Rapport de Stage

Autonomie pour Augustin

Rapport de stage 2A ECM

***

Par

Justin CANO

Élève Ingénieur à Centrale Marseille

jcano@centrale-marseille.fr

Destinataires :

Franck ANJEAUX, Président Directeur Général d'Axyn Robotique

Ronald AZNAVOURIAN, Ingénieur de Recherche à l'Institut Fresnel

Sébastien GUENNEAU, Directeur de Recherche à l'Institut Fresnel

Alain KILIDJIAN, Professeur Agrégé à l'École Centrale de Marseille

Muriel ROCHE, Maître de Conférences à l'École Centrale Marseille

Page laissée intentionnellement vide pour la présentation.

Table des matières

Édition du lundi 8 août 2016

Avant-Propos

1. Synopsis du Stage

Employeur : CNRS

Lieu : Institut Fresnel, 52 Avenue Escadrille Normandie Niemen, 13013 Marseille

Durée : Du 30 mai au 29 Juillet 2016

Description : L'objet du stage était de donner de l'autonomie à un robot du type Ubbo développé par Axyn Robotique. Le robot étant la propriété de l'équipe Epsilon appartenant à l'institut Fresnel de Marseille il n'a pas vocation a être autonome *a priori*. L'introduction de ce robot au sein de l'Institut Fresnel (une unité mixte de recherche entre Aix-Marseille Université, le CNRS et Centrale Marseille qui n'a pas de compétence en robotique) vient d'une rencontre fortuite des membres de Epsilon avec Monsieur Anjeaux durant la fête des sciences à l'automne 2015.

2. Équipe d'accueil

L'équipe qui m'a accueilli est l'équipe Epsilon, travaillant principalement sur les méta-matériaux et l'électromagnétisme macroscopique. Ces matériaux présentent de nombreux et nouveaux[1] intérêts physiques tels que la biréfringence[2] et la réfraction à indice négatif. Les méta-matériaux peuvent en outre être utilisés dans les génies civil et maritime : en effet, ces derniers ont la propriété de dévier les ondes sismiques et les vagues. En transposant les connaissances de l'optique ondulatoire dans les domaines connexes aux méta-matériaux, les membres de l'équipe aspirent à découvrir des propriétés permettant à terme de créer une « cape d'invisibilité » sismique. Ce procédé est révolutionnaire car il n'utilise pas la dissipation d'énergie cinétique canoniquement employée par les constructeurs, mais du détournement d'onde par anisotropie artificielle des sols[3].

Cette équipe utilise des procédés mathématiques parmi lesquels figurent : la théorie spectrale, l'analyse complexe et les méthodes asymptotiques. Toutefois, l'équipe reste fortement liée au domaine de l'ingénierie (du bâtiment notamment où se trouvent les principales applications des méta-matériaux) ainsi qu'au monde de l'industrie avec lequel elle collabore.

3. Remerciements

Je tiens à remercier toutes les personnes ayant permis de concrétiser ce stage, en particulier :

- Sébastien GUENNEAU & Ronald AZNAVOURIAN, membres de l'équipe Epsilon de Fresnel et co-tuteurs de mon stage. Je tiens à les remercier spécialement pour leur accueil chaleureux, pour le fait d'avoir initié les démarches qui ont abouti à ce stage et

1 La première publication dans ce domaine d'étude date de 2006

2 Anisotropie artificielle

3 obtenue avec des trous ou des colonnes de béton judicieusement répartis

pour les multiples discussions extrêmement constructives que nous avons eu ensemble.

- Muriel ROCHE, de l'équipe PHYTI de Fresnel et Maître de Conférences à Centrale Marseille, pour son soutien technique en traitement d'image. Je tiens également à la remercier d'avoir accepté de faire partie de mon jury de soutenance.
- Stefan ENOCH, Directeur de l'institut, pour avoir rendu ce stage possible.
- Nelly BARDET, Responsable des Ressources Humaines, pour ses démarches administratives qui ont été nécessaires à la poursuite du stage.
- Frédéric GALLAND, Antoine ROUEFF et Muriel ROCHE, de l'équipe PHYTI et enseignants bien connus à Centrale Marseille, pour leur soutien technique en traitement d'images
- Krishma GUPTA André DIATTA, Bogdan UNGUREANU, Younès ACHAOUI pour leur accueil au sein de la salle commune que nous avons partagé.
- Alain KILIDJIAN, pour son soutien comme à l'accoutumée dans ce type d'entreprise et ses conseils avisés.
- Franck ANJEAUX, Aurélien BISSOTI & Guillaume AGUIRAUD, travaillant à Axyn Robotique pour toute l'attention qu'ils ont porté à mes travaux ainsi qu'à leur support technique.

I. Appropriation du problème

1. *Le Robot Augustin*

Nommé ainsi en l'honneur du célèbre physicien français Augustin Fresnel (1788-1827) ayant donné son nom à l'institut éponyme, il s'agit du robot sur lequel j'ai travaillé durant mon stage. Acheté par l'Institut à la société Axyn Robotique[4], il s'agit d'un robot du modèle Ubbo. Le robot est un robot open source destiné à un marché *tech-friendly* et qui a vocation à évoluer vers un modèle un peu plus « clef en main », visant un public plus large.[5] Sa principale application, bien qu'on puisse lui affecter aisément plusieurs tâches, est la **visioconférence en mouvement**.

Ce concept est né du constat que lors des réunions de travail impliquant une visioconférence à distance pour un intervenant, il arrive souvent que ce dernier soit un peu délaissé au profit des personnes physiquement présentes. Or, la nature statique de l'écran et le fait de ne pas pouvoir mouvoir la caméra vers la personne qui parle est une des facteurs les plus prépondérant dans la logique d'oubli du vidéo-conférencier. Avec Ubbo, il est donc possible de mouvoir une tablette connectée à Skype ® (ou tout autre logiciel) avec l'aide d'une base roulante et ainsi pouvoir suivre les interlocuteurs. On peut aisément imaginer d'autres applications telles que la surveillance d'enfants en bas âge, la visite à distance de personnes malades etc...

2. *Architecture du robot Augustin*

a) Mécanique & structure

Le Robot Augustin est composé d'un montage différentiel à quatre roues striées. Ceci lui permet de translater selon **x** [6], de tourner sur lui même[7] et de translater selon **y** [8]. Sa structure est assez simple : il s'agit d'une base roulante pourvue de quatre capteurs à lumière rouge SICK sur les points cardinaux surmontée d'un mât. Ce mât sert à supporter la tablette servant à la fois de siège de l'application Skype et de poste de commande de la base roulante. Toutefois, la présence de cette tablette surélevée justifie la présence des capteurs anti-collision précédemment évoqués : en effet, il est impossible de voir des obstacles se trouvant à une hauteur modérée.

4 Startup basée à Meyreuil, dans les Bouches-du-Rhône

5 Notamment pour de l'aide à l'enseignement en écoles, collèges et lycées ou à visée thérapeutique en hôpitaux et maisons de retraite.

6 Utilisation simple des roues motrices avec les mêmes consignes pour les quatre moteurs

7 Utilisation des moteurs « avant » en vitesse de rotation opposées avec les moteurs « arrière »

8 Il suffit de donner une consigne opposée aux moteurs situés à gauche du robot par rapport à ceux de droite. Normalement, le robot devrait rester sur place ou glisser en faisant forcer ses deux moteurs, mais les stries des roues permettent un déplacement longitudinal de translation (à la manière des crabes).

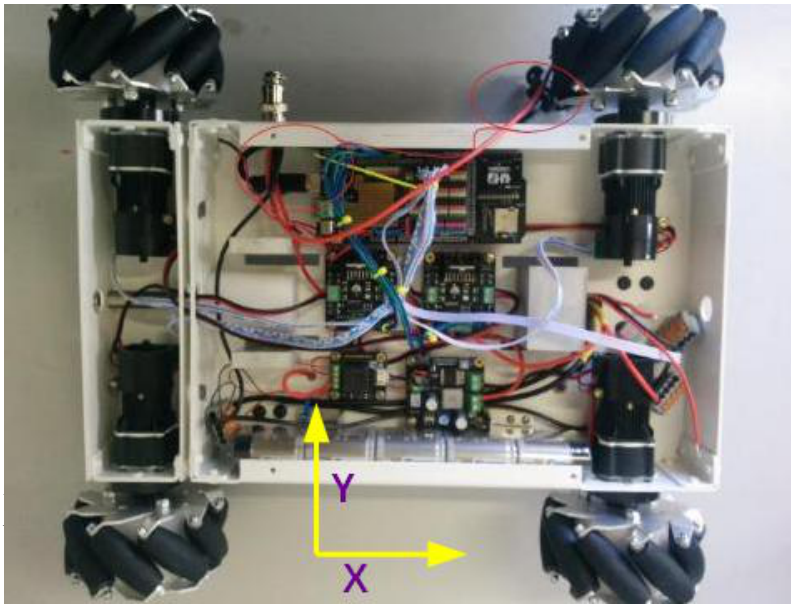


Illustration 1: La structure du robot Ubbo Maker (Source : documentation Axyn)

b)l Architecture de la commande

L'architecture de la commande des Ubbo Maker peut être vue en quatre étages :

- Un étage de haut niveau : interface avec l'utilisateur et application permettant de commander via un site de partage les mouvements de la base roulante. L'utilisateur ayant la main sur la base roulante est celui qui n'est pas présent physiquement via son ordinateur.
  - Il est composé d'une tablette servant à abriter toutes les applications nécessaires au bon fonctionnement du robot : logiciel de visioconférence, logiciel de commande de la base roulante, logiciels de protocoles Wi-Fi & Bluetooth, caméra…
- Un étage de bas niveau : ce dernier permet de faire appliquer les commandes transmises par la tablette sur le robot, de faire l'acquisition des signaux des capteurs etc...
  - Cet étage est exclusivement composé d'une carte Arduino Mega ® qui est open source.
  - Cette dernière permet de commander via ses variateurs (PWM) les quatre moteurs,
  - Toutefois, un servomoteur doté d'un protocole particulier (mais facilement commandable en incluant une bibliothèque Arduino spécifique) est présent sur la tête du robot. Il permet une rotation de la tablette autour de l'axe **y**, permettant à l'utilisateur plus d'aisance.
- Un étage d'adaptation, composé de cartes pré-actionneuses et de filtrage (tous des modules compatibles[9] Arduino : pont en H, driver pour servomoteurs, driver de

9 Appelés « Shields » .

capteurs filtrant...) . I

- Les capteurs et les actionneurs qui font le lien entre l'électronique et la physique de l'environnement. Sont présents :
  - Les quatre moteurs, équipés de capteurs à effet Hall pour l'odométrie et l'asservissement en vitesse des roues, indispensable au vu de la variabilité des contraintes mécaniques sur ces derniers.
  - Le servomoteur qui est au sommet du mât servant à orienter la tablette tactile
  - Les quatre capteurs à lumière rouge, permettant de procéder à l'évitement d'obstacle

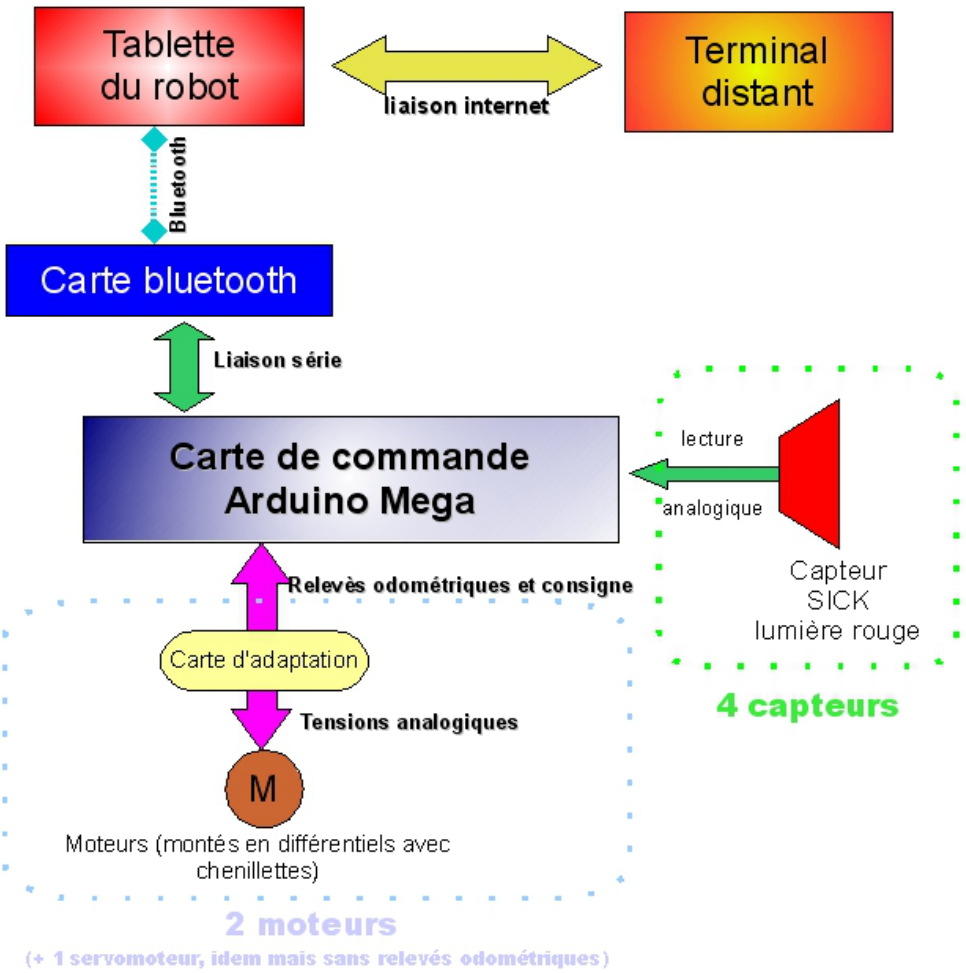


Illustration 3: Schéma électrique simplifié du robot (Source : Justin CANO)

c) Architecture logicielle

Deux niveaux de code existent :

- ✔ Un code dit « de bas niveau » implémenté sur l'Arduino Mega ®, il permet de gérer les actionneurs (moteurs et servomoteurs) et comporte les méthodes d'évitement d'obstacle (simple lecture digitale des capteurs). Ce code gère également une partie de la communication Bluetooth afin de piloter le robot via la tablette. Le programme est codé en langage Arduino ®[10], très proche du langage C++
- ✔ Un code dit de haut niveau codé en Java pour fonctionner sur le système d'exploitation Android ®. Développé dans le logiciel de développement Android Studio ®, il permet de gérer l'interface utilisateur (communication, pilotage manuel, à distance etc.) il est implémenté sous la forme d'une application gratuite disponible sur le Google App Store ®.

3. Problèmes de conception constatés

a) Risques principaux

Tout d'abord, j'ai procédé à une petite analyse de risques en vue de critiquer le modèle Ubbo dans l'optique de proposer des améliorations en vue de son industrialisation.

Intitulé	Probabilité	Gravité	Criticité
Décharge d'une des batteries[11]	Élevée	Inconfort mais gravité modérée	Élevée
Chocs mécaniques[12]	Moyenne	Moyenne	Moyenne
Chute avec basculement[13]	Élevée (selon la topologie du terrain)	Élevée (destruction de la tablette)	Très Élevée
Erreur de commande[14]		Faible	Modérée
Perte de connexion	Forte	Aucune (le robot est prévu pour s'arrêter en l'absence de renouvellement de consigne)	Aucune

10 Références : http://arduino.cc , Commencer avec Arduino, M. Banzi, ETSF. Mémo Arduino pour débuter, (http://wiki.centrale-marseille.fr/fablab) J.Cano, P.Salles, FabLab Marseille, 2014, *rév.* 2016.

11 Le robot Augustin présente trois batteries distinctes : une pour les moteurs, une pour la tablette ainsi qu'une pour les capteurs et la partie commande de bas niveau (carte Arduino et périphériques).

12 J'ai remarqué que dans certains cas le robot ne détectait pas un obstacle, je détaillerai ces constatations dans un paragraphe ultérieur, en proposant quelques solutions pour contrer le phénomène.

13 Je me situais au troisième étage de l'institut Fresnel, et donc en présence d'escaliers. Sans contre-mesure pour une chute dans des escaliers ou un obstacle dit « lacunaire ». Il est impossible pour Augustin de détecter ce dernier, donc le robot risque le basculement, qui peut s'avérer fatal pour la tablette le commandant.

14 Si le robot a pivoté sur lui même, ce n'est pas le cas de la fenêtre d'exécution de l'application sur la tablette : il est difficile pour l'utilisateur néophyte de distinguer l'avant de l'arrière. Ceci menant bien sûr à quelques erreurs bénignes de pilotage.

b) Accessibilité de la batterie et chargeur

J'ai vite constaté que l'accès à la prise chargeant la batterie n'était pas aisé. En effet, l'embase permettant le raccordement des batterie se trouve sur la paroi interne du robot. Donc, en aucun cas elle n'est saillante, alors que dans le cas contraire cela permettrait une accessibilité et une commodité d'utilisation plus évidente.

c) Système d'évitement d'obstacles

Le système d'évitement d'obstacles présente plusieurs défauts inhérents à l'emploi d'une seule famille de capteurs situés sur le même plan. A mon sens, un système d'évitement de collision complet doit être mis en place afin de rendre plus confortable la visioconférence. En effet, vu l'angle de la caméra[15], il est plus que nécessaire d'éviter toute collision. Avec le système actuel une collision est synonyme d'intervention d'un opérateur pour dégager le robot de sa mauvaise posture… Nous pouvons citer quelques points restant à améliorer :

- La technologie des capteurs utilisés : capteurs SICK à lumière rouge
  - La lumière rouge est souvent absorbée par les objets de couleur sombre ou terne et ces derniers ne sont pas détectés. Le retour du capteur étant basé sur l'amplitude des ondes lumineuses rouges réfléchies
  - L'angle d'incidence du faisceau lumineux. Si l'angle entre le robot et l'obstacle s'éloigne de 90° (et ce, quel que soit le sens), alors, si l'on considère que l'objet est un miroir on constate que le faisceau lumineux réfléchi ne viendra pas sur le capteur. Cela provoque une absence de détection selon l'angle d'incidence.
  - La portée de dispositif est aléatoire et petite. Petite car le coefficient d'absorbance n'est jamais celui d'un miroir parfait. En effet, seule une infime partie de l'onde lumineuse est réfléchie et donc une fraction encore plus petite est détectée (le capteur couvre une surface plus petite que le rayon qu'il émet). Aléatoire, car cela dépend du matériau de l'obstacle, de l'angle d'incidence et de la vitesse à laquelle arrive le robot (temps de réaction cumulé des capteurs, comparateurs et microcontrôleurs pour arrêter les moteurs).
  - La comparaison intégrée des capteurs ne permet pas de retourner des valeurs analogiques des mesures de distance. En effet, l'information est binaire : il y a un obstacle ou la voie est libre.
  - Donc, peut-on déterminer la distance entre le robot et l'obstacle ? Pour les raisons évoquées ci-dessus l'observateur ne peut pas avoir accès à cette information à l'aide des capteurs SICK.
- L'absence de contrôle dans les virages
  - Le code de bas niveau, implémenté dans la carte Arduino Mega située à l'intérieur de la base roulante, n'a pas de fonction pour éviter les obstacles lorsque le robot est en rotation.

15 Qui se trouve à l'extrémité d'un mât et qui ne peut pas aisément permettre de visualiser ce qu'il peut advenir à l'embase du robot.

- Le robot ayant une forme rectangulaire, lorsqu'il tourne, il couvre une zone assez étendue. Donc sans capteur opérant, un obstacle peut très bien se trouver dans ladite zone et percuter le robot.

- Les capteurs ne couvrent pas tous les obstacles
  - Les capteurs sont exclusivement situés dans un plan parallèle au sol et à mi-hauteur de la base roulante. Par conséquent :
    - Il est impossible pour Augustin de détecter un obstacle discontinu telle une table : les capteurs ne détectent que des obstacles situés environ à 5cm du sol, donc ne détectent pas la table[16] . Cela dit, le mât du robot, portant la tablette peut se retrouver percuté par la table, ce qui peut s'avérer grave suivant la vitesse d'impact (risque de chute de la tablette).
    - On ne peut pas détecter les obstacles lacunaires tels que les escaliers. Ceci est le danger principal, connaissant la topologie du terrain de l'Institut.
    - Des angles morts inhérents aux capteurs, notamment dûs au fait de la forme de rectangle allongé de la base roulante rendant ainsi impossible la détection de certains obstacles « ponctuels ». Par exemple, un pied de table ne peut être détecté que si il se trouve bien centré sur le capteur, dans le cas contraire c'est la collision.

- Un problème, résultant pour partie des points précités est l'algorithme d'évitement en lui même
  - Ce dernier ne permet que d'interdire des mouvements vers des obstacles, il ne peut débloquer le robot de manière automatisée.
  - Il n'inclut pas de retour pour l'utilisateur. Ce dernier ne sait pas où est l'obstacle ni même pour quelle raison le mouvement lui est interdit : défaillance électrique, mécanique, obstacle, problème logiciel, de connexion... Tant de possibilités peuvent interdire un mouvement. De fait, il serait souhaitable d'informer l'opérateur de la raison de l'arrêt inopiné d'Augustin.

d) Téléguidage : nécessité d'une expertise

Le point précédent, par ricochet, implique que pour être téléguidé, le robot Ubbo nécessite une certaine « expertise » dans le sens où l'utilisateur [17] doit connaître la mécanique du robot pour pouvoir s'en servir de manière adéquate. Le système d'évitement serait une bonne chose mais il n'y pas que ce point qui pose problème. J'ai personnellement mis du temps à m'habituer à l'orientation avant/arrière du robot en le pilotant manuellement : en effet, l'interface de l'application n'a pas de boutons distinctifs ni un schéma (dynamique ou statique) représentant le robot pour faire comprendre quel mouvement l'utilisateur s'apprête à ordonner. Un didacticiel serait salutaire pour cela, il permettrait d'avoir une aide graphique et intuitive pour manipuler le robot à distance. Et ce didacticiel devrait être présent sur le site de contrôle à distance du robot et devrait être proposé lorsqu'on crée un compte client. Ceci

16 *Exit* les pieds de ladite table.

17 L'utilisateur distant le pilotant, n'a pas accès directement au robot. Il est même possible qu'il n'en ai jamais vu auparavant (exemple : des chercheurs d'un pays étrangers contactant ceux de Fresnel n'ont pas le modèle à disposition.).

rendrait plus facile la prise en main des robots Ubbo.

e) Mouvement de rotation circulaire

Autre point de mes observations, mais c'est plus une mise en garde qu'un vrai défaut : le robot ne décrit pas un cercle centré lorsqu'on lui demande de tourner sur lui même mais une ellipse légèrement excentrée. Ceci paraît être un détail mais pourrait être un frein à un système d'évitement d'obstacles.

4. Solutions envisagées

a) Définition du cahier des charges

Après avoir consulté mes tuteurs de stages MM. Guenneau et Aznavourian (Fresnel) puis M.Kilidjian (ECM), j'ai décidé de compléter les objectifs suivants :

i. Optimisation du Système d'évitement
   (a) Étude du besoin précis
   (b) Étude des solutions envisageables
   (c) Chiffrage et proposition de solutions
ii. Autonomie pour Augustin via le Tracking des Conférenciers [18]
   (a) Étude du besoin
   (b) Tests sur des cas simples sur ordinateur
   (c) Implémentation dans un environnement réel sur tablette incorporée au robot

b) Compétences requises

Intitulé	État au début du projet
Maîtrise du langage de bas niveau Arduino	Étant déjà formé et même formateur dans ce domaine, je disposais d'un atout indéniable.
Maîtrise du langage de haut niveau Java	J'ai suivi les cours de tronc commun puis d'options dans ce domaine.
Savoir créer des applications Android pour systèmes embarqués	Je ne m'étais jamais confronté à cette difficulté mais des documentations claires existaient.
Reconnaissance de formes & de couleurs	Des spécialistes de l'équipe Phyti de Fresnel, plus les cours du parcours S8 SISN dispensés par ces mêmes professeurs m'ont été grandement utiles dans ce domaine.

18 Le tracking est le fait de suivre un objet donné en traitement d'image. Mon idée étant de faire suivre par Augustin les vidéo-conférenciers automatiquement afin de donner un confort optimal à l'utilisateur à distance. Les ressources pour moi furent facilement accessibles puisque l'Institut regorge d'experts en la matière.

c) Échéancier

Afin de remplir le cahier des charges avec un maximum d'efficacité, j'ai dressé un échéancier global des tâches qui m'incombaient

1. Prise en main du robot *(semaine 1)*
   A) Lecture du support client *(1 jour)*
   B) Prise de contact avec les développeur du robot *(2 jours)*
   C) Premiers tests *(2 jours)*
2. Établissement du cahier des charges (Semaine 1 et 2)
   A) Rapport oral aux tuteurs de l'Institut Fresnel (*pendant les premiers tests)*
   B) Visite du tuteur ECM (*immédiatement après le rapport avec mes tuteurs Fresnel)*
   C) Établissement puis validation du cahier des charges *(2 jours)*
3. Installation des IDE (Logiciels de développement du code source) *(semaine 2)*
   A) Eclipse (*1 jour, avec mise à niveau java)*
   B) Arduino (*en parallèle avec l'étape précédente)*
   C) Android Studio (*2 jours : je ne suis pas familier de cette IDE, de plus beaucoup de packages, bibliothèques et dépendances sont à installer avant de développer son application)*
4. Étude sur l'inférence de bas niveau (*Semaine 2 & 3)*
   A) Compréhension du code-source (*Deux jours, en comptant des questions aux ingénieurs d'Axyn)*
   B) Étude sur les capteurs anti-collision (*Un jour, en attendant les caractéristiques techniques de la part des ingénieurs d'Axyn)*
   C) Recommandations techniques à propos du système d'évitement (*Trois jours)*
5. Programmation d'un algorithme de tracking trivial *(Semaines 3 et 4)*
   A) Recherche d'une bibliothèque de Reconnaissance des Formes (RdF) (*Un jour)*
   B) Recherche bibliographique sur ce type d'algorithme *(Deux jours, avec des tests rapides pour voir si l'algorithme correspond bien au problème posé)*
   C) Implémentation en Java *(Deux jours, en configurant les bibliothèques et en faisant attention à la mémoire vive allouée, ce qui peut s'avérer être un problème une fois l'algorithme implémenté sur la tablette)*
   D) Implémentation dans l'application Androïd *(Quatre jours, incluant formation au développement Androïd sur Internet, compréhension du code, codage, tests, configuration de l'IDE...)*
6. Programmation d'un algorithme de tracking plus proche du besoin *(Cinquième semaine)*

   A) Recherche d'une bibliothèque de RdF (*déjà fait si la bibliothèque choisie est assez générale)*
   B) Recherche bibliographique sur ce type d'algorithme *(un jour)*
   C) Implémentation en Java *(deux jours)*
   D) Implémentation dans l'application (*deux jours)*
7. Rédaction des compte-rendus *(Sixième semaine)*
8. Rédaction d'un quitus technique pour poursuivre l'amélioration d'Augustin (*Septième semaine)*

II.Déroulé du plan d'action

1. Rectification du système d'évitement

L'inférence de bas niveau permettant l'évitement implémentée sur l'Arduino Mega de la base roulante est basée sur une détection booléenne autorisant ou non un déplacement. Je donne ici l'algorithme général en pseudo-code :[19]

- **booléen evitement()**
  - **si (déplacement == VersL'avant ET proximité(capteurAvant))**
    - // Cela signifie que l'on désire aller de l'avant alors qu'il existe un obstacle
    - alors **déplacement = arrêt**
  - *idem pour la translation arrière*
  - *idem pour la translation gauche*
  - *idem pour la translation droite*

Donc, on remarque assez aisément qu'il n'existe pas de méthode d'évitement pour les rotations. Or, comme je l'ai indiqué dans le chapitre précédent, le robot est de géométrie rectangulaire et ne décrit pas des cercles autour de son « centre » mais plutôt des ellipses. Ceci à pour conséquence de prendre une surface pour tourner assez conséquente et donc il est nécessaire d'implémenter un « garde-fou » pour des mouvements de rotation.

Afin de procéder ainsi, j'ai dû d'abord comprendre le système de transmission d'ordre dans les composantes de bas niveaux. La tablette communique via Bluetooth ses instructions et la base roulante les capte via un flux de données transmis par le module Bluetooth connecté à l'Arduino Méga. Or, les flux de données Blutooth sont composés de variables de type Byte (Octets en Français) et connaître le protocole de communication est indispensable avant de s'aventurer à capter lesdites données.

Fort heureusement pour moi, un ingénieur[20] travaillant actuellement pour la société Axyn m'a transmis la documentation suffisante pour me permettre de réussir à créer une méthode d'évitement suffisante.

19 J'ai simplifié ici l'algorithme, dans le code-source, la fonction retourne un booléen pour autoriser ou non le déplacement ce qui nécessite un ultime test ayant pour options **ne pas modifier la variable de déplacement** OU **de mettre à zéro ladite variable.**

20 M. Guillaume AGUIRAUD que je tiens ici à remercier.

Je vais donner ici les grandes lignes du protocole précédemment évoqué :

Numéro_octet	Utilité	Possibilités	Encodage
1	Commande	**Avant** **Arrière** **Stop** **Translation-gauche** **Translation-droite** **Mouvement_tablette**	0x01 0x02 0x03 0x08 0x09 0x10
2	Nombre d'octets restants	***Toujours présent***	7 bits « classique »
3	Angle de la commande	**Entier entre 0 et 127**	7 bits « classique »
4	Fin de trame 1	***Toujours présent***	0x7F
5	Fin de trame 2	***Toujours présent***	0x00
6	Fin de trame 3	***Toujours présent***	0x7F

L'encodage étant comme ci-dessus, il faut maintenant se pencher sur la physique du robot afin de coder une méthode d'évitement en rotation.

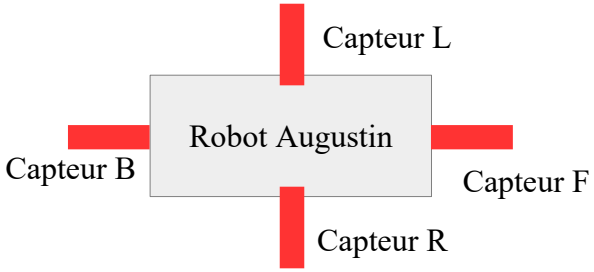


Illustration 4: Les zones en rouge représentent l'interdiction de mouvement vue par les capteurs

On remarque que lorsqu'on fait pivoter le robot sur lui même, les quatre côtés du robot peuvent heurter aisément un obstacle dû au fait qu'ils balayent une aire assez grande. Donc il est préférable qu'il n'y ait aucun obstacle détecté autour du robot lorsqu'il commence une rotation sur lui même [21]. Autrement dit, l'algorithme d'interdiction de rotation sera le suivant[22] :

Soit c la condition telle que

$$c = (octet_2 > 0) \times \sum_{i=1}^{4} proximité\,(capteur_i)$$

21 Même si l'obstacle n'est vu que par un seul capteur. En effet, il suffit que le robot décrive un angle de plus de 90° pour nécessairement le percuter. Donc je me suis permis de penser à faire un « garde-fou » draconien afin de prévenir toute collision inopinée.

22 Notation usuelles de l'Algèbre de Boole

alors

c = vrai → mouvement = arrêt

c = faux → mouvement = mouvement

En d'autres termes si l'angle de rotation est non nul (pas de translation pure) **ET** que l'un des capteurs détecte un objet **ALORS** la base roulante est forcée de s'arrêter quelle que soit la consigne demandée.

J'ai implémenté sur Arduino ce code, qui est assez simple à mettre en œuvre (composé d'équations logiques seules, sans réglage).

A noter qu'il est difficile de faire plus de modifications sans toucher à la partie matérielle du robot car comme je l'ai souligné dans le chapitre précédent, le robot est victime des faibles capacités des capteurs. En revanche, dans le chapitre suivant, je préconise quelques améliorations en ce sens.

2. Tracking de personnes

Le pilotage du robot étant fait par l'utilisateur cherchant à suivre l'interlocuteur à distance, il est apparu naturel pour mes tuteurs et moi même de l'automatiser. Réflexion d'autant plus naturelle puisque les compétences en traitement d'image et tracking d'image à l'Institut Fresnel sont regroupées. Le but étant de suivre le visage de l'interlocuteur avec le plus de robustesse possible tout en pouvant arrêter le « pilotage automatique » de la base roulante à tout instant afin de garantir une sécurité et un confort de pilotage optimaux.

a) Tracking grâce à la bibliothèque OpenCV de traitement d'images

En tant qu'ancien Président du Club Robotique de Centrale Marseille, je fus confronté à la problématique de tracking dynamique d'objets en milieu réel. En considérant la Webcam de la tablette du robot comme un capteur potentiel, le tracking m'a paru envisageable : nous avions utilisé au Club une caméra webcam avec encore moins de résolution et de mémoire vive disponible[23] que sur la tablette.

C'est ainsi que j'ai repris les travaux de mes collègues roboticiens de l'équipe du Pôle Haut Niveau du Club Robotique que je tiens à remercier[24]. Ces derniers avait utilisé la bibliothèque libre de droit **openCV** pour le traitement d'image et tous les algorithmes que je vais évoquer après, lesquels seront disponibles et documentés dans le Wiki du Club Robotique. Je précise que ce dernier est librement accessible à l'adresse suivante : http://wiki.centrale-marseille.fr/egab

OpenCV est une bibliothèque codée à l'origine en C++ mais les méthodes ont été traduites en des langages de plus haut niveau tels que Java ou Python. Néanmoins, j'ai dû utiliser la version 2.4 d'OpenCV (assez ancienne, datant de 2013) car les méthodes ne sont pas toutes traduites en Java. Ceci se justifie par le fait que les périphériques Androïd ont des applications

23 Nous avions utilisé une Raspberry Pi 2 qui a pour défaut principal sa mémoire vive. Nous avions dû recourir à un subterfuge afin de pallier à la surconsommation de mémoire vive pour le traitement d'images : procéder au traitement que si nécessaire et seulement sur une image donnée avec le robot immobile. Ceci limitait la capacité opérative du robot mais ce fut salutaire vu les contraintes de temps et de capacité de calcul imposées dans le cadre de la « petite » robotique embarquée.

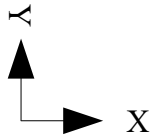
24 Annabelle TRINH (responsable Traitement d'Image 2015), Florian CALVET (RTI 2016), Constanza FIERRO (Responsable Inférence de Haut Niveau 2016), Sylvain BENTZ & Tarek BENHNINI (RIHN 2015).

fonctionnant exclusivement en Java et qu'il est presque obligatoire d'écrire en ce langage pour obtenir une application efficace compilée sous Android.

b) Stratégie de repositionnement d'Augustin

Le but de mon programme est de repositionner le robot Augustin face à son interlocuteur. J'ai donc procédé à une division de l'image captée par la caméra en neuf zones :

Z1	Z2	Z3
Z4	Z0	Z5
Z6	Z7	Z8



Le but étant de centrer l'image en ZO. Pour cela je procède par un recalage en trois étapes qui nécessite trois informations.

Premièrement, il me faut les coordonnées cartésiennes dans la matrice qu'est l'image du centre de l'objet visé. On notera ces dernières **x** et **y** : je précise par ailleurs que les programmes de reconnaissance automatique des formes et des couleurs que je présente ci-après nous donnent facilement accès à ces informations. Généralement, la forme recherchée est inscrite dans un rectangle qui suit ses contours : on nommera par la suite cette opération **« boxage »** et ce rectangle **« box »**.

L'aire de la box peut nous renseigner sur la profondeur de champ **z** : si l'on sait les dimensions de l'objet de l'on cherche à centrer alors forcément en ayant l'aire on en déduit **z**.

Algorithme :

En pratique, on va créer une chaîne de caractère différente en fonction de l'appartenance ou non aux zones géométriques précitées. Cette chaîne sera interprétée par l'application de commande de la tablette du robot (dans un premier temps pour simplifier)

Nom de la commande	Effet	Nom de la commande	Effet
toDOWN	Agit sur le servomoteur pour baisser l'image	toRIGHT	Idem pour la droite
toTOP	Agit sur le servomoteur pour rehausser l'image	toFORWARD	Fait avancer le robot en ligne droite (il s'approche de l'interlocuteur)
toLEFT	Provoque une rotation sur lui même d'Augustin le ramenant à Gauche	NULL	Le robot reste immobile

On va ajuster dans un premier temps le paramètre le plus crucial pour ne pas perdre l'interlocuteur, à savoir l'angle autour de l'axe du robot, matérialisé par **x** dans l'image. Puis,

on décide ou non de se rapprocher de l'interlocuteur (profondeur de champ **z**, matérialisée par l'aire de l'objet visé). Enfin, on règle la hauteur à l'aide du servo du mât du robot (axe **y**).

- **SI** ($x_{seuil}^{bas} \leqslant x \leqslant x_{seuil}^{haut}$)
  - **ALORS**
    - **SI** ($Aire_{seuil}^{bas} \leqslant Aire \leqslant Aire_{seuil}^{haut}$)
      - **ALORS**
        - **SI** ($y_{seuil}^{bas} \leqslant y \leqslant y_{seuil}^{haut}$)

          ALORS commande = NULL
        - **SINON** commande = toTOP/toDOWN (suivant le cas)
    - **SINON** commande = toFORWARD (grande aire) OU commande = NULL (trop petite aire : l'objet est considéré comme du bruit)
  - **SINON** commande = toRIGHT/toLEFT (suivant le cas)

Note : Ce programme est étudié pour être transposable à toutes les méthodes de détection d'objets qui suivent : il suffit juste de régler les seuils et/ou de moyenner plusieurs mesures afin de limiter le bruit.

c) Algorithme simple de tracking : suivi d'un disque rouge

Tout d'abord j'ai voulu, afin de comprendre les méthodes de Tracking et sous les conseils avisés de M.Fréderic GALLAND[25] tester un programme simple. En effet, reconnaître un signe distinctif que l'on connaît par avance est plus simple que de détecter un visage humain. Afin de comprendre plus en détail l'état de l'art dans ce domaine, j'ai lu une publication de l'équipe PHYTI[26] sur le sujet datant de 2008 qui mettait en avant l'extraction de quelques caractéristiques afin de « boxer » la personne cible[27] ce qui permettait de rendre les programmes plus robustes.

Or, dans un premier temps, il me restait à trouver un objet déterministe. De plus, je savais, en tant qu'initié dans le domaine du freeware qu'il existait des exemples de tracking géométriques monochrome déjà codés en Java. Donc, j'ai cherché et trouvé un code pouvant me retourner les coordonnées cartésiennes d'un disque rouge uni ainsi que l'aire de sa « box ».

Les essais furent plutôt concluants et le code s'avéra assez robuste même si il fut aisé de faire dysfonctionner ce dernier en présentant des objets de couleur rouge autour de la cible.

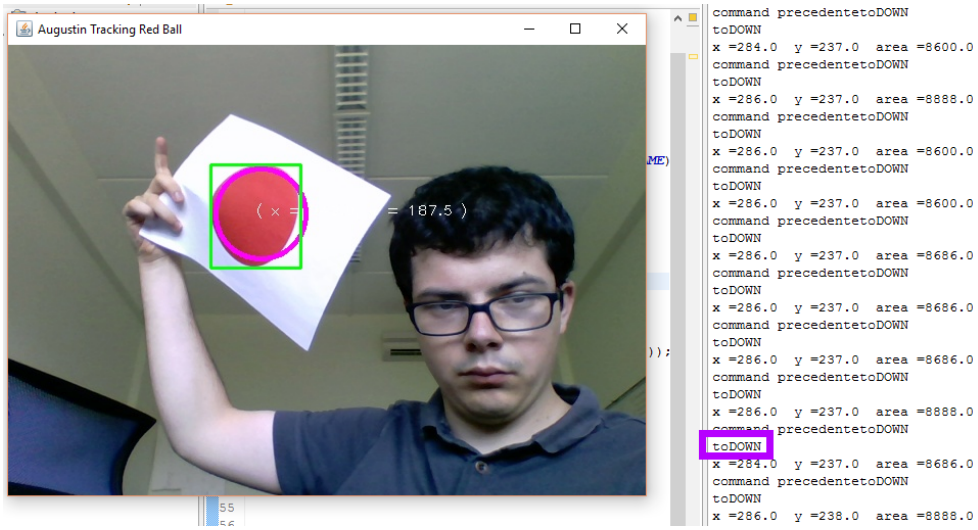


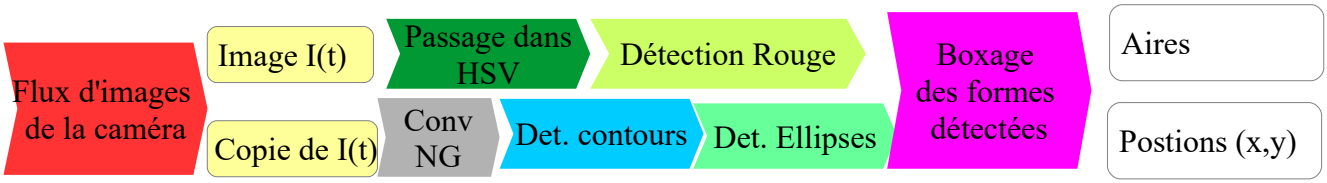
Illustration 5: Je déforme légèrement la feuille en la mettant en mouvement mais l'ellipse est tout de même repérée. Mieux : mon programme d'extraction me signale qu'il faut descendre l'image pour cadrer correctement.

25 Maître de conférences à Centrale Marseille, membre de l'Institut Fresnel, ayant publié dans des revues scientifiques sur la thématique du tracking. Je tiens à le remercier chaleureusement pour le temps qu'il m'a consacré et qui fut d'un grand appui pour mes travaux

26 Spécialistes du traitement d'images à Fresnel

27 Stochastic complexity integral image based technique for fast video tracking, J-F Boulanger, F.Galland, P.Martin, Ph.Réfrégier, AMU, Fresnel, Kaolab Tech. Article daté du 28 Octobre 2008, Optical Society of America

Explication de l'algorithme de détection :



On procède par étapes :On capture deux images identiques

(a) Image A :

i. On fait passer l'encodage des pixels dans l'espace HSV, plus propice à la détection des couleurs telles que le perçoit l'humain[28]

ii. Des seuils (hyperplans) dans l'espace HSV permettent de définir à partir de I(t) l'image binarisée suivante : (on note P(x,y) la valeur du pixel en (x,y)) $I^B(t) tq \quad \forall (x,y) \in I^B(t), P(x,y)=1 \quad si (x,y) \in HSV_{rouge} \quad et \, P(x,y)=0 \, sinon$

(b) Image B :

i. On convertit l'image en niveau de gris afin de faire une évaluation des contrastes

ii. En appliquant un filtre passe haut sur l'image en niveau de gris, on arrive à estimer la valeur locale du gradient de luminosité :

En effet, si je connais $\frac{\partial I(x,y)}{\partial x_i}$ alors je sais les variations brutales de luminosité et de ce fait les contours.

iii. Comparaison avec seuil de luminosité de l'image obtenue. Afin d'obtenir une image de pixels P(x,y) = 1 si (x,y) appartient à un contour et 0 sinon

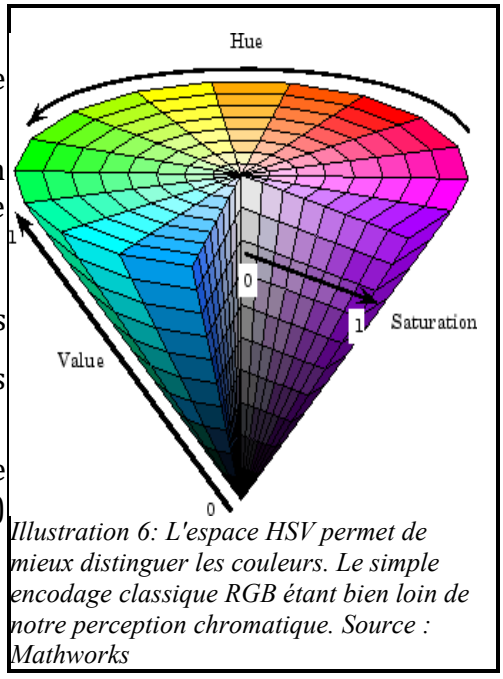


Illustration 6: L'espace HSV permet de mieux distinguer les couleurs. Le simple encodage classique RGB étant bien loin de notre perception chromatique. Source : Mathworks

iv. « Boxage » des contours contigus C_i dans des rectangles

v. Si la forme C_i est un cercle, alors le rectangle le plus petit l'entourant est un carré de côté 2R, avec R rayon du cercle. Dans ce cas : $\frac{Aire_{box}}{Aire_{C_i}} = \frac{4r^2}{\pi r^2} = \frac{4}{\pi}$ ce qui est un **invariant**[29] . Il suffit juste de faire la comparaison des deux aires (avec des seuils de tolérance, car le résultat mathématique n'est évidemment jamais obtenu de manière rigoureuse)

(c) On compare les deux résultats obtenus et on obtient les données voulues c'est à dire

28 cf. Illustration explicative

29 Notion très importante en Reconnaissance des Formes car les distances et les dimensions peuvent évidemment varier de manière imprévue.

(x,y) les coordonnées cartésiennes du plan de l'image pour le centre de la « box » et son aire A.

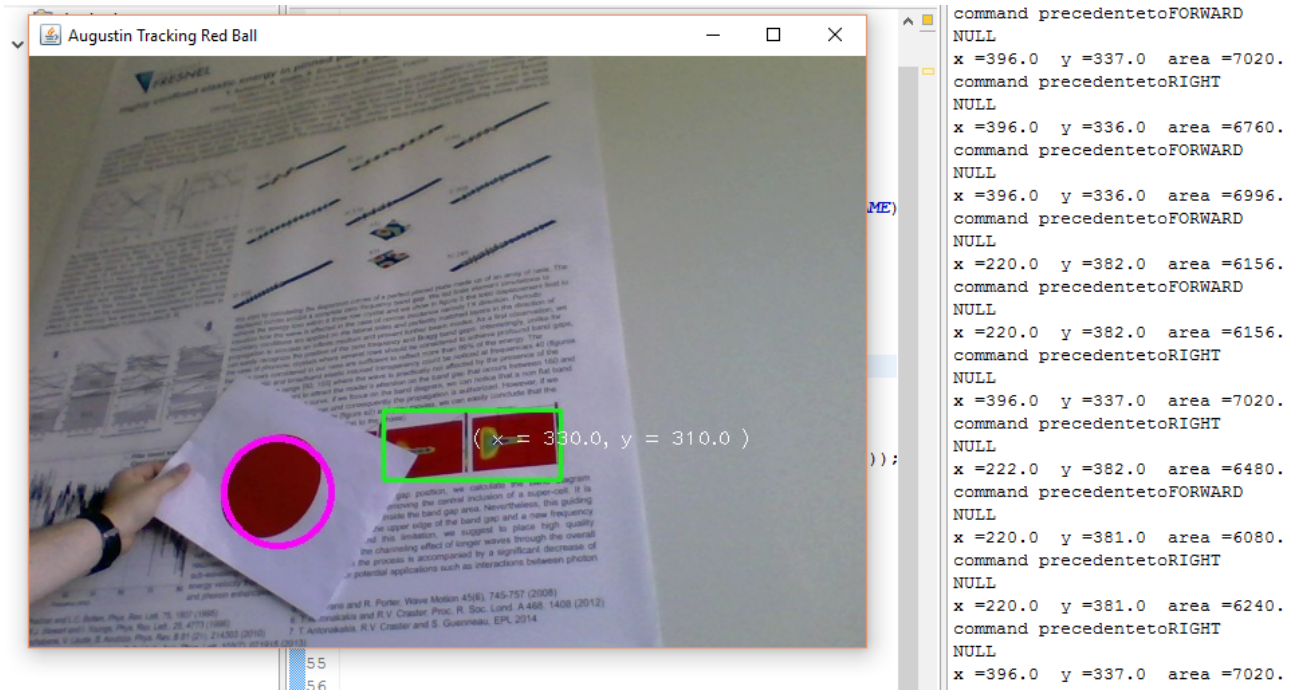


Illustration 7: Un fond bruité (autres objets rouges) peut induire en erreur le programme de boxing. La robustesse du code est ici mise à l'épreuve...

Algorithme d'exploitation des données :

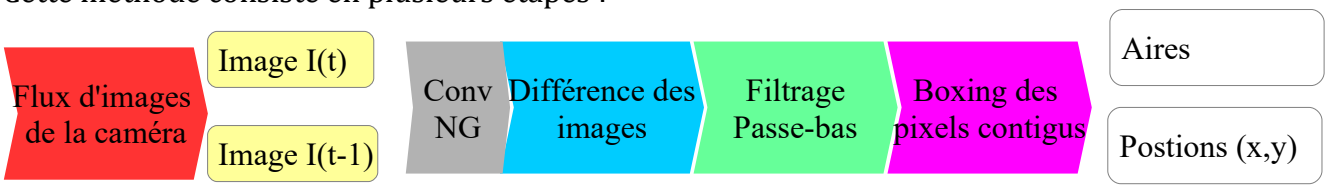
Il est obligatoire de prendre quelques précautions avant d'envoyer à l'algorithme de repositionnement précité les données brutes, je fais donc un petit traitement :

- *Notations :* pour chaque Box B(t) il existe une position de son centre (x(t),y(t)) ainsi que l'aire de cette dernière A(t)
- **TANT QUE** (tracking = vrai)
  - SI ($commande(t) = commande(t-1)$)
    - nombre_succes += 1 ;
    - commande(t-1) = commande(t) ;
    - commande(t) = correction(x(t), y(t), A(t))
  - SINON
    - nombre_succes, a_moyenne, x_moyen, y_moyen = 0
    - commande(t-1) = commande(t) ; commande(t) = NULL ;
  - SI (nombre_succes > N_requis)
    - applicationConsigne() ;
  - SINON
    - applicationConsigneNulle() ;

On soumet à une validation multiple les mesures. La validation permet de réduire les effets de bruits (« boxage » inopiné d'un objet considéré comme du bruit alors que la cible est suivie → arrêt de la machine par mesure de précaution).

d) Algorithme simple de détection de mouvement : méthode de soustraction

Cette méthode consiste en plusieurs étapes :



- On convertit en Niveau de gris (Conv NG) les deux images
- On obtient de la différence $I(t)_{NG} - I(t-1)_{NG} = I_{NG}^{Resultante}$
- $I_{NG}^{Resultante}$ étant potentiellement bruitée, on lui applique un filtrage discrétisé gaussien passe bas sur trois pixel de côté afin d'éliminer les trop petits groupements :

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

en dimension pour une fenêtre de trois pixels de côté :

On note M(x ,y) un pixel courant du plan et $V_3(M(a,b))$ le voisinage immédiat de degré du pixel situé aux coordonnées (a,b) du plan, ie la région à moins de trois pixels de M(a,b)

f(x,y) représente ici l'éclairement du pixel de coordonnées (x,y)

h(k,l) = g_0 (k,l) si $(i',j') \in \mathbb{R}^2, M(i',j') \in V_3(M(k,l))$ et h(l,k) = 0 sinon

Et donc ici, $g_o(x,y) = \frac{1}{\sigma_x\sqrt{2\pi}} e^{-\frac{(x-\mu_x)^2}{2\sigma_x^2}} \frac{1}{\sigma_y\sqrt{2\pi}} e^{-\frac{(y-\mu_y)^2}{2\sigma_y^2}}$ fonction gaussienne 2D dont les paramètres sont évalués à partir du pixel de référence (ici M(k,l))

- On compare les pixels restants en éclairement afin d'obtenir une matrice de pixels binarisée.
- Les pixels contigus sont regroupés au sein d'ensembles
- OpenCV détecte leur contour et les « boxe » dans des rectangles dont on se servira par la suite pour estimer la position de l'objet en mouvement.

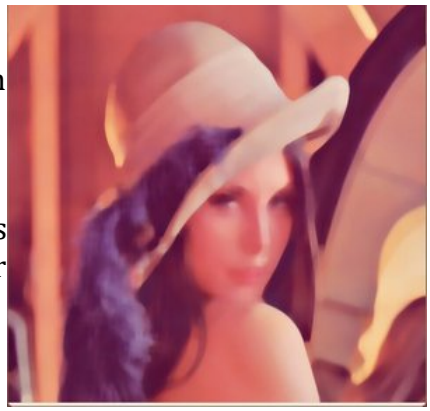


Illustration 8: Image floutée (Lena) par un filtre gaussien Source : openCV.org

Tests : Limites de l'algorithme par soustraction

Conditions du test 1 : visage en mouvement (légèrement)

L'algorithme crée bien des « boxes » autour des pixel qui ont bougé mais bien souvent du bruit apparaît et fausse complètement l'inférence du robot. J'ai dès lors tenté la même approche que pour le problème du disque rouge.

J'ai donc ré implémenté l'algorithme d'exploitation [30] permettant une extraction de données assez robuste en rajoutant une étape moyennant les variables car le flux de données est important mais bruité :

- *Notations :* pour chaque Box **B(t)** il existe une position de son centre (**x(t),y(t)**) ainsi que l'aire de cette dernière **A(t)**
- **TANT QUE** (tracking = vrai)
  - **POUR** dix points de mesures
    - faire la moyenne des variables : x_m(t), y_m(t), A_m(t)
  - **SI** ($commande(t) = commande(t-1)$)
    - nombre_succes += 1 ;
    - commande(t-1) = commande(t) ;
    - **commande(t) = correction(x_m(t), y_m(t), A_m(t))**
  - **SINON**
    - nombre_succes, a_moyenne, x_moyen, y_moyen = 0
    - commande(t-1) = commande(t) ; commande(t) = NULL ;
  - **SI (** nombre_succes > N_requis)
    - **applicationConsigne() ;**
  - **SINON**
    - **applicationConsigneNulle() ;**

30 Déjà utilisé pour le tracking du disque rouge

Illustration :

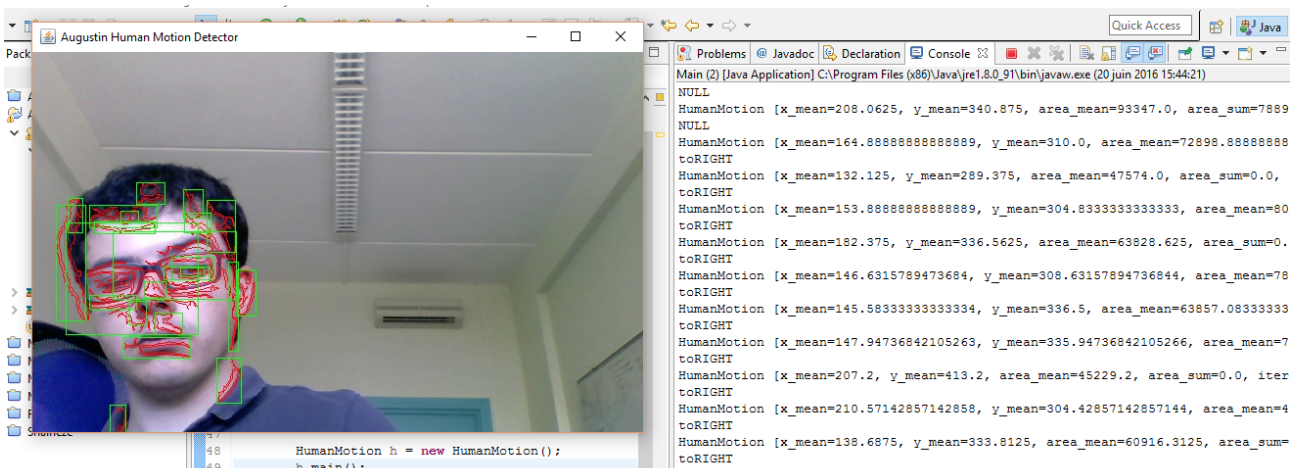


Illustration 9: Test mettant en œuvre la méthode HumanMotion, on remarque que le robot a pris la bonne décision à savoir centrer mon visage sur la droite ("toRIGHT" dans la console) on peut également apercevoir les valeurs de x, a et y moyennes.

Conditions du test 2 : bouche seule en mouvement

Ce test permet de mettre en évidence un défaut de l'algorithme : la méthode d'estimation aréolaire de la profondeur de champ n'est pas adéquate.

Le robot prendrait la décision d'avancer alors que l'interlocuteur est déjà très près !

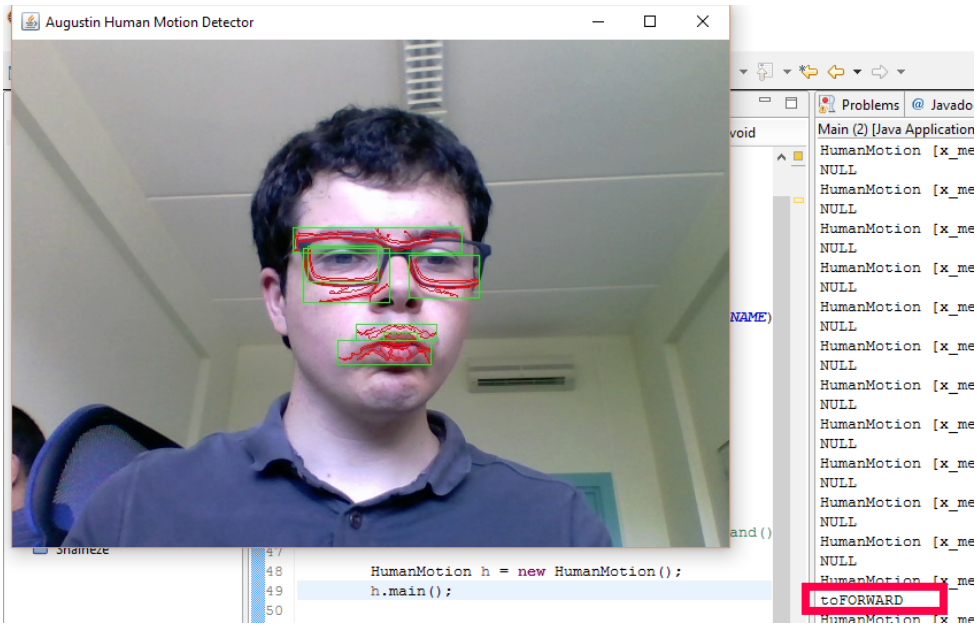


Illustration 10: Le robot décide d'avancer de son propre chef alors que mon visage est convenablement cadré

Conditions du test 3 : Caméra « tremblante »

Je le rappelle ici : le robot est pourvu d'une caméra sur un mât et donc la prise de vue est quelque peu tremblante. Regardons les effets d'un tel comportement sur l'algorithme : Cf. Illustration 11

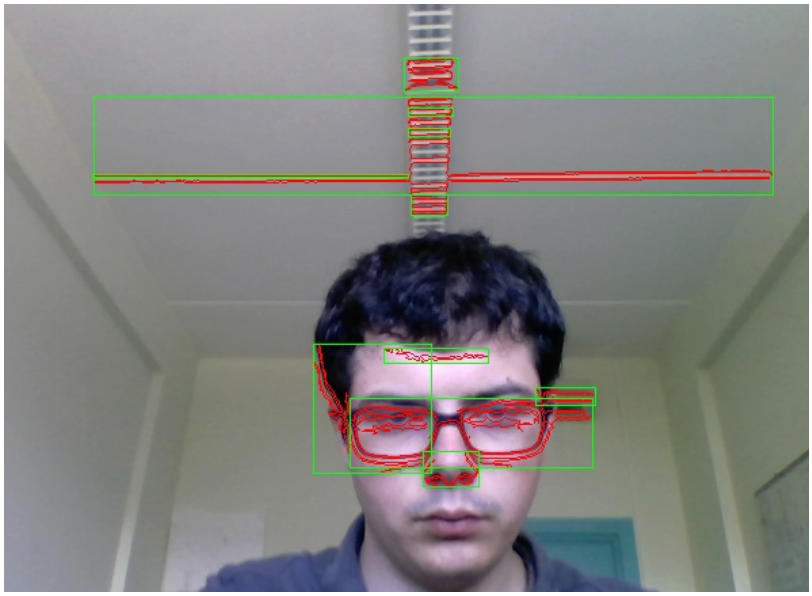


Illustration 11: On voit que l'aire la plus grande : ie la box ayant la plus grande pondération est ici de manière malheureuse du bruit.

Hélas on ne peut avoir de caméra rigoureusement immobile sur le mât, il semblerait que la détection par soustraction ne soit pas adaptée au problème.

Conclusion :

Certes, l'expérience de la détection par soustraction se révéla ici non transposable mais je pus rajouter quelques critères à mon cahier des charges afin de garantir une plus grande robustesse :

- Moyenner les points obtenus par boxing
- Faire un filtrage par itérations successives
- Extraire des caractéristiques géométriques propres à des êtres humains afin de limiter le bruit
- Utiliser un invariant dans l'image (le visage par exemple) afin de déterminer l'aire de ce dernier et par ricochet la profondeur de champ.

e) Algorithme définitif : suivi d'un visage

L'algorithme de soustraction simple nous retournant une image fort bruitée et de plus partant sur une hypothèse de mouvement de la personne, il fallait en trouver un plus performant. Nous avons besoin de réseauter le visage d'une personne avec un nuage de point afin de pouvoir en extraire quelques caractéristiques géométriques et espérer ainsi obtenir des invariants du visage humain.

Algorithme choisi :

Non seulement, il permet de repérer les visages mais en plus en flux dynamique, et il permet de faire également la même chose avec les yeux à distance réduite.

Démonstrations :

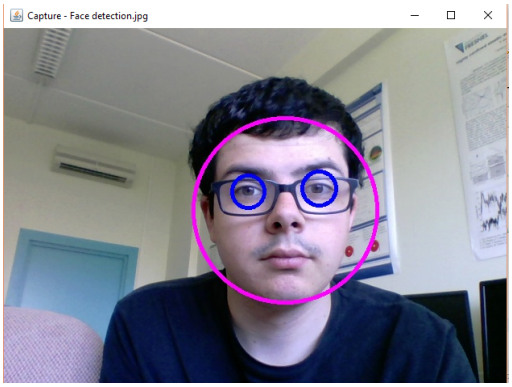


Illustration 13: Malgré mes lunettes de vue, l'algorithme arrive non seulement à détecter mon visage mais de plus mes yeux

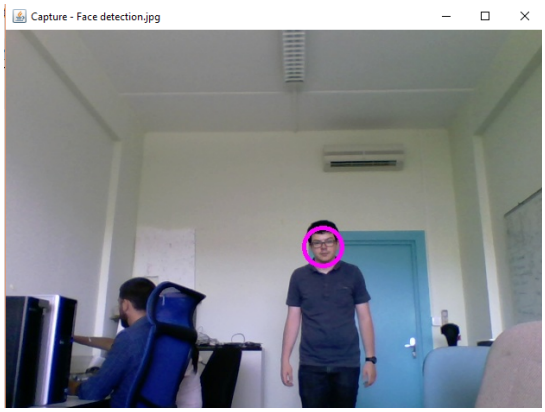
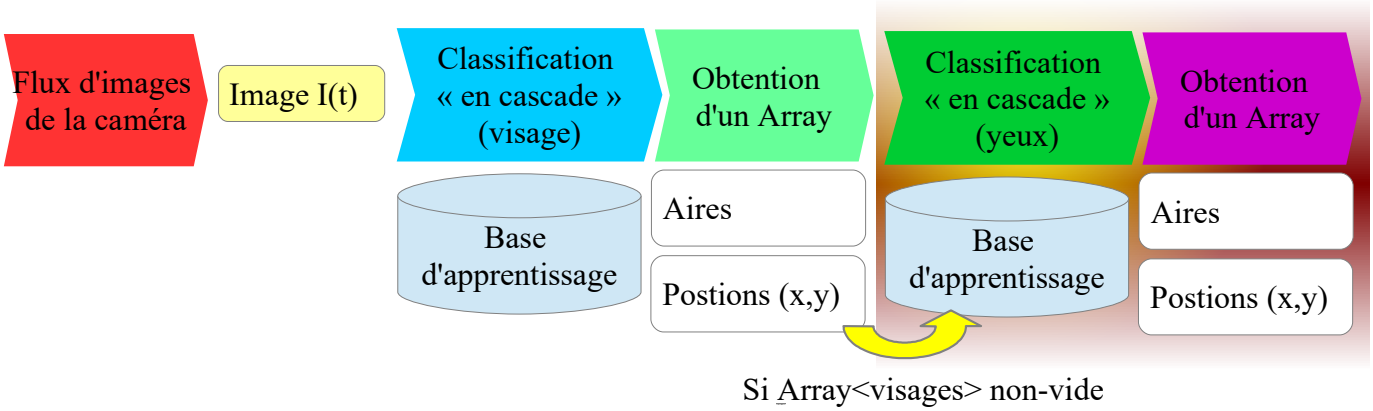


Illustration 12: Il se révèle robuste à longue distance. Mon visage est détecté car je fais face à la caméra, celui de mon collègue Bogdan (à gauche) ne l'est pas car il est de profil.

Principe de l'algorithme :



NB : Un « Array » est le terme qui désigne un tableau d'objets en Java.

(a) Classification en cascade

Une classification en cascade est une classification nécessitant une base d'apprentissage pour déterminer si l'objet est celui recherché ou non. Dans un premier temps, l'image est traitée

afin de découvrir des ROI[31] nécessaires pour isoler les objets candidats. Des algorithmes incluant une détection de contours par la méthode du filtre passe-haut dans l'espace des niveaux de gris peuvent isoler lesdites régions.

S'en suit un **test en cascade[32]** pour chacune des sous-images d'objets candidats : les objets sont soumis à un test de corrélation entre les objets de la base test classés selon des orientations Θ_i différentes[33] et selon des caractéristiques χ_i distinctes[34].

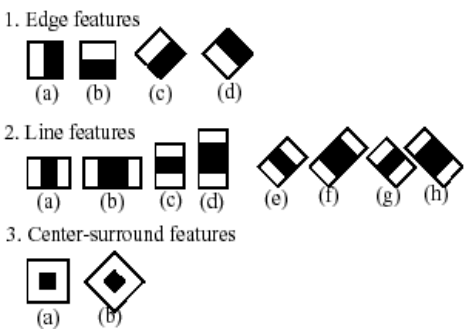


Illustration 14: La méthode de reconnaissance en cascade. Source : opencv.org

Le test décidant si oui ou non l'objet est celui recherché peut s'écrire de la manière suivante:

Notations : Les produits et sommes sont à prendre ici au sens de l'algèbre de Boole. On note :

- la variable $Is_X(y)$ la variable booléenne caractérisant l'appartenance de l'objet y à la catégorie des X.
- x_i^j un objet appartenant à la base d'apprentissage de la catégorie des X, dont l'image a été capturée dans sa j ème orientation.
- $\chi_j(x_i)$ la j ème caractéristique d'un objet x_i
- la corrélation booléenne[35], paramétrée par une tolérance donnée[36] α comme suit : $C_\alpha(\chi_j(x_i), \chi_k(x_l))$

On a : $$Is_X(y) = \prod_{k=0}^{dim(CaracteristiquesX)} \sum_{j=0}^{dim(BaseObjetsX)} \sum_{i=0}^{dim(OrientationsX)} C_\alpha(\chi_k(x_j^i), \chi_k(y))$$

On remarque donc que pour **un objet candidat** d'une ROI donnée, l'algorithme est déjà au pire[37] en terme de complexité :
$O(dim(caracteristiques_x) \times dim(Orientations_x) \times dim(BaseObjets_x))$

Ceci peut paraître grand **mais** si l'on trouve les caractéristiques correspondants aux meilleurs invariants de la catégorie des X, alors on limite l'indice k de la fonction ET (produit booléen).

(b) Deux boucles imbriquées pour un tracking optimal

En fait, on va chercher à tester en continu la présence de visage et si cette dernière s'avère fructueuse faire une recherche d'yeux en fonction de la position et de la taille du visage détecté. Ce subterfuge limite très fortement la complexité du programme : la ROI est choisie et

31 Region of Interest
32 D'où le nom du présent algorithme
33 (a), (b)… sur la figure illustrative. Ceci étant fait afin de limiter l'impact de l'orientation de l'objet sur sa reconnaissance par l'algorithme.
34 (1), (2)… sur la figure illustrative. Ce sont les tests consécutifs du cascadage.
35 Qui répond à la question : les deux caractéristiques comparées sont-elles similaires ?
36 Exemple, dans l'espace RGB 8 bits, la valeur R de l'objet candidat doit être égale à 200 **à 5 % près**
37 Il n'est pas obligatoire de faire toutes les corrélations : si l'on retrouve toutes les caractéristiques discriminante de l'objet y dans la base d'apprentissage, le programme s'arrête et conclut à son appartenance. Le cas le plus défavorable est la présence d'objets ressemblant à un visage mais avec une caractéristique aberrante (bonnes proportions mais absence de bouche par exemple). L'algorithme alors va chercher à tester toutes les orientations et exemples d'objets de la base avant de renoncer.

les objets candidats sont au nombre de deux. La base de données est également de taille réduite du fait que l'on peut s'affranchir d'une partie du problème d'orientation.

(c) Interprétation par le programme de repositionnement

J'ai repris le même algorithme que pour le tracking d'une balle rouge. En y adaptant quelques modifications (la détection des yeux dans la région $x_{seuil}^{bas} \leqslant x \leqslant x_{seuil}^{haut}$) entraîne l'annulation des « toFORWARD » car j'estime que lorsqu'on distingue les yeux d'une personne, on se trouve suffisamment près de cette dernière pour converser. En pratique, j'ai affecté une valeur très grande et aberrante dans ce cas à l'aire détectée pour ne pas réécrire la méthode de correction[38]

- *Notations :* pour chaque Box B(t) il existe une position de son centre (x(t),y(t)) ainsi que l'aire de cette dernière A(t)
- **TANT QUE** (tracking = vrai)
  - SI ($commande(t) = commande(t-1)$)
    - nombre_succes += 1 ;
    - commande(t-1) = commande(t) ;
    - **SI(** $x_{seuil}^{bas} \leqslant x \leqslant x_{seuil}^{haut}$ **&& taille(Array_Yeux) > 0)**
      - **A(t) = 1e8 ;**
    - commande(t) = correction(x(t), y(t), A(t))
  - SINON
    - nombre_succes, a_moyenne, x_moyen, y_moyen = 0
    - commande(t-1) = commande(t) ; commande(t) = NULL ;
  - SI (nombre_succes > N_requis)
    - applicationConsigne() ;
  - SINON
    - applicationConsigneNulle() ;

(d) Ressource mémoire utilisée

Avec l'affichage graphique dynamique sur le retour vidéo, mon micro-processeur et ma mémoire vive étaient fortement sollicités. A tel point que j'ai estimé que ceux de la tablette du robot Augustin n'allait plus suffire. J'ai donc pris la décision d'alléger le programme en terme de ressources requises.

f) Version allégée du programme pour une application mobile

En créant un attribut booléen dans ma classe autorisant ou non le déploiement de l'interface graphique, en implémentant l'algorithme de telle manière à ne retenir que les données géométriques et de commande de la base roulante après une itération. Avant tout en ne créant pas d'autres attributs que ceux déclarés dans le constructeurs de la classe, j'ai réussi à faire un programme qui nécessite moins de 150Mo de RAM sans retour graphique, ce qui est acceptable pour faire fonctionner ce dernier sur une application mobile.

NB : J'ai utilisé l'interface de développement Android Studio ® afin de compiler l'application.

38 Cela revient au même, si le lecteur est prévenu.

III.Idées pour l'optimisation du robot

1. *Capteurs ultrasoniques*

Je préconise des capteurs ultrasoniques plutôt que des capteurs à réflexion photonique. Plusieurs raisons imposent ce choix :

- On ne cherche pas à faire une mesure odométrique précise mais bien une mesure d'évitement. Donc, les cônes d'émission larges des capteurs ultrasoniques sont plus adaptés que les cônes étroits des capteurs à lumière rouge.
- Le problème de la réflexion partielle est conséquent avec les capteurs photoniques :

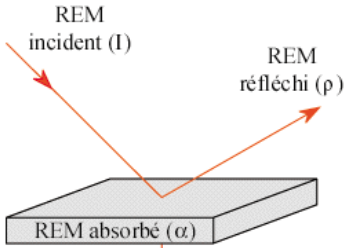
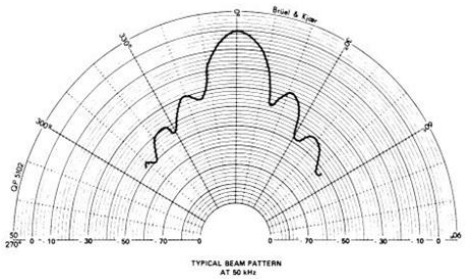


Illustration 15: Illustration du phènomène de réflexion électromagnétique (source : UVED)

En effet, cette dernière dépend pour une part non négligeable de la nature du matériau cible du rayon : ainsi $\alpha+\rho=I$ en termes de puissance transmise. Si par exemple, le rayon incident est de couleur rouge (ce qui est notre cas) et que le matériau-cible est foncé et mat, alors le terme d'absorption α devient prépondérant.

Cette discrimination en fonction du matériel est **moins prépondérante si les ondes sont acoustiques** et si le matériau est rigide, ce qui pousse à choisir la technologie ultrasonique.

- Des bibliothèques en .cpp sont existantes sur le web pour retourner la distance. Bien souvent, on utilise un estimateur statistique de type maximum de vraisemblance pour retourner le temps d'aller-retour T de l'onde. Ensuite, en supposant que la célérité V de l'onde acoustique dans l'air est constante et vaut approximativement 340m/s (Mach 1). Alors il est possible de déterminer la distance à l'obstacle aisément : $d=\frac{T\times V}{2}$
- L'estimateur statistique permet de repérer la première réflexion et cela donne le **minimum** des distances repérées dans le cône de détection du module ultrason. Ceci est exactement ce que l'on veut : l'obstacle le plus proche fait foi.

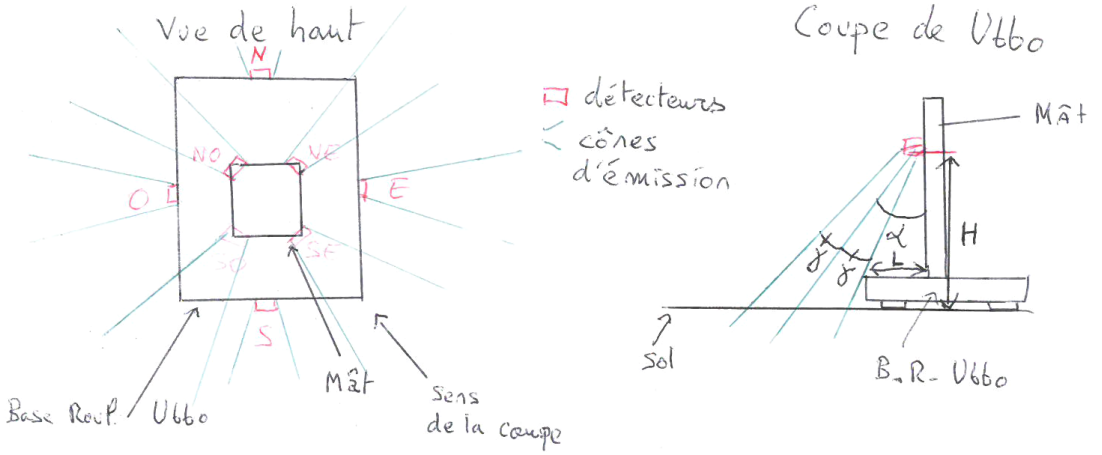


Structure typique d'un cône d'émission d'un capteur à ultrason.
Source : Génération Robots

2. *Robe ultrasonique protectrice*

Nous devons également détecter les obstacles lacunaires : que ce soit des cages d'escaliers[39] tout comme des tables[40]. Pour cela, je propose de créer une « **robe ultrasonique** » qui serait comme suit :

- Si l'on note les quatre détecteurs existants sur la base roulante comme les points cardinaux : **N, S, E,** O. Alors, on ajoute quatre détecteurs sur le mât à α d'incidence par rapport au sol que l'on place aux quatre points cardinaux secondaires **SE, SO, NE, NO**. Je laisse ici deux schémas pour mieux saisir la situation :



- <u>Calcul de l'angle d'incidence :</u>

  On note γ , l'angle du cône caractérisant le faisceau du détecteur. On prend par exemple 60°, qui est une valeur assez large mais commercialisée de cône de détection (on se place donc dans le pire des cas). Il faut prendre en compte cette valeur car on risque de détecter la base du robot <u>et non le sol</u>. On considérera que la demi-longueur de la diagonale de la base roulante fait *L = 30cm*. Si l'on place les capteurs à *H = 1m* du sol (ce qui ne gênerait pas le déplacement de la tablette). On obtient :

$$\tan(\alpha-\gamma)=\frac{L}{H}\Leftrightarrow\arctan\left(\frac{L}{H}\right)+\gamma=\alpha \quad car \quad \alpha-\gamma\in[0,90°]\Rightarrow\alpha\approx77°$$

  L'inclinaison des capteurs par rapport au sol doit être donc assez modérée. On remarque également que ma figure n'est pas à l'échelle, ce n'est qu'une illustration.

- <u>Synchronisation des capteurs :</u>

  Nous nous devons de faire attention à ceci : les capteurs de la **base roulante** peuvent émettre des signaux <u>induisant en erreur</u> les capteurs du **mât** (et vice-versa). Ainsi, il faut les **faire fonctionner par alternance** mais au rythme le plus élevé que possible afin de garantir un déplacement sûr du robot.

39 On nommera ce type d'obstacle lacunaire positif : le mât du robot se trouve à une distance au sol plus grande par rapport à celle relevée en situation normale. Le sol se dérobe face à Augustin/Ubbo en d'autres termes.

40 On nommera cette fois ce type d'obstacle lacunaire négatif : le mât se trouve à une distance inférieure à celle de la situation normale. Il y a quelque chose entre Augustin/Ubbo et le sol.

Si l'on suppose que l'on détecte les obstacles se trouvant à *2 mètres* du robot au plus loin, on peut faire fonctionner une série de quatre détecteurs durant une période T :

$T = \frac{2 \times d}{V_{son}} = \frac{2 \times 2}{340} \approx 12ms$ donc, les obstacles seront contrôlés dans une boucle durant *24ms*, ce qui est suffisamment rapide pour arrêter Ubbo en cas de danger de collision ou de chute.

3. Gestionnaire d'alimentation & chargeur intelligent

J'ai remarqué que le système d'alimentation présentait plusieurs défauts auquel je donne quelques pistes pour le parfaire :

- La prise du chargeur se trouve sur la partie interne du robot :
  - C'est problématique pour charger la batterie et ce n'est pas intuitif.
  - Il faut tout simplement la déplacer sur la partie externe, en prenant garde que Ubbo soit toujours démontable pour les réparations/améliorations éventuelle dans sa version maker.
- Les batteries doivent être au nombre de trois.
  - Nous avons donc celle de la tablette et idéalement deux autres : une batterie de puissance (moteurs) avec la batterie de commande (cartes et moteurs). Ceci limite les effets d'irrégularités de courant induites par l'appel en intensité du moteur.
  - Les batteries de puissance/commande devraient utiliser la technologie gel.
    - Ces dernières seraient bien plus lourdes et stabiliseraient ainsi la base roulante
    - Les chargeurs sont plus nombreux par le marché par rapport à la technologie au nickel (se chargent comme des batteries au plomb)
    - L'entretien est minime et les batteries résistent à de multiples cycles de charge, à l'inverse des accus au Nickel
    - On peut aisément en loger deux de manière symétrique dans la base.
  - Il faudrait un seul chargeur pour les trois batteries,
    - Ceci garantirait une meilleure expérience pour le client.
    - **MAIS :** les technologies sont différentes, une batterie au gel ne se charge pas de la même manière qu'une batterie au Lithium.
    - Je propose donc la chose suivante : créer un **module de charge** qui lorsque le chargeur de batterie au gel (identique à un chargeur de batterie au plomb) est branché interdit la charge de la batterie au lithium[41]. Une fois les batteries au gel chargées, un témoin lumineux s'allume et la tablette se retrouve chargée par la batterie au plomb « commande » au travers d'un régulateur de tension 5V « conventionnel ».

41 Celle de la tablette.

4. Vers une application plus complète

Dans l'optique d'un développement de Ubbo « clef en main », il me semble judicieux d'apporter quelques améliorations à l'application :

- Graphisme et ergonomie :
  - Quelques images seraient très bien vues du public dans l'application, notamment une vision schématique de Ubbo. L'application étant pas assez customisée à mon sens.
  - L'activité[42] de connexion du robot ne devrait pas être intégrée par défaut : elle ne présente qu'un bouton et fait un doublon avec l'activité de configuration. Je préconise soit de mettre un bouton avec témoin de connexion[43] dans l'activité principale.
  - L'activité de configuration devrait être plus lisible (faire un jeu de couleurs vives, bleu et vert[44] la rendraient très agréable).
  - L'activité de pilotage manuel devrait intégrer un schéma du robot : de cette manière, il faudrait faire en sorte que l'utilisateur comprenne du premier regard où se trouve l'avant du robot. De plus, un bouton de pivotage d'écran devrait être présent. Ceci est justifié par le fait que les commandes se retrouvent inversées lors d'un demi-tour du robot. Notons toutefois qu'il serait alors préférable que le schéma fasse aussi un « demi-tour » sur l'écran de contrôle.

- Intégrer l'application de tracking :
  - Permettrait d'augmenter le « confort de pilotage » du robot pour les visio-conférenciers.
  - Permettrait de développer des applications à but éducatif (suivi de formes, reconnaissance de visages).
  - **MAIS :** nécessite tout d'abord un système d'évitement d'obstacles fiable.

- Prévention des erreurs :
  - Faire un retour sur les alertes du système d'évitement serait souhaitable et facile à mettre en place. Répondre aux questions « où se trouve l'obstacle ? Est-il lacunaire (positif/négatif) ou « plein » ? » permettrait une meilleure prise en main d'Ubbo surtout si le visio-conférencier distant n'a jamais vu le robot.
  - Une alerte de batterie faible et un moniteur de niveaux avec estimateur de capacité serait également parmi mes recommandations. Je pense que c'est faisable en travaillant sur l'Arduino Mega de commande

42 Terme technique pour désigner un menu sous Android.
43 Ex : le bouton est bleu si le robot est connecté et grisé sinon.
44 De surcroît les couleurs de la société !

5. Bouton d'arrêt d'urgence

Il s'agit d'un détail mais je pense que la présence d'un bouton d'arrêt d'urgence rassurera les utilisateurs quant à la fiabilité d'Ubbo. De plus, je pense qu'ajouter un bouton de marche/arrêt plus visible et plus lumineux est souhaitable car l'actuel est assez petit et isolé.

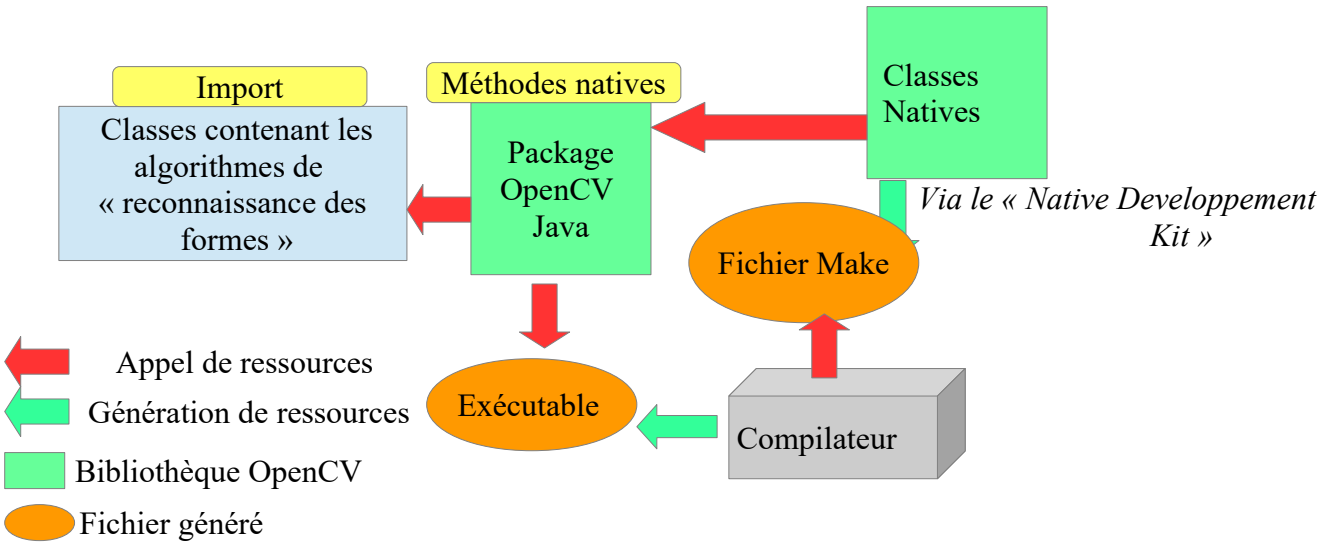
IV.Conclusion du stage et extensions possibles

1. *Difficultés à gérer OpenCV sous Android*

a) OpenCV, une bibliothèque native

Native signifie que cette dernière est codée en C++ , langage de programmation objet compilé ayant pour vocation d'être « de bas niveau ». Ce choix se motive par la nécessité d'avoir une gestion de mémoire parfaitement maîtrisée. Ainsi, les éléments de la bibliothèque OpenCV Java, appellent des fonctions dites « natives ». Ceci signifie que l'on doit configurer un compilateur pour qu'il rende exécutable les bibliothèques requises.

En substance on a ce schéma pour un fonctionnement en **Java « seul »**[45] :



b) Complexité du développement natif en Androïd

La structure d'une application Android ou plutôt du projet sous Android Studio est complexe. Je donne un chiffre surprenant faisant état de cela : le projet d'application que j'ai entrepris dépasse les 2000 fichiers et frôle le Giga Octet ! Ceci étant dit, je donne l'arborescence de mon projet afin de mieux comprendre (page suivante).

- Le projet doit au final se retrouver compilé en format **apk**, une archive conventionnée et exécutable avec le système d'exploitation Android.

45 Et non sous le format « Androïd », plus contraignant comme je vais le révéler par la suite.

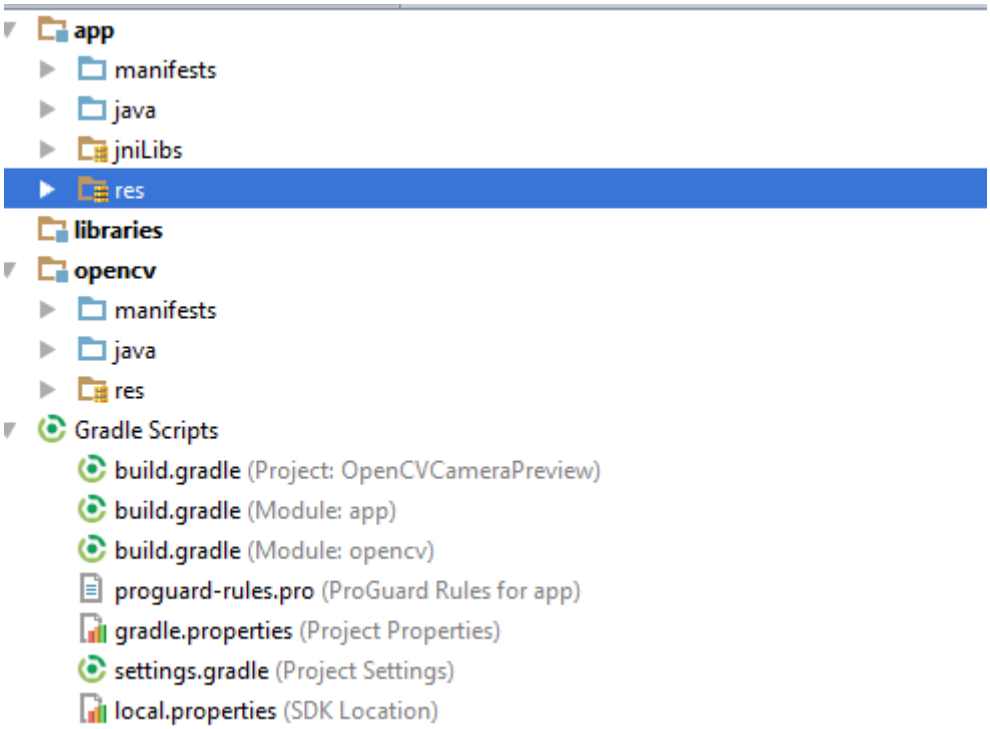


Illustration 16: Arborescence (non détaillée, faute de place) de mon projet Androïd

- Cette dernière est mise en forme par un compilateur Gradle. On distingue ces scripts en fin de l'arborescence : il y en a un par module et il s'agit de ceux qui sont lus en premier par le compilateur (haut niveau).
  - Ce dernier dispose de ses propres scripts avec leur propre convention syntaxique.
  - Le problème est qu'il faut indexer les modules nécessaires et obtenir des versions compatibles pour tous ces derniers afin de compiler le tout.
  - Une compilation réussie, fusse-t-elle longue[46], n'est pas synonyme de fin du problème. Des bugs[47] peuvent survenir à l'exécution sous émulateur (avec retour terminal et par ricochet origine du problème) ou pire dans un essai sur un appareil physique (pas de retour précis de la part de l'USB debug).
- Chaque librairie/application doit avoir son propre répertoire indexé par un script Gradle dans l'arborescence haute.
  - App est mon application et a besoin du module OpenCV pour fonctionner
  - Ceci est précisé dans son .gradle en haut de l'arborescence (OpenCV sera alors compilé en premier lieu)

46 Une compilation durait typiquement 1 minute avec un PC prêté par le CNRS et disposant de 48Go de RAM, ce qui était une contrainte forte pour une application relativement simple. Mais, pour une première compilation, il n'y a peu de chances que les modules nécessaires soient déjà installés dans l'Android Starter Developpement Kit, ce qui prend un certain temps pour les installer. Hélas bien souvent les ordres de grandeur des tailles des composants requis sont le Giga Octet.

47 Les fameuses « Java Exceptions », notamment qui forcent la machine virtuelle à s'arrêter (ex : bibliothèque native ayant reçu l'ordre d'être compilée sans être ajoutée dans l'exécutable faute de bonne version. Ceci provoque un arrêt immédiat de l'exécution sans trop d'explication, les fonctions « natives » n'étant qu'en phase d'essai sous Android Studio).

    - MAIS : les deux versions du SDK (« kernel Android ») et les deux versions du compilateur Gradle **doivent être compatibles ET identiques** (sans quoi, c'est l'erreur critique assurée).

- Un manifeste, fichier au format XML, indexant les ressources et les classes du module est présent dans chaque dossier.
    - MAIS : tout doit être compatible là encore et les droits de l'application sont scrupuleusement examinés[48].
- Les Classes quant à elles sont dans le répertoire Java parmi lesquelles :
    - Les classes « interface homme-machine » appelées **les activités** (dans le module App), elles appellent une ressource présentation (dans res/layout) qui est un fichier de mise en forme .xml de l'application. On peut manipuler les interfaces homme machine à la fois sur du code .xml et sur un éditeur de palette graphique, ce qui est bien pratique pour donner un corps ergonomique à l'application. Mais il se peut que les fichiers .xml appellent des bibliothèques du « kernel Android », et ces dernières DOIVENT être compatibles avec l'application. Par exemple, faire des affichages de type « slideshow »[49] dynamique de vidéo n'est pas de base dans la version 2 d'Android qui date déjà de plus de cinq ans. Notons que quelques méthodes classiques, initialisation, routine, et auto-générées (lorsqu'on déclare un bouton dans le .xml) doivent être présentes dans cette classe. Elles ne présentent par ailleurs pas de constructeur classique mais une fonction **void OnCreate** créant tous les objets nécessaires au fonctionnement de l'activité et associant leurs éléments de présentation respectifs.
    - Les autres classes (ressource), classiques au sens de Java et sans fiche de style. On y retrouve les notions de constructeurs, d'attributs, de partage (privé, public, protégé) du Java. Elles m'ont servi à interfacer les fonctions propres à Augustin (on crée un objet FaceTracking qui appelle lui-même OpenCV au lieu de tout appeler dans l'activité, ceci permettant de transporter plus facilement la librairie [ou classe] à plusieurs activités distinctes.
- **Src** sert à stocker toutes les ressources dont :
    - La mise en forme .xml (déjà traité)
    - Les fichiers intrinsèques générés avec droit de modification (ex : image en format JPEG émanant d'une capture d'image)
    - Les fichiers de base de données nécessaires à l'exécution de certains de mes algorithmes.
- **JnLibs** sert à stocker le code en format natif (C++) nécessaire au bon fonctionnement de openCV. Par ailleurs, il est nécessaire d'installer une **version compatible** du NDK (Native Developpement Kit, comprenant le compilateur C++) pour Android afin d'intégrer proprement le code de la bibliothèque OpenCV dans l'application.

48 Il faut se rappeler que l'ultime but d'un développeur Android est de publier son application sur l'App Store Google Play. Pour cela, il faut qu'elle satisfasse aux exigences de qualité requises par Google. Notamment, il faut préciser si l'on accède à des ressources de l'appareil physique tels que le WiFi, l'appareil Photo, la carte Bluetooth….

49 Un exemple de « slideshow » peut se trouver sur mon site personnel http://jcano.perso.ec-m.fr sur la page d'accueil. Ceci nécessite une certaine mémoire vive, difficilement supportable par les premiers smartphones.

c) Nécessité de choisir une autre plate-forme

Pour résumer la situation, la recherche de compatibilité ainsi que mes tentatives d'intégration du code natif m'ont pris aisément deux semaines. Je le rappelle ici, je n'avais aucune connaissance du développement d'application Android *a priori* . De ce fait, il a fallu que je me forme à la philosophie de développement.

Hélas, Android n'est pas facilement applicable au domaine de la R&D car il impose des contraintes normatives fortes. Je parle évidemment ici de compatibilité, mais aussi d'autorisation et de déclaration de « frameworks ». Ces derniers sont la norme en équipe de programmation mais en aucun cas pour des scientifiques considérant que l'informatique est un outil et un support pour des applications physiques. De surcroît, en robotique, les programmes changent fréquemment, ont des paramètres qui se règlent (de manière automatisée ou non) très fréquemment. Et comme tout système embarqué, la recherche de la complexité moindre des algorithmes ainsi que la gestion de l'allocation optimisée de ressources sont synonymes d'efficacité[50] en robotique. Il est clair que Android n'est pas adapté pour la phase de conception d'un robot. En revanche, une fois ce dernier conçu, cela peut être un attrait pour le client que de disposer d'une application sur son smartphone, mais la conception de cette dernière nécessite d'engager des experts dans le développement mobile[51] car ce n'est en aucun cas aujourd'hui à la portée de chacun.

Pour conclure, mon opinion quant à son emploi dans le domaine de la robotique est conforme en tout point à celui de M.Anjeaux[52], Android est trop jeune pour supporter des ressources natives efficacement et ne dispose pas de source de documentation sérieuse[53] à ce propos (à la date où ce rapport est publié).

2. *Vers une application multi-plateformes sous Java*

a) Une bibliothèque transportable

Cette dernière est codée et disponible sur le site du Wiki du Club Robot de l'école Centrale Marseille (http://wiki.centrale-marseille.fr/egab). La bibliothèque Augustrack est en fait composée des algorithmes présentées dans la partie II du présent rapport.

Changements notables :

- Trois classes contenant les trois Algorithmes, permettant une gestion mémoire au besoin. En effet, une instance de classe peut être détruite dès que le mode de tracking est quitté, libérant la mémoire.
- Transposable sur un grand nombre de plate-formes
  - Il suffit d'installer Java sur le micro-ordinateur choisi, que ce soit une carte portable (Raspberry, BeagleBoard…) comme sur un PC portable ou notebook.

50 Un algorithme allouant trop de mémoire sera lent, donc on aura une discrétisation temporelle des décisions et le robot se devra alors d'être ralenti pour ne pas devenir instable.

51 Et donc de disposer déjà du code et de l'avoir éprouvé et réglé sur une plate-forme plus ouverte.

52 Avec qui j'ai échangé mes impressions lors de mon stage à propos de Ubbo et de la reconnaissance des formes en robotique.

53 Les seules données que j'ai pu extraire venaient du site stackoverflow.com et n'étaient que des réponses d'internautes à des questions précises de développeurs, difficilement transposables à mon problème dans la plupart des cas.

- Java est indépendant du Support : il suffit de créer un script (seule partie dépendante du support) indiquant d'exécuter la commande Java sur la classe principale pour lancer la machine virtuelle. Et cela se fait en quelques lignes de code et sans beaucoup de conflits de versions de Java, par opposition avec le triplet SDK, NDK et Gradle.

- Code réglable, modifiable et « dé-buggable » facilement. Le temps d'une compilation en archive Jar est très court et l'installation/configuration de OpenCV est extrêmement détaillée sur le site de Polytechnique Turin (voir bibliographie).
- Utilisation de l'interface de développement Eclipse
  - Interface plus légère, mais qui comporte moins d'aide que Android Studio (supporté par l'IDE InteliJ).
  - Cette dernière est bien documentée, surtout en ce qui concerne OpenCV, la majorité de la communauté développe sur Eclipse et des modules ouverts sont facilement téléchargeables sur le Web.
  - Dispose d'une communauté ouverte, qui a tendance à améliorer l'interface elle-même. A l'inverse d'Android Studio, qui met du temps à corriger ses défauts du fait du « versionning » commercial.

b) Quel support choisir ?

Comme dans toute application embarquée, la question du support est primordiale. Je pense qu'une Raspberry Pi, fonctionnant sur Linux pourrait être une solution autant peu coûteuse qu'efficace.

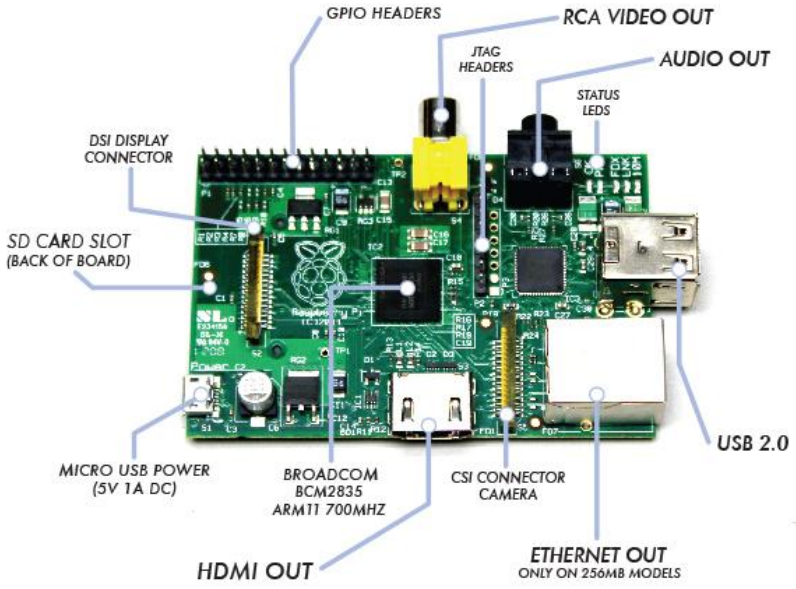


Illustration 17: Une Raspberry Pi (Source : pcmag.com)

Motivations :

- Il est facile d'installer les environnements Java (Java Developpement Kit et Java Runtime Environement) nécessaires à l'exécution de notre programme sous OpenCV. De même, l'archive contenant OpenCV peut être stockée directement à Bord et non compilée permettant de modifier l'algorithme au sein même du système embarqué (supprimant la phase fastidieuse et chronophage de la compilation).
- On peut connecter aisément **des** Webcams sur les bus sériels de la Raspberry. Ceci permettra peut-être de couvrir une plus large zone à moindre frais.
- La consommation d'énergie est moindre en comparaison avec celle d'une tablette.
- Cela est déjà utilisé en robotique, et une forte communauté s'est développée autour de cet environnement depuis quelques années. Notamment, il existe de nombreux moteurs d'inférences immédiatement installables, permettant des applications robotiques.
- L'alliance Raspberry-Arduino est connue, éprouvée et fortement usitée par tous les roboticiens. Des interfaces graphiques permettent de repérer les bugs et de lire les valeurs de capteurs à souhait, permettant le réglage du robot de manière aisée.

3. Mes conseils pour la reconduite du projet

NB : Cette section pourrait éventuellement servir à la reconduite du projet.

a) Quelques modifications au niveau Hardware

Comme évoqué précédemment, l'implémentation de nouveaux composants au sein de Ubbo serait souhaitable dans le but de l'améliorer :

➔ Les capteurs ultrason anti-collision (*Cf. partie II)*

- Les quatre à la place des quatre capteurs Sick déjà présents.
- Les quatre nouveaux sur le mât, avec leur support incliné (Je pense qu'il faudra l'usiner dans un FabLab, au vu de sa spécificité) .

➔ Les câbles d'alimentation et d'émission/réception des capteurs précités

- Un branchement sur la carte d'alimentation sera à établir.
- Des branchements sur la carte d'acquisition Arduino Mega seront également à réfléchir, au vu du nombre de fils mis en jeu (8 capteurs, donc 8 fils de déclenchement TRIGGER ainsi que 8 fils de retour [54])

➔ Des caméras webcam

- Une webcam pourrait être installée sur la base roulante afin de faire des lectures au sol : line-tracking, suivi des carrelages, détection d'escaliers…
- Une autre sur le mât, devant se substituer à celle déjà présente sur la tablette dans des applications de tracking de personnes.

54 Penser peut-être à un déclenchement synchrone des capteurs par batteries de 4, ce qui nécessiterait seulement 2 fils contre 8 en branchant les batteries en parallèle.

➔ Une vision infra-rouge

- On pourrait penser à une vision nocturne pour opérer dans des endroits sombres (ex : chambre noire dans l'Intitut) qui pourrait se révéler opportune pour des applications telles que la surveillance d'expériences en photonique par exemple (Augustin n'est pas sujet à l'éblouissement).
- Ceci nécessiterait un collier de diode IR autour des caméras ainsi qu'une carte d'activation avec transistor et alimentation à découpage adaptée pour permettre une illumination plus puissante et à souhait. Ces derniers se trouvent dans le marché facilement, mais ils peuvent être routés[55] et gravés aisément du fait de leur simplicité (combinaison de transistors, hacheur[56] et résistances).

➔ Un (petit) circuit de détection de décharge, adapté à la (ou aux) batterie(s), si elles venaient à changer de type.

➔ Une liaison sérielle entre les deux cartes Arduino et Raspberry et non le Bluetooth (permet de s'affranchir de problèmes dans le cadre d'un prototype.

➔ Un boîtier sur la base roulante (pour Augustin) rendant l'accès à la carte aisé (pour par exemple insérer une carte SD, nécessaire au chargement du système d'exploitation, de Java et de OpenCV sur la raspberry ; ou encore pouvoir ajouter une caméra supplémentaire).

b) Quelques idées de possibilités de Software

En découlent plusieurs pistes qui tendraient à améliorer Augustin (ou plus généralement Ubbo) :

➔ Partie « Bas-niveau », *ie codes sur l'Arduino Mega*

- Robe ultrasonique protectrice (*Cf. Partie II)*
- Retour des capteurs (implémenté par Axyn mais pas encore exploité dans la version d'Augustin) à destination de la Raspberry Pi.
- Module de commande par liaison sérielle de la base roulante et du servomoteur de la « tête »(en calquant celui qui est déjà codé pour le Bluetooth) de l'Arduino par la Raspberry Pi).
- Implémentation du code pour activer les diodes infra-rouge de vision nocturne (si il y a lieu) et le réglage du rapport du hacheur en vue de les alimenter.

➔ Partie « Haut-niveau », *ie codes sur la Raspberry PI*

- Création d'un petit moteur d'inférences[57] permettant de donner de l'intelligence

55 Terme du jargon électronique évoquant le « routage » des pistes de cuivre durant la conception assistée par ordinateur du circuit imprimé.

56 Jargon électronique désignant le hachage de tension utilisé pour les moteurs mais aussi les diodes IR. En effet, leur tension nominale est faible mais pour obtenir une illumination forte sans endommager par surtension ces dernières, on peut utiliser une tension plus élevée mais arrivant par intermittence aux bornes de la diode, empêchant par ce biais la destruction de la diode. Explication du montage sur Wikipédia : https://fr.wikipedia.org/wiki/Hacheur

57 Terme désignant les moteurs d'intelligence artificielle. Ces derniers fonctionnent à partir de règles et de faits. En appliquant les règles aux faits, les faits sont modifiés et donc on arrive par itérations successives à un système d'intelligence artificielle. Pour en savoir plus, j'ai laissé un exemple que j'ai co-réalisé à Centrale il y a un an : il se

artificielle au robot en lui donnant des règles à suivre.

- Création d'une petite interface pour Raspberry PI, permettant de donner des instructions au robot à distance :
  - Parmi les possibilités, on peut utiliser des commandes vocales, nous en avons déjà réalisé sous ce support au FabLab Marseille, association de prototypage dont je fus Vice-Président chargé de l'électronique en 2015.
  - Une télécommande par smartphone est possible, des modules Bluetooth et des logiciels clefs en main (mais libres de droits, conformément aux usages de la communauté) sont disponibles pour Raspberry PI.
- Implémentation de ma bibliothèque de reconnaissance de visage humain et des méthodes de tracking intégrées. *Cf. Partie II*
- Création possible d'une base de données sous openCV de clichés d'utilisateurs du robot. Certains codes sont très efficaces sur le tutoriel de Polytechnique Turin, permettant de discriminer un petit groupe de personnes. Une fois l'utilisateur reconnu, il serait alors possible d'émettre une bande sonore saluant ce dernier d'une façon personnalisée. Ceci pourrait être intéressant à des fins ludiques mais aussi éducatives (compréhension de la classification en cascade, perceptron….).
- Implémentation et amélioration de l'algorithme de line-tracking au sol déjà développé par Axyn, un balisage dans les couloirs de l'Institut pourrait permettre par exemple un tour guidé par Augustin des équipements du laboratoire. Cela ferait appel à la caméra près du sol.

c) <u>Vers une interaction plus forte avec Axyn Robotique</u>

Créer une plus forte dynamique entre Axyn, entreprise dynamique dans le monde de la robotique des environs de Marseille et entre les compétences phocéennes, est pour moi très enrichissant en termes de compétences. Mais au-delà de l'attrait personnel, je pense qu'ouvrir la voie à de nouvelles opportunités, à des Centraliens dans les domaines de la robotique en R&D et de la robotique de service, est extrêmement adapté.

Donc, nouer des liens entre E-Gab, Club de Robotique de Centrale Marseille[58] mais avant tout laboratoire de R&D étudiant dans les domaines des drones et des robots autonomes terrestres, me paraît être évident. Car E-Gab regorge chaque année d'élèves-ingénieurs désireux d'accomplir des stages passionnants, professionnalisants et également utiles dans les domaines concernés. Mus d'une forte interaction avec des composantes phocéennes du CNRS telles que l'Institut Fresnel (Robotique), le LIF[59] (Intelligence Artificielle) ou encore l'ISM[60] (Drones, pour le site de Lumigny et robots dans le domaine biomédical pour le site de Saint Charles) , les adhérents de E-Gab aspirent à s'engager davantage dans le projet que j'ai initié avec l'aide, le soutien et le crédit de l'équipe Epsilon de l'Institut Fresnel[61].

De plus, M.Anjeaux, dirigeant de Axyn, est personnellement un partenaire de confiance du

trouve dans les annexes du présent rapport.

58 Dont je fus président pour la session 2015-2016.

59 Laboratoire Fondamental d'Informatique.

60 Institut des Sciences du Mouvement.

61 Je remercie chaleureusement Ronald AZNAVOURIAN et Sébastien GUENNEAU qui ont rendu possible ce stage, initiant la démarche qui a abouti à un tel résultant.

club de robotique. Ce dernier nous ayant aidés matériellement pour la Coupe de France de Robotique 2016 en permettant un partenariat avec la boutique de robotique Arobose, qu'il dirige également.

4. Bilan personnel de mon stage

Tout d'abord, je tiens à souligner que l'obtention d'un tel stage est due à un concours de circonstances favorables. En effet, le projet est né d'une discussion en marge de la Journée de la Recherche à Centrale Marseille en 2015. Mais force est de constater que ceci a permis de mener un projet ambitieux et profondément valorisant.

Du point de vue des compétences techniques, ce dernier m'a appris à développer des applications sous Android, à faire du traitement d'images sous openCV[62] et à comprendre la conception d'un robot du point de vue d'un industriel. Quant aux compétences humaines, travailler dans une équipe de recherche m'a permis de comprendre aisément le fonctionnement d'un laboratoire en y étant acteur. D'une manière évidente, les contacts humains furent nombreux et j'ai eu extrêmement de plaisir à converser avec les collègues de laboratoire dont certains, paradoxalement, furent jadis[63] mes professeurs ! L'accueil fut très cordial de la part des membres de l'équipe et plus généralement de l'Institut : j'ai eu une totale autonomie sur le projet, tout en tenant informé,de manière rapprochée, mon tuteur Ronald Aznavourian, ingénieur en informatique, qui me fut d'une grande aide.
Certes, l'application sur Android est incomplète, ce qui ne correspond pas au cahier des charges initial. Toutefois, notons que les prolongements possibles du projet et surtout les contacts avec le milieu industriel n'y figuraient pas *a priori* et c'est une grande fierté pour moi d'y avoir contribué.

Il ne reste plus qu'à souhaiter bonne chance à mon potentiel successeur, Farid HOUDI[64], pour l'édition 2017 du projet, qui pourra donner, je l'espère, encore plus d'autonomie à Augustin.

62 Je ne connaissais que Matlab sur lequel on enseigne la Reconnaissance des Formes et le Traitement du Signal à Centrale.
63 Le semestre précédant le stage...
64 Farid HOUDI est également l'actuel Président de E-Gab.

V.Annexes

Annexe 1 : Code des méthodes de reconnaissance des formes & de tracking :

http://wiki.centrale-marseille.fr/egab

Annexe 2 : Informations pratiques relative à mon stage

http://jcano.perso.ec-m.fr/fresnel

Annexe 3 : Code-source de l'application modifiée et documentation

Ces derniers seront en la possession de Sébastien GUENNEAU, Ronald AZNAVOURIAN et de moi même à l'issue du stage. Pour des raisons de confidentialité et de taille de stockage (supérieure à 2GB) , je ne peux pas les mettre sur un site mobile.

Cependant, j'ai réussi à en mettre une partie sur le Wiki précédant par le biais d'un Google Drive : http://wiki.centrale-marseille.fr/egab

Annexe 4 : Exemple de moteur d'inférence

En 2015, j'ai crée avec un collègue du Club Robotique, Armand SIBILLE, un moteur d'inférence libre de droit sous Java. On l'avait appelé ARSTIN : cela pourrait inspirer quelques lecteurs donc je donne le répertoire Internet sur lequel j'ai documenté et mis les code-source :

http://jcano.perso.ec-m.fr/arstin

Annexe 5 : Site du Club Robotique de l'ECM | E-Gab

Je fus président de cette association en 2015-2016, s'approchant du fonctionnement d'un laboratoire de R&D . Pour en savoir plus, voici le site Internet de cette dernière :

http://assos.centrale-marseille.fr/clubrobot

VI.Bibliographie & références

- Manuel de l'utilisateur du robot UBBO, Axyn Robotique
- Notes sur les protocoles de communication intrinsèques à UBBO, Axyn Robotique
- http://arduino.cc , Compléments sur des bibliothèques spécifiques utilisées
- http://github.com, Références diverses en termes de programmation
- www.oracle.com, Références sur le langage Java
- http://developer.android.com, Documentation sur l'IDE Androïd Studio & la conception d'application sur ce Système d'exploitation.
- Datasheet des capteurs SICK à lumière rouge, SICK industries
- Opencv.com & infoRobotique.com : références sur le tracking en utilisant la bibliothèque de traitement d'images OpenCV
- Algorithme de Tracking pour un objet circulaire rouge : Adapté de : https://ratiler.wordpress.com/2014/09/21/detection-et-suivi-dun-objet-colore/
- Documentation d'Android Studio
- Tracking de visage : algorithme incomplet mais m'ayant beaucoup aidé pour l'écriture du mien sous java dans Eclipse : **http://answers.opencv.org/question/12395/real-time-face-tracking-java/**
- http://romanhosek.cz Programme de tracking de visage adapté à la bibliothèque d'openCV d'Androïd, fort similaire au précédent.
- http://opencv-java-tutorials.readthedocs.io/en/latest/03-first-javafx-application-with-opencv.html
  Tutoriel de Polytechnique Turin sur l'utilisation de OpenCV en Java (ce dernier est très complet et je le recommande vivement à ceux qui veulent se lancer dans le traitement d'image avec OpenCV)