**MASTER Ingénierie des Systèmes Industriels et des Projets**

SPÉCIALITÉ : SYSTÈMES DYNAMIQUES ET SIGNAUX

Année 2009 / 2010

**Thèse de Master SDS**

Présentée et soutenue par :

Clément Aubry

le 16 juillet 2009

Au sein de l'Institut des Sciences et Techniques de l'Ingénieurs d'Angers

---

# Localisation de robots humanoïde par des approches ensemblistes

---

JURY

| | | |
|---|---|---|
| Président : | Laurent Hardouin | Responsable du Master Systèmes Dynamiques et Signaux de l'université d'Angers et de l'équipe Modèles Dynamiques et Système du Lisa |
| Examinateurs : | Sébastien Lagrange | Maître de Conférences |
| | Bertran Cottenceau | Maître de Conférences |

Directeur de thèse : **Mr Sébastien Lagrange**

Laboratoire :

62, avenue Notre Dame du Lac
49000 Angers

# Table des matières

# 1 Introduction

La plateforme utilisée pour traiter ce problème de localisation est le robot humanoïde Nao. Ce robot 100 % français est une plateforme idéale pour la recherche. Il est équipé de 2 caméras, 4 micro, 2 haut-parleurs, 2 émetteurs / récepteurs ultrasons, 1 émetteur et un récepteur infrarouge, 1 centrale inertielle, 2 bumpers et 23 moteurs équipés de capteurs à effet hall qui lui confèrent 25 degrés de liberté (ddl). Vous trouverez en annexe de ce rapport un guide d'utilisation du robot Nao.

Un premier rapport (établi en mars 2010) détaillait les recherches bibliographiques liées à ce sujet, la première partie de ce rapport sera donc un rappel et un complément bibliographique du premier rapport. Cette partie mettra aussi en évidence les solutions retenues et les raisons de ces choix.

La deuxième partie sera consacrée à l'implémentation de la solution retenue, aussi bien sur la partie localisation que sur la partie vision, les deux grands thèmes de cette thèse.

L'étude du sujet se fera en deux parties, la partie localisation et la partie vision.
La première partie permettra la localisation et la prise en compte de l'erreur de mesure sur angles entre chaque amers et pourrait nous permettre d'ajouter des imprécisions sur d'autres paramètres(tels que la position des amers ou même la position du repère absolu). C'est cet outil mathématique qui garantira le résultat obtenu par la méthode de localisation.
La seconde partie traitera du travail fait au niveau de la reconnaissance d'amer, avec des recherches sur des techniques de reconnaissance assez souples pour fonctionner sur la plateforme embarquée et assez fiables pour permettre un bon fonctionnement de la méthode dans toutes les situations.

## 1.1 La localisation

Cette partie est un rappel des méthodes de localisation énoncés plus précisément dans le rapport bibliographique :

**Localisation par odométrie** : L'odométrie est utilisée pour connaître la position d'un robot mobile par rapport à sa dernière position connue. En général, cette technique n'est pas utilisée seule puisqu'elle génère une grande imprécision de par la nature incrémentale des erreurs.

**Localisation par approches probabilistes** : Dans [GBFK98], Gutmann *et al.* comparent des méthodes qui permettent de décrire la posture d'un robot à partir de distributions probabilistes.

**Localisation par images de profondeur** : Les images de profondeur, produites par télémétrie laser, par radar ou par stéréo vision, sont utilisées pour déterminer la position d'un robot ou pour l'établissement d'une carte. Ces données sont souvent utilisées dans le cadre des SLAM [1]. Nous trouverons des exemples de ces méthodes dans [BB96, Lar03, RLDL05]. Kieffer et *al.* dans [KJWM00] utilisent des images de profondeur traitées par une technique d'estimation utilisant l'analyse par intervalles pour trouver la posture du robot. L'utilisation de l'analyse par intervalles est un point important puisqu'elle apporte une garantie

1. Simultaneous Localization And Mapping

quand à la posture du robot.

**Localisation par la vision** : Ces méthodes vont extraire des images prises par le robot. Betgé-Brezetz utilise, dans [BB96], la reconnaissance d'amers pour compléter sa localisation. La publication de Betke et Gurvits [BG97] nous explique quand à elle une méthode de localisation basée sur la reconnaissance d'amers avec l'utilisation du principe de triangulation pour repérer la position du robot.

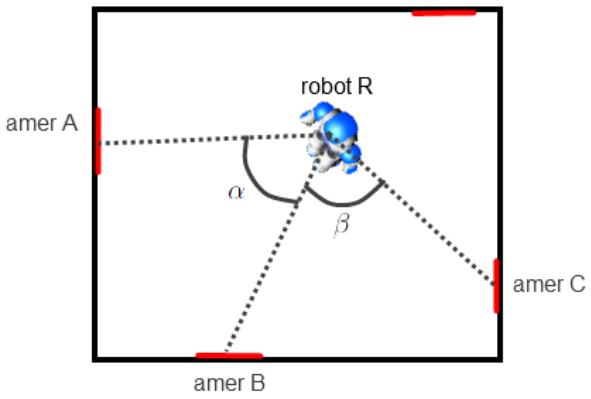



FIGURE 1 – Schéma de principe

La technique de localisation retenue pour l'implémentation sera une localisation par triangulation comme celles utilisée par Betgé-Brezetz dans [BB96] ou celles utilisées pour la navigation maritime. La figure 1 nous donne un schéma de principe de la méthode de localisation retenue.

Le robot mesure les angles $(\alpha, \beta)$ entre les amers $A$, $B$ et $C$ puis construit les cercles circonscrits aux triangles $ABR$ et $BCR$ , illustré sur les figures 2(a) et 2(b). La position du robot est l'intersection des cercles $\mathcal{C}_1$ et $\mathcal{C}_2$.

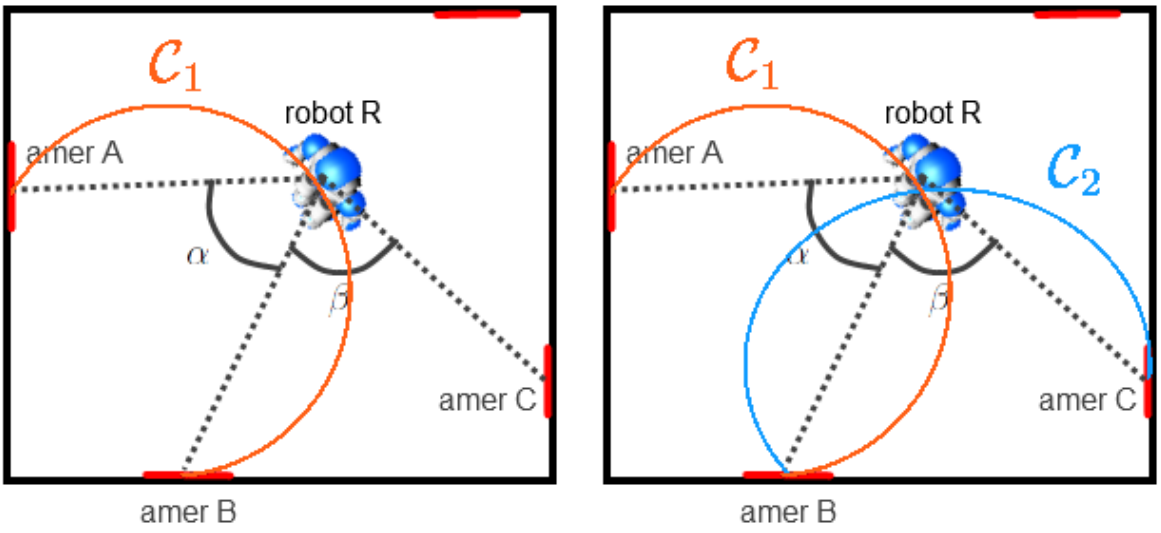
Je détaillerais dans la partie implémentation les choix retenus ainsi que l'utilisation de cette méthode avec utilisation de l'analyse par intervalles, l'apport le plus important pour cette thèse.

## 1.2 La vision

La partie vision a été la plus problématique dans ce sujet. La puissance de calcul du robot ne lui permet pas d'utiliser des techniques de reconnaissance d'objet avancées comme les SIFT ou les SURF (il serait possible d'utiliser ces techniques mais les développements et recherches associées seraient trop important pour ce sujet). Ces techniques sont actuellement très utilisées pour leurs propriétés d'invariance en rotation et changement d'échelle, elles supportent aussi les changements de points de vue.

Il faut donc trouver une technique qui nous permettra de repérer les amer rapidement et de façon fiable. Il faudra aussi être capable de différencier les amers.

Compte tenu de la qualité de la caméra et de la puissance de calcul disponible, plusieurs méthodes se basant sur l'utilisation d'amers lumineuses actives ont été envisagées. Voici les




(a) Construction du cercle $\mathcal{C}_1$ (b) Construction du cercle $\mathcal{C}_1$

FIGURE 2 – Construction des cercles circonscrits pour la localisation

différentes possibilités :

- Amers clignotantes différentiables par leur fréquence de clignotement.
- Amers lumineuses de couleurs différentes.
- Amers lumineuses commandées à distance.

Ces différentes méthodes se basent sur un principe simple, si le robot prend une première photo quand la source lumineuse est éteinte puis une seconde avec la source lumineuse allumée, la différence de ces deux images doit lui permettre de localiser la source lumineuse.

La première idée, repérer les amers lumineuses par leur fréquence de clignotement n'a pas été choisi puisqu'il aurait fallu synchroniser chaque amer avec le robot. De plus, le temps de réponse de la caméra n'étant pas certain, il aurait été difficile d'assurer la différentiabilité des amers.

En ce qui concerne les sources lumineuses de différentes couleurs, les essais n'ont pas été révélateurs puisque, face aux changements de luminosité ambiant, les amers n'étaient pas toujours détectées.

La troisième solution, amers commandées à distance, à été bien plus prometteuse. En effet, si l'on peut commander chaque amer indépendamment, la question de différentiabilité ne devrait plus se poser.

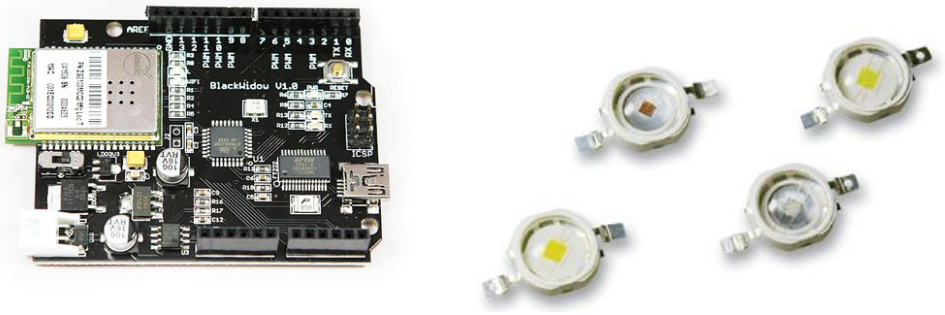
Différentes techniques de traitement d'image comme la recherche de contours par les transformées de Hough ou la segmentation ont été testées à partir des images traitées mais les résultats n'étant pas probants, je n'ai pas continué sur cette piste.

# 2 Méthode de localisation par l'approche ensembliste

## 2.1 Les amers

### 2.1.1 Construction des amers

J'ai choisi d'utiliser des amers commandées par le robot. Après quelques réflexions et au vues des technologies de communication disponibles sur le robot (Wifi, infrarouge), mon choix s'est porté vers des modules Wifi Arduino : il s'agit d'une carte à microcontroleur Atmel équipée d'un module Wifi appelée BlackWidow (voir figure 3(a)).
Cette carte sera équipée d'une source lumineuse à Led (voir figure 3(b)). J'ai choisi d'utiliser des Led fortes puissances (>1W) avec un flux lumineux de 65lm, afin qu'elles puissent être visibles de loin. Ainsi, la méthode est invariante aux changements d'échelles. Cette particularité sera utile lorsque le robot sera loin des amers qu'il devra repérer. Ces Led possèdent aussi un grand angle visuel, 130°qui confère à la méthode une invariance aux changements de point de vue. Ce qui permettra au robot de pouvoir reconnaitre les amers quelque soit l'angle d'observation.
Ces fonctionnalités se rapprochent (à moindre coût) de celles des traitement type SIFT/SURF que nous avions imaginés au lancement du projet.




(a) BlackWidow 1.0 (Arduino Wifi) (b) LED forte puissance

FIGURE 3 – Construction des amers

### 2.1.2 Communication robot / amers

La communication entre les amers et le robot se fera par l'intermédiaire d'une infrastructure Wifi. Le robot enverra les ordres d'allumage et extinction des Leds par des sockets. Ce protocole de communication type client/serveur sera basé sur la pile TCP/IP. Les cartes BlackWidow auront le rôle de serveur alors que le robot sera le client.

Au niveau des BalckWidow, un programme simple créera la socket puis lira en permanence les ordres arrivant sur le buffer d'entrée. La figure 4 illustre la communication entre le robot et une amer.

Comme on peut le voir sur la figure 4, la communication établie entre une blackWidow et le robot est toujours active, en effet, la connexion relativement lente en Wifi, n'est effectuée qu'une fois puis la blackwidow "écoute" en permanence le client dans l'attente d'une requête. Cela nous permet d'accélérer le temps de réponse des amers.

Les premières versions de l'implémentation utilisaient le protocole de communication classique en communication (connexion/déconnexion a chaque envoi) mais le besoin de rapidité de la

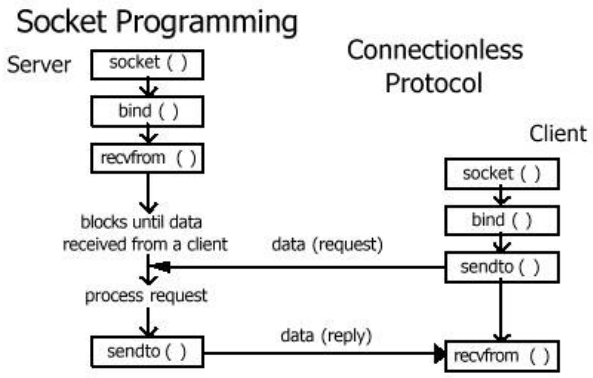



FIGURE 4 – Principe de communication socket

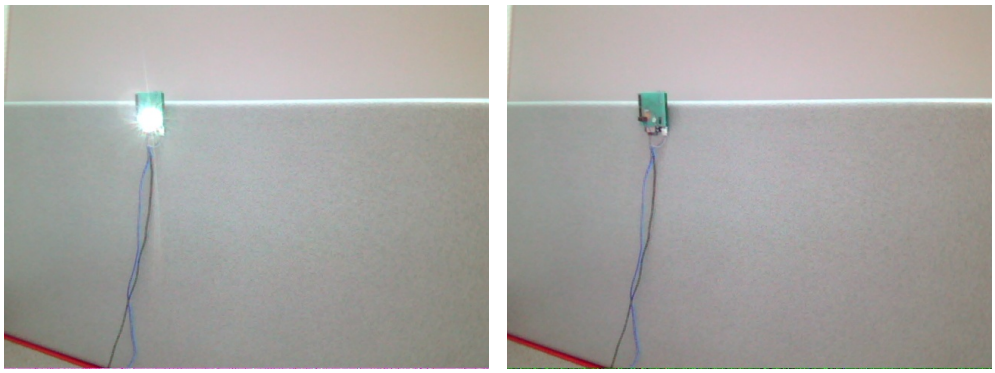
méthode nous a fait choisir cette solution "Connectionless".

## 2.2 Recherche et reconnaissance d'amer

Concernant la partie vision, j'ai utilisé des amers lumineuses commandées indépendamment. Plusieurs algorithmes ont étés testés mais tous se basent - pour la reconnaissance des amers - sur les différences d'images amers allumées / amers éteintes. J'expliquerai dans une premier temps ce principe ainsi que le travail fait sur les images pour le repérage des points lumineux. Puis, je parlerais les algorithmes de détection explorés pour en venir à celui implémenté actuellement.

### 2.2.1 Différences d'images - repérage de "blobs"

Comme la méthode de reconnaissance d'amer ne doit pas consommer en terme de puissance de calcul, il faut utiliser une méthode simple qui sera chargée de capturer deux images puis d'en faire la différence. La figure 5 nous montre une première image Led éteinte et une seconde Led allumée.

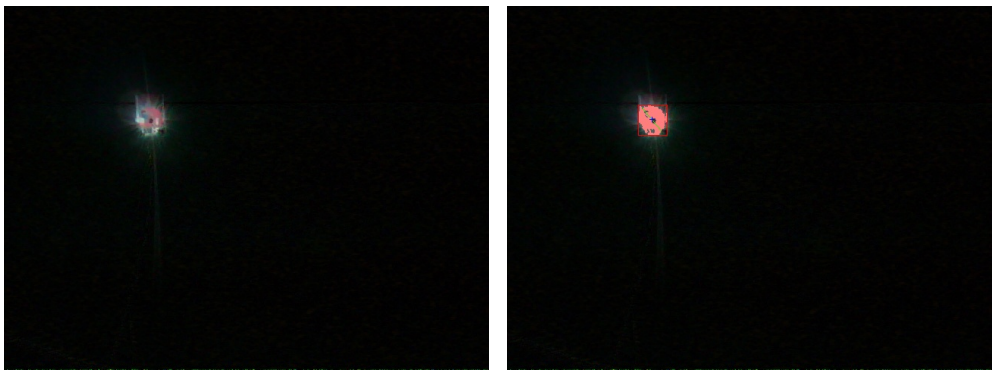


(a) Amer éteinte (b) Amer allumée

FIGURE 5 – Capture d'images

La différence de ces images est ensuite binarisée avant d'être envoyée vers une étape d'analyse des blobs. Un blob - terme anglais signifiant "tache" ou "goutte" - est une caractérisation des régions connexes d'une image. En d'autres termes, un blob sur un image binarisée regroupe les

pixels blancs adjacents d'une image. La figure 6(b) nous montre un repérage de blobs sur une image binarisée (figure 6(a)) issue de la différence des images de la figure 5.

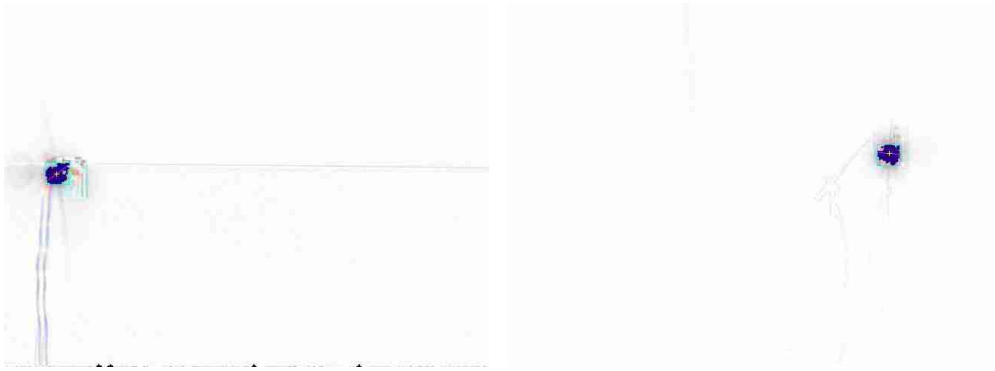


(a) différence d'images binarisée (b) Repérage des Blobs

FIGURE 6 – Repérage d'amer

L'exemple illustré par les figures 5 et 6 nous montre le procédé de repérage d'amer lorsqu'une amer allumée est présente. Cette technique appliquée à une image où il n'y aurait pas d'amer ne devrait pas rendre de blob puisque la différence entre deux images identiques est nulle.
Cependant, l'étape de binarisation nécessite un seuil, c'est pourquoi cette étape est sensible. En effet, si le seuil est trop bas, on obtient plusieurs blobs dans l'image correspondant à des légers changements d'intensité lumineuse ou par exemple des reflets sur un mur. Au contraire, si le seuil est trop haut, les blobs correspondant aux amers ne seront pas repérés.

Pour assurer le repérage des amers, plusieurs "sécurités" ont étés mises en place. La première vient de la construction des amers. Comme l'intensité lumineuse des Leds est importante, le seuil peut être fixé assez haut pour éviter les outliers [2]. Cette première sécurité permet d'éviter certains outliers mais il en subsiste encore, ces outliers sont en général issus de reflets ou bien d'effets de bord des images. La figure 7(a) montre un repérage de blobs avec des outliers sur le bord inférieur de l'image (les couleurs ont étés inversées pour plus de visibilité).



(a) avec outliers (b) sans d'outliers

FIGURE 7 – Repérage de blobs

Ces derniers outliers qui ne peuvent pas être évités vont être traités après la construction des blobs. En effet, la librairie choisie pour le traitement des blobs (je reviendrais plus tard sur les choix logiciels) propose une fonction de filtrage des blobs en fonction de leurs aires. Comme ces

2. comprendre les blobs non désirés mais repérés

outliers sont en général de taille remarquable - petits pour des pixels isolés ou grand pour un fort changement d'intensité lumineuse ambiant - ils peuvent facilement être évités par filtrage d'aires. La figure 7(b) montre un repérage de blobs sans outliers.

A présent, les blobs sont récupérés et une information utile peut en être extraite - leurs dimensions et leurs centres - cette technique est utilisée pour que le robot puisse repérer les amers dans son environnement.

### 2.2.2 Stratégie de mouvement pour la recherche d'amers

Plusieurs algorithmes ont étés testés, je vais en détailler deux qui ont étés les plus probants. Ces deux solutions se basent sur un principe de balayage (de "scan") de l'environnement.
L'articulation de la tête du robot lui autorise un mouvement de $-119.5°$ à $+119.5°$. L'angle d'ouverture horizontal de la caméra est de $46.4°$. Il est possible, sans modifier la position du robot de balayer un angle de $(119.5° * 2) + 46.4° = 285.4°$. Pour assurer une certaine robustesse par rapport aux effets de bords des images et pour garantir que tout l'environnement possible à bien été visualisé, j'ai choisi de rechercher des amer tous les quarante degrés. L'environnement est donc balayé en 6 images (car $(119.5° * 2)/40 = 5.97$). Il faut aussi prendre en compte, pour toutes les techniques utilisées, qu'une photo ne peut pas être prise directement après un mouvement de la tête, il faut attendre que cette dernière se stabilise (environ 400ms) pour que la photo ne soit pas bruitée.

**Première méthode** La première implémentation de la recherche de blobs se basait sur un principe simple, si on voit une des amers, on les test toutes indépendamment pour savoir laquelle prendre en compte. Pour chacun des 6 angles de recherche, Le robot prend une image amers toutes éteintes puis une autre amers toutes allumées et retourne une liste contenant les informations sur les blobs. Si un blob à été détecté, on recommence l'opération pour comparer les résultats en allumant tour à tour les amers et ainsi différencier laquelle à été repérée si le blob trouvé n'est pas un outlier. Dans le cas ou cette phase de différentiation des amers ne retourne aucun blob similaire, nous pouvons dire que le blob repéré durant la première phase est un outlier.
Cette méthode est très efficace puisqu'un changement d'intensité ambiant n'altérera pas la recherche de blobs, la différence étant calculée a chaque fois sur deux images très rapprochée dans le temps.

L'inconvénient de cette méthode vient du nombre de photos à prendre. Il faut compter deux prises d'image par angle de vue plus un calcul de blob (temps d'exécution non négligeable), à ces images s'ajoute deux autres photos et un autre calcul de blob par amer, si l'on a potentiellement trouvé une amer. Par exemple, lors de la première prise de vue, si l'on a repéré une amer, on devra prendre $(2 * n)$ (avec $n$ le nombre d'amer restante à détecter) photos pour repérer l'amer en question. Imaginons que nous recherchons 3 amers, le robot aurait alors pris 8 photos et 4 calculs de blobs.

Le temps nécessaire à une prise d'image est d'environ 50ms pour une résolution de 640*480 pixels d'une image en noir et blanc (configuration utilisée dans l'implémentation). Ajouté au temps d'allumage des leds (90 ms en moyenne, le temps de communication via le Wifi ne pouvant être déterminé) et au temps de calcul des blobs (environ 50 ms aussi), la méthode demande entre 35 et 45 secondes avant de repérer toutes les amers.

**Deuxième méthode** La deuxième méthode a été mise en place pour palier au problème de rapidité de la première. Il a fallut réduire le nombre de photos.

Le scan se déroule donc plus simplement. A chaque angle de prise de vue, le robot prend une image led éteinte puis une image led allumée pour chaque amer restante à détecter. Soit $n+1$ images par point de vue.

La différence est calculée pour chaque image "led allumée" à partir de la première image "led éteinte", on compte donc $n$ calculs de blobs. Dans le cas ou un blob est repéré, il faut garantir que ce blob est bien une amer, on prend donc une dernière photo avec l'amer en question allumée puis de nouveau un calcul de blob pour assurer le reconnaissance. Dans notre exemple précédent, il fallait détecter 3 amers, les 8 photos et 4 calculs de blobs sont réduits à 5 photos et 3 calculs de blobs.

La deuxième technique donne des résultats en détection similaires à la première méthode mais dans un temps plus réduit. En effet, dans le meilleur des cas, la totalité des amers (pour 3 amers recherchées) est détectée entre 8 et 30 secondes et fonction des conditions de l'expérience.

### 2.2.3 Calcul de l'angle visuel entre deux amers

A ce stade de la lecture, les images sont prises avec un angle de vue connu et les coordonnées des amers dans ces images sont connues. Il faut extraire des angles visuels entre les amers pour les fournir à la partie localisation.
Par exemple, supposons que :

- L'amer A a été repérée pour un angle de la tête de $-40°$ et le blob qui représente l'amer a pour centre $(513; 243)$ dans le repère caméra

- L'amer B a été repérée pour un angle de la tête de$80°$ et le blob qui représente l'amer a pour centre $(131; 203)$ dans le repère caméra

- On cherche l'angle$\alpha$ entre les amers

Le repère de la tête étant axé avec celui de la caméra, on pourrait dire que l'angle observé entre les deux amers serait approximativement de $80-(-40) = 120°$ mais les amers ne sont pas forcément situées au milieu de l'image, ou l'on peut projeter le repère caméra. La figure 8 illustre bien cet exemple.

On sait que la caméra à un angle d'ouverture horizontal de $46.4°$et vertical de $34.8°$. On connait aussi les dimensions de nos images $640 * 480$ pixels. Donc, 1 pixel sur l'image correspond à $46.4/640 = 0.0725°$ en vertical et $34.8/480 = 0.0725°$ en horizontal. On sait aussi que le centre de l'image est situé en $(320, 240)$ .
L'angle d'observatoin de l'amer A sera appelé "angleA". L'angle d'observatoin de l'amer B sera appelé "angleB". On peut donc corriger les angles de cette manière :

$$\text{angleA} = -40 + (320 - 513) * 0.0725 \qquad \text{angleA} = -53.9925° \tag{2.1}$$

$$\text{angleB} = 80 + (320 - 131) * 0.0725 \qquad \text{angleB} = 93.7025° \tag{2.2}$$

$$\text{Soient : } \alpha = \text{angleB} - \text{angleA} \tag{2.3}$$

$$\alpha = 147,695° \tag{2.4}$$

Si ce traitement est effectué non plus sur le centre des amers mais sur les coordonnées min et max du blob, on obtiendra un encadrement de l'angle désiré. Par exemple, sur la figure 8, on pourrait trouver l'angle $\alpha = [\underline{\alpha}; \overline{\alpha}]$

C'est de cette façon que l'on obtient l'angle sous forme d'intervalle avec une précision relative à la distance avec laquelle on observe l'amer (entre $3°$et $5°$en fonction de la distance).

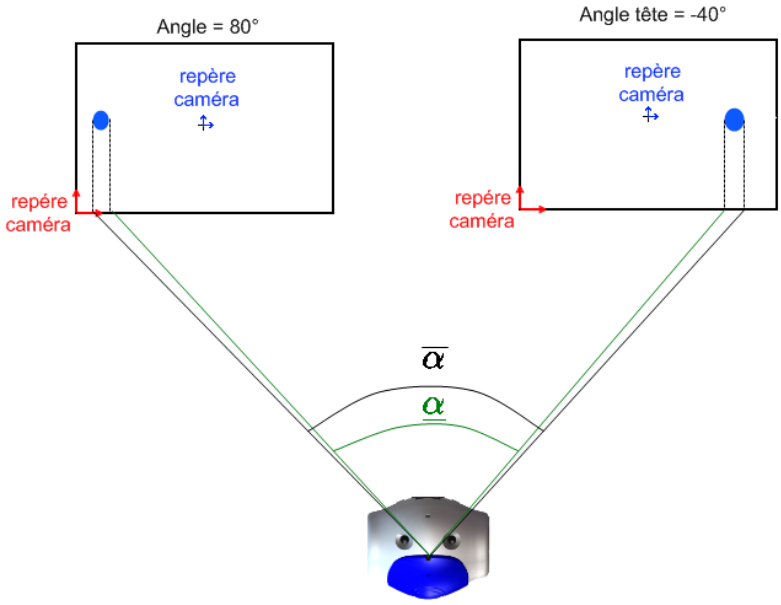



FIGURE 8 – Calcul de l'angle entre deux amers

## 2.3 Localisation du robot

Dans cette partie, je vais détailler l'implémentation de la partie localisation sur le robot Nao. Je ne rentrerai pas dans les détails techniques, un second rapport plus précis sur la programmation du robot Nao est actuellement en cours d'écriture.

### 2.3.1 Localisation par triangulation

Reprenons la figure 2 vu au 1.1. Pour localiser le robot, on doit trouver les trois cercles circonscrits aux triangles ABR, BCR et ACR.

**Construction du cercle circonscrit au triangle** *ABR*

Pour une meilleure compréhension, nous nous baserons sur la figure 9.

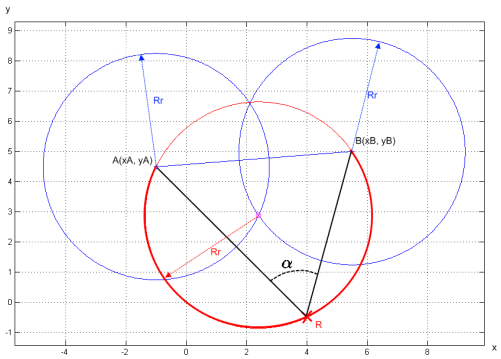



FIGURE 9 – Recherche du cercle circonscrit au triangle *ABR*

La loi des sinus nous donne le rayon du cercle circonscrit au triangle $ABR$, $R_r = \frac{AB}{2.sin(\alpha)}$.

Le centre de ce cercle est un des points d'intersection des deux cercles de rayon $R_r$ et de centres respectifs $A$ et $B$.

Le lecteur trouvera en annexe du rapport bibliographique une méthode de calcul des points d'intersections de deux cercles. Le choix du point d'intersection se faisant par un simple test sur la position (gauche ou droite) du vecteur $\overrightarrow{AB}$.

On a maintenant les centres et rayons des trois cercles circonscrits. Pour trouver la position du robot, il faut trouver l'intersection des trois cercles, soit résoudre le système suivant :

$$\begin{cases} (x-x_r)^2 + (y-y_r)^2 - R_r = 0 \\ (x-x_b)^2 + (y-y_b)^2 - R_b = 0 \\ (x-x_o)^2 + (y-y_o)^2 - R_o = 0 \end{cases}$$

$$\text{avec} \begin{cases} x_r,\ y_r \text{ et } R_r \text{ les coordonnées et rayon du cercle circonscrit au triangle } ABR \\ x_b,\ y_b \text{ et } R_b \text{ les coordonnées et rayon du cercle circonscrit au triangle } BCR \\ x_o,\ y_o \text{ et } R_o \text{ les coordonnées et rayon du cercle circonscrit au triangle } ACR \end{cases}$$

### 2.3.2 Algorithme d'inversion ensembliste

Il faut maintenant résoudre le système d'équations ci-dessus. Pour cela, nous utiliserons l'algorithme SIVIA [3]. Un algorithme d'inversion ensembliste. Cet outil permet, entre autres, la résolution de systèmes d'équations non-linéaires. L'algorithme SIVIA peut résoudre notre problème.

Il est nécessaire, pour comprendre l'algorithme SIVIA, de connaitre la définition mathématique d'une fonction d'inclusion. Une fonction d'inclusion pour la fonction $f$ définie de l'ensemble $[\mathbf{E}]$ vers $[\mathbf{F}]$ vérifie :

$$\forall [x] \in [\mathbf{E}], f([x]) \subset [f]([x])$$

Les entrées de cet algorithme sont :

| | |
|---|---|
| $[f]$ | une fonction d'inclusion pour la fonction f à inverser. |
| $[X_0]$ | un domaine initial de recherche. |
| $\epsilon$ | une précision. |
| $\mathbb{Y}$ | l'ensemble à inverser. |

En sortie, l'algorithme fournit un encadrement de $f^{-1}(\mathbb{Y}) = \{[x] \in [X_0] \mid f([x]) \in \mathbb{Y}\}$

$$\mathbb{X}^- \ \subset \ f^{-1}(\mathbb{Y}) \ \subset \ \mathbb{X}^+ \tag{2.5}$$

De façon itérative, l'algorithme va construire un sous pavage pour $\mathbb{X}^+$ et $\mathbb{X}^-$.
La figure 10 illustre les résultats possibles d'un SIVIA pour une fonction d'inclusion $f$ : $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$.

3. Set Inversion Via Interval Analysis

**Algorithm 1** Algorithme SIVIA

$L \leftarrow [X_0]$
**while** $L$ is not empty **do**
  $[x] = L$.pop()
  **if** $[f]([x]) \subset \mathbb{Y}$ **then**
    $[x]$ is a solution
    $\mathbb{X}^- \leftarrow [x]$
  **end if**
  **if** $[f]([x]) \cap \mathbb{Y} \neq \emptyset$ **and** $[x]$.size() $> \epsilon$ **then**
    $[x_1], [x_2] =$ bissect$([x])$
    $L \leftarrow [x_1]$
    $L \leftarrow [x_2]$
  **else**
    //$[x]$ may be a solution
    $\mathbb{X}^+ \leftarrow [x]$
  **end if**
  **if** $[f]([x]) \cap \mathbb{Y} = \emptyset$ **then**
    //$[x]$ is not a solution
    $\mathbb{X}^- \leftarrow [x]$
  **end if**
**end while**

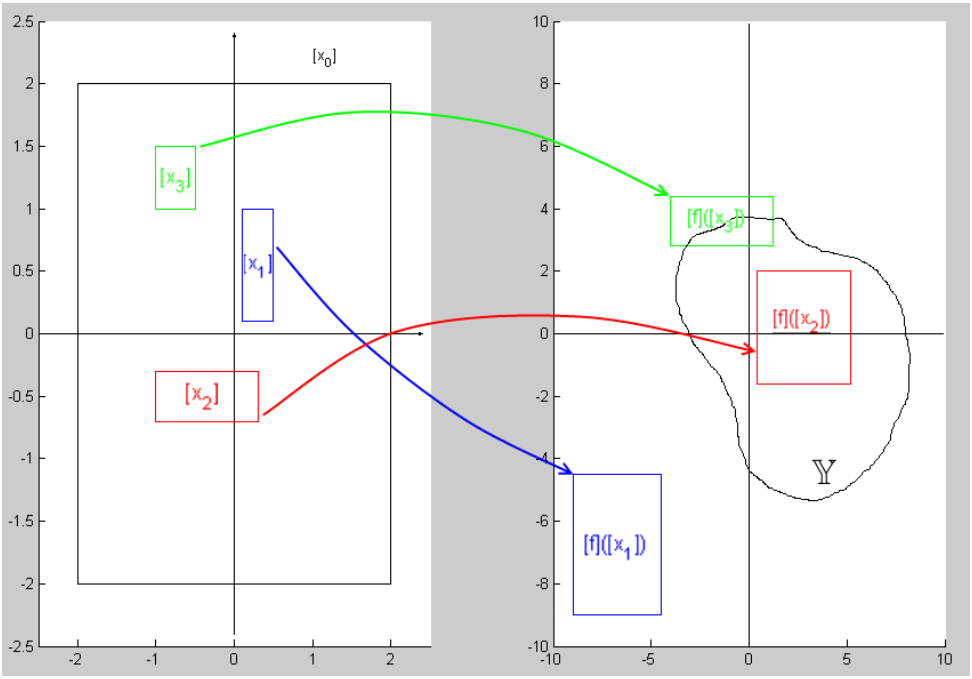



FIGURE 10 – Exemple de SIVIA

- En rouge, l'image du pavé $[\mathbf{x_2}]$ par la fonction d'inclusion de $f$ est inclue dans l'ensemble $\mathbb{Y}$. Ce pavé est donc solution car $[\mathbf{f}]([\mathbf{x_2}]) \subset \mathbb{Y}$.

- En bleu, l'image du pavé $[\mathbf{x_1}]$ n'intersecte pas l'ensemble $\mathbb{Y}$. Ce pavé n'est pas solution car $[\mathbf{f}]([\mathbf{x_1}]) \cap \mathbb{Y} = \emptyset$.

- En vert, l'image du pavé $[\mathbf{x_3}]$ intersecte l'ensemble $\mathbb{Y}$. On ne peut pas déterminer si le pavé est solution ou non de façon garantie, mathématiquement :

$$\begin{cases} [\mathbf{f}]([\mathbf{x_3}]) \cap \mathbb{Y} \neq \emptyset \\ [\mathbf{f}]([\mathbf{x_3}]) \not\subset \mathbb{Y} \end{cases}$$

.

Lorsque ce cas se produit, on va bissecter le pavé pour recommencer le test sur les deux pavés résultants. Le découpage se fera jusqu'à ce que l'on ait ateint la précision désirée sur la taille d'un pavé. La figure 11 montre le résultat (en zoom) d'un SIVIA pour une fonction f définie comme suit :

$$f : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$$
$$f([x],[y]) = ([x]-[x_c])^2 + ([y]-[y_c])^2 - [R_c]^2$$

avec : $[\mathbf{x_c}], [\mathbf{y_c}]$ les coordonnées du centre du cercle et $[\mathbf{R_c}]^2$ le rayon du cercle.

### 2.3.3 Recherche de position par triangulation

Le paragraphe 2.3.1 nous a expliqué comment trouver une position par triangulation, il faut résoudre un système d'équations non linéaire. Pour ce faire, on utilise l'algorithme SIVIA vu au 2.3.2.
La fonction d'inclusion va donc tester si le pavé $[X] = ([x],[y])$ est solution des équations de

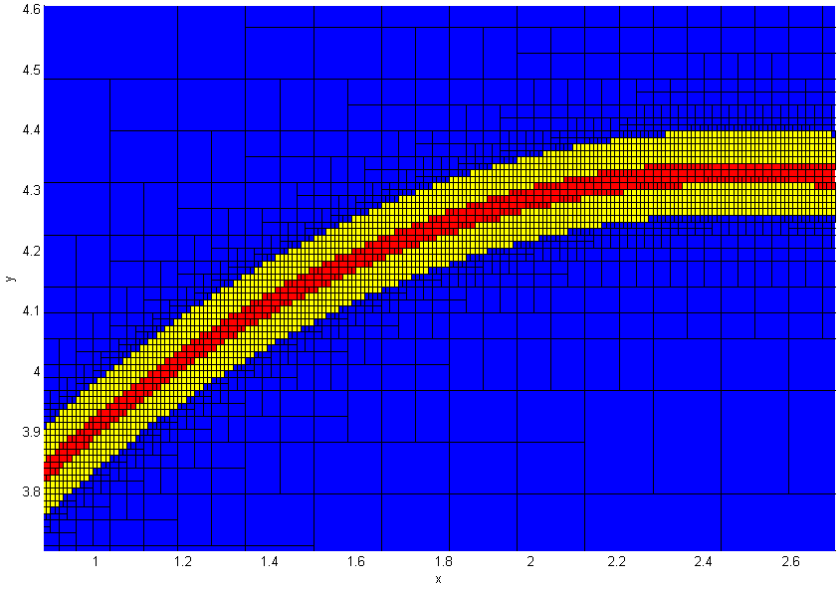



FIGURE 11 – Résultats d'un SIVIA

cercles.On se donne un domaine de recherche $[\mathbf{X_0}]$ définit par les dimensions de la pièce et une précision $\epsilon$ arbitraire.

On reprend donc les équations de cercles nécessaires à la localisation et on les écrit sous forme d'intervalles. Cette écriture sous forme d'intervalles nous permeet directement de prendre en compte l'erreur sur la mesure d'angle puisque les équations de ces cercles dépend de cet angle ($R_r = \frac{AB}{2.sin(\alpha)}$).
On recherche donc l'ensemble défini par :

$$([x] - [x_r])^2 + ([y] - [y_r])^2 - [R_r] = 0 \tag{2.6}$$
$$([x] - [x_b])^2 + ([y] - [y_b])^2 - [R_b] = 0 \tag{2.7}$$
$$([x] - [x_o])^2 + ([y] - [y_o])^2 - [R_o] = 0 \tag{2.8}$$
$$\tag{2.9}$$

L'algorithme suivant est, en quelques sortes, la fonction d'inclusion (ce n'est pas réellement le cas puisque nous n'avons pas à faire à une fonction). Rappelons nous, dans l'algorithme SIVIA, un pavé était déterminé en fonction de 3 tests :

1 $[f]([x]) \subset \mathbb{Y}$
2 $[f]([x]) \cap \mathbb{Y} \neq \emptyset$ ET $[x]$.size() $> \epsilon$
3 $[f]([x]) \cap \mathbb{Y} = \emptyset$

Dans notre cas, ces trois tests sont remplacés par un appel à l'algorithme suivant :

$[testC1] = ([x] - [x_r])^2 + ([y] - [y_r])^2 - [R_r]$
$[testC2] = ([x] - [x_b])^2 + ([y] - [y_b])^2 - [R_b]$
$[testC3] = ([x] - [x_o])^2 + ([y] - [y_o])^2 - [R_o]$
**if** $[testC1] \subset [0,0]$ **and** $[testC2] \subset [0,0]$ **and** $[testC3] \subset [0,0]$ **then**
  le pavé $[X]$ est solution car le pavé est inclut dans les deux cercles.
**else**
  **if** $[testC1] \cap \emptyset$ **or** $[testC2] \cap \emptyset$ **or** $[testC3] \cap \emptyset$ **then**

le pavé $[X]$ n'est pas solution, ce pavé n'est pas inclut dans un des cercles (voir dans plusieurs).

**else**

le pavé $[X]$ est peut être solution et sera bissecté si sa taille dépasse $\epsilon$.

**end if**

**end if**

Comme une fonction d'inclusion, cet algorithme prend en entrée un pavé et retourne le statut de ce pavé. Il est donc adapté pour fonctionner avec SIVIA.

L'algorithme SIVIA retourne deux ensembles, $X^-$ et $X^+$. Tous les pavés de l'ensemble $X^-$ sont solutions des équations 2.6, 2.7 et 2.8. Les pavés de l'ensemble $X^+$ sont les pavés pour lesquels on ne peut pas statuer. Comme on ne peut pas dire que ces pavés ne sont pas solution, on les intègre à la solution. Ce résultat est garanti par l'équation 2.5.

# 3 Implémentation

Pour répondre à la problématique de localisation, j'ai développé une librairie pour le robot Nao. Cette librairie est découpée en plusieurs classes et chacune à une une fonctionnalité bien particulière. La figure 12 montre le schéma de la librairie LOCALIB.

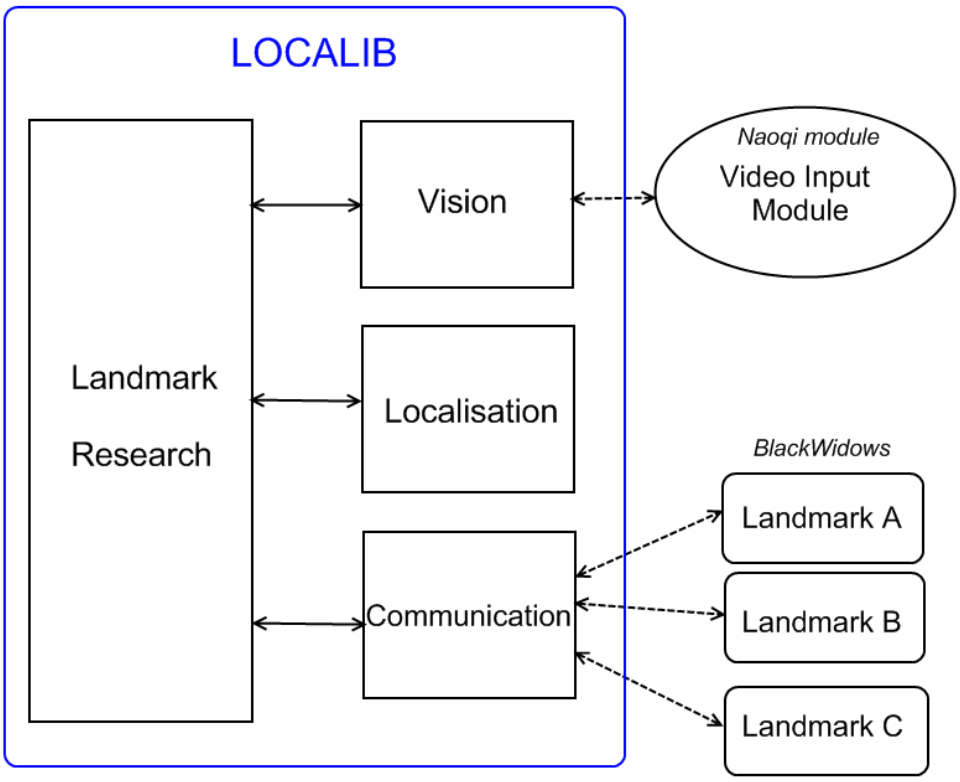



FIGURE 12 – Schéma de la librairie LOCALIB

Classe Vision : Cette classe est l'interface entre le "Video Input Module(VIM)" du robot qui donne accès aux caméras, c'est ce module qui enregistre les photos et fait le traitement de repérage des blobs. Ces traitements sont effectués par les librairies opencv (disponible sur le robot) et cvblob (une librairie adaptée pour opencv).

Classe Communication : Cette classe est la classe qui communique avec les amers. Elle permet la connexion, la déconnexion et l'envoi de données vers chaque amer

Classe Localisation : Cette classe est celle qui va localiser le robot en fonction des angles qu'on lui donne. Elle utilise la librairie Boost pour le calcul par intervalles.

Classe LandmarkResearch : C'est la classe qui coordonne le fonctionnement de la méthode, c'est ici que sont codés les algorithmes de recherche d'amers. Cette classe est aussi un module Naoqi, ce qui lui permet d'être accessible en C++ et python depuis le réseau.

La partie implémentation s'est vue complétée par un logiciel qui permet l'affichage des données. Un quadrillage identique aux dimensions de la zone d'expérimentation est rafraichi toutes

les 500ms avec la position du robot, telle qu'elle est donnée par le module motion [4]de Naoqi [5]. Cette position est calculée dans le repère de la salle (le repere "WORLD" du robot se trouve sous ses pieds au démarrage) puis affichée par un point. Le module motion ne proposant pas d'odométrie, il faudrait calculer un modèle de déplacement pour le robot qui prendrait en compte les glissements dus à la marche pour afficher une odométrie avec estimation d'erreurs.

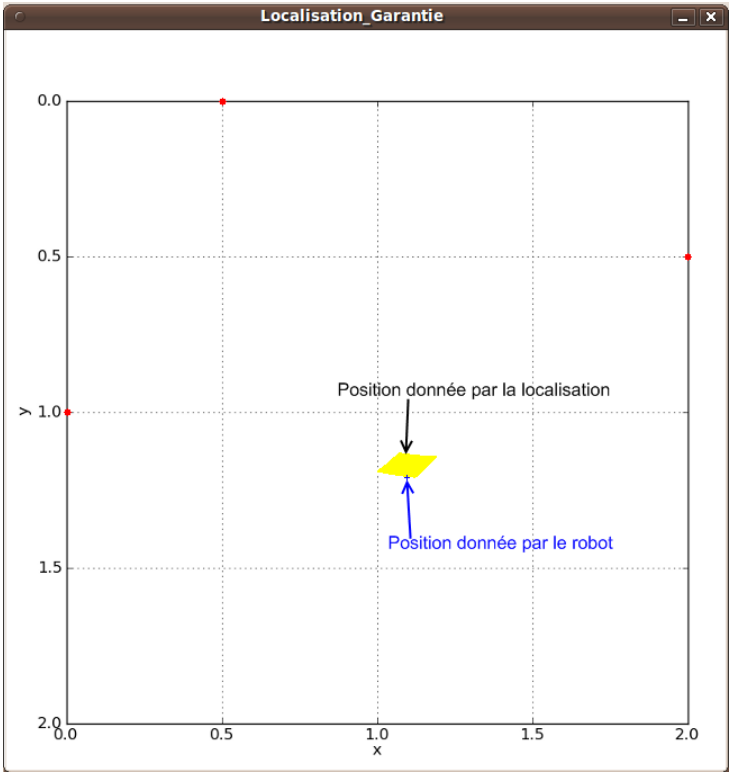



FIGURE 13 – Impression d'écran du logiciel d'affichage

La position courante du robot est affichée dans le repère de la salle. A la fin d'un déplacements, le robot va chercher à se localiser puis sa position sera affichée a la fin du calcul. Le repère du robot sera alors remis a zéro à partir de la dernière mise à jour de localisation.

Le logiciel client qui permet l'affichage (un module "remote" de naoqi) est abonné à un certains nombre de variables en mémoire de naoqi. Lorsque ces dernières sont modifiées, un événement lancé par Naoqi coté robot va renvoyer la valeur "surveillée" au logiciel qui va pouvoir effectuer le traitement et l'affichage en quelques instants. La figure 13 donne un aperçu des résultats affichés. Sur cette figure, nous pouvons voir une position marqué par une croix en bas de la zone plus claire, issue de la localisation. Cette marque est la position donnée par le robot. Le logiciel d'affichage n'étant pas encore achevé, je ne peux pas encore afficher de résultats avec plusieurs positions.

Il est aussi possible d'afficher les résultats en python grâce à la librairie matplotlib qui permet (a quelques exceptions près) les même traitements et fonctions que la toolbox image sous matlab. L'affichage obtenu est alors similaire à celui présenté sur la figure 13 mais, sur la figure 14, on peut voir en plus affichés les cercles issus de la localisation. Pour cette exemple, le robot s'est localisé en 9 secondes et les angles calculés entre les amers avaient une précision entre 3°et 4°. La

4. le module en charge de la commande et de la marche du robot

5. Naoqi est le software interne du robot. Il permet l'accès au différents modules et donc aux différentes instrumentations du robot

précision utilisée pour le SIVIA à été fixée à 0.01 soit 1 cm. La boite englobantes des résultats nous donne une localisation précise à 12cm sur l'axe $\vec{x}$ et 9cm sur l'axe $\vec{y}$.

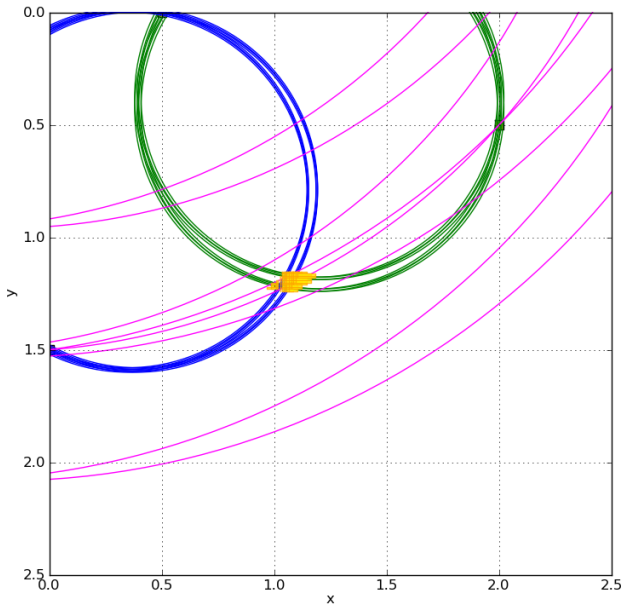



FIGURE 14 – Impression d'écran du résultat affiché en python

Pour l'expérimentation, une salle de l'ISTIA a été équipée d'un quadrillage au sol. Un carré de $2m * 2m$ quadrillée tous les 50cm. Trois amers ont étés disposées sur les murs à hauteur de tête de Nao.

Les essais ont étés implémentés en python pour plus de facilitée. Le robot se localise actuellement entre 8 et 30 secondes pour une précision entre 10 et 30 cm, dépendante de la distance d'observation des amers. Des améliorations restent à faire puisqu'il arrive qu'un changement d'intensité ambiant vienne perturber les résultats.

# 4 Conclusion et perspectives

Voici les deux perspectives possibles pour ce sujet :

La première serait de travailler sur une autre technique de reconnaissance d'amer. Se diriger vers une technologie sans amer, une utilisation des points naturels de son environnement. Cette adaptation est envisageable au vues de la modularité du programme.

La deuxième serait d'adapter cette méthode statique en dynamique. Le robot pourrait, par exemple, calculer grâce à l'odométrie les déplacements effectués entre chaque photo (il faudrait la aussi travailler sur une tête "stabilisée" qui permette de prendre des photos exploitables en mouvement) et ainsi se localiser dès qu'il verrait une nouvelle amer. En quelque sorte, continuer le projet pour l'utiliser en SLAM.

**Conclusion**

La triangulation avec utilisation de l'analyse par intervalles a montré ses preuves puisque le robot est maintenant capable de se localiser avec une précision satisfaisante. L'apport de l'analyse par intervalles permet de garantir les résultats.

Pour la partie vision, un certain nombre de difficultés techniques ont ralenties le projet et je pense que la reconnaissance de forme, d'objet ou autres techniques auraient former un sujet orientées vision à lui seul.

La localisation en robotique est un sujet très complet puisqu'elle nécessite la connaissance de nombreux domaines pour arriver à un résultat. C'est une problématique de fusion de données, par exemple, dans le cadre de SLAM, on pourrait très bien imaginer ajouter les résultats des capteurs ultrasons à nos résultats pour permettre une cartographie. C'est une des pistes de recherche à explorer pour ce thème.

## Références

[BB96] Stéphane Betgé-Brezetz. *Modélisation incrémentale et localisation par amers pour la navigation d'un robot mobile autonome en environnement naturel.* PhD thesis, Université Paul Sabatier , Toulouse, février 1996.

[BG97] Margrit Betke and Leonid Gurvits. Mobile robot localization using landmarks. In *IEEE transactions on robotics and automation*, volume 13, April 1997.

[GBFK98] Jens-Steffen Gutmann, Wolfram Burgard, Dieter Fox, and Kurt Konolige. An experimental comparison of localization methods. 1998.

[KJWM00] M. Kieffer, Luc Jaulin, E Walter, and D Meizel. Localisation et suivi robustes d'un robot mobile grâce a l'analyse par intervalles. *Traitement du signal, numéro spécial sur la robotique : "fusion de données pour véhicules intelligents"*, 17(3), 2000.

[Lar03] Frédéric Large. *Navigation autonome d'un robot mobile en environnement dynamique et incertain.* PhD thesis, Université de Savoie, septembre 2003.

[RLDL05] E Royer, M Lhuillier, M Dhome, and J.M. Lavest. Localisation par vision monoculaire pour la navigation autonome. In *ORASIS'05*, Fournol (France), mai 2005.

Titre : Localisation de robots humanoïde par des approches ensemblistes
Mots clés : Robot, Localisation, Analyse par intervalles, Amers

Résumé :
Ce rapport a été réalisé dans le cadre du Master SDS à l'ISTIA, il traite de la "Localisation de robots humanoïde par des approches ensemblistes". Un robot humanoïde doit pouvoir se repérer dans un espace connu par repérage d'amer, des points de repère situées dans son environnement qu'il devra repérer et différencier. Cette méthode utilisera l'analyse par intervalles qui nous permettra de garantir les résultats.
La première partie est un rappel des recherches bibliographiques d'un premier rapport. On y trouvera une liste non-exhaustive des méthodes de localisation utilisées dans la litérature. On traitera également la partie Vision, qui nous permet de repérer les amers dans l'environnement du robot. Enfin, une partie implémentation expliquera les choix retenus et leurs résultats.

---

Title : humanoïd robot localization using interval analysis
Keywords : Robot, Localisation, Interval analysis, Landmarks

Abstract :
This report is written for the end of a training for the master degree SDS at ISTIA. This report is about "Localization of an humanoïd robot using interval analysis". A humanoïd robot had to localize itself using known landmarks in his environment. Interval analysis will give us a guarantee for the results.
The first part is a state of art from a first report. We will find a non-exhaustive list about localization process used in the litterature. We also talk about landmark recognition in this part. A second part about implementation will explain the method we choose and why. This last part will also give results of the method.

Laboratoire :
62, avenue Notre Dame du Lac
49000 Angers