



École Polytechnique de l'Université de Tours  
64, Avenue Jean Portalis  
37200 TOURS, FRANCE  
Tél. +33 (0)2 47 36 14 14  
[www.polytech.univ-tours.fr](http://www.polytech.univ-tours.fr)

**Département Informatique**  
**4<sup>e</sup> année**  
**2012 - 2013**

**Rapport de mini-projet Robotique**

**Test de l'accessibilité d'un point par un  
bras manipulateur**

**Encadrants**

Pierre Gaucher  
[pierre.gaucher@univ-tours.fr](mailto:pierre.gaucher@univ-tours.fr)

Université François-Rabelais, Tours

**Étudiants**

Zheng LIU  
[zheng.liu@etu.univ-tours.fr](mailto:zheng.liu@etu.univ-tours.fr)  
Lei SHANG  
[lei.shang@etu.univ-tours.fr](mailto:lei.shang@etu.univ-tours.fr)

DI4 2012 - 2013

Version du 6 janvier 2013



# Table des matières

---

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Présentation de l'algorithme IAA</b>	<b>6</b>
2.1	Problème réel concerné . . . . .	6
2.2	Catégories des solutions existantes . . . . .	6
2.3	Algorithme IAA . . . . .	7
<b>3</b>	<b>Travail effectué</b>	<b>8</b>
3.1	Complément de l'algorithme . . . . .	8
3.1.1	Premier cas . . . . .	8
3.1.2	Deuxième cas . . . . .	8
3.2	Implémentation de l'algorithme IAA . . . . .	9
3.2.1	Choix de langage . . . . .	9
3.2.2	Le diagramme de classes . . . . .	10
3.2.3	Le code source critique de l'algorithme . . . . .	11
3.3	Test de l'algorithme IAA . . . . .	13
3.3.1	Un exemple spécifique . . . . .	13
3.3.2	Interprétation de la solution . . . . .	14
<b>4</b>	<b>Difficultés rencontrées</b>	<b>16</b>
4.1	Problèmes résolus . . . . .	16
4.1.1	Problème sur la précision . . . . .	16
4.2	Problèmes non résolus . . . . .	16
4.2.1	Problème des minimums locaux . . . . .	16
4.2.2	Problème de vitesse de convergence . . . . .	17
<b>5</b>	<b>Conclusion</b>	<b>18</b>

# Table des figures

---

3.1	Diagramme de classes . . . . .	10
3.2	Le medèle utilisé ( $h=2m$ ; $l_1, l_2, l_3=1m$ ) . . . . .	13
3.3	Résultat obtenu . . . . .	15
4.1	Message affiché quand un minimum local apparaît . . . . .	16

# Introduction

---

Ce projet est développé dans le cadre du mini-projet robotique en quatrième année du département informatique. Le but du projet est de pratiquer les connaissances que nous avons acquises durant les séances du cours Robotique.

Pour cela, une dizaine de mini-projets sont proposées, et sont distribués aux autant de binômes. Nous avons obtenu ce sujet qui consiste des travaux autour un algorithme : "Algorithme à Approximation Incrémentale"(IAA). Étant donné un bras manipulateur et les coordonnées d'un point dans l'espace, cet algorithme sert à vérifier si ce point est atteignable par ce bras. Et si oui, il peut trouver au moins un ensemble des valeurs articulaires possibles du bras pour que son organe terminal puisse atteindre ce point.

Nous avons commencé notre travail par l'étude de l'article[1], qui a présenté cet algorithme ainsi que l'arrière-plan du problème concerné. Après avoir compris cet algorithme, nous l'avons implémenté en C++, et l'avons testé en utilisant une configuration d'un bras manipulateur donné en TD.

# Présentation de l'algorithme IAA

---

## 2.1 Problème réel concerné

Après avoir obtenu ce sujet, nous avons tout d'abord étudié l'article[1] proposé par notre encadrant. Nous nous apercevons que le sujet de notre projet peut être une modélisation de beaucoup de problèmes réels. Dans l'article nous avons lu, le problème concret est d'évaluer l'accessibilité dans un environnement bâti lorsque les capacités physiques de la personne ne correspondent plus aux nécessités de l'habitat ce qui survient en général après un accident de la vie.

Ce problème, selon les auteurs de cet article, peut alors être formulé comme un problème de résolution de l'inverse cinématique d'une structure articulée composée de la personne et de son dispositif de mobilité en considérant l'amplitude de ses capacités physiques résiduelles.

## 2.2 Catégories des solutions existantes

Trois types de méthodes sont disponibles actuellement pour établir l'existence d'une inverse cinématique de la chaîne articulaire par rapport au point à atteindre :

- Les méthodes analytiques.
- Les méthodes de linéarisation.
- Les méthodes d'optimisation.

La troisième catégorie est la plus intéressante et souvent utilisée lorsque le nombre de variables est important et si l'on désire obtenir des solutions respectant certains critères. Le principe consiste à formuler le problème comme un problème d'optimisation par minimisation d'une fonction de coût. Ici la fonction de coût[1] est de forme :

$$\epsilon = |f([\Theta]) - [X]|$$

Dans cette fonction :

- $\Theta$  représente l'ensemble des variables articulaires qu'on cherche.
- $f([\Theta])$  représente le point qu'on peut atteindre avec les variables  $\Theta$ .
- $[X]$  représente le point qu'on veut tester l'accessibilité.
- $\epsilon$  est donc la distance entre le point cible et le point qu'on peut atteindre pour le moment, et donc la valeur de  $\epsilon$  est à minimiser.

Si on peut obtenir, à la fin de l'algorithme, un ensemble de  $\Theta$  qui peut assurer que la valeur de  $\epsilon$  est inférieure à une valeur prédéfinie, alors on peut dire que le point cible est atteignable avec les variables  $\Theta$ . Et l'algorithme doit aussi traiter correctement le cas où le point n'est pas atteignable.

Dans le problème réel, il faut aussi prendre en compte que les variables articulaires qu'on obtient sont bien raisonnables. Par exemple, le joint coude humain n'a (normalement) pas de degré de liberté entre  $[0, 2\pi]$ .

Dans la section suivante, on va présenter l'algorithme IAA, qui nous permet de résoudre ce problème.

## 2.3 Algorithme IAA

L'algorithme IAA[1] est représenté ci-dessous.

Initialiser les variables  $\Theta_i$  de manière aléatoire;

**repeat**

    Définir l'incrément  $Inc(i)$  (Incrément par variable articulaire);

**foreach** *variable*  $\Theta_i$  **do**

$\Theta_i = \Theta_i + Inc(i)$ ;

        Calculer la distance entre la position courante et le but tel que  $\epsilon = |f([\Theta]) - [X]|$ ;

**if** ( $\Delta\epsilon < 0$ ) **then**

            garder  $\Theta_i$  (On est plus proche que le point cible)

**else**

$\Theta_i = \Theta_i - 2 * Inc(i)$  (chercher vers l'autre sens);

            Calculer  $\epsilon = |f([\Theta]) - [X]|$ ;

**if** ( $\Delta\epsilon < 0$ ) **then**

                garder  $\Theta_i$

**else**

$\Theta_i = \Theta_i + Inc(i)$  (garder la valeur d'origine)

**end**

**end**

**end**

**until** les condition d'arrêts sont vérifiées;

Selon l'article [2], nous avons pu trouver une définition possible de la fonction  $Inc(i)$  :

$$Inc(i) = (max(i) - min(i)) * IncrementRate$$

avec  $max(i)$  et  $min(i)$  sont les limites maximales et minimales de l'articulation  $i$ . La valeur de  $IncrementRate$  ajuste la vitesse de convergence de l'algorithme. La convergence est rapide au départ et ensuite devient plus faible à proximité de la solution. Une modification de  $IncrementRate$  est aussi proposée :

$$\text{If}(\Delta\epsilon = 0) \text{ Alors } IncrementRate = IncrementRate/2.$$

# Travail effectué

---

## 3.1 Complément de l'algorithme

La description de l'algorithme IAA qu'on a pu trouver n'est pas précise. Pour implémenter cet algorithme, il nous reste du travail à faire pour préciser tout l'algorithme. Le plus important est de fixer la condition d'arrêt de cet algorithme.

Pour cela, on considère deux cas possibles que notre algorithme rencontrera :

1. Le point cible est atteignable
2. Le point cible n'est pas atteignable

### 3.1.1 Premier cas

Pour le premier cas, l'algorithme doit s'arrêter quand la distance entre le point courant et le point cible est inférieure à une valeur prédéfinie (0.0001 par exemple), et il doit renvoyer l'ensemble des variables articulaires courantes.

La définition de la valeur de bornes est très souple selon le problème réel. Par exemple dans le problème présenté dans le chapitre précédent, la précision n'est pas trop importante, car on peut dire sans problème que "une personne peut atteindre un point dans sa chambre si la distance entre ce point et le bout de son doigt est seulement 1mm". Dans notre implémentation, cette valeur est 0.0001, qui est à la fois assez petite et pas trop coûteuse par rapport au temps d'exécution essentielle pour arriver à la fin de l'algorithme.

### 3.1.2 Deuxième cas

Pour le deuxième cas, puisque le point cible n'est réellement pas atteignable, on ne va jamais avoir une distance assez petite entre le point courant et le point cible. En analysant l'algorithme, nous pensons intuitivement que l'algorithme doit s'arrêter quand il arrive un état "consistant". Ça veut dire que l'algorithme ne peut déjà pas améliorer aucune variable articulaire  $\Theta_i$  dans deux itérations consécutives.

Nous avons testé notre algorithme avec cette condition d'arrêt, mais le résultat n'est pas préférable. Après une analyse plus détaillée, nous avons pu trouver le truc : même si le point cible est atteignable, on peut aussi avoir des états consistants pendant l'exécution de l'algorithme à cause de la valeur de *IncrementRate*. C'est assez raisonnable, car quand le changement est trop grand, c'est possible qu'aucun changement n'ait utile pour améliorer le résultat, et c'est pour ça qu'on diminue la valeur *IncrementRate* en ce moment-là en apportant des changements plus délicats. Et donc, on ajoute une autre condition en dehors de la consistance d'état, c'est que la valeur de *IncrementRate* doit être assez petite pour assurer qu'il n'y a vraiment pas de possibilité d'améliorer le résultat.

Nous avons utilisé dans notre program, pour *IncrementRate[i]*, une borne calculée comme ça :

$$BORNE[i] = (0.0001/180 * PI)/(QUAconfig[i].maxTheta - QUAconfig[i].minTheta);$$

Notre motivation de le faire, c'est pour avoir une précision de 0.0001 par rapport aux chaque changement apporté par *Inc(i)*. Autrement dit, quand *IncrementRate[i]* égale à *BORNE[i]*, ça veut dire que la ième articulation essaie d'améliorer la solution en apportant un changement de précision 0.0001.

En un mot, l'algorithme IAA (avec notre implémentation) doit s'arrêter quand l'une des deux conditions ci-après est vérifiée : soit la distance entre le point courant et le point cible est inférieur à une valeur



prédéfinie ; soit la valeur de *IncrementRate* est inférieur à une valeur prédéfinie et les variables articulaires  $\Theta$  ont un état consistant.

## 3.2 Implémentation de l'algorithme IAA

### 3.2.1 Choix de langage

Nous avons choisi C++ en tant que langage de programmation avec les considérations ci-dessous :

- Ce travail consiste à plutôt les calculs au lieu des logiques commerciales ou interface d'interaction. C'est très efficace d'utiliser C++ que les autres langages supérieurs.
- C++ possède des caractéristiques orientées objets, qui peut nous faciliter le travail.
- Nous venons de finir les cours de C++ avant de commencer ce projet, donc c'est une opportunité pour nous de pratiquer ce langage.

### 3.2.2 Le diagramme de classes

Le travail principal est d'implémenter l'algorithme IAA, mais pour faciliter notre traitement des données pendant les calculs, nous avons aussi défini quelques classes. Nous vous montrons notre diagramme de classes au-dessous :

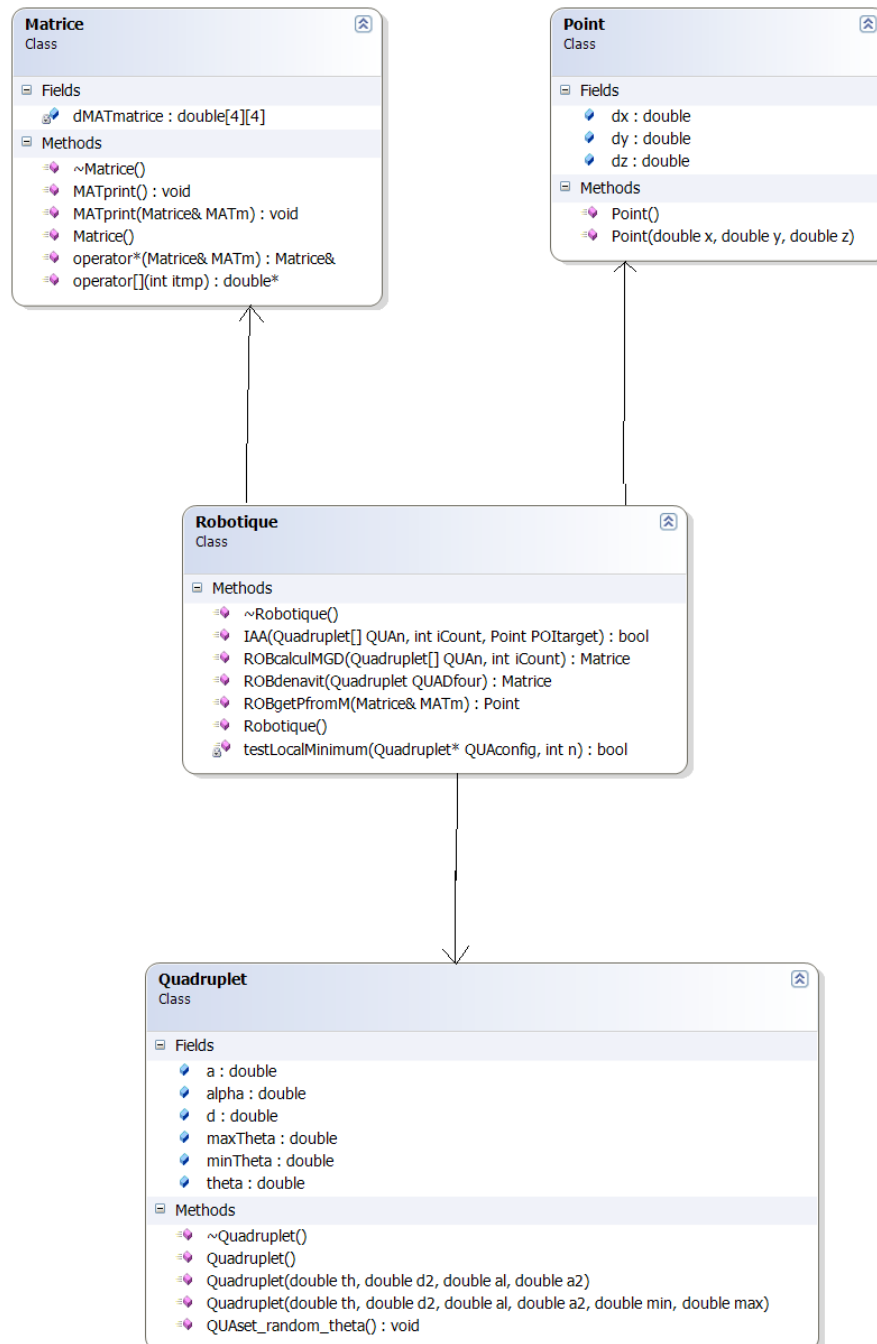


FIGURE 3.1 – Diagramme de classes

### 3.2.3 Le code source critique de l'algorithme

Dans cette section, nous mettons le code source de la fonction critique qui réalise l'algorithme IAA :

```

/**L'algorithme qui vérifie si un point est atteignable.
*@param QUAn Un tableau de quadruplet qui représente la configuration du robot.
* Les valeurs QUAn[i].theta sont à chercher, donc les valeurs initiales
* sont ignorés.
*@param iCount Le nombre de quadruplet dans le tableau QUAn
*@param POLtarget Le point cible.
*@result false Si le point cible n'est pas atteignable
* true Si le point cible est atteignable, les valeurs articulaires
* sont affecté à QUAn[i].theta.
*/
bool Robotique::IAA(Quadruplet QUAconfig[], int n, Point POLtarget)
{
    int i;
    bool flag = true;
    double lastX; //The theta of the last time

    //A table of increment rate
    double *incRate = new double[n];

    //stop condition
    double *BORNE_RATE = new double[n];

    //The coordinate of the current point that we can reach
    Point POLcurrent;

    //The distance between the current point and the target point
    double lastDist, newDist;

    //Initialisation
    for(i=0; i<n; i++)
    {
        QUAconfig[i].QUAset_random_theta();
        incRate[i]=0.15; //cf the article of Abdelhak MOUSSAOUI

        //Stop condition: If the value of IncrementRate is less
        //than this threshold, and the values of theta cannot be
        //improved, then the function return false, which means that
        //the target point is not reachable.
        BORNE_RATE[i]=0.0001*PI/180/ //precision:0.00...1°
        (QUAconfig[i].maxTheta - QUAconfig[i].minTheta);
    }

    POLcurrent = ROBgetPfromM(ROBcalculMGD(QUAconfig, n));
    lastDist = distance(POLcurrent, POLtarget);

    int j=0, times = 0;
    while(flag)

```

```

{
    flag = false;

    //for each value of theta
    for(i=0; i<n; i++)
    {
        POLtarget.dx, POLtarget.dy, POLtarget.dz);
        lastX = QUAconfig[i].theta;

        //inc
        QUAconfig[i].theta+=(QUAconfig[i].maxTheta-QUAconfig[i].minTheta)*incRate[i]
        if(QUAconfig[i].theta > QUAconfig[i].maxTheta)
            QUAconfig[i].theta = QUAconfig[i].maxTheta;

        POLcurrent = ROBgetPfromM(ROBcalculMGD(QUAconfig, n));
        newDist = distance(POLcurrent, POLtarget);

        if (newDist < lastDist)
        { //We get a better value
            //Keep the new value
            lastDist=newDist;
            flag=true;
        }
        else
        {
            //search another possibility
            QUAconfig[i].theta = lastX -
                (QUAconfig[i].maxTheta-QUAconfig[i].minTheta)*incRate[i]
            if(QUAconfig[i].theta < QUAconfig[i].minTheta)
                QUAconfig[i].theta=QUAconfig[i].minTheta;

            //calculate new distance
            POLcurrent = ROBgetPfromM(ROBcalculMGD(QUAconfig, n));
            newDist = distance(POLcurrent, POLtarget);
            if(newDist < lastDist)
            { //We get a better value
                //Keep the new value
                lastDist = newDist;
                flag = true;
            }
            else
            {
                //Keep the original value
                QUAconfig[i].theta=lastX;

                //Adjust increment rate
                incRate[i]/=2;
                if(incRate[i]>BORNE_RATE[i])
                    flag = true;
                newDist = lastDist;
            }
        }
    }
}

```

```

    }

    }
    if (newDist < BORNE_DIS)
    {
        delete [] incRate;
        return true;
    }
}

if (true == testLocalMinimum(QUAconfig, n))
{
    printf("We've probably encountered a \"Local minimum\" situation. Try again\n");
}
delete [] incRate;
return false;
}

```

### 3.3 Test de l'algorithme IAA

Après avoir fini le codage de l'algorithme, nous l'avons testé avec quelques exemples de robots qu'on a vus dans les cours de la robotique.

#### 3.3.1 Un exemple spécifique

Nous présentons un exemple spécifique que nous avons testé, concernant un manipulateur de 5 rotations, représenté dans la figure au-dessous :

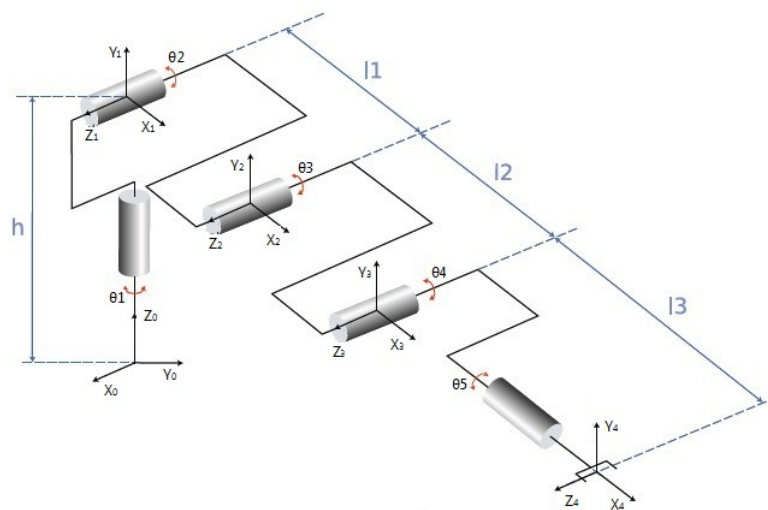


FIGURE 3.2 – Le medèle utilisé ( $h=2m$  ;  $l_1, l_2, l_3=1m$ )

Selon notre implémentation, pour représenter cette configuration du bras manipulateur, il suffit de créer un tableau d'objet de classe **Quadruplet**, dont chaque objet représente une transformation de repère. La dernière articulation est particulière, elle n'intervient pas dans notre travail, car la rotation de cette articulation ne change pas les coordonnées du point de l'organe terminal. Donc on a 4 objets au total dans le tableau :

```
//La représentation de la configuration du bras manipulateur (On a imposé des limites)
Quadruplet pQUAconfig1[4];
pQUAconfig1[0] = Quadruplet(PI/2, 2, PI/2, 0, 0, PI);
pQUAconfig1[1] = Quadruplet(0, 0, 0, 1, -PI/2, PI/2);
pQUAconfig1[2] = Quadruplet(0, 0, 0, 1, 0, PI/2);
pQUAconfig1[3] = Quadruplet(0, 0, 0, 1, 0, PI/2);

//La représentation du point cible à tester
Point POLtarget1(0,0,3);
```

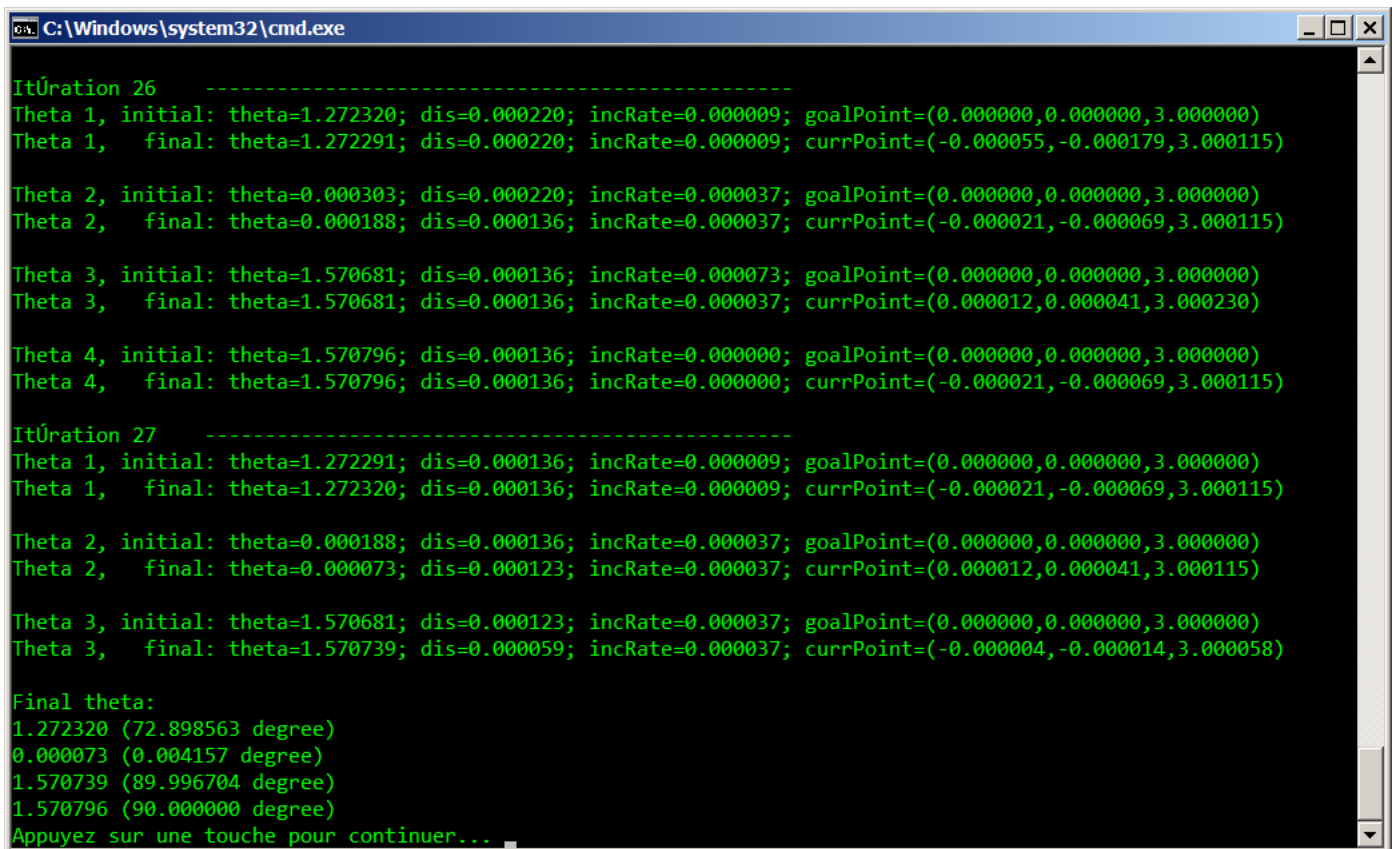
Et puis on peut appeler la fonction IAA :

```
//La valeur renvoyée indique si le point cible est bien atteignable ou non.
bool res = Robotique::IAA( pQUAconfig1, 4, POLtarget1);

//C'est une fonction définie pour afficher les valeurs des
//variables articulaires.
printResIAA(pQUAconfig1, 4, res);
```

### 3.3.2 Interprétation de la solution

En conséquence de cet exemple spécifique, nous avons obtenu dans la fenêtre de terminal :



```

C:\Windows\system32\cmd.exe

Itération 26 -----
Theta 1, initial: theta=1.272320; dis=0.000220; incRate=0.000009; goalPoint=(0.000000,0.000000,3.000000)
Theta 1, final: theta=1.272291; dis=0.000220; incRate=0.000009; currPoint=(-0.000055,-0.000179,3.000115)

Theta 2, initial: theta=0.000303; dis=0.000220; incRate=0.000037; goalPoint=(0.000000,0.000000,3.000000)
Theta 2, final: theta=0.000188; dis=0.000136; incRate=0.000037; currPoint=(-0.000021,-0.000069,3.000115)

Theta 3, initial: theta=1.570681; dis=0.000136; incRate=0.000073; goalPoint=(0.000000,0.000000,3.000000)
Theta 3, final: theta=1.570681; dis=0.000136; incRate=0.000037; currPoint=(0.000012,0.000041,3.000230)

Theta 4, initial: theta=1.570796; dis=0.000136; incRate=0.000000; goalPoint=(0.000000,0.000000,3.000000)
Theta 4, final: theta=1.570796; dis=0.000136; incRate=0.000000; currPoint=(-0.000021,-0.000069,3.000115)

Itération 27 -----
Theta 1, initial: theta=1.272291; dis=0.000136; incRate=0.000009; goalPoint=(0.000000,0.000000,3.000000)
Theta 1, final: theta=1.272320; dis=0.000136; incRate=0.000009; currPoint=(-0.000021,-0.000069,3.000115)

Theta 2, initial: theta=0.000188; dis=0.000136; incRate=0.000037; goalPoint=(0.000000,0.000000,3.000000)
Theta 2, final: theta=0.000073; dis=0.000123; incRate=0.000037; currPoint=(0.000012,0.000041,3.000115)

Theta 3, initial: theta=1.570681; dis=0.000123; incRate=0.000037; goalPoint=(0.000000,0.000000,3.000000)
Theta 3, final: theta=1.570739; dis=0.000059; incRate=0.000037; currPoint=(-0.000004,-0.000014,3.000058)

Final theta:
1.272320 (72.898563 degree)
0.000073 (0.004157 degree)
1.570739 (89.996704 degree)
1.570796 (90.000000 degree)
Appuyez sur une touche pour continuer...
  
```

FIGURE 3.3 – Résultat obtenu

L'algorithme a passé 27 itérations pour trouver la solution. Les valeurs des 4 variables articulaires finales sont affichées à la fin. Cette solution peut être interprétée comme ceci : pour atteindre le point (0,0,3), les 4 angles de rotation  $\Theta_1, \Theta_2, \Theta_3, \Theta_4$  sont respectivement 73°, 0°, 90°, 90°. Il peut aussi y avoir des autres solutions possibles.

# Difficultés rencontrées

---

## 4.1 Problèmes résolus

### 4.1.1 Problème sur la précision

Au début de l'implémentation de l'algorithme, nous obtenions souvent des résultats déraisonnables. Après avoir discuté avec notre encadrant, nous nous sommes aperçus que ça soit un problème de précision. Ça concerne la représentation interne des données en C++. Et puis nous avons changé notre type de donnée à **double** qui a enfin résolu ce problème.

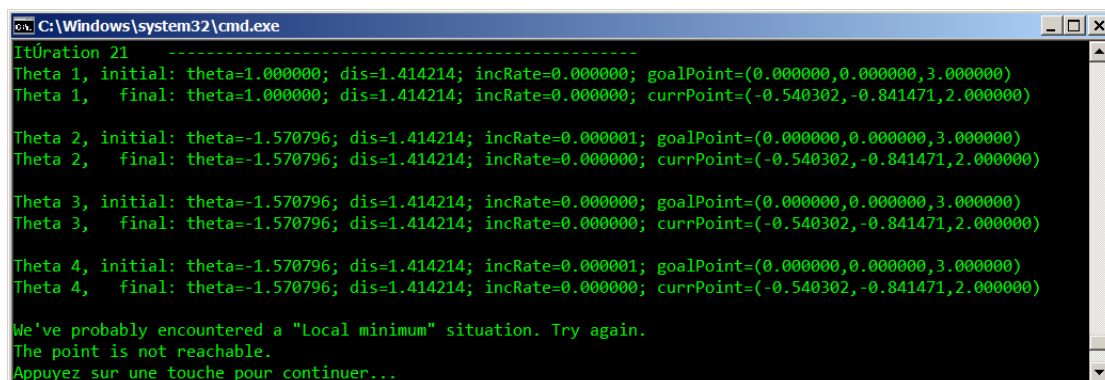
## 4.2 Problèmes non résolus

### 4.2.1 Problème des minimums locaux

Un avantage de l'algorithme IAA est qu'on peut imposer des limites sur les variables articulaires, pour que la solution respecte certains critères particuliers. Mais ça nous amène aussi le problème des minimums locaux. Parfois l'algorithme se converge vers un minimum local (mais pas global), et donc dans un état consistant. En conséquence, il nous dit qu'il y a pas de solution (le point cible n'est pas atteignable), mais le point est en fait atteignable, et la solution n'était pas trouvée à cause des minimums locaux.

C'est un problème commun des algorithmes similaires. C'est à cause des limites qu'on a imposées aux variables articulaires et à la initialisation aléatoire d'elles. Nous ne pouvons pas éliminer ce problème, mais nous pouvons essayer de l'examiner, pour qu'on puisse relancer l'algorithme au lieu de croire qu'il n'y a pas de solution.

Nous constatons que quand un "minimum local" apparaît, il y a au moins une variable articulaire qui a atteint sa limite qu'on l'a imposée. Selon ça, dans notre programme, si aucune solution n'est trouvée, on va tester les variables articulaires courantes, s'il existe une variable articulaire qui a atteint sa limite, alors on affiche un message disant qu'on a **probablement** rencontré un minimum local. À savoir qu'on ne l'a pas prouvé mais plutôt le tester selon les expériences.



```
C:\Windows\system32\cmd.exe
Itération 21 -----
Theta 1, initial: theta=1.000000; dis=1.414214; incRate=0.000000; goalPoint=(0.000000,0.000000,3.000000)
Theta 1,  final: theta=1.000000; dis=1.414214; incRate=0.000000; currPoint=(-0.540302,-0.841471,2.000000)

Theta 2, initial: theta=-1.570796; dis=1.414214; incRate=0.000001; goalPoint=(0.000000,0.000000,3.000000)
Theta 2,  final: theta=-1.570796; dis=1.414214; incRate=0.000000; currPoint=(-0.540302,-0.841471,2.000000)

Theta 3, initial: theta=-1.570796; dis=1.414214; incRate=0.000000; goalPoint=(0.000000,0.000000,3.000000)
Theta 3,  final: theta=-1.570796; dis=1.414214; incRate=0.000000; currPoint=(-0.540302,-0.841471,2.000000)

Theta 4, initial: theta=-1.570796; dis=1.414214; incRate=0.000001; goalPoint=(0.000000,0.000000,3.000000)
Theta 4,  final: theta=-1.570796; dis=1.414214; incRate=0.000000; currPoint=(-0.540302,-0.841471,2.000000)

We've probably encountered a "Local minimum" situation. Try again.
The point is not reachable.
Appuyez sur une touche pour continuer...
```

FIGURE 4.1 – Message affiché quand un minimum local apparaît



#### 4.2.2 Problème de vitesse de convergence

La vitesse de convergence est aussi un problème de cet algorithme, surtout quand le point cible est sur la frontière de la zone atteignable du bras manipulateur. En ce cas particulier, l'algorithme peut enfin trouver la solution, mais la vitesse de convergence devient très lente quand l'organe terminal approche du point cible.

# Conclusion

---

Notre projet concerne la compréhension, implémentation et le test d'un l'algorithme(IAA) qui peut vérifier l'accessibilité d'un point dans l'espace par un bras manipulateur. Nous commençons notre travail par la lecture de deux articles proposés par notre encadrant. Après avoir compris l'algorithme, nous l'avons implémenté en C++, et l'avons testé avec quelques exemples de bras manipulateur qu'on a vu en cours.

Ce projet nous a renforcé la compréhension sur la cinématique robotique, surtout le *Modèle Géométrique Direct* et le *Modèle Géométrique Inverse*. Ça nous permet aussi d'avoir une impression sur l'application de robotique dans la vie réelle.

# Bibliographie

---

- [1] Otmani R., Pruski A., Belarbi K., *La réalité virtuelle comme outil pour l'évaluation, la visualisation et la validation de l'accessibilité d'un lieu de vie*. Conférence Handicap 2010, juin 2010, Paris.
- [2] Abdelhak MOUSSAOUI, *Prise en charge de psychothérapie et du handicap par la réalité virtuelle*

# Test de l'accessibilité d'un point par un bras manipulateur

---

Département Informatique  
4<sup>e</sup> année  
2012 - 2013

Rapport de mini-projet Robotique

**Résumé :** Ce mini projet consiste à étudier, puis implémenter et tester l'Algorithme à Approximation Incrémentale (IAA), qui est une méthode permettant de vérifier si un point situé dans l'espace est atteignable par un bras manipulateur. Nous avons réalisé cet algorithme en C++.

**Mots clefs :** bras manipulateur, accessibilité d'un point, IAA

**Abstract:** This project is about our study of the Incremental Approximation Algorithm (IAA), which is a method telling whether a point in the space is reachable by a robot arm. We have studied, implemented and tested this method using C++.

**Keywords:** robot arm, accessibility of a point, IAA

---

## Encadrants

Pierre Gaucher  
[pierre.gaucher@univ-tours.fr](mailto:pierre.gaucher@univ-tours.fr)

Université François-Rabelais, Tours

## Étudiants

Zheng LIU  
[zheng.liu@etu.univ-tours.fr](mailto:zheng.liu@etu.univ-tours.fr)  
Lei SHANG  
[lei.shang@etu.univ-tours.fr](mailto:lei.shang@etu.univ-tours.fr)

DI4 2012 - 2013