doit permettre de piloter le robot tout en affichant les informations des capteurs. La communication bidirectionnelle pour une application web n'est réalisable que par le biais de WebSocket disponible depuis la nouvelle norme du HTML5. J'ai donc choisi d'utiliser cette technologie afin de pouvoir rafraichir les informations des capteurs sans intervention du client. Le HTML5 est un standard en devenir, cependant seul les navigateurs récents sont capables actuellement de le supporter.

Principe de fonctionnement :

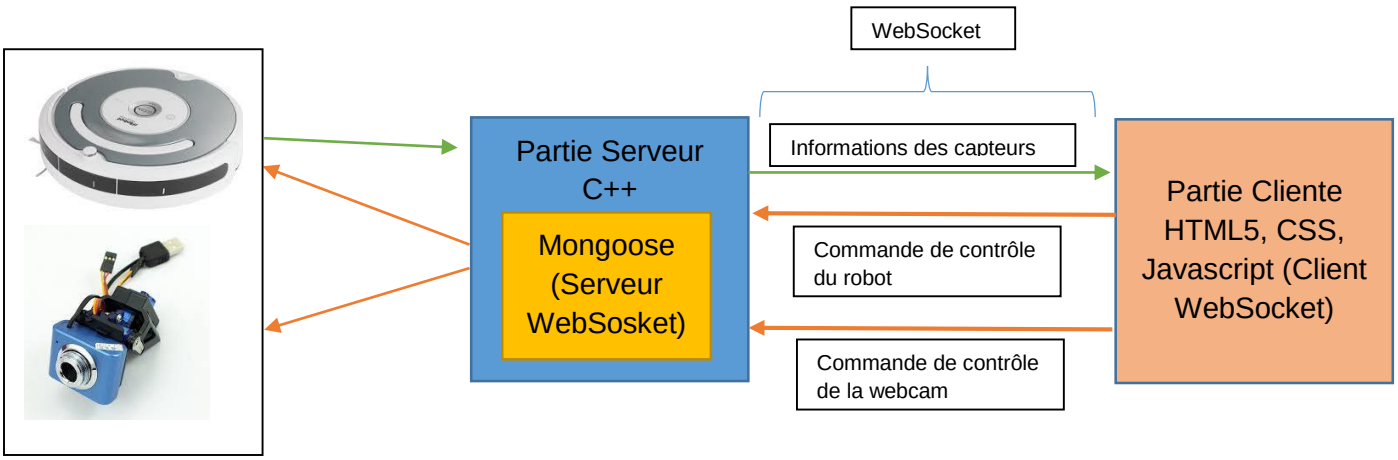


Figure 14 : *Echanges client/serveur de l'application Web*

a. Coté Serveur

Afin de réaliser la partie serveur j'ai décidé d'utiliser une librairie dénommé « Mongoose ». Cette librairie écrite en C/C++ a l'avantage de proposer un serveur web léger proposant de nombreuses fonctionnalités comme le https (pour une navigation encrypté) ou même la réalisation de WebSocket. Un objet a été réalisé dans le programme principal afin de permettre l'encapsulation de cette librairie ainsi que la gestion des interactions avec les autres objets précédemment crée.

b. Coté Client

Figure 15 : *logo HMTL5*

La partie cliente a été réalisée principalement en HTML 5, CSS et Javascript. L'appel aux différents fichiers a été géré directement dans Mongoose. Le HTML ainsi que les fichiers CSS permettent de d'écrire l'affichage et les feuilles de style utilisés pour la page principale. Le Javascript quant à lui permet d'ajouter de l'interaction coté client et notamment la connectivité WebSocket. Le Framework Javascript JQuery a été utilisé afin de gagner du temps dans la réalisation de certaines interactions.

Voici une représentation de l'application depuis un navigateur :

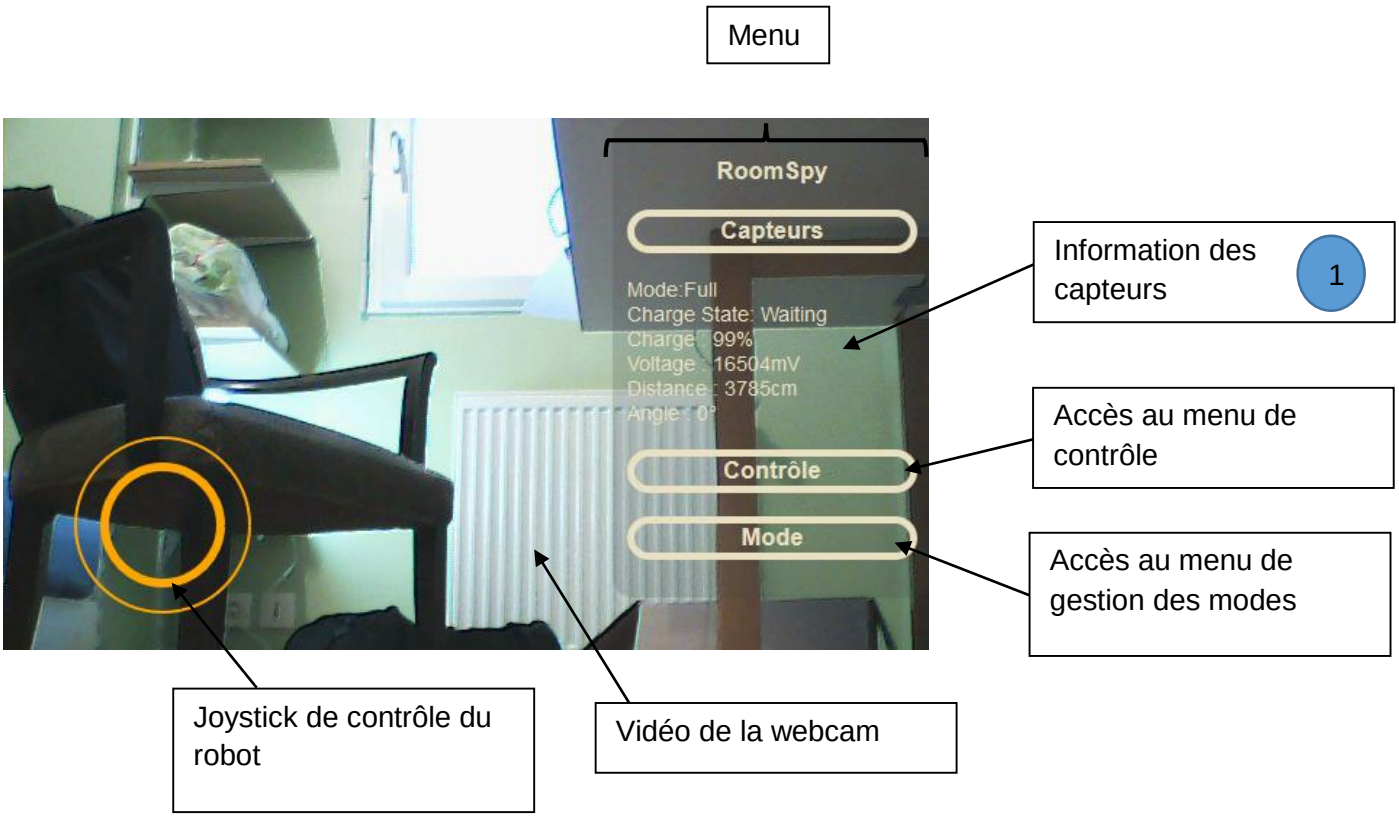


Figure 16 : *Interface web*

L'interface propose sur la partie droite un joystick permettant de piloter le robot selon différentes directions :

-  Position haut le robot avance
-  Position bas le robot recule
-  Position centre arrêt du robot
-  Position droite le robot pivote à droite (en restant sur place)
-  Position gauche le robot pivote à gauche (en restant sur place)

Pour cette fonctionnalité, j'ai utilisé une librairie Javascript dénommé « virtualjoystick ». Cette librairie permet notamment, en plus de permettre l'intégration du joystick, de gérer les écrans tactiles.

Sur la partie gauche se trouve un menu permettant d'accéder au différentes fonctionnalités :

- ⇨ 1. Visualisation des données capteurs (voir figure 16)
- ⇨ 2. Passage en mode plein écran (Menu contrôle)
- ⇨ 3. Déclenchement d'une alarme (Menu contrôle)
- ⇨ 4. Retour à la base (Menu contrôle)
- ⇨ 5. Passage en mode Passif pour autoriser le rechargement (Menu contrôle)
- ⇨ 6. Contrôle de la webcam par slider (Menu contrôle)
- ⇨ 7. Changement de mode (Menu mode)

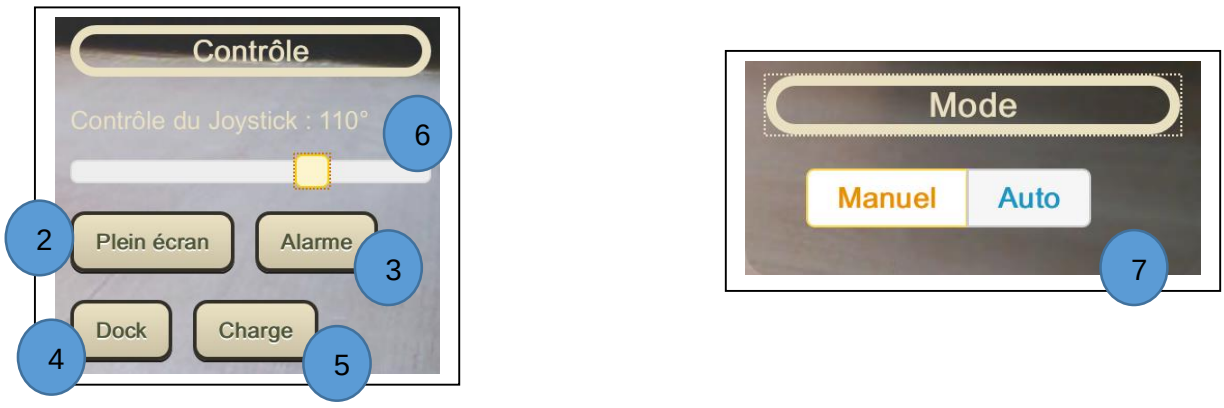


Figure 17 : *Menus*

6. Détection d'intrus, alarme et notification

Dans cette partie il est question d'utiliser la webcam du robot afin de pouvoir faire de la détection d'intrusion. J'ai choisi la librairie OpenCV afin de réaliser le traitement d'image.
Comme pour les étapes précédentes, j'ai tout d'abord réalisé des programmes de test en C permettant notamment de tester unitairement différents modes de détection.

Figure 18 :
Logo OpenCV

a. Détection de visage

OpenCV utilise une méthode de détection d'objet dite « Méthode de Viola et Jones ». Le principe est le suivant, plutôt que de travailler sur des valeurs de pixels, la méthode propose de travailler sur des caractéristiques. Une caractéristique est une représentation synthétique et informative. Voici des exemples de caractéristiques :

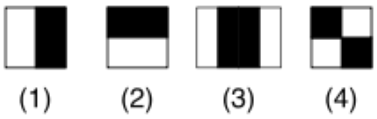


Figure 19 : *Exemple de caractéristiques*

Un algorithme dénommé « image intégrale » permet de calculer rapidement les caractéristiques d'une image. Une fenêtre de détection de petite taille est utilisée pour calculer les caractéristiques à toutes les positions et à toutes les échelles. La méthode utilise ensuite une suite de classifier afin de déterminer en fonction des caractéristiques si l'image contient l'objet recherché. Un classifier est un arbre de décision qui a été entrainé a détecté certains ensembles de caractéristiques en fonction d'un apprentissage. En effet, il est nécessaire d'entrainer un classifier avec un certain nombre d'images d'exemple dans lesquelles se trouve l'objet. Un premier programme a été écrit en C à l'aide d'OpenCV en utilisant ce principe. La librairie met à la disposition de l'utilisateur des fichiers xml permettant l'apprentissage d'un classifier en fonction du besoin. On trouve ainsi des fichiers permettant de faire de la détection de visage, des yeux voir même de certaines parties du corps (main, buste).

Les tests ont été réalisés avec succès l'algorithme détecte les visages comme le montre la photo suivante :

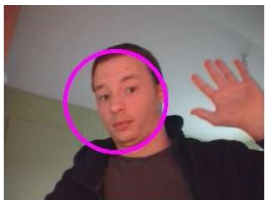


Figure 20 : *Exemple de détection*

b. Détection de mouvements

Pour la détection de mouvements l'algorithme est plus simple il consiste à comparer pixel par pixel une image à l'instant T avec l'image à l'instant T-1. Il est tout de même nécessaire de passer par quelques étapes de prétraitement afin de faciliter la comparaison. Ce traitement est assez long, afin d'améliorer les performances la taille de l'image été réduite dans ce cas.

c. Détection de buste

Lors des différents tests j'ai essayé la détection de buste. Tout comme la détection de visage celle-ci utilise la même méthode de fonctionnement. Seul le fichier d'apprentissage change. L'avantage de ce type de détection et qu'il prend en compte les épaules de l'individu en plus du visage. La fenêtre de détection est donc agrandi par rapport à une détection de visage simple ce qui entraine un temps d'exécution moins important. Suite à cette observation j'ai décidé d'abandonner la détection de visage au profit d'une détection de buste qui donne un résultat similaire mais avec un temps de réaction bien plus rapide.

d. Interaction avec le programme principal

Au vue des ressources nécessaires pour faire le traitement, j'ai décidé de créer un programme C++ dédié à la détection. Ce programme est donc dissocié du programme principal qui permet de contrôler le Roomba et la webcam par le biais du serveur web. Néanmoins afin d'éviter de mettre place un nouveau moyen de communication j'ai décidé de réutiliser la connectivité WebSocket pour la communication entre les deux programmes. L'avantage est de pouvoir réutiliser la même interface de contrôle quel que soit le cas d'utilisation (interface web ou programme de détection). La librairie Mongoose ne permettant pas de faire un client WebSocket,

j'ai utilisé une autre librairie dénommé « POCO » qui permet de gérer cette partie dans le programme de détection.
Voici le principe de fonctionnement :

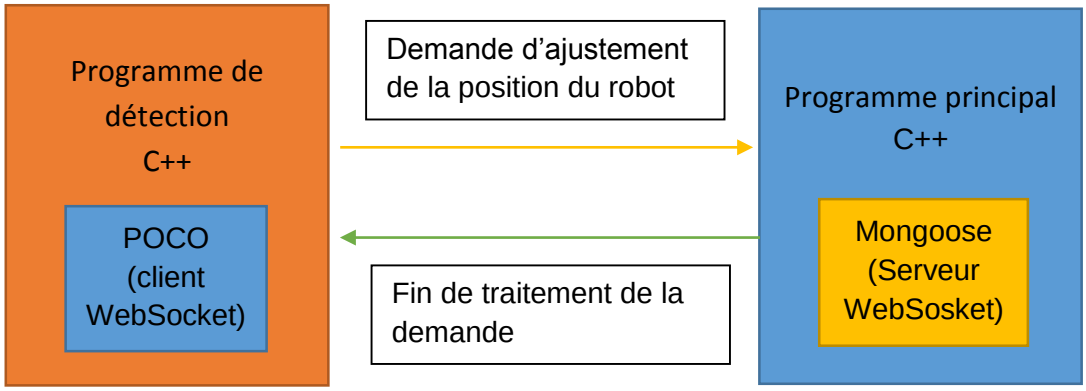


Figure 21 : *Echange entre le programme de détection et le programme principal*

Remarque :
Afin de passer du mode manuel au mode automatique, le programme principal arrête le flux vidéo en cours du programme mjpeg_streamer. Puis il lance le programme de détection. De la même façon, en cas de retour au mode manuel, le programme arrête le programme de détection pour ensuite relancer le programme mjpeg_streamer. Le passage d'un mode à l'autre se fait par le biais des boutons disponibles dans le menu mode de l'interface web.

e. Alarme et Notifications

Afin de pouvoir effrayer l'intrus détecté, j'ai mis en place, par le biais du ROI, une alarme en utilisant le hautparleur intégré dans le Roomba. Une alarme visuelle est également visible, le bouton power clignote en rouge lors d'une détection de buste. Des notifications par mails ont également été mises en place afin de conserver une trace des photos. De ce fait, même si le robot viendrait à être détérioré (par l'intrus par exemple), il serait toujours possible de l'identifier (si celui-ci n'est pas masqué). Un mail est envoyé à chaque alerte afin de prévenir l'utilisateur au plus vite avec l'image qui a provoqué la détection en pièce jointe. Les applications « ssmtp » et « mutt » ont été utilisés pour configurer l'accès et l'envoi sur la boite mail de destination.

7. Programmation de rondes

Faute de temps, je n'ai malheureusement pas pu aborder cette partie. Un système de détection pour un couple forme/couleur (ex : un rond rouge) avait été envisagé afin de pouvoir faire réaliser au robot un circuit bien déterminer dans une pièce.

8. Bilan

Ci-dessous, un état des lieux des actions effectuées :

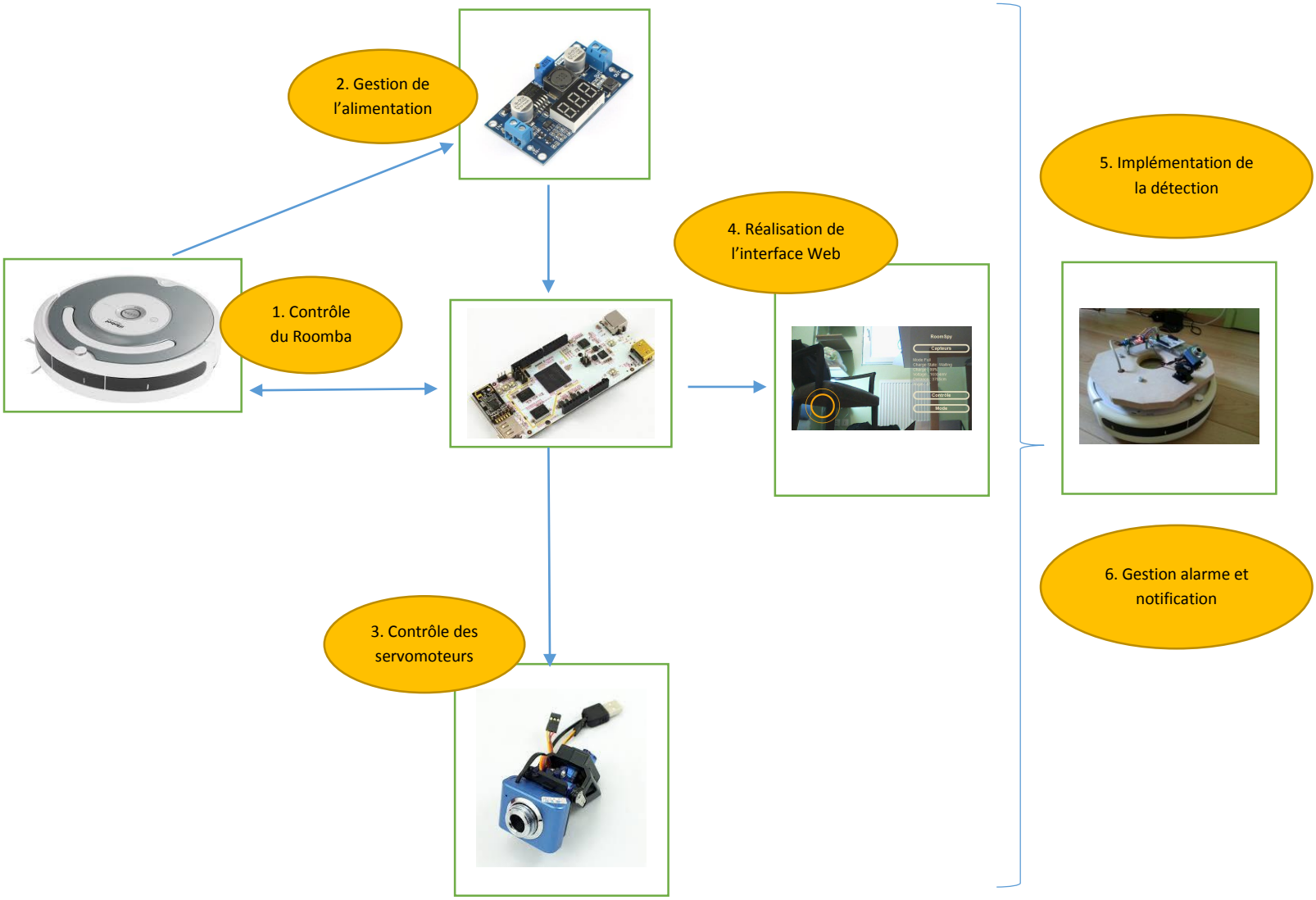


Figure 22 : *Présentation des différentes étapes de ce projet*

1. Difficultés rencontrées

J'ai rencontré deux principaux problèmes lors de ce projet.

Tout d'abord la communication avec le Roomba. Car même si les commandes étaient décrites dans la documentation constructeur du ROI, il n'a pas toujours été évident de faire obéir le robot. Lors des premiers tests, il s'est avéré que le Roomba répondait de façon aléatoire aux différentes commandes. En effet, dans un premier temps le robot ne répondait en général qu'une fois sur deux. J'ai également observé des réactions non désirées de la part du robot (beep d'erreur ou déclenchement d'un mouvement rapide sans arrêt). Il s'est avéré, en lisant la documentation, que certaines commandes nécessitent un laps de temps entre deux appels. J'ai donc décidé d'utiliser un délai entre l'appel de chaque instruction afin d'éviter de perturber le système interne du robot. J'ai également pour la même raison choisis de diminuer la vitesse du port série, je suis ainsi passé de 115200 bauds à 19200 bauds.
Suite à ces modifications les tests se sont avérés concluants le Roomba répondait de façon cohérente aux différentes sollicitations.

Un autre problème à résoudre fut les problèmes de performance du traitement d'image sur le système embarqué. Car malgré tout, même si celui-ci est relativement puissant comparé aux autres cartes du marché, le processeur est loin d'égalé les performances des processeurs de nos ordinateurs actuels. En effet, le temps de traitement des images s'avère très long. Le processeur du système embarqué est utilisé à 100% de sa capacité lors du traitement d'image. Un décalage de la vidéo de plus d'une dizaine de seconde s'est produit lors des premiers tests. Deux actions ont alors été mises en place afin d'améliorer la vitesse du traitement.

- ⇨ Diminution de l'image : plus l'image est grande plus l'algorithme passe du temps à la traiter. J'ai donc décidé de passer d'une résolution d'image de 640x480 pixels à 320x240pixels

⇨ Modification du paramétrage du classifier en fonction de la détection précédente. Il est possible de configurer la taille de la fenêtre utilisé pour la détection de l'objet à rechercher. J'ai décidé d'implémenter un algorithme qui utilise la taille de la fenêtre de détection précédente, si elle existe, afin d'adapter la fenêtre de la détection en cours.

Voici le principe de fonctionnement :

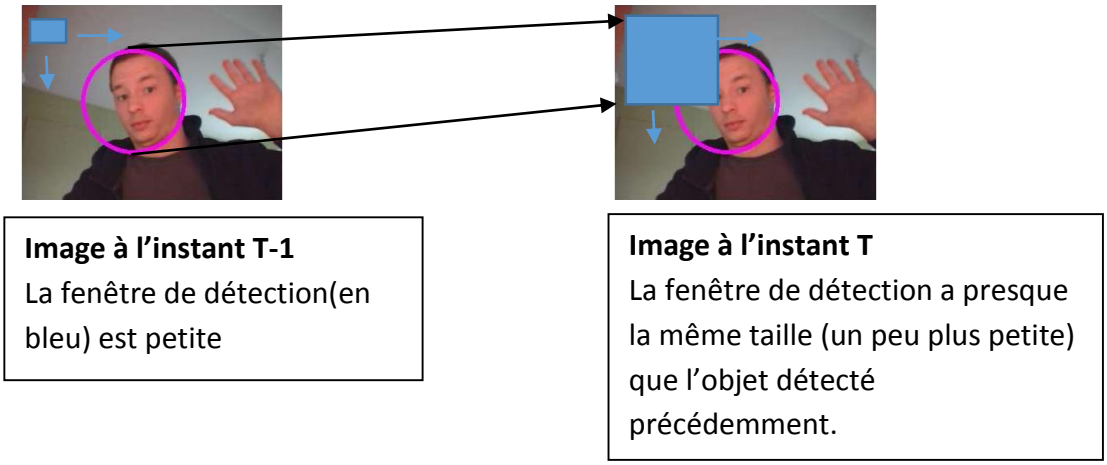


Figure 23 : *Explication de l'adaptation de la fenêtre de détection*

Le traitement est plus long dans le premier cas car on va chercher à détecter des visages plus petits alors que ce n'est pas nécessaire. L'image d'après, le traitement est beaucoup plus rapide car la fenêtre de détection a été adaptée en fonction du besoin. Finalement plus on se rapproche de la caméra plus l'algorithme est rapide.

Mes modifications ont permis d'améliorer les temps de réaction. La vidéo est décalée uniquement de 0.5 à 4 secondes en fonction de la distance de l'objet à détecter.

2. Bilan personnel

J'ai trouvé ce projet très enrichissant tout d'abord d'un point de vue technique, il m'a permis de mettre à profit mes compétences acquises tout au long de mon cursus et d'en acquérir de nouvelles.

En effet, j'ai pu lors de ce projet faire un peu d'électronique, du système embarqué, du développement et du traitement d'image. J'ai également utilisé de nombreuses librairies et outils que je ne connaissais pas à l'origine. J'ai pu mettre en pratique la méthodologie d'apprentissage acquise lors de ma formation. C'est important de pouvoir à tout moment s'adapter à un nouvel environnement, l'étudier, apprendre et analyser le traitement afin de pouvoir obtenir ce que l'on veut d'un système. C'est une des qualités que doit avoir un ingénieur : l'adaptabilité.

Ensuite d'un point de vue gestion de projet, travaillant seul sur le projet j'ai dû gérer la priorisation de mes taches. De même, il est important de savoir organiser son temps afin de pouvoir gérer dans certains cas plusieurs tâches de façon simultanée. En effet, il arrive que l'on soit en attente de matériel ou que l'on rencontre un problème technique. Mais ces blocages n'empêchent pas la réalisation d'autres tâches secondaires, qui peuvent contribuer à l'avancer globale du projet.

A l'exception d'une fonctionnalité, le cahier des charges a été respecté correctement dans les temps voulus. La partie programmation de rondes n'a pas pu être traitée faute de temps mais aurait certainement pu être réalisé dans le cas d'une durée normale du PFE.

D'autres idées ont également été envisagées afin d'améliorer le système. Comme l'intégration d'un dispositif microphone avec hautparleur permettant d'interagir avec des personnes se trouvant à proximité du robot. Cela permettrait également de mettre en place un dispositif de commande vocale. Il serait intéressent également par la suite de mettre en place un système d'authentification afin de sécuriser l'accès à l'application web.

Conclusion

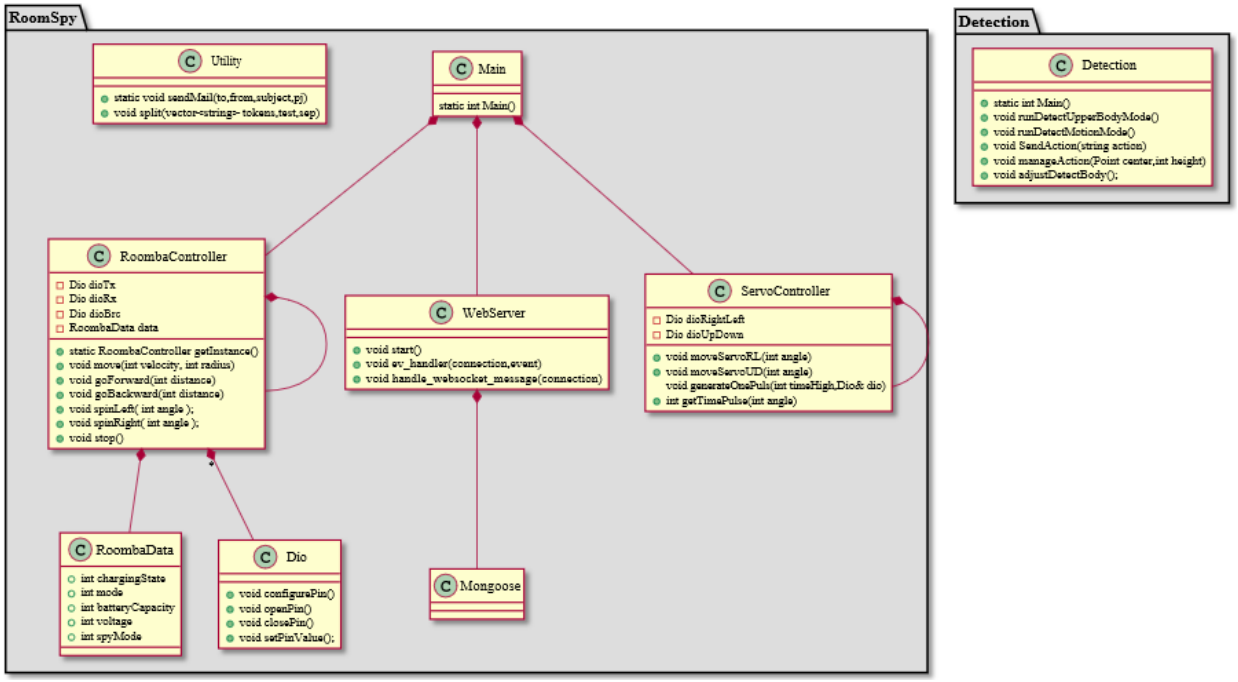
J'ai effectué ce projet de fin d'étude dans le cadre de ma cinquième année dans la spécialité Informatique Micro-électronique et Automatique à Polytech'Lille.

Le projet consistait à créer un système de surveillance domestique à base d'un robot aspirateur existant. Le robot a été agrémenté d'un système embarqué disposant du wifi ainsi que d'une webcam mobile. Il est désormais capable d'être piloté de l'extérieur ou même de bouger en toute autonomie afin de détecter un intrus. Les détections engendrent une alarme sonore ainsi que des notifications par mail.

J'ai pu, lors de ces onze semaines de projet, mettre en pratique mes connaissances acquises durant la formation et d'en développer de nouvelles. Cette expérience m'a permis d'anticiper les différents problèmes qui se sont dressés devant moi. Les différentes capacités que j'ai dû mettre en œuvre me permettront d'être plus efficace dans la gestion de mes futurs projets.

Annexes

Annexe 1 : Diagramme de classes



Légende :

: Librairie

: Classe

: Agrégat (signifie que la classe où le symbole est collé possède un objet de la classe vers laquelle il pointe)

Remarque :

Le diagramme n'est pas exhaustif, mais il permet de donner un point de vue global de l'application et explicite le rôle de chacune des classes. A noter, l'utilisation du design pattern singleton à deux reprises concernant RoombaController et ServoController. L'utilisation de ce pattern est particulièrement utile dans ce cas. Car il est important qu'une seule instance de ces

objets soient créé à un instant T afin d'éviter l'envoi de commande en simultanée sur les mêmes entrées/sorties. De plus cela facilite leurs utilisations lors du traitement du Handler WebSocket qui est une méthode statique déclenché dans un autre thread que le thread principal.

Annexe 2 : Description d'une commande du ROI (Roomba Open Interface)

Drive **Opcode: 137** **Data Bytes: 4**

This command controls Roomba's drive wheels. It takes four data bytes, interpreted as two 16-bit signed values using two's complement. The first two bytes specify the average velocity of the drive wheels in millimeters per second (mm/s), with the high byte being sent first. The next two bytes specify the radius in millimeters at which Roomba will turn. The longer radii make Roomba drive straighter, while the shorter radii make Roomba turn more. The radius is measured from the center of the turning circle to the center of Roomba. A Drive command with a positive velocity and a positive radius makes Roomba drive forward while turning toward the left. A negative radius makes Roomba turn toward the right. Special cases for the radius make Roomba turn in place or drive straight, as specified below. A negative velocity makes Roomba drive backward.

NOTE:

Internal and environmental restrictions may prevent Roomba from accurately carrying out some drive commands. For example, it may not be possible for Roomba to drive at full speed in an arc with a large radius of curvature.

- Serial sequence: [137] [Velocity high byte] [Velocity low byte] [Radius high byte] [Radius low byte]
- Available in modes: Safe or Full
- Changes mode to: No Change
- Velocity (-500 – 500 mm/s)
- Radius (-2000 – 2000 mm)

  Special cases:

  Straight = 32768 or 32767 = hex 8000 or 7FFF

  Turn in place clockwise = -1

  Turn in place counter-clockwise = 1

La figure ci-dessus montre la commande à envoyer au Roomba pour faire un mouvement. La première instruction 137 explicite le type de commande. Elle indique au Roomba que l'on va le faire bouger. Il est ensuite nécessaire d'envoyer 4 octets pour la vélocité et le radius.
La vélocité détermine la vitesse moyenne des roues en millimètre par seconde. Si la vélocité est positive le Roomba avance et inversement si la vélocité est négative il recule. De même plus elle est élevé plus le Roomba avancera rapidement et inversement.
Le radius permet quant à lui de faire tourner le Roomba par rapport au centre de celui-ci en millimètre.