Balises laser

CVRA

Eurobot 2008

Documentation pour la
programmation du système

Patrick Eugster
Mars 2009

Version 3

Table des matières

1. Principe de fonctionnement

Un laser tourne autour d'un axe vertical dans une balise positionnée au sommet d'un robot. Ce laser balaye tout autour de lui des balises fixes disposées autours de la table ou est posé le robot. Ces balises fixes étant balayées les unes après les autres, elles peuvent mesurer des délais qui sont proportionnels aux angles vu depuis le robot. Ceci permet de positionner le robot moyennant un calcul géométrique.

L'information de position est transmise depuis les balises fixes qui communiquent entre elles par câble, vers la balise mobile qui est sur le robot qui doit récupérer les informations de positions. Le système est fait de telle sorte que deux balises laser mobiles peuvent balayer en même temps toutes les balises fixes, sans s'interférer. Cela permet de positionner deux robots à la fois.

Voici le terrain représenté avec tout le système de balises et deux robots à positionner :

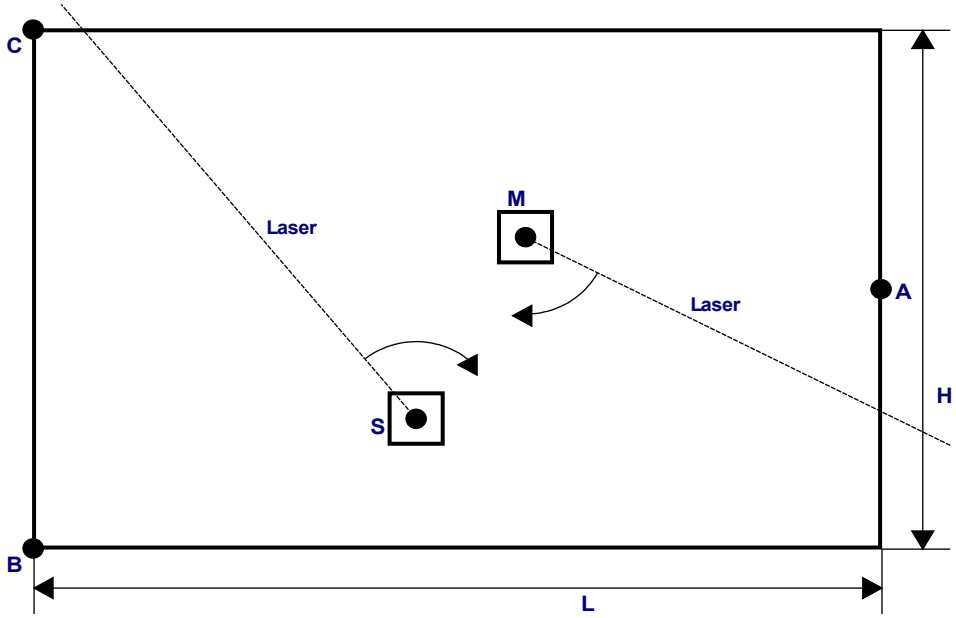

Figure 1 : représentation du terrain avec deux robots et la rotation des lasers

Les dimensions sont :

L : Largeur du terrain
H : Hauteur du terrain

Attention, ceci est relatif à la position exacte des récepteurs optiques sur les balises fixes, et non pas au coins du terrain.

Les balises sont :

A :

Balise fixe principale qui mesure les délais d'après son récepteur optique est les balises B et C, et qui renvoie tout ça à la balise M.
Cette balise à un microcontrôleur.

B et C :

Balises fixes secondaires, avec un récepteur optique pour les lasers, qui communiquent par câble avec la balise A.
Ces balises n'ont pas de microcontrôleurs.

M :

Balise mobile principale, qui à un laser qui tourne pour pouvoir la positionner, et qui reçoit les informations des balises fixes, et les renvoient au robot.
Cette balise à un microcontrôleur.

S :

Balise mobile secondaire, qui à juste un laser qui tourne pour positionner le robot adverse. Cette balise n'a pas de microcontrôleur.

2. Calculs de positionnements

Les calculs sont tous doublés vu que l'on souhaite positionner deux robots avec un système qui mesure des délais en distinguant les deux robots, le raisonnement est donc le même dans les deux cas. Les calculs traitent donc un seul cas général, applicable pour les deux robots.

2.1 Mesure des temps et conversion en angle

Voici un graphe temporel qui représente les impulsions mesurées pour un robot :

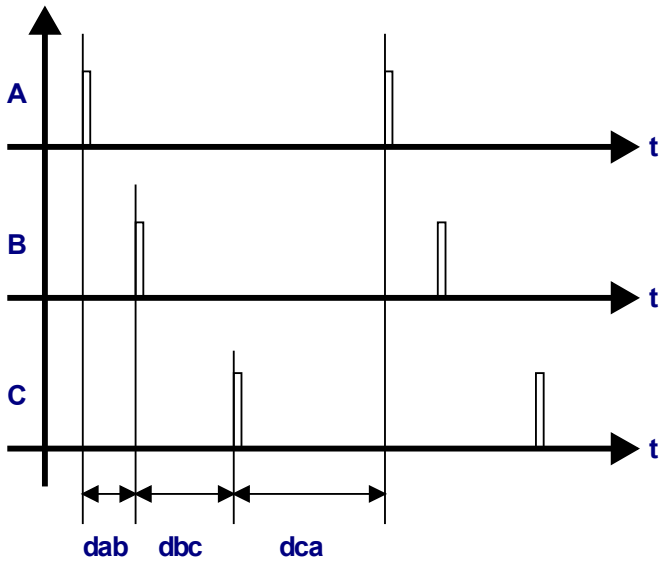

Figure 2 : représentation temporelle des impulsions sur les balises fixes pour un robot

Les temps mesurés sont :

dab: temps mesuré entre la balise A et la balise B
dbc: temps mesuré entre la balise B et la balise C
dca: temps mesuré entre la balise C et la balise A

A partir de ces temps, on peut déduire les angles qui correspondent à ces temps, en partant du principe que la somme des trois temps est l'équivalent d'un tour complet. Ceci est vrai car la vitesse de rotation est considérée comme constante.

Voici la situation d'un point de vue géométrique :

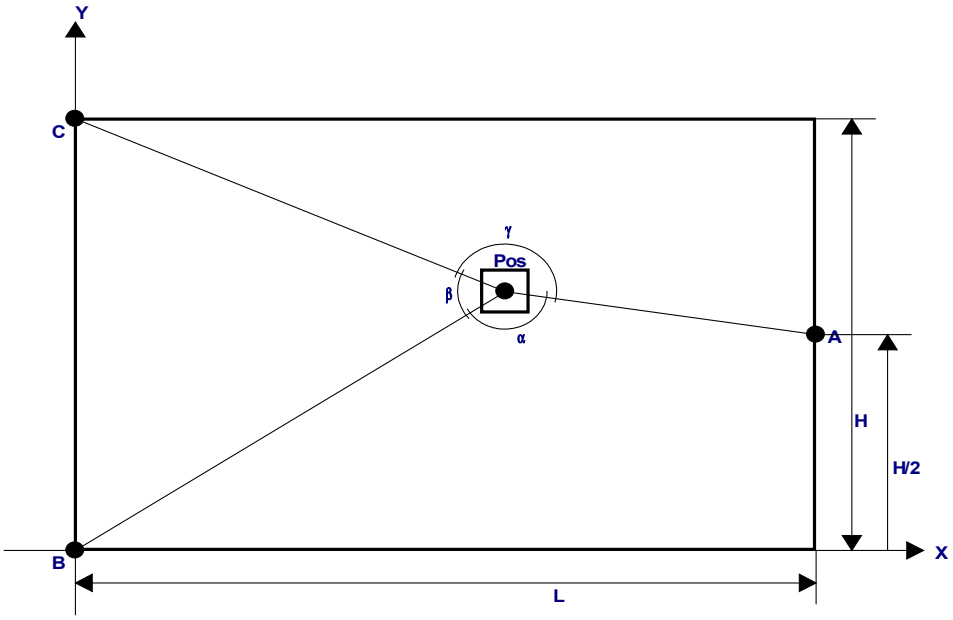

Figure 3 : problème géométrique

Les balises sont situées à des endroits connus :

$$A=(L\,;\frac{H}{2}) \qquad B=(0\,;0) \qquad C=(0\,;H)$$

On a les angles :

α : angle qui est proportionnel à dab
β : angle qui est proportionnel à dbc
γ : angle qui est proportionnel à dca

Que l'on détermine d'après les temps dab, dbc et dca, en degrés :

$$\alpha=\frac{dab}{dab+dbc+dca}\cdot 360\,° \qquad \boxed{\beta=\frac{dbc}{dab+dbc+dca}\cdot 360\,°} \qquad \gamma=\frac{dca}{dab+dbc+dca}\cdot 360\,°$$

Et comme les angles sont complémentaires :

$$\alpha+\beta+\gamma=360\,°$$

D'après les angles mesurés, ainsi que les dimensions L et H, et les positions de A, B et C, il est possible de calcules la position du robot.

2.2 Théorème de l'angle inscrit

Afin de résoudre la position du robot, il faut passer par le théorème de l'angle inscrit :

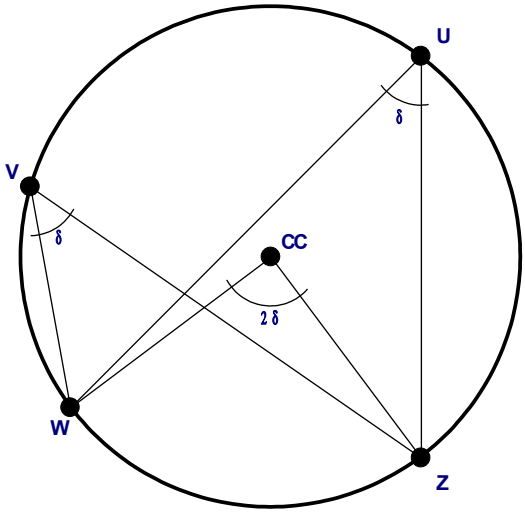

Figure 4 : théorème de l'angle inscrit

Le théorème de l'angle inscrit fait la relation entre l'angle formé par deux points W et Z sur un cercle, vu depuis le centre CC et, vu depuis n'importe quel point sur le cercle U et V. L'angle du centre 2δ est toujours deux fois plus grand que l'angle vu depuis n'importe quel point situé sur le cercle δ.

Voici un cas particulier du théorème, qui peut être déduit du cas général :

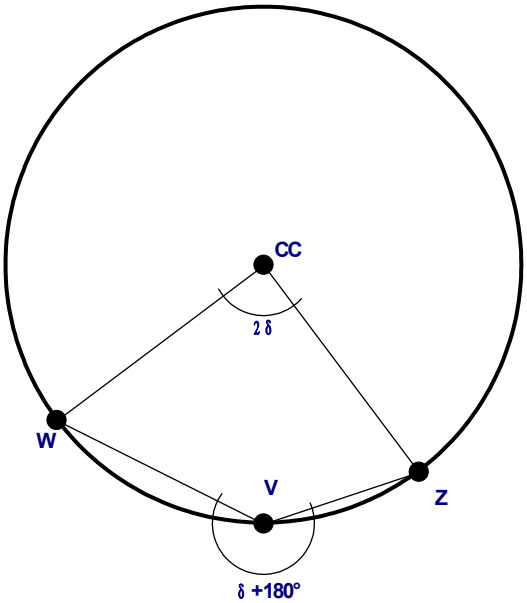

Figure 5 : théorème de l'angle inscrit, cas particulier

Dans le cas des balises laser, les balises fixes sont les points W et Z, le robot est le point U ou V, et on cherche le centre du cercle CC, pour chaque angle, avec les balises qui y correspondent.

2.3 Problème géométrique

Voici le problème d'un point de vue géométrique dans le cas des balises laser, en prenant pour l'exemple l'angle β, et en appliquant le théorème de l'angle inscrit :

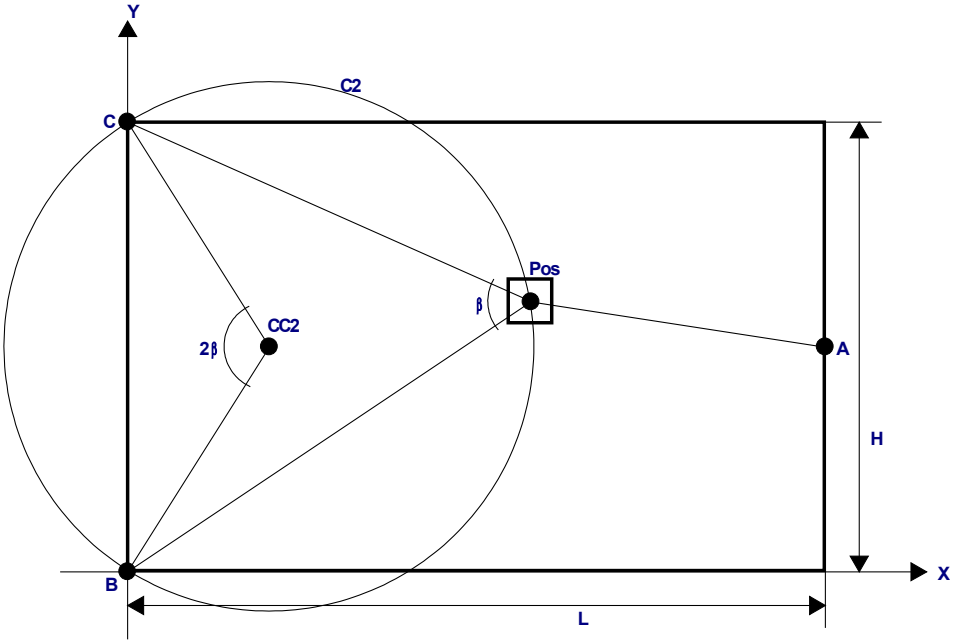

Figure 6 : cercle des possibilités déduit d'un angle

Pour chaque cercle relatif à un angle, la notation est la suivante :

C1 : cercle relatif à α
C2 : cercle relatif à β
C3 : cercle relatif à γ

CC1 : centre du cercle relatif à α
CC2 : centre du cercle relatif à β
CC3 : centre du cercle relatif à γ

On remarque que l'on peut trouver le centre du cercle CC2 avec l'angle β et en connaissant la position des balises fixes B et C. De ce centre peut être déduit le rayon du cercle avec sa distance jusqu'à une balise fixe, ou le robot. On connaît donc l'équation complète d'un des cercles, sur lequel doit se trouver le robot.

Avec uniquement deux balises, on trouve un angle, et donc un seul cercle, sur lequel se situe le robot, ce n'est donc pas suffisant pour trouver la position. Avec trois balises, on obtient deux angles significatifs, car le troisième est complémentaire aux deux autres, et donc redondant.

Voici une illustration de l'intersection de deux cercles C1 et C2, issus de α et β :

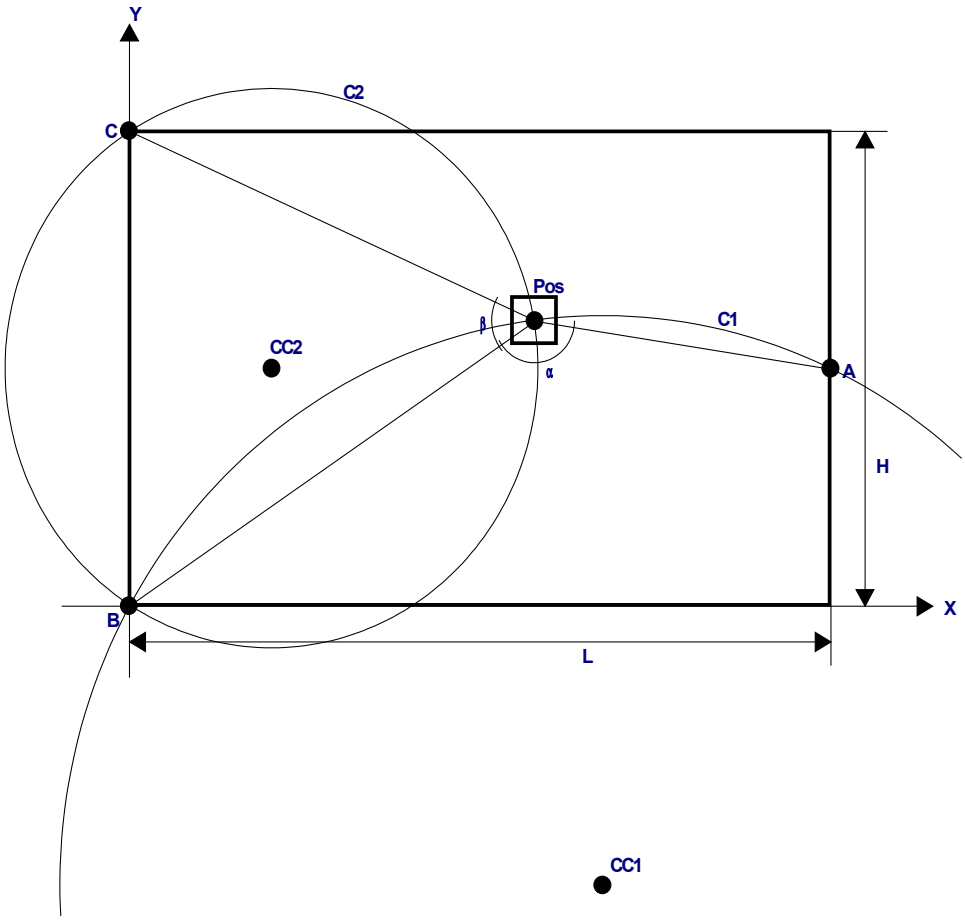

Figure 7 : intersection entre les cercles C1 et C2

On remarque clairement deux intersections; une dans la balise B, et une autre dans le robot. Avec l'intersection issue de trois cercles, C1, C2 et C3, on aurait eu seulement une seule intersection commune aux trois cercles, mais comme les autres intersection sont situées sur les balises, c'est inutile de faire ce calcul.

Pour résoudre la position du robot, il faut donc commencer par trouver les deux centres de cercles qui correspondent aux angles α et β.

2.3.1 Cercle C2 relatif à β

Le cercle C2 est le plus simple à trouver car celui-ci passe toujours par les points B et C qui sont situé sur une droite verticale, ce qui à pour conséquence de toujours avoir un cercle qui est centré verticalement au milieu du terrain. Voici différentes possibilités de cercles C2 avec leurs centres respectifs pour illustrer ceci :

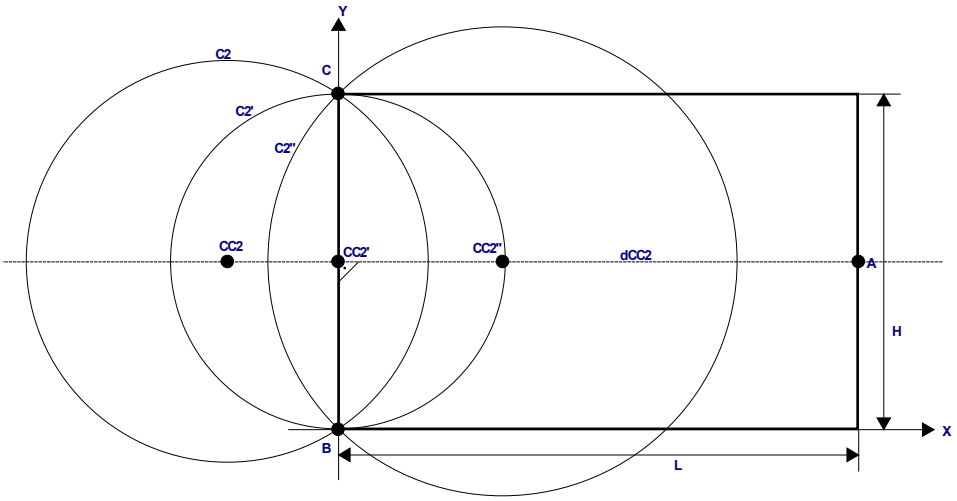

Figure 8 : quelques cercles relatifs à β

On remarque de manière évidente que le centre du cercle se balade sur une droite dCC2 qui est perpendiculaire à la droite BC, et qui passe à mi-chemin entre les points B et C. Cette observation est utile pour comprendre comment résoudre plus tard le même problème appliqué aux autres angles, bien que pour le cas de β cela soit plutôt évident :

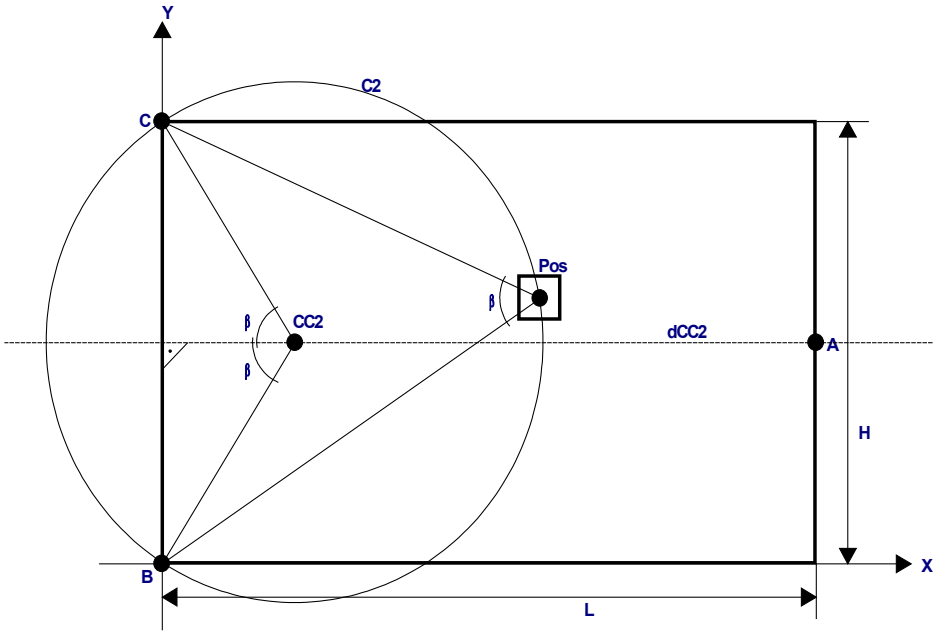

Figure 9 : recherche du centre du cercle C2

$$CC2y = \frac{H}{2} \qquad CC2x = \frac{H}{2} \cdot \frac{1}{\tan(\beta)}$$

2.3.2 Cercle C1 relatif à α

Tout comme le cercle C2, le cercle C1 à un centre qui se balade le long d'une droite dCC1 qui passe à mi-chemin entre A et B, en étant perpendiculaire à la droite AB :

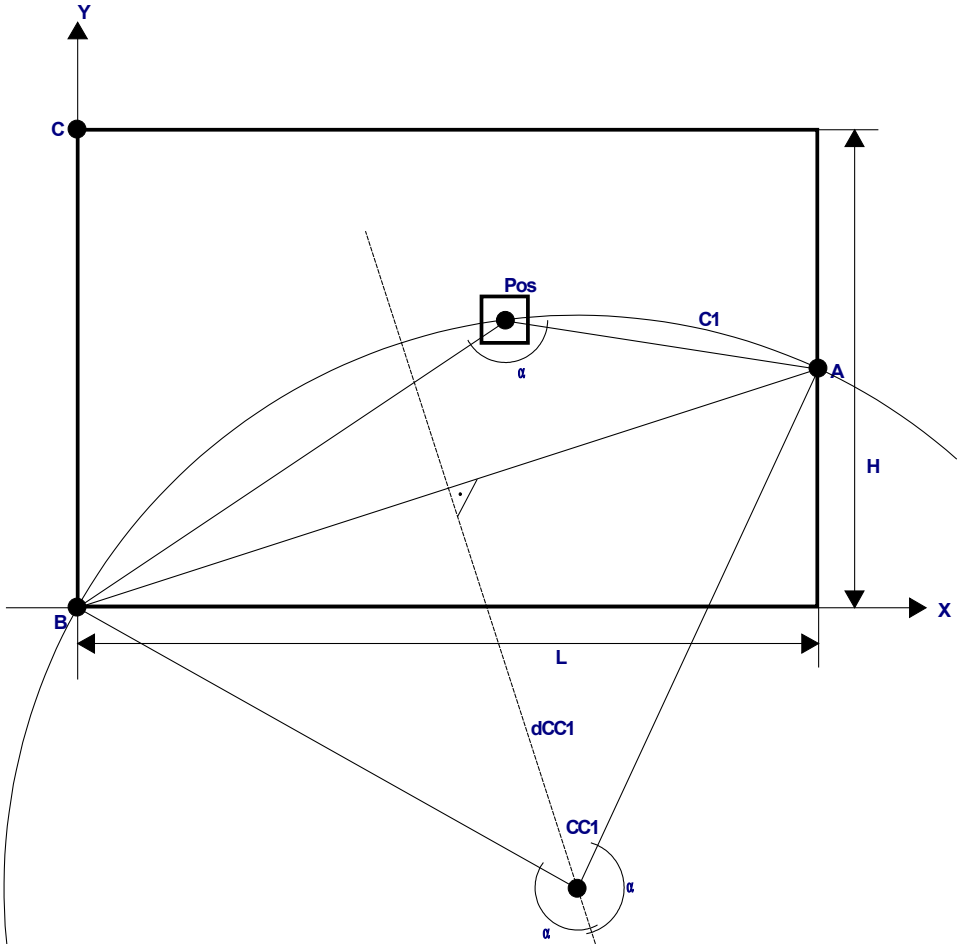

Figure 10 : droite dCC1 des possibilités pour CC1

Les calculs trigonométriques sont nettement plus compliqué que pour le cas de l'angle β, car la droite dCC1 n'est parallèle ni à l'axe X ni à l'axeY, ou autrement dis, elle est ni verticale, ni horizontale, mais de biais. Sur cette droite dCC1 se situe le centre du cercle C1 que l'on recherche.

Voici la résolution trigonométrique du problème :

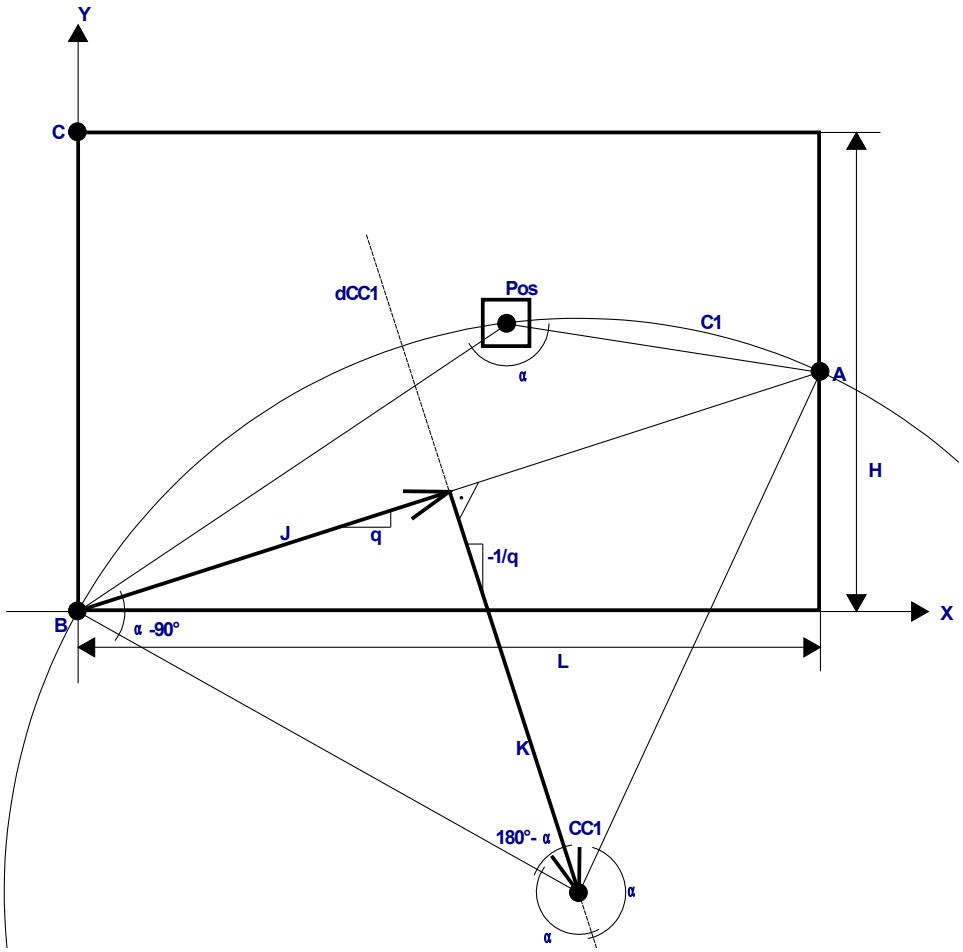

Figure 11 : recherche du centre du cercle C1, α<180°

Les vecteurs J et K ont été introduits ainsi que leurs pentes. Étant perpendiculaire, on connaît la relation entre leurs pentes, qui sont mutuellement inverses et opposées (c'est une manière de dire que leur produit scalaire est nul) :

$$J=(\frac{L}{2};\frac{H}{4})=(Jx;Jy) \qquad Jx+Kx=CC1x \qquad Jy+Ky=CC1y \qquad \frac{Jy}{Jx}=q=-\frac{Kx}{Ky}$$

On remarque donc que l'angle α contraint le rapport entre le module de J et celui de K avec une tangente :

$$\tan(180°-\alpha)=\tan(-\alpha)=-\tan(\alpha)=\frac{\|J\|}{\|K\|}=r$$

Attention, cette déduction est propre à la figure 11.

Visiblement, sur le cas donné par la figure 11, l'angle α est compris entre 90° et 180°, ce qui donne des valeurs de tangentes négatives et compense le signe négatif devant celle-ci. Néanmoins, si l'angle α venait à aller en dessous de 90°, alors CC1 monterait au-delà de la droite AB, et un rapport négatif entre les modules de J et K sous-entendrais que le vecteur K, l'inconnue du problème, serait orienté contre le haut :

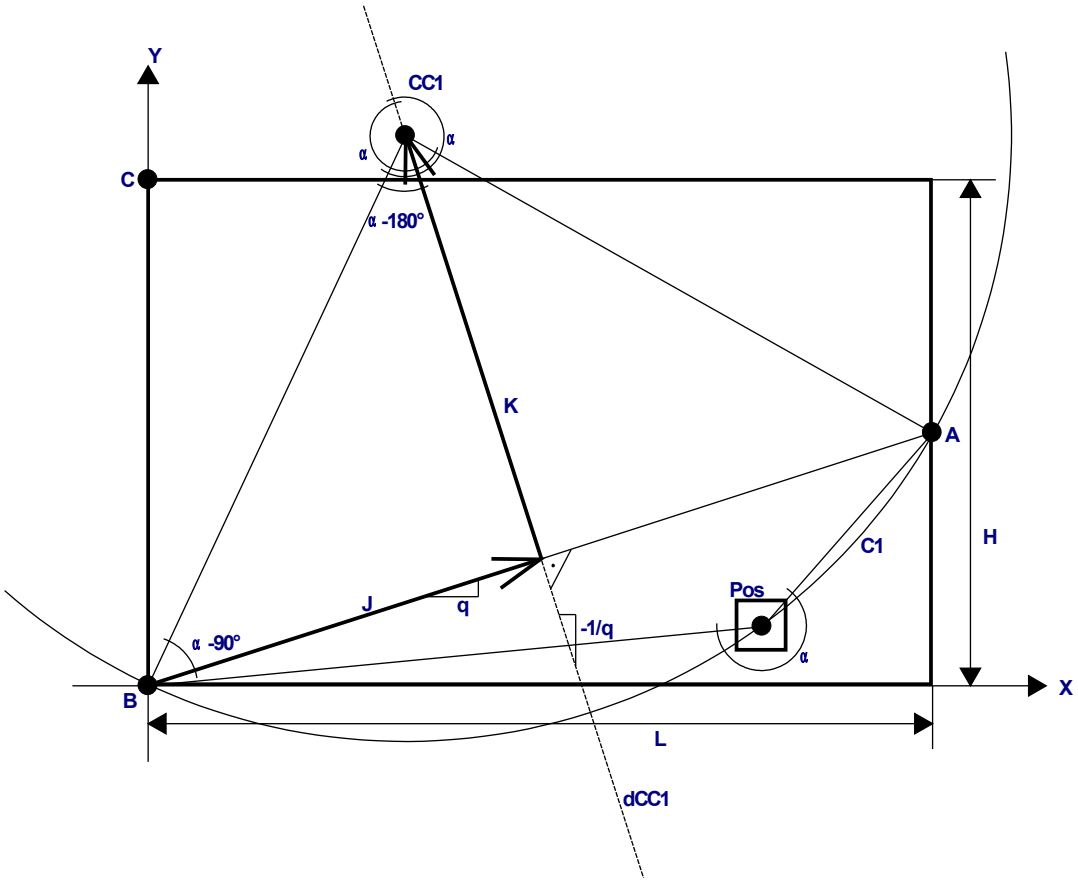

Figure 12 : recherche du centre du cercle C1, α>180°

Déduction propre à la figure 12 :

$$\tan(\alpha - 180°) = \tan(\alpha) = \frac{\|J\|}{\|K\|}$$

Dans le cas particulier de α à 180°, la tangente devient nulle, ce qui à pour conséquence de fixer un module du vecteur K comme infini, et donc le centre du cercle 1 se situe à une distance infinie de la table, le long de la droite dCC1.

Revenons au cas de la figure 11. Rappelons que module des vecteurs évolue proportionnellement aux composantes x et y de chacun (car q est constant) :

$$\frac{Jy}{Jx}=q=-\frac{Kx}{Ky}$$

Aussi, le rapport entre le modules des vecteurs, issu de la figure 11 :

$$\tan(180°-\alpha)=\tan(-\alpha)=-\tan(\alpha)=\frac{\|J\|}{\|K\|}=r$$

Alors vient :

$$r^2=\frac{Jx^2+Jy^2}{Kx^2+Ky^2} \qquad r^2\cdot(Kx^2+Ky^2)=(Jx^2+Jy^2)$$

$$r^2\cdot Kx^2\left(1+\frac{1}{q^2}\right)=Jy^2\left(1+\frac{1}{q^2}\right) \qquad r^2\cdot Ky^2(1+q^2)=Jx^2(1+q^2)$$

$$r\cdot Kx=\pm Jy \qquad r\cdot Ky=\pm Jx$$

En observant attentivement les figures 11 et 12, on fixe le choix des signes; tous les cas de figures ont pour solution commune :

$$Kx=Jy\cdot\frac{-1}{\tan(\alpha)} \qquad Ky=Jx\cdot\frac{1}{\tan(\alpha)}$$

Le fait d'avoir traité deux cas séparément pour l'angle α plus grand ou plus petit que 180° nous à donné une fois un rapport de modules entre J et K négatif, et une fois positif. Ceci s'est compensé par l'orientation du vecteur, une fois en haut, et une fois en bas, pour ne donner au final qu'une solution globale.

On trouve donc finalement les coordonnées du centre du cercle C1, comme la somme de J et K :

$$\boxed{CC1x=\frac{L}{2}-\frac{H}{4}\cdot\frac{1}{\tan(\alpha)}} \qquad \boxed{CC1y=\frac{H}{4}+\frac{L}{2}\cdot\frac{1}{\tan(\alpha)}}$$

2.3.3 Intersection entre C1 et C2

Les formules générales d'intersections de cercles sont du deuxième ordre, et mettent donc en œuvre des racines carrées, ce qui est très peu avantageux à l'implémentation software. Dans le cas particulier de l'intersection entre les cercles C1 et C2, on connaît déjà une des deux solutions; c'est la balise B, qui est à l'origine. D'un point de vue mathématique, en rapport avec l'équation du second ordre, on peut dire que l'on à donc pas besoin de calculer le delta, mais seulement la partie commune.

Si l'on connaît CC1 et CC2 :

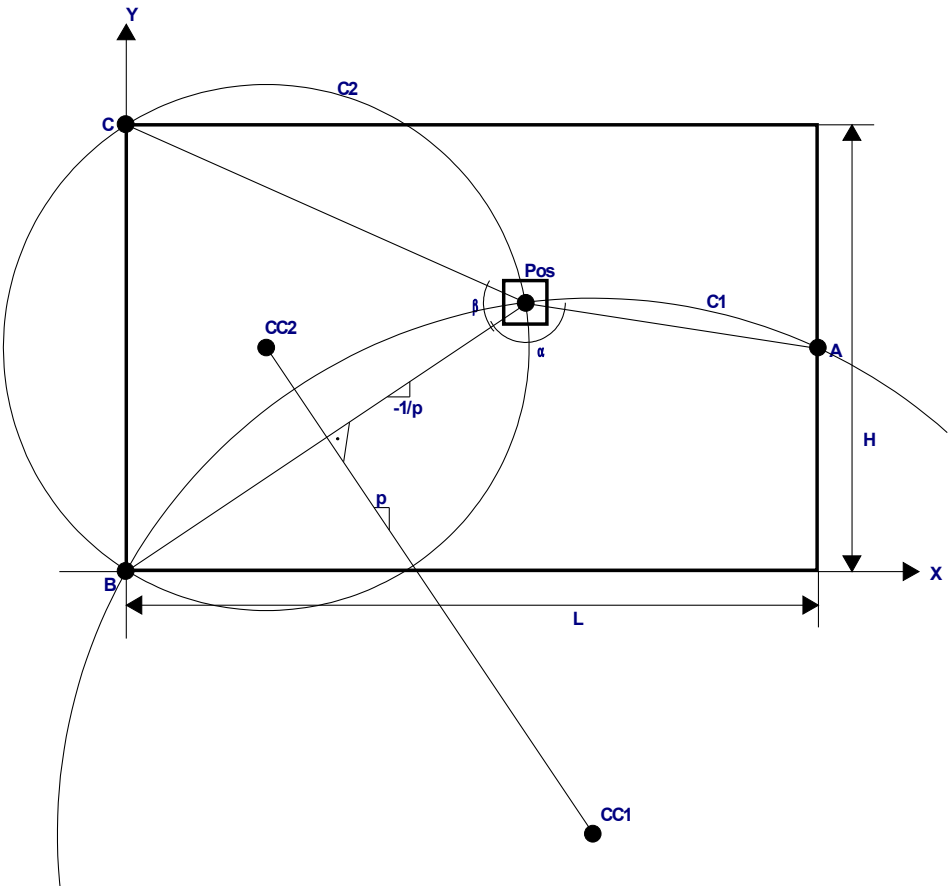

Figure 13 : recherche de la pente entre CC2 et CC1

La droite B-Pos est toujours perpendiculaire à la droite CC1-CC2. La pente p est la pente de la droite CC2-CC1. Donc par déduction la pente de la droite B-Pos est l'opposé de l'inverse de p, (c'est une manière de dire que leur produit scalaires est nul).

$$p = \frac{CC1y - CC2y}{CC1x - CC2x} \qquad \frac{-1}{p} = \frac{CC2x - CC1x}{CC1y - CC2y}$$

Désormais, on peut trouver la position du robot en faisant l'intersection d'un cercle avec un droite. Cette simplification est singulière au fait que l'une des solutions de l'intersection entre les deux cercles C1 et C2 est connue. On note aussi que le fait d'effectuer les calculs géométriques avec α et β (et non pas β et γ , ou α et γ) simplifie grandement cette étape, car une des deux intersection tombe sur B, qui est à l'origine.

2.3.4 Position du robot

Voici comment retrouver la position du robot :

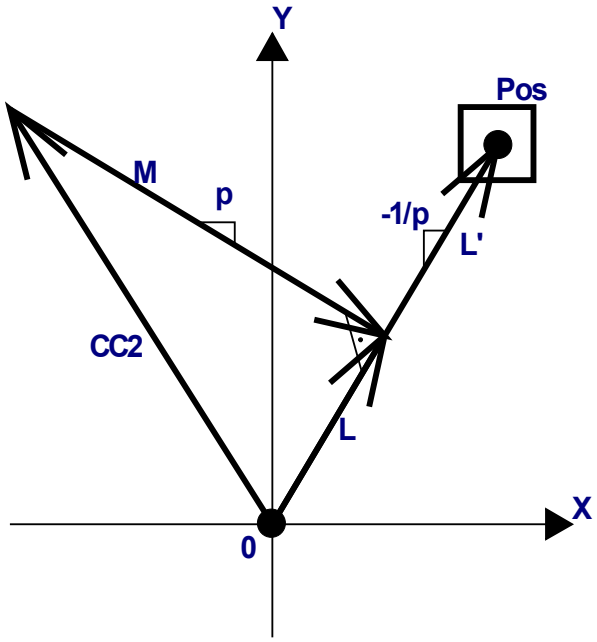

Figure 14 : recherche de la position du robot avec CC1 et CC2

Le centre du cercle 2, CC2 est directement "convertis" en vecteur, la position du robot est égale à L + L'. En observant les relations entre vecteurs :

$$p=\frac{My}{Mx}=-\frac{Lx}{Ly} \qquad Lx=CC2x+Mx \qquad Ly=CC2y+My$$

Ce qui donne :

$$CC2x+\frac{My}{p}=-p\,(CC2y+My)$$

$$My=\frac{-CC2x-p\cdot CC2y}{p+\frac{1}{p}}$$

Ainsi apparaît la position du robot :

$$Posy=2\cdot Ly=2\cdot(CC2y+My)=2\cdot\left(\frac{-CC2x-p\cdot CC2y+p\cdot CC2y+\frac{CC2y}{p}}{p+\frac{1}{p}}\right)=2\cdot\frac{CC2y-CC2x\cdot p}{p^2+1}$$

$$Posx=-p\cdot Posy$$

3. Cas spécial

En admettant que le robot se déplace contre la droite dans le coins en haut à droite :

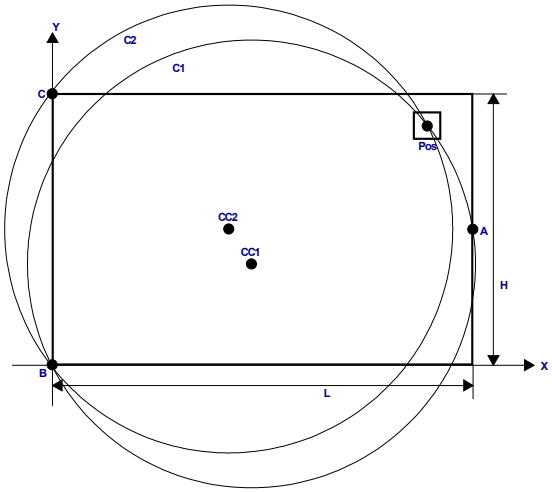

Figure 15a

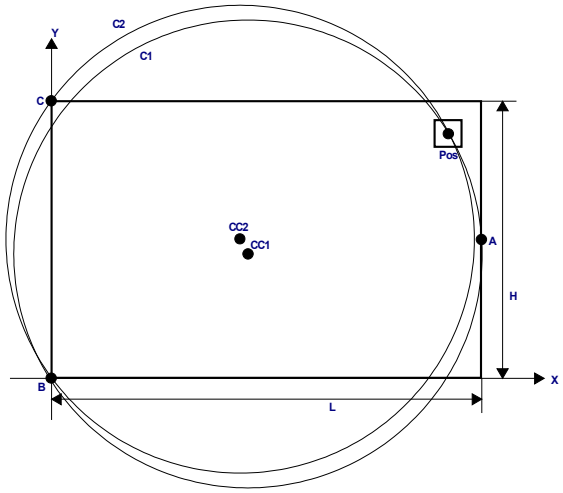

Figure 15b

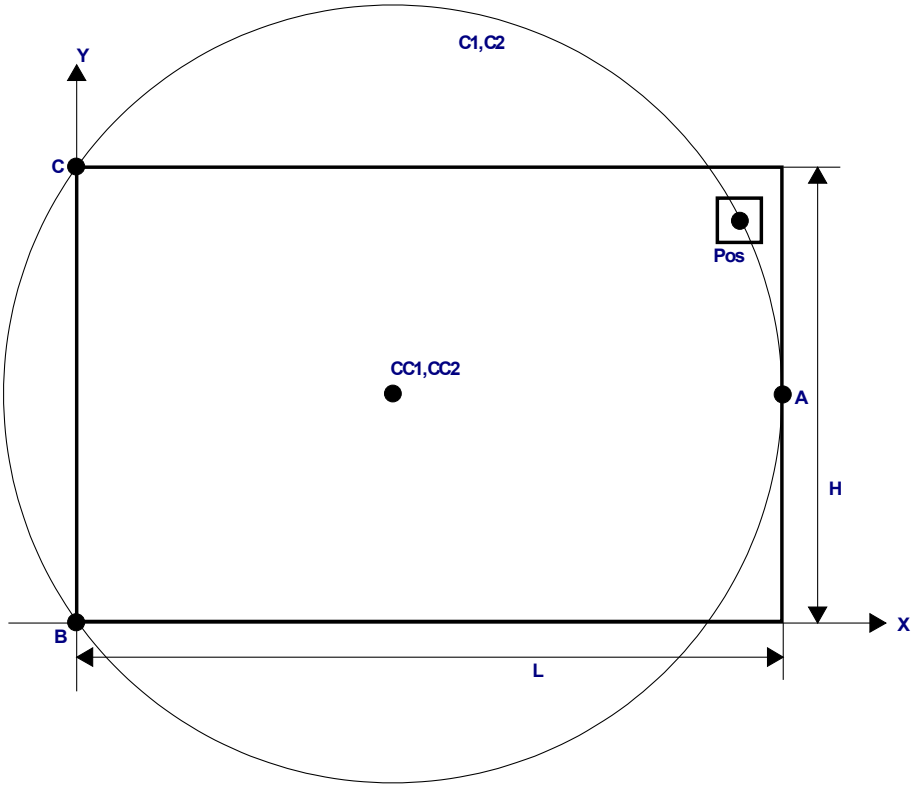

Figure 15c : cas particulier du cercle qui passe par toutes les balises fixes

On remarque que ce cas n'est possible que lorsque le robot va dans les coins à droite du terrain ou aucune balise n'est positionnée, et que si la balise A était plus éloignée ou à un coin de la table, jamais le robot ne pourrait venir sur ce cercle.

Chaque angle mesuré donne un cercle qui passe par les deux balises fixes qui le mesure. D'après cet angle est déduit un cercle qui passe par les deux balises en question. Au moment ou le robot s'approche d'un cercle qui passe par toutes les balises, la précision du système devient de moins en moins bonne, car l'intersection entre cercles devient de moins en moins précise, jusqu'à ce que le robot soit sur le cercle, et donc qu'il y ait une infinité de possibilités.

Afin de contourner cette limite, un second capteur optique est monté sur la balise fixe A, en profondeur du capteur principal, de telle sorte que lorsque le robot vient latéralement à la balise, les deux capteurs soient balayés l'un après l'autre.

3.1 Balise A et capteur latéral

Voici le graphe temporel des impulsions mesurées par les balises fixes avec le capteur secondaire sur la balise A qui est balayé par le laser :

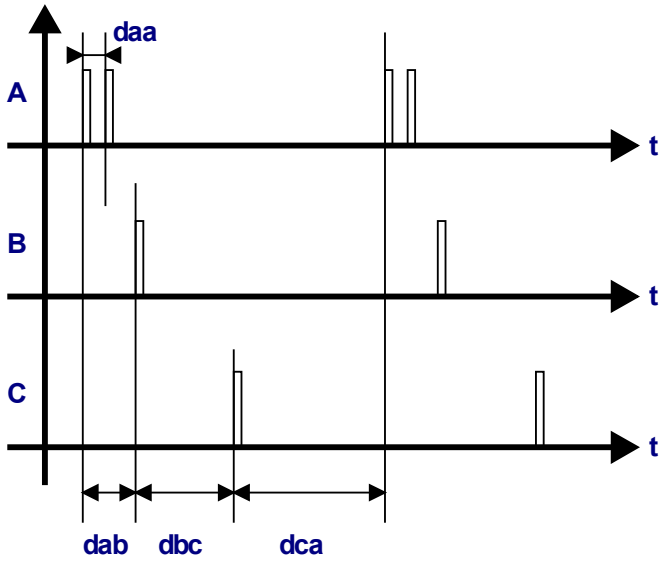

Figure 16 : impulsions mesurées sur les balises avec le capteur latéral de la balise A

Avec le capteur secondaire, l'impulsion de la balise A est doublée; le temps daa est donc rajouté :

daa: temps mesuré entre le capteur principal de la balise A, et son second capteur
dab: temps mesuré entre la balise A et la balise B
dbc: temps mesuré entre la balise B et la balise C
dca: temps mesuré entre la balise C et la balise A

Lorsque le capteur secondaire de la balise A, qui est orienté sur les deux cotés de cette dernière, n'est pas balayé par le laser, par exemple si le robot est au milieu du terrain, alors l'impulsion de la balise A n'est pas doublée. Il est donc indispensable de bien gérer la lecture des impulsions, et de faire attention à bien choisir laquelle des deux impulsions est l'impulsion A ou A', ce qui va dépendre de la position du robot sur le terrain. En effet si le robot en étant en haut balaye d'abord le capteur principal puis le secondaire, alors en bas du terrain les capteurs seront balayés selon un ordre inverse.

La situation géométrique à laquelle on est confronté pour résoudre la position du robot dans le cas spécial, aux coins de la table, est la suivante :

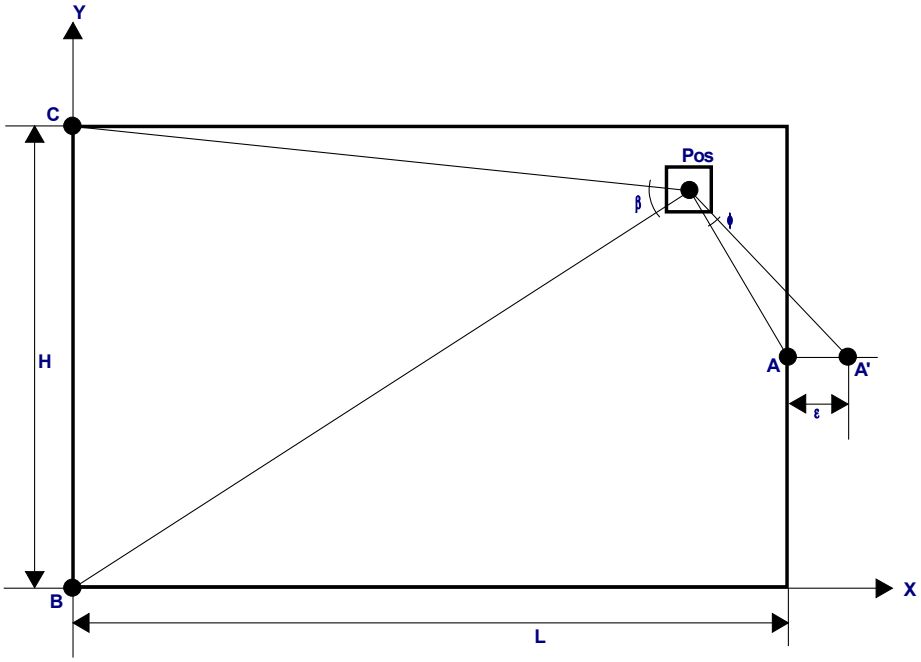

Figure 17 : problème géométrique avec le capteur latéral de la balise A

Les balises sont situées à des endroits connus :

$$A=(L\,;\frac{H}{2}) \qquad A'=(L+\varepsilon\,;\frac{H}{2}) \qquad B=(0\,;0) \qquad C=(0\,;H)$$

Avec dans le cas précis des balises laser :

$$\varepsilon=41\text{mm}$$

On a les angles :

β : angle qui est proportionnel à dbc
φ : angle qui est proportionnel à daa

Que l'on détermine d'après les temps daa et dbc, en degrés :

$$\varphi=\frac{daa}{dab+dbc+dca}\cdot 360\,° \qquad \beta=\frac{dbc}{dab+dbc+dca}\cdot 360\,°$$

D'après ces angles mesurés, ainsi que les dimensions ε, L et H, et les positions de A, B et C, il est possible de calculer la position du robot.

3.2 Problème géométrique

Tout comme pour la résolution de la position du robot dans le cas normal vers le milieu de la table, lorsque l'on est dans cas spécial il faut pour retrouver la position du robot, effectuer une intersection entre deux cercles :

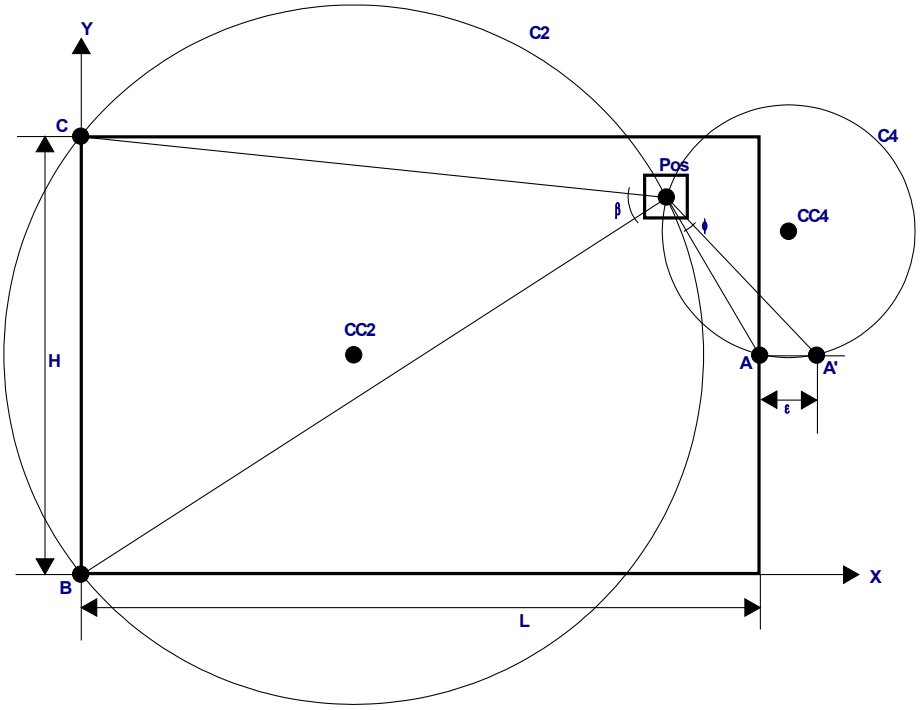

Figure 18 : intersection entre les cercles C2 et C4

Pour chaque cercle relatif à un angle, la notation est la suivante :

C2 : cercle relatif à β
C4 : cercle relatif à φ

CC2 : centre du cercle relatif à β
CC4 : centre du cercle relatif à φ

Le cercle C2, relatif à l'angle β, est trouvé de la même manière que pour la méthode normale (voir 2.3.1). Le centre du cercle CC4 peut être trouvé avec l'angle φ et en connaissant la position de la balise A et de son second capteur A'. De ce centre peut être déduit le rayon du cercle avec sa distance jusqu'à la balise A. On connaît donc l'équation complète d'un des cercles, sur lequel doit se trouver le robot.

Contrairement au premiers calculs d'intersection de cercles pour le positionnement du robot dans le cas normal, aucune des deux solutions n'est connue à l'avance. Pour remédier à ça, il faudrait utiliser l'angle α, et son cercle C1 qui passe par la balise A, tout comme le cercle C4, mais ce n'est pas optimal car l'imprécision de la mesure est nettement plus grande cet endroits que avec l'angle β.

3.2.1 Cercle C4 relatif à φ

Le cercle C4 est facile à trouver car son centre est situé sur une droite verticale située à mi-chemin entre A et A'. Aussi le cercle passe par A et A'. Pour la recherche du centre du cercle C4, on a donc :

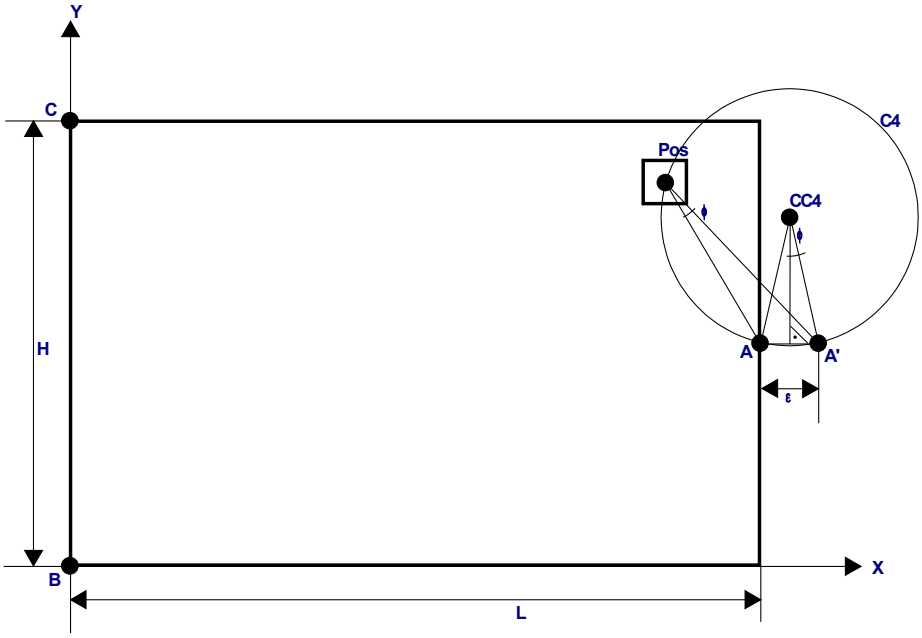

Figure 19 : recherche du centre du cercle C4

$$CC4x = L + (\frac{\varepsilon}{2}) \qquad CC4y = \frac{H}{2} \pm \frac{\varepsilon}{2 \cdot \tan(\varphi)}$$

Le paramètre CC4y à deux possibilités, en haut ou en bas, selon une symétrie horizontale. Pour faire le choix, il suffit de comparer les angles α et γ.

Si le robot est en haut :

$$\alpha < \gamma \qquad \alpha < (180° - \frac{\beta}{2})$$

$$CC4y = \frac{H}{2} + \frac{\varepsilon}{2 \cdot \tan(\varphi)}$$

Si le robot est en bas :

$$\alpha > \gamma \qquad \alpha > (180° - \frac{\beta}{2})$$

$$CC4y = \frac{H}{2} - \frac{\varepsilon}{2 \cdot \tan(\varphi)}$$

3.2.2 Intersection de cercles

Dans le cas spécial, l'intersection de cercles aboutissant à deux possibilités inconnues, on ne peut pas simplifier l'équation du second ordre, et on devra donc faire face à une racine carrée. Afin de mieux comprendre le problème, voici un cas très général d'intersection de cercles :

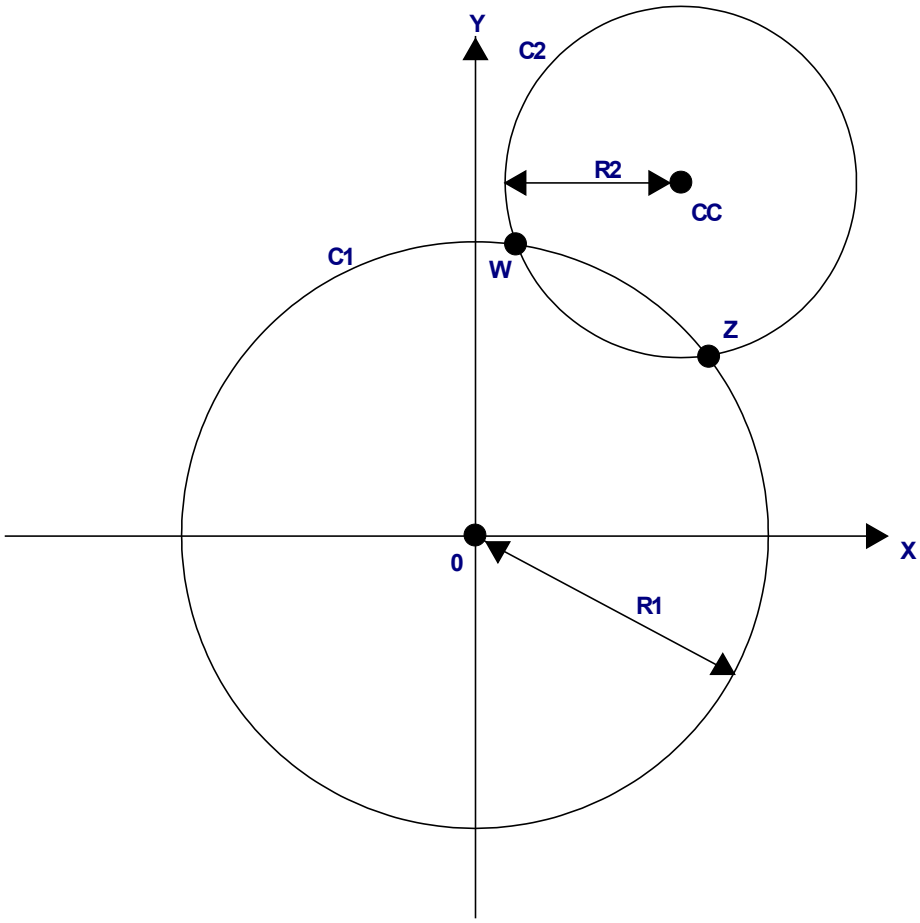

Figure 20 : problème d'intersection de cercles

Le problème est traité uniquement pour le cas ou les deux cercles ont deux intersections. Si les rayons des cercles ne le permettent pas, alors il peut y avoir une ou aucune intersection. A partir du centre du cercle C2, ainsi que des rayons R1 et R2, on cherche les intersections W et Z.

En admettant que :

$$CC = (CCx\,;\,CCy)$$

$$W = (Wx\,;\,Wy)$$

$$Z = (Zx\,;\,Zy)$$

Alors on a les équations des cercles :

$$R1^2 = Wx^2 + Wy^2$$

$$R2^2 = (CCx - Wx)^2 + (CCy - Wy)^2$$

$$R2^2 = CCx^2 - 2\,CCx\,Wx + Wx^2 + CCy^2 - 2\,CCy\,Wy + Wy^2$$

En faisant la différence, on cherche à isoler Wy :

$$R2^2 - R1^2 = CCx^2 + CCy^2 - 2\,(CCx\,Wx + CCy\,Wy)$$

$$N = \frac{R1^2 + CCx^2 + CCy^2 - R2^2}{2} = CCx\,Wx + CCy\,Wy$$

$$Wx = \sqrt{R1^2 - Wy^2}$$

$$N - Wy\,CCy = CCx\,Wx = CCx\sqrt{R1^2 - Wy^2}$$

$$N^2 - 2\,N\,CCy\,Wy + Wy^2\,CCy^2 = CCx^2\,(R1^2 - Wy^2)$$

$$(N^2 - CCx^2\,R1^2) - (2\,N\,CCy)\,Wy + (CCy^2 + CCx^2)\,Wy^2 = 0$$

D'après cette équation du deuxième ordre, on peut trouver les coordonnées des deux intersections W et Z :

$$Wy\,,\,Zy = \frac{N\,CCy \pm \sqrt{N^2\,CCy^2 - (CCy^2 + CCx^2)(N^2 - CCx^2\,R1^2)}}{CCy^2 + CCx^2}$$

$$Wx = \sqrt{R1^2 - Wy^2} \qquad\qquad Zx = \sqrt{R1^2 - Zy^2}$$

Afin de choisir entre le plus ou le moins, pour trouver Wy ou Zy, il faut observer le problème géométrique, et, dans le cas présent, pour Wy, on choisirait le plus, car selon la figure 20, W est situé plus haut en Y. Ce raisonnement est juste car la racine est forcément positive, et que le dénominateur l'est aussi, ainsi en choisissant le plus, on s'assure d'avoir une valeur plus grande.

3.2.3 Position du robot

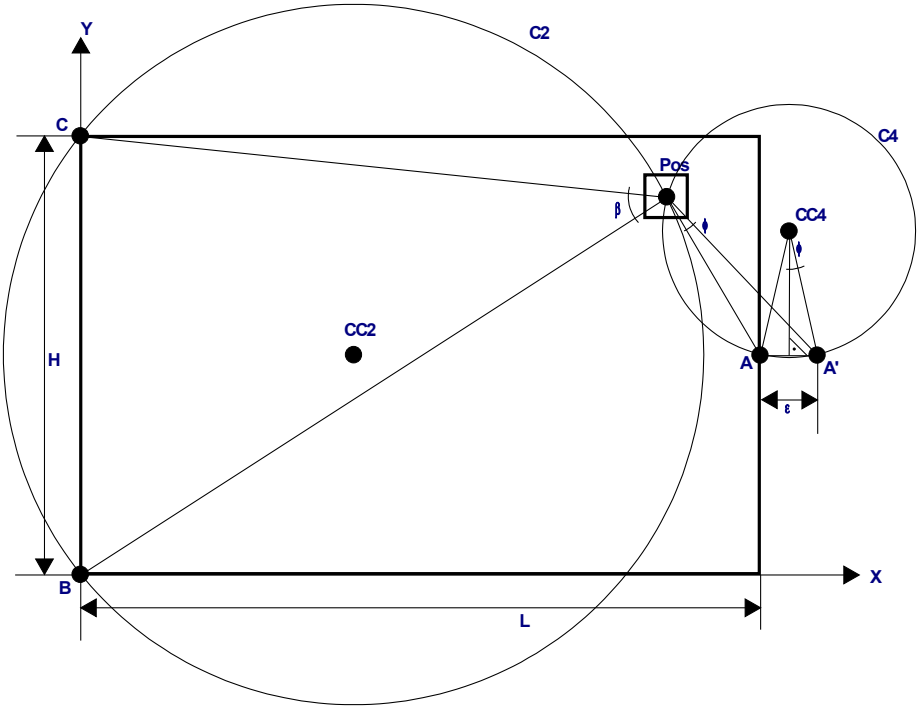

Figure 21 : détail de l'intersection entre les cercles C2 et C4

On cherche à appliquer la formule générale d'intersection de cercles à ce problème, ainsi on calcule :

$$CCy = CC4y - CC2y \qquad CCx = CC4x - CC2x$$

Et pour le décalage de l'origine, il suffira d'ajouter au résultat final les coordonnées de CC2, les résultats intermédiaires étant relatifs au centre du cercle C2.

Les rayons des cercles sont :

$$R2 = \sqrt{CC2x^2 + CC2y^2} \qquad R4 = \sqrt{\left(\frac{\varepsilon}{2}\right)^2 + \left(CC4y - \frac{H}{2}\right)^2}$$

Puis on introduit des variables temporaires pour regrouper un maximum de termes, et minimiser le nombre de calculs :

$$M = CCx^2 + CCy^2$$

$$N = \frac{R2^2 + M - R4^2}{2}$$

$$K = N\,CCy$$

Ce qui donne la formule finale de la position :

Si le robot est en haut :

$$\alpha < \gamma \qquad \alpha < (180° - \frac{\beta}{2})$$

$$Posy = \frac{K + \sqrt{K^2 - M(N^2 - CCx^2 R2^2)}}{M} + CC2y$$

Si le robot est en bas :

$$\alpha > \gamma \qquad \alpha > (180° - \frac{\beta}{2})$$

$$Posy = \frac{K - \sqrt{K^2 - M(N^2 - CCx^2 R2^2)}}{M} + CC2y$$

Avec dans tous les cas :

$$Posx = \sqrt{R2^2 - (Posy - CC2y)^2} + CC2x$$

4. Angle du robot

Lorsque le système des balises laser est en fonction, deux robots peuvent être positionnés d'après les impulsions mesurées sur les balises fixes, et les instants auxquels elles apparaissent. Les informations sont ensuite envoyées vers la balises fixe principale, M, dans laquelle tourne un des deux lasers. A l'intérieur de cette balise, est également monté une fourche optique, qui permet donc d'après l'instant auquel sont reçues les informations des balises fixes, et d'après l'instant auquel cette fourche est balayée de calculer l'angle du robot sur la table. Ceci est vrai à condition que les informations envoyées par les balises fixes soient retardé d'un délai constant après l'impulsion sur une des balises fixes.
Aussi on suppose la vitesse de rotation du laser comme constante.

4.1 Mesure des temps

Si on prend la balise A comme balise de référence pour l'angle, cela signifie qu'après un temps constant suite à l'impulsion A, on transmet les informations par l'infrarouge et/ou le module HF. Attention de bien compter un délai constant après l'impulsion du capteur principal de la balise A, et non pas le secondaire :

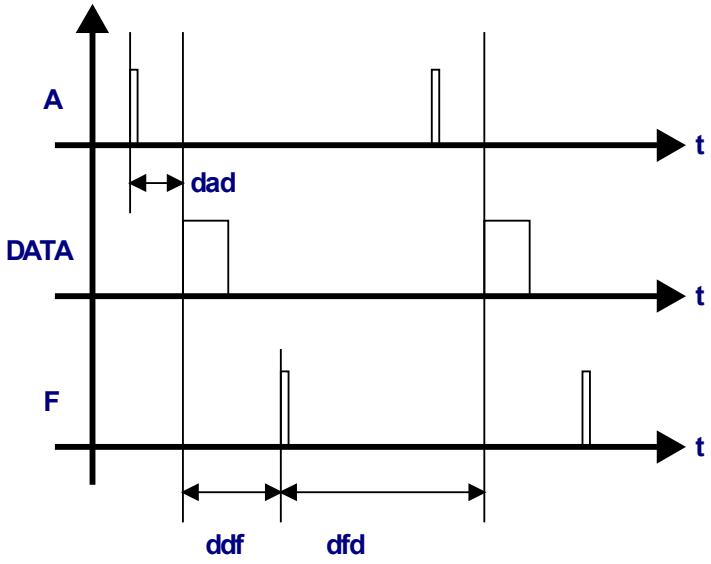

Figure 22 : représentation temporelle pour la mesure de l'angle du robot

On a :

A : instant de balayage de la balise A
DATA : instant de réception des données envoyées depuis la balise A
F : instant de passage dans la fourche optique montée dans la balise M

Le délai dad est constant, et va dépendre du temps de traitement des balises fixes, pour envoyer les informations d'angles ou de temps à la balise fixe M.
A partir des temps ddf et dca, on peut déduire les angles qui correspondent à ces temps, en partant du principe que la somme des deux temps est l'équivalent d'un tour complet. Ceci est vrai car la vitesse de rotation est considérée comme constante.

4.2 Situation géométrique

Voici le problème d'un point de vue géométrique

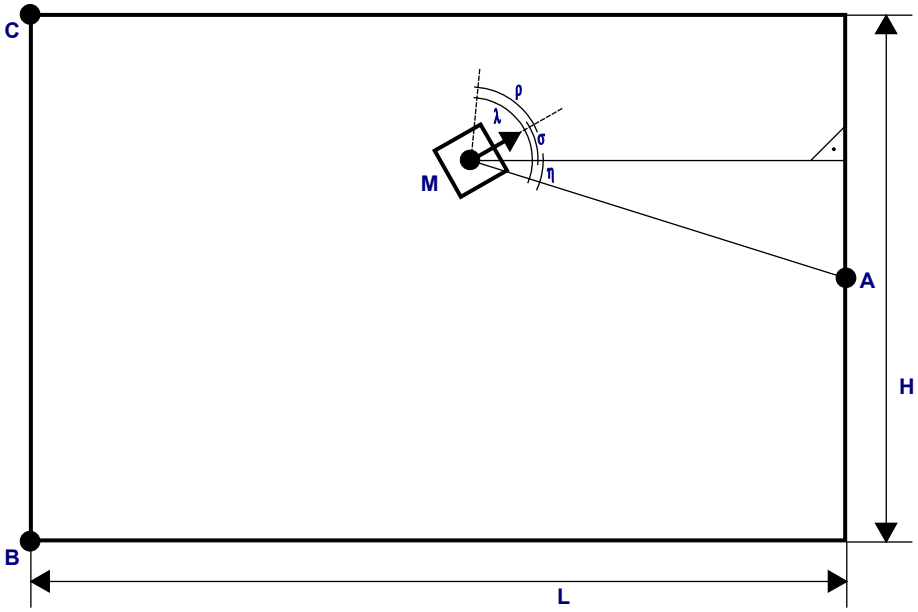

Figure 23 : mesure de l'angle du robot avec la balise M

La balise A ainsi que le robot M sont situés à des endroits connus :

$$A=(L\,;\frac{H}{2}) \qquad M=(Mx\,;My)$$

On a les angles :

σ : angle que l'on recherche, angle du robot par rapport à la table
η : angle qui dépend de la position du robot sur la table
ρ : angle de la fourche optique par rapport au robot, constant
λ : angle mesuré

On détermine l'angle λ d'après les temps dad, ddf et dfd, en degrés :

$$\lambda=\frac{ddf+dad}{ddf+dfd}\cdot 360\,°$$

L'angle η vient avec la relation trigonométrique suivante :

$$-\eta=\arctan\left(\frac{My-Ay}{Mx-Ax}\right)$$

Et donc on a un angle de robot, exprimé en degrés :

$$\sigma = \lambda - \eta - \rho = \frac{ddf}{ddf + dfd} + \arctan\left(\frac{My - Ay}{Mx - Ax}\right) + \frac{dad}{ddf + dfd} - \rho$$

En supposant que la vitesse de rotation soit quasiment constante, on peut regrouper deux constantes :

$$G = \frac{dad}{ddf + dfd} - \rho$$

Cette constante dépends donc du délais dad et de l'angle ρ. Pour trouver cette constante, il suffit de la mettre à zéro, de faire une mesure en mettant le robot sur la table avec un angle σ réel nul, puis d'observer l'angle σ mesuré. D'après sa valeur dans ce cas-la, on obtient la constante G

On a donc :

$$\sigma = \frac{ddf}{ddf + dfd} + \arctan\left(\frac{My - Ay}{Mx - Ax}\right) + G$$

Attention, cette méthode va donner un angle qui variera légèrement en fonction de la vitesse de rotation, qui peut varier à long terme, mais aussi du rapport entre dad et le temps de faire un tours complet. Pour faire mieux, il est nécessaire de connaître le temps dad; la meilleure façon étant de mesurer le délai entre l'impulsion du capteur principal sur la balise A, et l'émission des données de cette même balise. Cette mesure peut se faire entièrement sur la carte du microcontrôleur de la balise A.

Une fois cette mesure effectuée, il s'agit de la convertir dad dans la même base de temps que les mesures effectuées en interne de la balise M, et d'utiliser désormais l'équation :

$$\boxed{\sigma = \frac{ddf - dad}{ddf + dfd} + \arctan\left(\frac{My - Ay}{Mx - Ax}\right) + G}$$

Avec G qui est la constante qui ne compense désormais que la position de la fourche optique qui n'est pas forcément à l'avant du robot. Cette constante doit être mesurée à nouveaux, car dans ce second cas de figure plus précis pour la mesure de l'angle du robot, elle sera légèrement plus petite, car elle n'inclura plus le temps dad.

5. Implémentation software

Ce chapitre traite de l'implémentation software des calculs géométriques uniquement. Les calculs étant long et faisant appels à une multitude de fonctions différentes, il est indispensable de bien respecter le travail mathématique à effectuer pour avoir une bonne précision avec le système, et une rapidité de calcul maximale.

5.1 Tâches pour les calculs

Voici un organigramme simplifié qui résume l'ordre des tâches à effectuer :

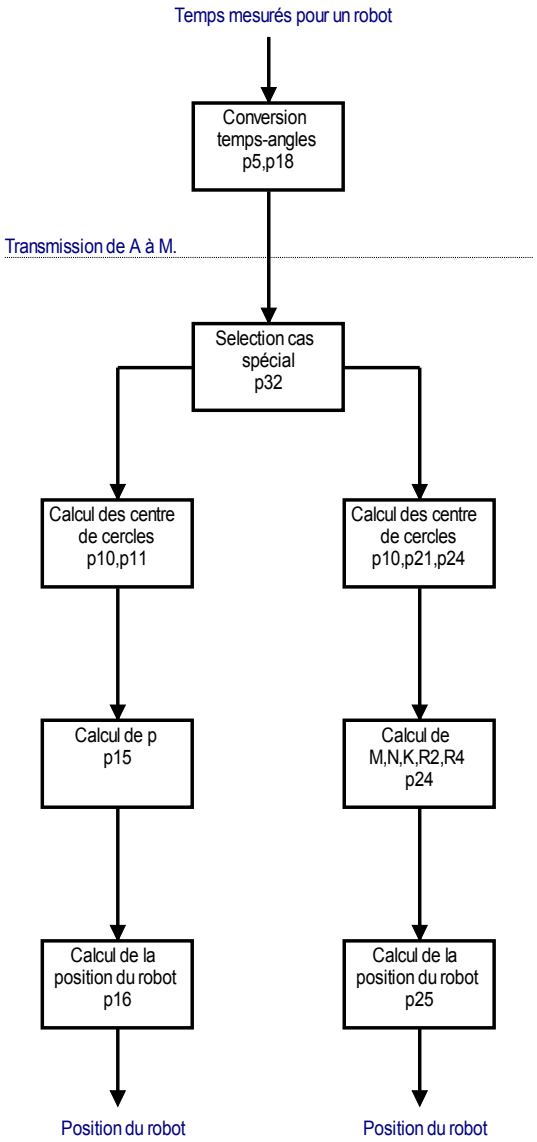

Figure 24 : Tâches à effectuer pour les calculs géométriques

Les numéros de pages au bas des cases font références aux numéros de pages de cette documentation, principalement la ou la tâche est abordée d'un point de vue théorique.

5.2 Pièges à éviter

Il faut particulièrement bien faire attention à certaines finesses lors de la programmation, qui sont mentionnées ici.

5.2.1 Conversion temps en angles

Lors de la lecture des impulsions, qui est implémentée par une routine d'interruption, et du comptage des temps entre les impulsions, il se peut que l'on se retrouve dans deux cas de figures :

Le capteur en profondeur de la balise A n'est pas balayé, car le robot se trouve par exemple au milieu de la table, et qu'il ne le "voit" pas :

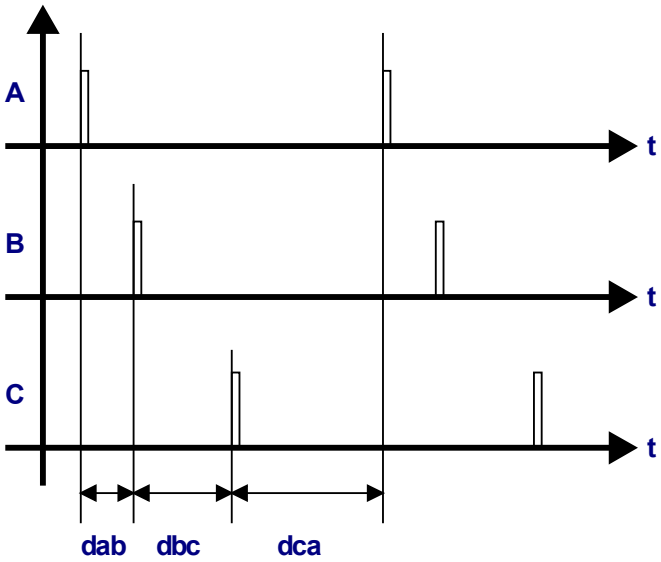

Figure 25 : représentation temporelle des impulsions sur les balises fixes pour un robot

Le robot est dans une position qui lui permet de balayer le capteur en profondeur :

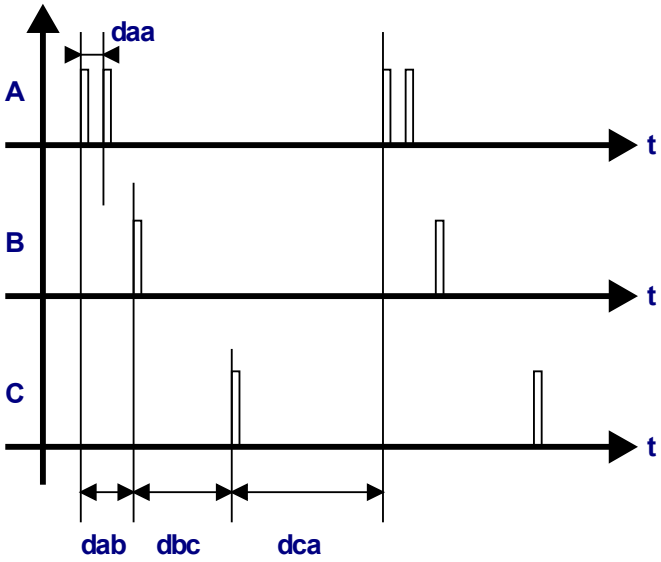

Figure 26 : impulsions mesurées sur les balises avec le capteur latéral de la balise A

Dans les deux cas, on peut se trouver dans une la zone de la table qui n'est pas réservée au cas spécial, avec son calcul particulier, bien que le capteur latéral soit balayé. Donc, la routine d'interruption doit être capable de mesurer des temps dab, dbc et dca avec précision, et sans confondre les deux impulsions sur la balise A. Aussi il n'est pas sur que la première impulsion sur la balise A soit celle du capteur principal, et donc on risque de faire un calcul de positionnement avec des angles faux. Pour éviter ce problème, il faut mesurer les temps comme indiqué sur le graphe de la figure 26, et corriger α si nécessaire :

Si il n'y a qu'une impulsion mesurée sur la balise A, il n'y a pas de question à se poser, et le temps daa est nul, donc pas de risque de confondre les deux capteurs de A :

$$\alpha = \frac{dab}{dab + dbc + dca} \cdot 360\,°$$

Si deux impulsions sont mesurées :

Si : $dab < dca$

$$\boxed{\alpha = \frac{(dab - daa)}{dab + dbc + dca} \cdot 360\,°}$$

Si : $dab > dca$

$$\boxed{\alpha = \frac{dab}{dab + dbc + dca} \cdot 360\,°}$$

Cette petite correction peut apparaître comme négligeable, mais en réalité elle est très importante, car sans cela la qualité du positionnement diminue très fortement.

La routine d'interruption qui mesures les temps est très importante, si cette dernière présente des défauts, c'est tout le positionnement qui peu devenir très imprécis.

Selon la bande passante de l'électronique du système, le temps de réaction des capteurs est de l'ordre de 4µs, ce qui implique que le timer qui sert de base de temps pour compter doit être cadencée au moins à cette vitesse. La vitesse de rotation du laser étant de 10 à 14 tours par seconde, cela représente une résolution de l'ordre de 20'000 incréments par tours, ce qui justifie une très bonne précision du système, et donc la nécessité de mesurer les angles au moins aussi précisément.

5.2.2 Détection du cas spécial

Afin de détecter le cas spécial, la solution la plus simple et fiable est de détecter si un des deux angles α ou γ est supérieur à 236°. On est dans le cas spécial si :

$$\alpha > 236\,°$$

Ou si :

$$\gamma > 236\,° \qquad 360\,° - \alpha - \beta > 236\,° \qquad \alpha + \beta < 124\,°$$

Cette détection peut se faire immédiatement après la mesure des temps et la conversion en angle, voici sur un terrain la zone concernée :

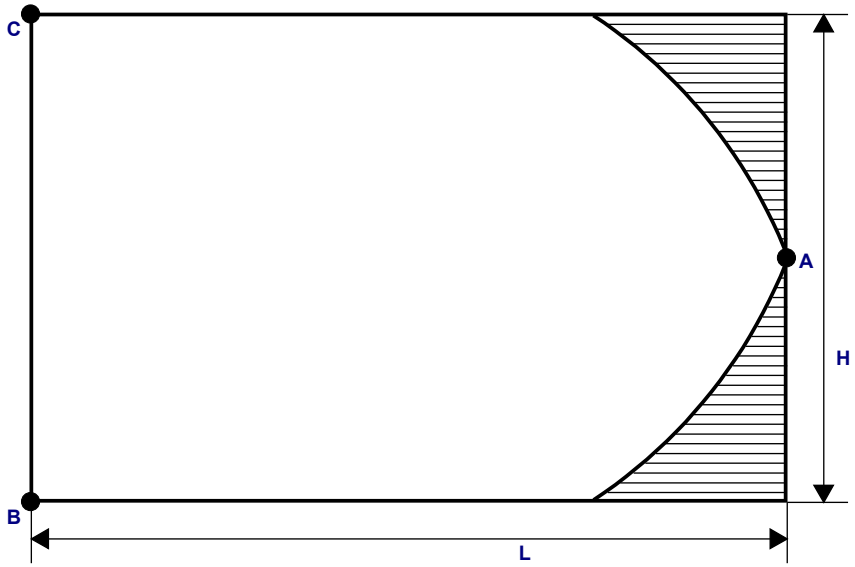

Figure 27 : zone concernée par le cas spécial

Ceci est valable pour une table dont les dimensions sont :

$$L = 310\,cm \qquad H = 200\mathrm{cm}$$

Si ces dimensions varient légèrement ce n'est pas très important de revoir la méthode de détection du cas spécial, car elle présente une certaine tolérance (qui dépend de la précision de l'électronique et de la précision du positionnement dans chaque cas).

5.2.3 Position du robot

Une fois les calculs terminés, il est très important de corriger la position du robot, car tous les calculs supposent que l'origine du problème géométrique est située sur le capteur de la balise B. Aussi la largeur L est relative à l'espace en largeur entre les capteurs de la balise A, et la balise B. La hauteur est relative à l'espace entre le capteur de la balise B et de la balise C.

5.3 Fonctions mathématiques en C

Afin de s'assurer un bon fonctionnement du système, tant en terme de temps d'exécution des calculs, que en terme de précision, il est nécessaire pour certaine opérations mathématiques de bien faire attention aux fonctions employées. Inutile d'avoir trop de précision, inutile de prendre trop de temps.

A la base, les temps mesurés par les timers des microcontrôleurs sont des nombres entiers, de 16 bits. Le travail effectué par la balise fixe A devrait être le plus petit possible pour éviter d'augmenter et de rendre trop variable le temps dad (voir 4.1). Ces temps mesuré en entiers sont convertis en float, et les angles sont calculé, puis normalisé en radiant, compris entre 0 et 2π, et transmis vers la balise M.
La balise M qui reçoit ces angles fait tout le travail de positionnement, et effectue toutes les opérations mathématique en float. Finalement, une fois la position retrouvée, elle est testée pour voir si elle est valide (dans la table), puis retournée au robot en entier, selon un quadrillage du terrain, par exemple en mm, ou plus petit.

Selon la documentation du compilateur MPLAB relative aux dsPIC, le format des nombre à virgule flottante est le standard IEEE 754, sur 32bits.

Tout le système à été simulé dans MATLAB, ainsi qu'un set de fonctions C spécialement conçues pour les balises. Ces fonctions C ont été compilées en dll pour pouvoir les utiliser directement avec MATLAB (et non pas simulées), afin de s'assurer de leurs précision.

Voici tous les opérateurs mathématiques employés pour retrouver la position du robot :

Opérateur	**Fonction C**	**Description**
$x+y$	x+y	Fonction originale adaptée et optimale
$x-y$	x-y	Fonction originale adaptée et optimale
$x \cdot y$	x*y	Fonction originale adaptée et optimale
x^2	x*x	Fonction originale adaptée et optimale
$\frac{1}{x}$	finv(x)	Fonction d'inversion, très rapide et suffisamment précise.
$\sqrt{x}$	fsqrt(x)	Fonction de racine carrée, très très rapide et suffisamment précise.
$\arctan(x)$	farctan(x)	Fonction d'arctangente rapide, et suffisamment précise.
$\frac{1}{\tan} x$	fcotan(x)	Fonction d'inverse de tangente rapide, et suffisamment précise.

Les quatre premières foncions sont plutôt rapides à faire avec un microcontrôleur 16 bit, même s'il s'agit d'opérer sur des float 32 bits. Les quatre autres sont plus difficiles a effectuer, et nécessitent la mise en place d'algorithme spécifiques.

5.3.1 Racine carrée

5.3.1.1 Méthode Babylonienne

Le problème de la racine carrée est depuis très longtemps étudié afin de trouver une méthode numérique rapide et efficace. Une méthode itérative déjà connue par le Babyloniens il y a plus de deux milles ans, est la suivante :

Pour :

$$y=\sqrt{x}$$

Alors :

$$y_{n+1}=(y_n+\frac{x}{y_n})\cdot\frac{1}{2}$$

En parcourant les indices n, on calcule chaque fois un résultat plus proche de la solution exacte. Pour la valeur initiale y(0), on pose une approximation de la racine. Exemple :

$$x=9,5 \qquad y=\sqrt{x}=\sqrt{9,5}$$

$$y_0\approx\sqrt{9}=3$$

$$y_1=(3+\frac{9,5}{3})\cdot\frac{1}{2}=3,08333333$$

$$y_2=(3,08333+\frac{9,5}{3,08333})\cdot\frac{1}{2}=3.08220721$$

Alors que la réponse "exacte" était :

$$y=\sqrt{9,5}=3.08220700$$

5.3.1.2 Méthode de Newton

Plus tard, cette méthode fut revisitée par Newton, qui de manière plus générale décrivit une méthode itérative pour approcher des fonctions :

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)}$$

Avec f(x) qui est la fonction qui trouve un zéro.

Voici ce que l'on a pour la racine carrée de x :

$$y = \sqrt{x}$$

$$f(y_n) = y_n^2 - x = 0$$

$$f'(y_n) = 2\,y_n$$

Alors vient :

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)} = y_n - \frac{y_n^2 - x}{2\,y_n} = \frac{1}{2}\left(y_n + \frac{x}{y_n}\right)$$

Qui est exactement la même chose que la méthode Babylonienne. En terme de performance lors d'implémentation informatique, une grande quantité de temps est gaspillé à faire la division. Cette dernière est aussi approchable avec Newton, mais imbriquer un algorithme dans un autre n'est pas optimal. C'est pourquoi pour éviter de diviser, il est nettement plus rapide d'approximer l'inverse de la racine carrée, la racine carrée étant retrouvable en multipliant le résultat par la valeur d'entrée.

Voici ce que l'on a pour l'inverse de la racine carrée de x :

$$y = \frac{1}{\sqrt{x}} \qquad \frac{1}{y^2} = x$$

$$f(y_n) = \frac{1}{y_n^2} - x = 0$$

$$f'(y_n) = \frac{-2}{y_n^3}$$

Alors vient :

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)} = y_n - \frac{\frac{1}{y_n^2} - x}{\frac{-2}{y_n^3}} = y_n + (y_n - y_n^3 x)\frac{1}{2} = y_n(3 - y_n^2 x)\cdot 0{,}5$$

Cette approximation de l'inverse de la racine carrée ne nécessite aucune division, et finalement pour retrouver la racine de x, il suffit d'une multiplication supplémentaire :

$$\sqrt{(x)} = x \cdot \frac{1}{\sqrt{(x)}}$$

5.3.1.3 Valeur initiale

Lors de l'implémentation de l'algorithme de l'inverse de la racine carrée, il faut trouver une méthode pour avoir une valeur initiale, aussi appelée estimateur.
Cette méthode doit présenter un bon compromis entre rapidité et précision, sinon autant utiliser un constante, et rajouter des itérations supplémentaires pour obtenir la précision voulue.

Lorsque le code source de Quake III à été rendu public, certaines personnes ont remarqué que ce jeux calculait l'inverse de racine carrée selon la méthode Newtonienne citée ci-dessus. Ce n'était pas surprenant car cet algorithme sans division est très souvent utilisé. Néanmoins une petite surprise était cachée du coté de la méthode pour trouver l'estimateur. Voici la fonction d'inversion de racine carrée en question :

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;  // evil floating point bit level hacking
    i  = 0x5f3759df - ( i >> 1 ); // what the fuck?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
    // y  = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iteration, this can be removed

    #ifndef Q3_VM
    #ifdef __linux__
        assert( !isnan(y) ); // bk010122 - FPE?
    #endif
    #endif
    return y;
}
```

On remarque l'algorithme itératif employé, qui est celui de l'approche de la valeur finale par Newton. Pour Quake, ils n'ont eu besoin que d'une itération :

```
y  = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
// y  = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iteration, this can be removed
```

$$y_{n+1} = y_n (3 - y_n^2 x) \cdot 0{,}5$$

Voici la manière dont l'estimateur est crée :

```
i  = * ( long * ) &y;  // evil floating point bit level hacking
i  = 0x5f3759df - ( i >> 1 ); // what the fuck?
y  = * ( float * ) &i;
```

La première ligne de code est une conversion de type directe, un peu particulière, qui "voit" les données hexadécimales qui représentent un float directement comme un int. Le but est uniquement de récupérer les données.

La seconde ligne de code est la ligne de codes "magique" qui permet de trouver un estimateur de l'inverse de la racine carrée avec uniquement une soustraction d'entiers, et un décalage à droite. La constante magique 0x5f3759df est la clé de l'astuce.

Cette méthode géniale pour trouver un estimateur de très bonne qualité à été découverte par un mathématicien de MathWorks (Matlab), dans les années huitante. Étonnamment, Quake III à été le premier jeu vidéo à utiliser cette astuce pour calculer des racines carrées. Encore aujourd'hui, de très nombreuses librairies utilisent la méthode classique de Newton pour trouver les racines carrées, ce qui est dans la pratique environ quatre fois moins rapide, pour la même précision de calcul, car d'une part on emploie des divisions, et d'autre part il n'y a pas d'astuce comparable pour trouver un estimateur de la racine carrée.

En 2003 un mathématicien, Chris Lomont, effectua des travaux sur la constante magique employée pour trouver un estimateur (www.lomont.org/Math/Papers/2003/InvSqrt.pdf), selon ce dernier, il vaut mieux employer la constante 0x5f375a86, qui donne des résultats très légèrement plus précis.

5.3.1.4 Fonction C

Voici donc la fonction de racine carrée qu'il est recommandé d'utiliser dans tous les cas ou on travaille en simple précision (aussi sur PC) :

```
float fsqrt( float in )
{
    long estimator;
    float out,inhalfs;
    const float threehalfs=1.5F;

    inhalfs=in*0.5F;

    estimator=*(long*)&in; // get bits from float to int
    estimator=0x5f375a86-(estimator>>1); // strange estimator...
    out =*(float*)&estimator; // get bits back from int to float

    out*=(threehalfs-(inhalfs*out*out)); // Newton iteration
    out*=(threehalfs-(inhalfs*out*out)); // Newton iteration
    //out*=(threehalfs-(inhalfs*out*out)); // Newton iteration
    //out*=(threehalfs-(inhalfs*out*out)); // Newton iteration

    return out*in;
}
```

Deux itérations sont nécessaire pour avoir une précision satisfaisante pour les balises laser, une n'étant pas du tout suffisante, trois étant excessif. Pour avoir plus de précision, si cette fonction est réemployée ailleurs il est possible de décommenter les deux itérations supplémentaires.

Une fois la fonction compilée en dll, elle à été testée dans Matlab, voici l'imprécision relative de la fonction :

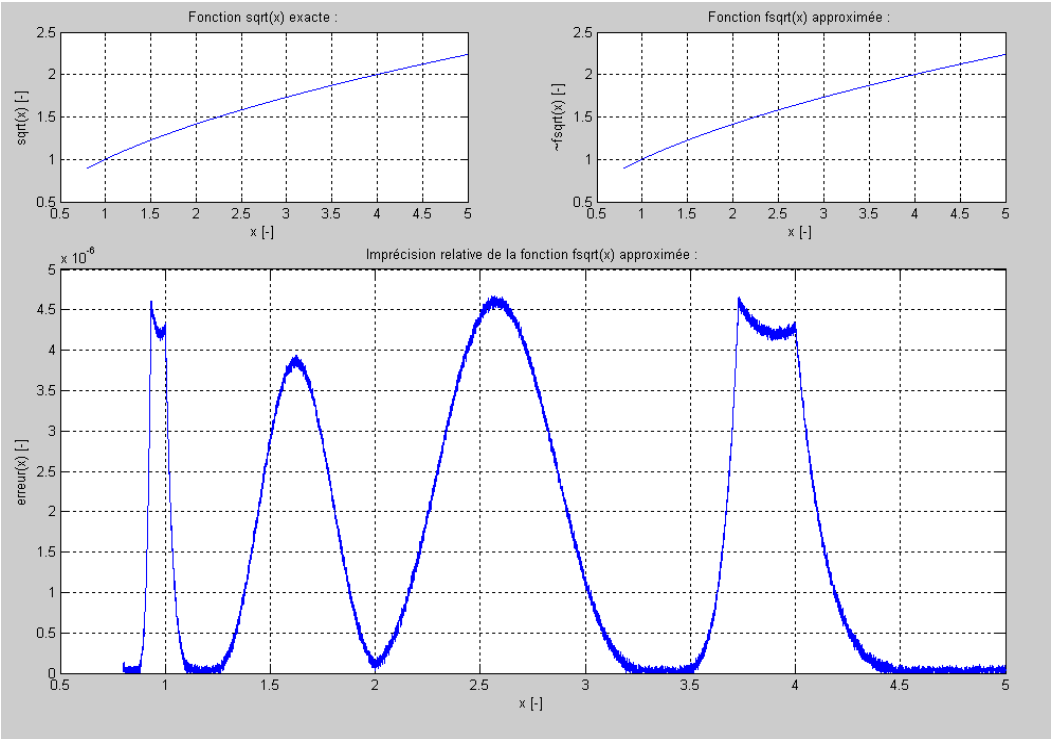

Figure 28 : imprécision relative de la fonction de racine carrée

L'imprécision relative est cyclique par quatre, ce qui est logique par rapport à la méthode pour trouver l'estimateur, qui travaille que sur la mantisse du nombre à virgule flottante, cyclique par deux, deux étant la racine de quatre.

En faisant quatre itérations, on obtiendrais une précision maximale par rapport au format à virgule flottante simple précision, cinq itérations sont inutiles.

5.3.2 Inversion

Trouver l'inverse d'un nombre à virgule flottante n'est pas une chose facile, de plus, pour positionner le robot, il est nécessaire de faire une grande quantité d'inversions. Pour cette raison, il est indispensable de bien maîtriser cette fonction, pour avoir suffisamment de précision, et prendre le moins de temps au niveaux des calculs.

Lorsque ce problème vient avec la programmation sur PC, certaines instructions assembleurs x86 permettent de faciliter grandement cette tâche, et d'accélérer le processus. La méthode décrite ici est donc optimale uniquement sur un microcontrôleur dont le set d'instruction assembleur est simple, avec au mieux des multiplications d'entiers. C'est donc le cas du dsPIC employé pour les balises laser. Il est donc indispensable de remplacer tous les opérateurs de division (/) en dessous desquels il y a une variable, par la multiplication de l'inverse de la variable, inversée par la fonction décrite ici. Ceci n'est pas valable pour les division par des constantes, qui sont automatiquement remplacées par des multiplication par le compilateur.

5.3.2.1 Méthode de Newton

En appliquant la méthode itérative de Newton à ce problème :

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)}$$

En cherchant f(x) qui est une des fonctions qui trouve un zéro, et qui donne bien sûr un résultat final sans division. Si ce n'était pas le cas, alors cela serait inutile de chercher à approximer une inversion, si l'approximation nécessite elle même une inversion.

Voici ce que l'on a pour l'inverse de x :

$$y = \frac{1}{x} \qquad 1 = \frac{1}{x\,y}$$

$$f(y_n) = \frac{1}{x\,y_n} - 1 = 0$$

$$f'(y_n) = \frac{-1}{x\,y_n^2}$$

Alors vient :

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)} = y_n - \frac{\frac{1}{x\,y_n} - 1}{\frac{-1}{x\,y_n^2}} = y_n + (\frac{1}{x\,y_n} - 1)(x\,y_n^2) = y_n(2 - x\,y_n)$$

Cet algorithme itératif est de très bonne qualité et/ou très rapide, car pour chaque itération, on a besoin que d'une soustraction et deux multiplication.

5.3.2.2 Valeur initiale

Lors de l'implémentation de l'algorithme de l'inverse, il faut trouver une méthode pour avoir une valeur initiale, aussi appelée estimateur. Cette méthode doit présenter un bon compromis entre rapidité et précision, sinon autant utiliser un constante, et rajouter des itérations supplémentaires pour obtenir la précision voulue.

En général, il est recommandé d'utiliser la méthode suivante pour l'estimateur :

$$\frac{1}{x} \approx 1{,}5 - \frac{x}{2}$$

Ceci est valable pour des x qui sont compris entre un et deux, car lors de l'implémentation de ceci avec des nombre à virgule flottante, l'inversion ne travaille que sur la mantisse, l'exposant étant simplement le même avant qu'après, moyennant une inversion de signe. Plus précisément, si on recherche à améliorer l'estimation dans la plage mentionnée :

$$\frac{1}{x} \approx \frac{3}{2} - (\frac{3}{4} - \frac{2}{3})\frac{1}{2} - \frac{x}{2} = 1{,}458333 - \frac{x}{2}$$

Néanmoins, cette méthode est plutôt longue à implémenter, car beaucoup d'opérations supplémentaire sont nécessaires par rapport à la méthode très rapide utilisée pour trouver un estimateur de l'inverse de la racine carrée. Cette dernière méthode est d'ailleurs réutilisable ici, en multipliant simplement son résultat par lui-même, car :

$$\frac{1}{x} = (\frac{1}{\sqrt{x}})^2$$

Les trois méthodes pour trouver un estimateur d'inversion ont été codées en C, et testés compilées dans Matlab. Entre la première et la seconde méthode, le peaufinage de la constante permet de diminuer l'imprécision par quatre. Entre la seconde méthode et la méthode de l'estimateur de l'inverse de la racine carrée, la vitesse est augmentée de 50% , et le résultat un peu plus précis. On choisit donc de réutiliser la méthode de l'estimateur de l'inverse de la racine carrée, que l'on multiplie par lui même.

5.3.2.3 Fonction C

Voici donc la fonction d'inversion qu'il est recommandé d'utiliser pour les balises laser :

```
float finv ( float in )
{
    long estimator;
    float out;
    const float two = 2.0F;

    estimator=*(long*)&in;  // get bits from float to int
    if (estimator&0x80000000){
        estimator=0x5f375a86-((estimator&0x7FFFFFF)>>1); // strange estimator...
        out=*(float*)&estimator; // get bits back from int to float
        out*=-out; // estimator was ~1/sqrt(-x)
    }else{
        estimator=0x5f375a86-(estimator>>1); // strange estimator...
        out=*( float*)&estimator; // get bits back from int to float
        out*=out; // estimator was ~1/sqrt(x)
    }

    out*=(two-(in*out)); // Newton iteration
    out*=(two-(in*out)); // Newton iteration
    //out*=(two-(in*out)); // Newton iteration
    //out*=(two-(in*out)); // Newton iteration

    return out;
}
```

Deux itérations sont nécessaire pour avoir une précision satisfaisante pour les balises laser, une n'étant pas du tout suffisante, trois étant excessif. Pour avoir plus de précision, si cette fonction est réemployée ailleurs il est possible de décommenter les deux itérations supplémentaires. Une fois la fonction compilée en dll, elle à été testée dans Matlab, voici l'imprécision relative de la fonction :

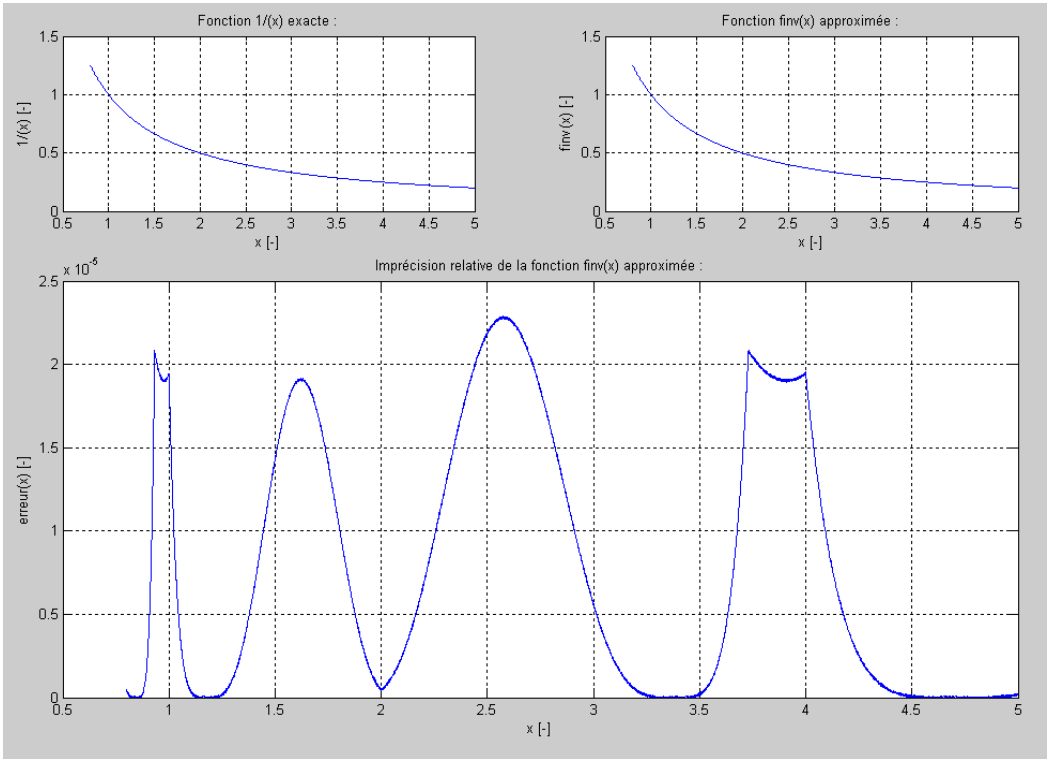

Figure 29 : imprécision relative de la fonction d'inversion

En faisant cinq itérations, on obtiendrais une précision maximale par rapport au format à virgule flottante simple précision, six itérations sont donc inutiles.

5.3.3 Tangente inversée

Une opération qu'il faut effectuer sur chaque angle pour le positionnement du robot est la fonction un sur tangente. On appelle également cette fonction cotangente. Attention à ne pas confondre avec la fonction réciproque de la tangente, l'arctangente.

En informatique, il n'y a pas de solution miracle, ni d'algorithme itératif pour calculer des fonction trigonométriques, donc en général on développe des séries de Taylor pour sinus et cosinus, et on trouve les autres fonctions par déduction, ou on utilise une table de lookup, énorme tableaux avec beaucoup de valeurs calculées à l'avance.

5.3.3.1 Séries de Taylor

Si l'on utilisait directement une série de Taylor de degré 15 pour approximer directement la cotagente, ce qui représente au moins 30 multiplications, la précision n'est pas suffisante pour les balises laser. Il faut donc trouver une autre solution.

Quelques rappels de trigonométrie :

$$\tan(x) = -\tan(-x) = \tan(x+\pi) = -\tan(\pi - x)$$

$$\tan(x) = \frac{1}{\tan(\frac{\pi}{2} - x)}$$

$$\boxed{\tan(x) = \frac{\sin(x)}{\cos(x)}}$$

Selon ces définitions, et disposant d'une fonction d'inversion rapide, on peut approximer la cotangente par un cosinus divisé par un sinus. Ceci est optimal, car cela permettra de bien approcher le comportement asymptotique de la fonction cotangente. Seul les valeurs entre 0° et 90° doivent être approximée, les autres valeurs sont obtenues par inversion ou opposition.

En développant en série de Taylor sinus et cosinus, on arrive assez facilement à avoir de bons résultats, surtout de 0 à 90°. Pour avoir une précision suffisante pour les balises laser, il suffit de développer sinus et cosinus seulement jusqu'au degré 9 et de retrouver la cotangente en divisant cosinus par sinus :

$$\boxed{\sin(x) \approx x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \frac{x^9}{9!} = x + \frac{x^3}{6} + \frac{x^5}{120} + \frac{x^7}{5040} + \frac{x^9}{362880}}$$

$$\boxed{\cos(x) \approx 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \frac{x^8}{8!} = 1 + \frac{x^2}{2} + \frac{x^4}{24} + \frac{x^6}{720} + \frac{x^8}{40320}}$$

5.3.3.2 Fonction C

Voici donc la fonction de cotangente qu'il est recommandé d'utiliser pour les balises laser :

```
float fctan ( float in )
{
    float out,in_,cos,sin;
    const float PI_2   = 6.28318530717958647692528676655900F;
    const float PI_3_2 = 4.71238898038468985769396507491930F;
    const float PI     = 3.14159265358979323846264338327950F;
    const float PI_1_2 = 1.57079632679489661923132169163980F;

    if (in>PI){
        if (in>PI_3_2){ // 4
            in_=PI_2-in;
        }else{          // 3
            in_=in-PI;
        }
    }else{
        if (in>PI_1_2){ // 2
            in_=PI-in;
        }else{          // 1
            in_=in;
        }
    }

    cos=1.0F;
    sin=in_;

    out=in_*in_;
    cos-=out*0.5F;
    out*=in_;
    sin-=out*0.16666666666666F;

    out*=in_;
    cos+=out*0.04166666666666F;
    out*=in_;
    sin+=out*0.00833333333333F;

    out*=in_;
    cos-=out*0.00138888888889F;
    out*=in_;
    sin-=out*0.00019841269841F;

    out*=in_;
    cos+=out*0.00002480158730F;
    out*=in_;
    sin+=out*0.00000275573192F;

    out=cos*finv(sin);

    if (in>PI){
        if (in>PI_3_2){ // 4
            return -out;
        }
    }else{
        if (in>PI_1_2){ // 2
            return -out;
        }
    }
    return out;
}
```

Attention, l'angle donné à la fonction doit être en radians, normalisé entre 0 et 2π.

Une fois la fonction compilée en dll, elle à été testée dans Matlab, voici l'imprécision relative de la fonction :

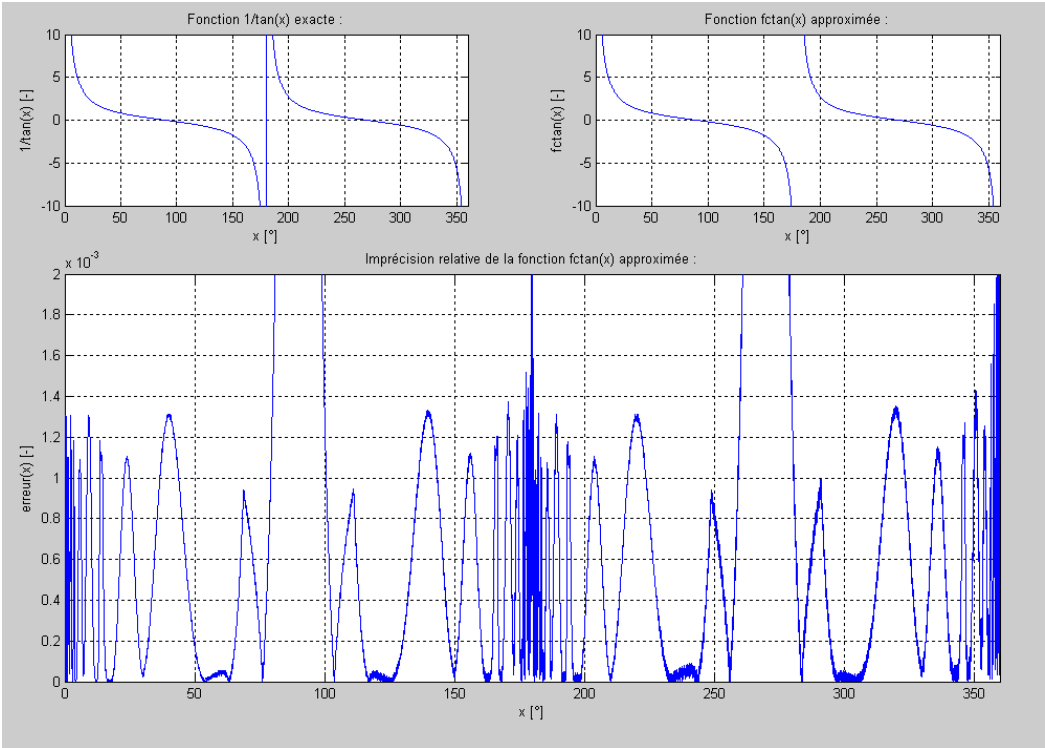

Figure 30 : imprécision relative de la fonction cotangente

La fonction présente une précision qui est suffisante pour les balises laser. Rajouter des termes aux séries de Taylor du sinus ou du cosinus n'est donc pas nécessaire. La fonction nécessite 17 multiplications, 2 soustractions, et quelques embranchements conditionnels. On remarque qu'elle emploie la fonction d'inversion rapide finv(). Sans cette fonction d'inversion, alors la fonction perd tout son sens, car ses performances serraient très diminuées.

5.3.4 Arctangente

Une arctangente est utilisée une seule fois, dans la balise M. Il s'agit d'une opération qui permet de trouver l'angle du robot sur la table. Voici quelques rappels des relations de base pour la fonction arctangente :

$$atan(x) = -atan(-x)$$

$$atan(x) = \frac{\pi}{2} - atan\left(\frac{1}{x}\right)$$

L'angle du robot ne nécessite pas une grande précision en comparaison des angles nécessaire au positionnement du robot. Une précision 50 fois moins grande permet d'avoir l'angle du robot à 1° près.

Il est donc judicieux de faire une approximation de l'arctangente spécialement rapide et dédiée aux balises laser, avec une imprécision sur l'angle de l'ordre de 1° ou moins.
Les séries de Taylor approximant de manière très médiocre la fonction arctangente, en pratique on choisit une autre approche.

5.3.4.1 Approximation par Padé

L'approximation par Padé pour l'arctangente est un bon compromis entre simplicité et rapidité, on a :

$$\boxed{atan(x) = \frac{\pi}{4}(1 - r)} \qquad r = \frac{1 - |x|}{1 + |x|}$$

Ceci est valable et optimal pour des angles entre 0° et 45°. Pour le calcul des autre plages d'angles, on utilise les conversions d'après les relations de bases.

Suite à de nombreux essais dans Matlab, une légère correction de r permet de gagner en précision :

$$atan(x) = \frac{\pi}{4}(1 - r) \qquad \boxed{r = \frac{1 - |x|}{1 + |x| \cdot 0.415}}$$

Cette correction permet d'atteindre 0.5° d'imprécision, avec une addition, une soustraction, une multiplication, et une division.

5.3.4.2 Fonction C

Voici donc la fonction d'arctangente qu'il est recommandé d'utiliser pour les balises laser :

```
float fatan ( float in )
{
    bool pos;
    float absin,r,out;
    const float PI_1_4 = 0.78539816339744830961566084581988F;

    pos=(in>0);

    if (pos){
        absin=in;
    }else{
        absin=-in;
    }

    if (absin<1){
        r=(absin-1.0F)*finv(absin*0.415F+1.0F);
        out=PI_1_4*(1.0F+r);
    }else{
        r=(1.0F-absin)*finv(absin+0.415F);
        out=PI_1_4*(1.0F-r);
    }

    if (pos){
        return out;
    }else{
        return -out;
    }
}
```

Attention, l'angle retourné par la fonction est en radians, normalisé entre -π et π.
Une fois la fonction compilée en dll, elle à été testée dans Matlab :

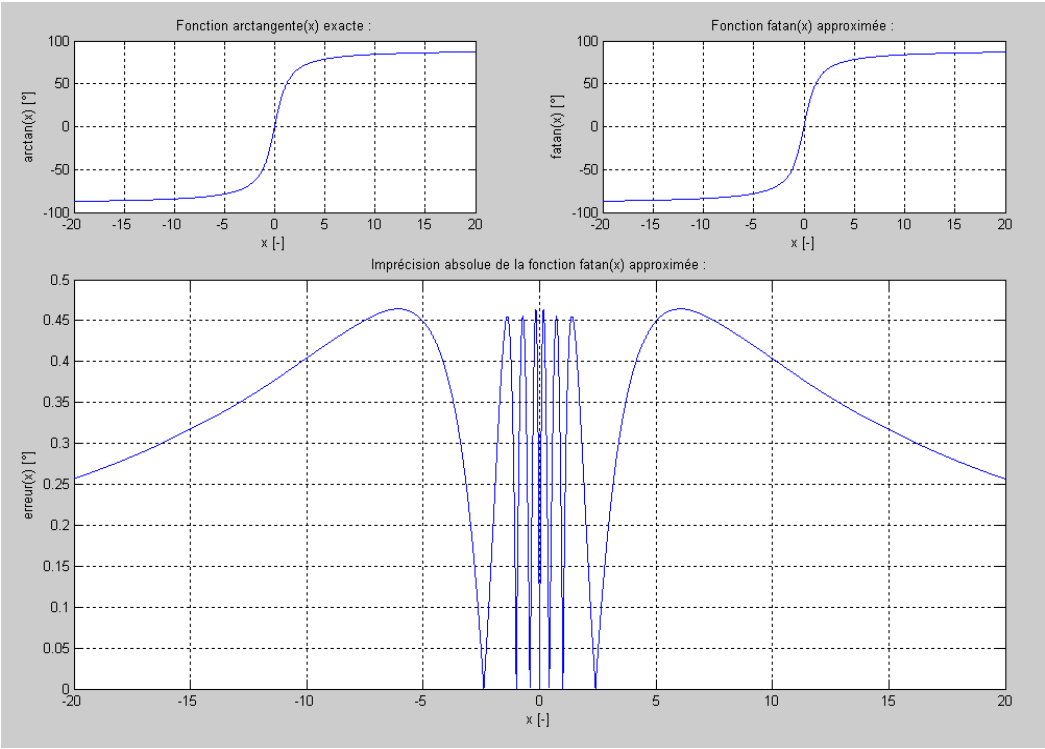

Figure 31 : imprécision absolue de la fonction arctangente

5.4 Imprécision du système

Lorsque tout le système est programmé conformément à cette documentation, et que les fonctions utilisées sont celles décrites dans cette documentation, on peut calculer l'imprécision du système en chaque point de la table.

Tout les calculs effectué dans Matlab sont conformes à ce qui se passe dans le microcontrôleur. Toutes les variales sont en virgule flotante, et tout le travail se fait de la même manière que le système programmé. Une marge de 15cm est appliquée aux bords de la table, zone inaccessible au robot.

5.4.1 Imprécision software

Voici l'imprécision en millimètres pour chaque position sur la table, engendrée par le software, par les arrondis du format float, et par l'imprécision des fonctions mathématiques implémentées en C :

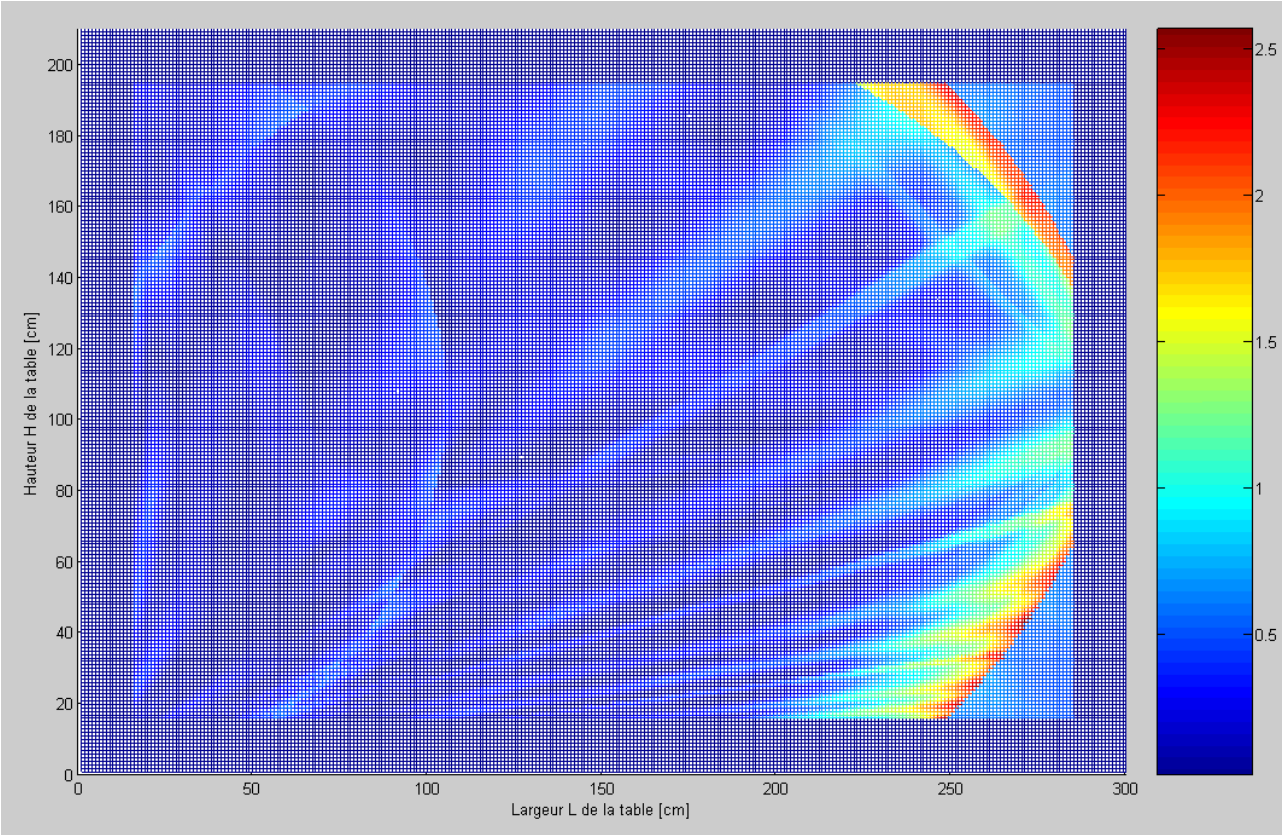

Figure 32a

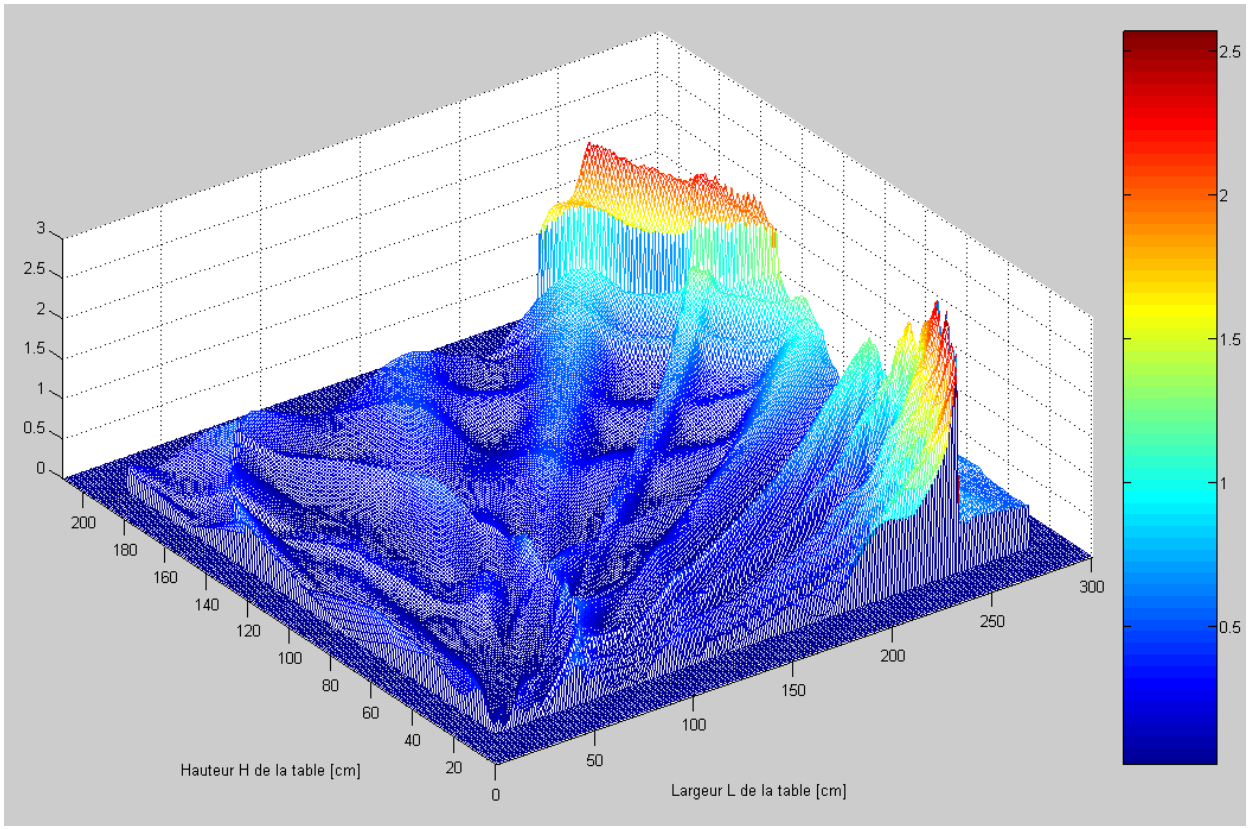

Figure 32b

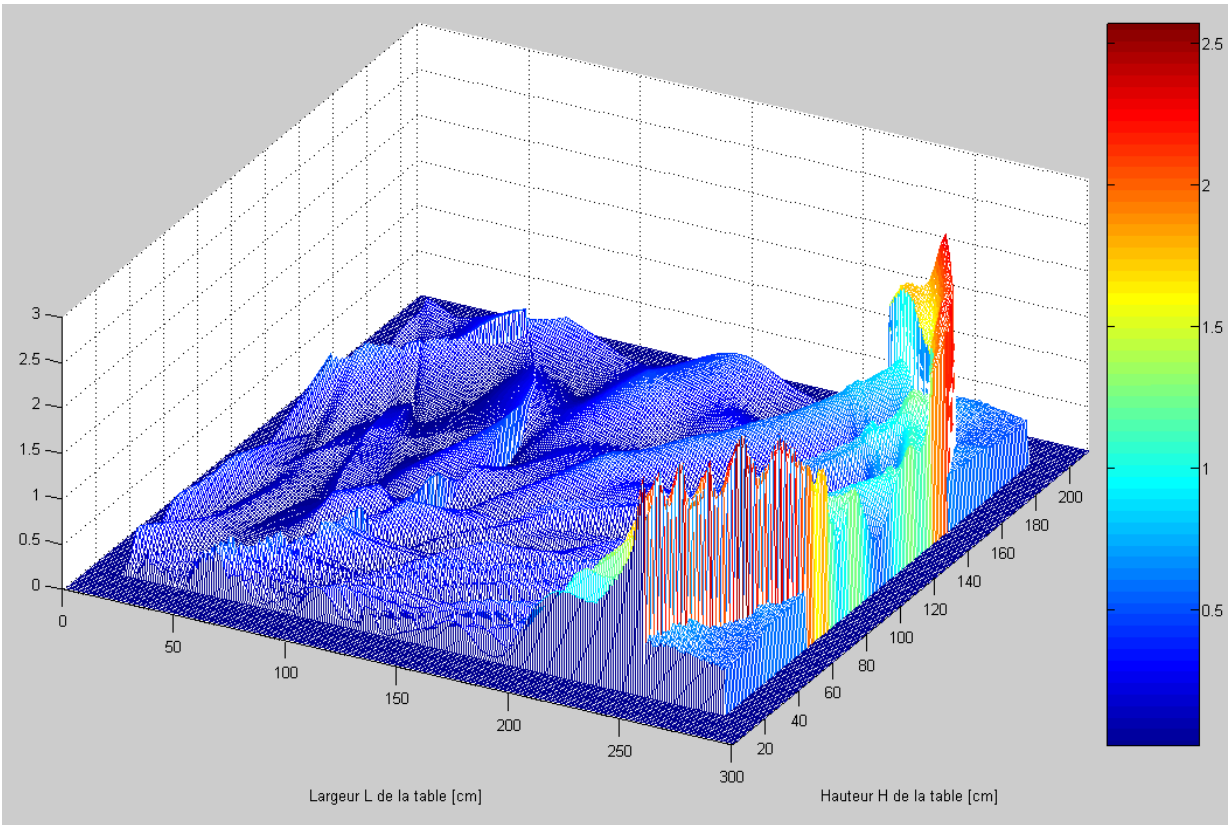

Figure 32c : imprécision software

On remarque dans les coins de droite, la zone ou l'on positionne le robot avec le capteur latéral de la balise A.

5.4.2 Imprécision hardware

Voici l'imprécision en millimètres pour chaque position sur la table, engendrée par le hardware, par le temps de réaction des capteurs, par la bande passante du système électronique :

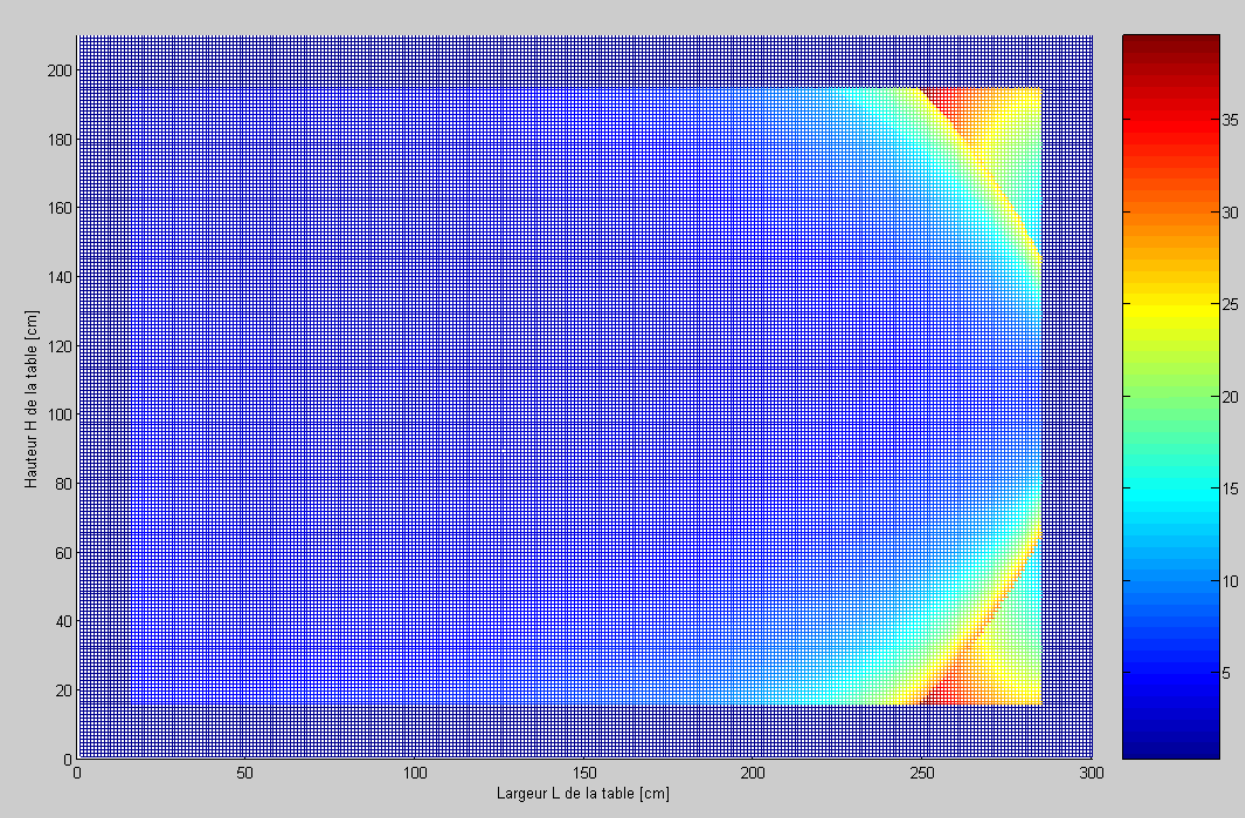

Figure 33a

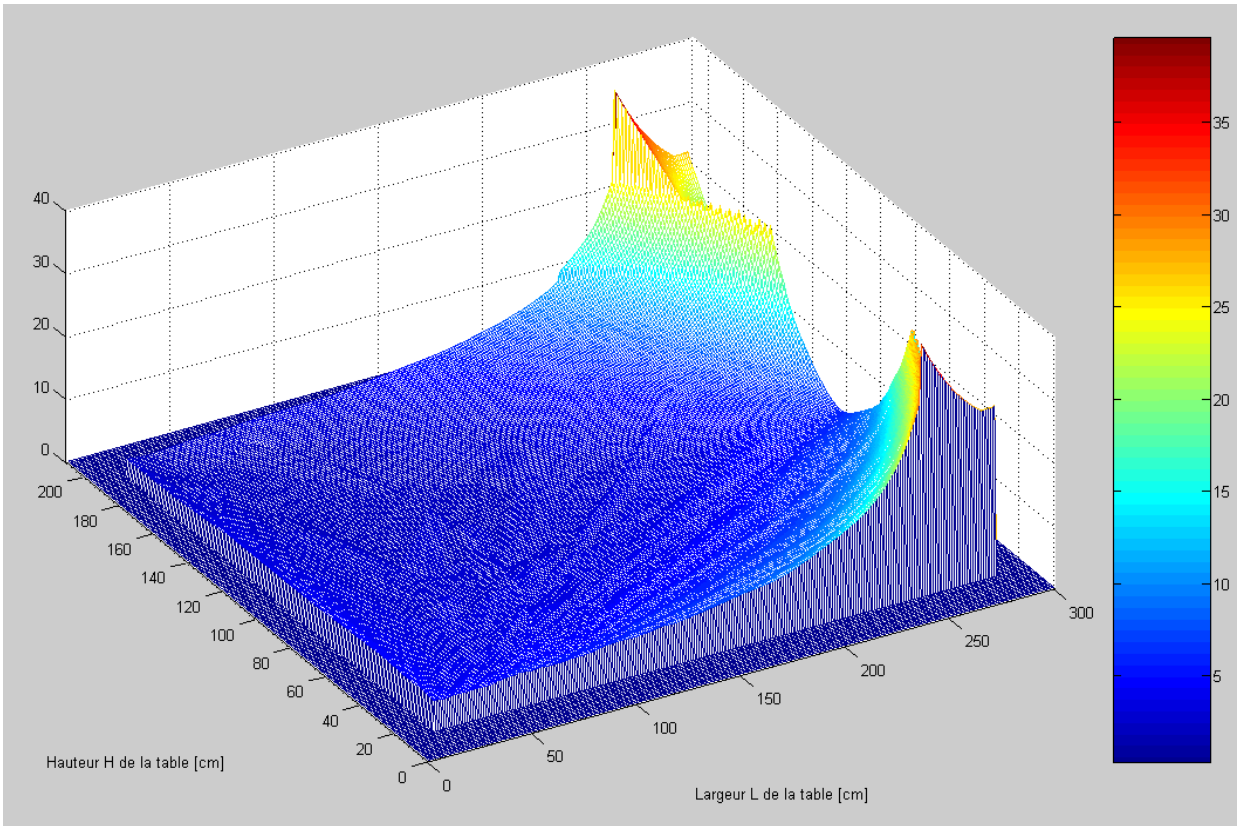

Figure 33b

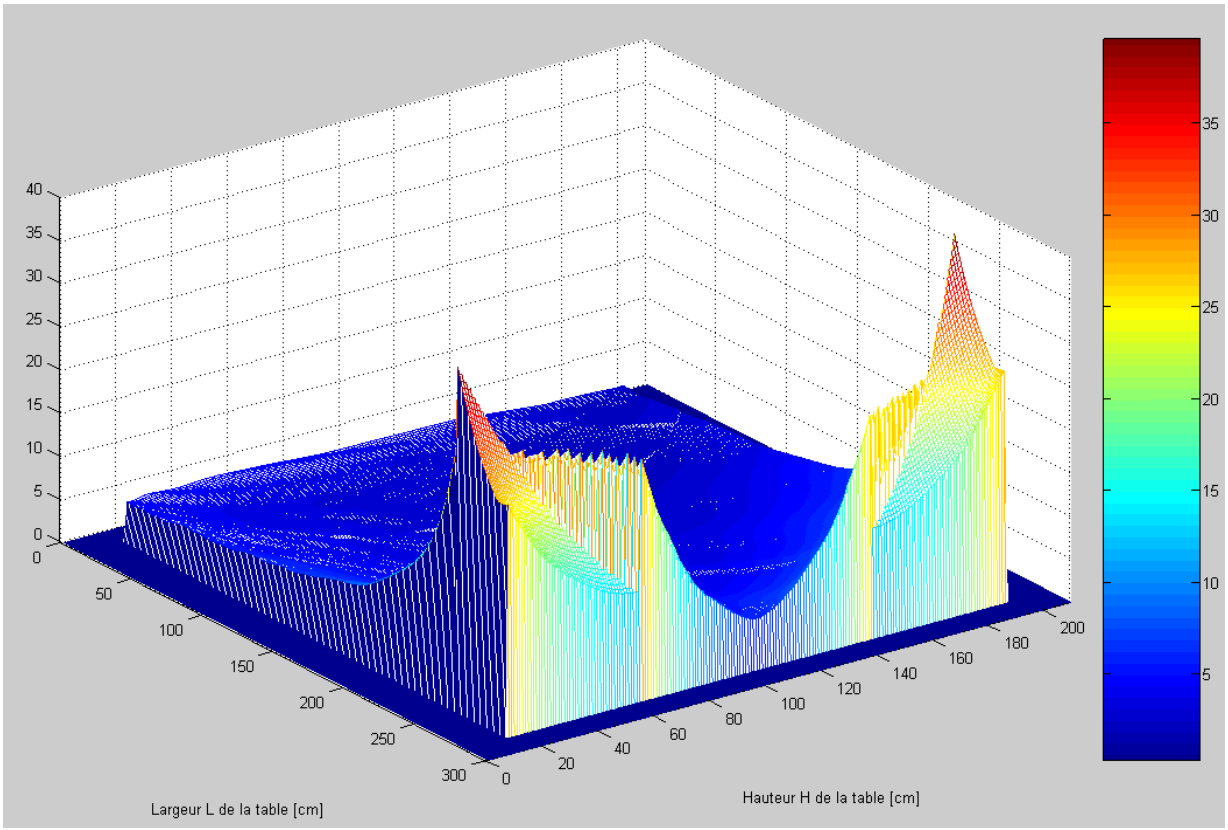

Figure 33c : imprécision hardware