TP `Unity`:
Cinématique inverse avec l'algorithme CCD

Steve Tonneau
IRISA

12 novembre 2013

Introduction

Ce TP propose de traiter un problème concret, récurent dans le domaine de l'animation, celui de la cinématique inverse. La partie 1 de ce document est consacrée à la description du problème. La partie 2 est consacrée à la présentation de l'algorithme CCD (pour Cyclic Coordinate Descent), très utilisé, en particulier dans le domaine du jeu vidéo, pour résoudre le problème de cinématique inverse. La partie 3 a pour but de guider les étudiants dans l'implémentation de l'algorithme CCD. On l'implémentera d'abord en 2, puis en 3 dimensions.

Public concerné

Ce tp s'adresse à des étudiants en première année d'informatique. Il requiert une connaissance minimale du moteur `Unity` 3d [1] (Menus, système de scripts). Les signatures des méthodes utilisées sont présentées en C#, mais il est facile de les transposer pour les langages Boo et javascript également acceptés sous `Unity`. Aussi il est également nécessaire d'avoir des connaissances dans un de ces langages cibles.

Pour comprendre et implémenter l'algorithme CCD sous `Unity`, des connaissances mathématiques de niveau lycée sont nécessaires.

Durée

TODO

Matériel

Un squelette de projet est disponible à l'adresse TODO. Il nécessite une version d'`Unity` (gratuite ou pro) supérieure ou égale à la version 4.1.

1. http://unity3d.com/unity.

Le projet comprend :
- Une scène `testScene.unity`. Elle comprend elle-même :
  - Une hiérarchie de sphères qui servira de chaîne cinématique pour l'exercice ; la première sphère s'appelle `root`, la dernière `effector` ;
  - Un cube `target`, qui représentera la cible à atteindre par notre chaîne cinématique.
- Deux squelettes `CCD2d.cs` et `CCD3d.cs`, qu'il va falloir modifier ;
- Un script `TargetController.cs`, déjà implémenté, qui permet de contrôler la cible au clavier.

1 Problématique : La cinématique inverse

1.1 Exemple introductif : réglage de l'heure

Considérons la montre à gauche de la figure 1 :

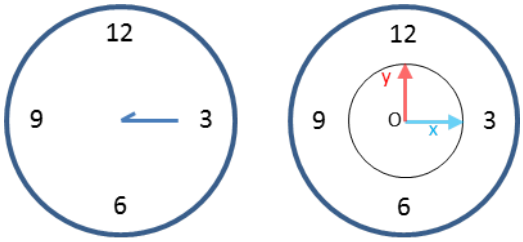

FIGURE 1 – *A gauche : Une montre indiquant 2h15. A droite : la même montre munie d'un repère cartésien* $(\vec{Ox}, \vec{Oy})$.

Cinématique directe

Nous munissons notre montre d'un repère cartésien $(\vec{Ox}, \vec{Oy})$ (figure 1, droite), dans lequel la longueur de la grande aiguille vaut 1. Nous appelons l'extrémité extérieure de l'aiguille **effecteur**. On observe en fait que l'effecteur se déplace le long du cercle trigonomètrique associé au repère $(\vec{Ox}, \vec{Oy})$.
Ce déplacement est réalisé par une rotation d'un angle θ autour d'un troisiéme axe $\vec{Oz} = \vec{Ox} \wedge \vec{Oy}$, qui prend son origine en O.

On définit donc la fonction f, telle que

$$f(\theta) = \begin{pmatrix} cos(\theta) \\ sin(\theta) \end{pmatrix}$$

avec $\theta \in [-2\pi, 2\pi[$.

Grâce à f on peut connaître la position de l'effecteur de la grande aiguille en fonction de l'angle θ. Par exemple, si $\theta = -\pi/2$, l'effecteur est à la position $\begin{pmatrix} 0 \\ -1 \end{pmatrix}$, comme montré sur la figure 2

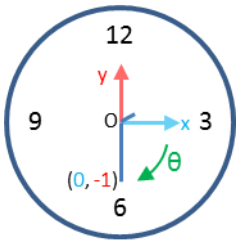

FIGURE 2 – *Si on choisit $\theta = -\pi/2$, la montre indique maintenant 2h30.*

Ceci constitue un exemple de **cinématique directe** : à partir d'un angle θ, nous calculons une position $\begin{pmatrix} x \\ y \end{pmatrix}$.

Cinématique inverse

On considère le problème inverse : On souhaite régler l'heure à 2h30, c'est à dire que l'extrémité de la grande aiguille aie pour coordonnées $\begin{pmatrix} 0 \\ -1 \end{pmatrix}$. Quelle valeur doit prendre l'angle θ pour atteindre cet objectif ?

On définit la fonction f^{-1}, telle que

$$f^{-1}(x, y) = acos(x) * sgn(y)$$

avec $sgn(y) = -1$ si $y < 0$, $sgn(y) = 1$ sinon.

f^{-1} permet de calculer l'angle θ nécessaire à l'atteinte de la position $\begin{pmatrix} x \\ y \end{pmatrix}$. Ceci est un exemple de **cinématique inverse**.

1.2 Cinématique en deux dimensions

Définition d'une chaîne cinématique

Pour ce tp, nous définissons une chaîne cinématique comme un ensemble n de barres rigides $c_i, i = 0, 1, ..., n-1$ maintenues deux à deux ensemble à leurs extrêmités par des articulations q_i. L'articulation $q_j, j = 1, ..., n-1$ connecte les barres $c_j - 1$ et c_j. q_0 connecte la première barre et l'environnement. Cette articulation est appelée racine. La dernière barre n'est connectèe qu'à une de ses extrémités ; on appelle l'extrémité libre **effecteur** e.

Par exemple la grande aiguille de la montre présentée (figure 3) est une chaîne composée d'une seule barre rigide, qui comprend donc la racine et l'effecteur.

Un angle θ_i qui décrit la rotation associée à une articulation q_i, comme le montre la figure 4.

FIGURE 3 – *La grande aiguille de la montre est une chaîne cinématique composée d'une seule barre rigide.*

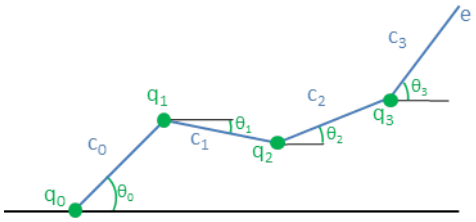

FIGURE 4 – *Une chaine cinématique composée de quatre corps rigides.*

Cinématique directe

Dans un problème de cinématique directe, on cherche à déterminer la position de l'effecteur e en fonction des valeurs d'angles θ_i de la chaîne cinématique.

On veut donc trouver la fonction f telle que :

$$f\begin{pmatrix}\theta_0\\ ...\\ \theta_{n-1}\end{pmatrix} = \begin{pmatrix}x_e\\ y_e\end{pmatrix}$$

Cinématique inverse

Dans un problème de cinématique inverse, on cherche à déterminer une combinaison de valeurs d'angles θ_i qui permette d'atteindre la position $\begin{pmatrix}x_e\\ y_e\end{pmatrix}$.

On veut donc trouver la fonction f^{-1} telle que :

$$f^{-1}\begin{pmatrix}x_e\\ y_e\end{pmatrix} = \begin{pmatrix}\theta_0\\ ...\\ \theta_{n-1}\end{pmatrix}$$

Le problème est que, dans les cas complexes, f n'est pas bijective :
– elle n'est pas surjective car f^{-1} n'est pas forcément définie partout (figure 5) ;

– elle n'est pas injective car il peut exister plusieurs solutions qui permettent d'atteindre la position $\begin{pmatrix} x_e \\ y_e \end{pmatrix}$ (figure 6).

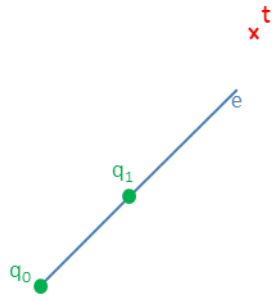

FIGURE 5 – *Le point t ne peut être atteint par l'effecteur e.*

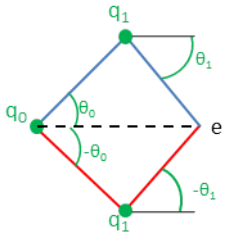

FIGURE 6 – *Différentes combinaisons d'angles peuvent aboutir à la même position pour l'effecteur e.*

Quand une chaîne cinématique est trop complexe, il est difficile, voire impossible de définir correctement f^{-1} ; c'est pour cela que des méthodes numériques ont été mises au point pour calculer les valeurs de f^{-1}, comme la méthode CCD.

1.3 Cinématique en 3 dimensions

Nous avons vu qu'en deux dimensions nous considérons des rotations autour de l'axe $\vec{Oz} = \vec{Ox} \wedge \vec{Oy}$. En 3 dimensions, il nous faudra également considérer des rotations autour des axes $\vec{Ox}$ et $\vec{Oy}$ (figure 7). En théorie cela fait donc 3 fois plus d'angles à prendre en compte !

Une autre manière de voir les choses est ramener la combinaison de 3 rotations à une seule rotation d'angle θ autour d'un axe $\vec{u}$ à définir. Comme pour le cas 2d, étant donnés 2 vecteurs $\vec{a}$ et $\vec{b}$ $\vec{u}$ se calcule grâce au produit vectoriel :

$$\vec{u} = \vec{a} \wedge \vec{b}$$

Ceci est illustré par la figure 8.

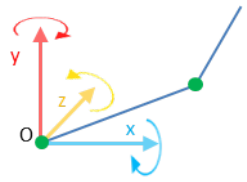

FIGURE 7 – *Articulation en 3 dimensions : 3 rotations sont possibles.*

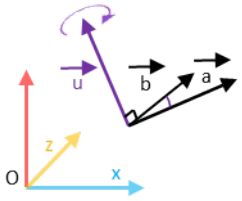

FIGURE 8 – *La transformation permettant d'aligner $\vec{a}$ à $\vec{b}$ peut s'exprimer comme une seule rotation autour de l'axe $\vec{u}$.*

2 L'algorithme CCD

Définitions

- p_i est la position d'une articulation q_i ;
- p_e est la position de l'effecteur ;
- p_t est la position de la cible à atteindre.

Déroulement de l'algorithme

La figure 9 illustre le déroulement de l'algorithme. On commence par choisir l'articulation $q_i = q_{n-1}$ la plus proche de l'effecteur.

- On calcule l'angle α formé entre les vecteurs $q_i\vec{p}_e$ et $q_i\vec{p}_t$;
- On effectue la rotation de centre p_i et d'angle α qui aligne $q_i\vec{p}_e$ et $q_i\vec{p}_t$, autour de l'axe $q_i\vec{p}_e \wedge q_i\vec{p}_t$;
- Si la distance entre l'effecteur et la cible $||p_e\vec{p}_t||$ est infèrieure à un nombre $\epsilon \in R$, on arrête l'algorithme, car l'objectif est atteint ;
- Sinon deux cas sont possibles :
  - si $i > 1$, on répète ces opérations avec l'articulation q_{i-1} ;
  - sinon nous sommes arrivés à la racine ; on recommence alors avec q_{n-1}.

3 Questions

1. Charger la scène testScene.unity. Depuis l'explorateur de projet, ouvrir le fichier `CCD2d.cs`. Au besoin, regarder la documentation pour comprendre le type

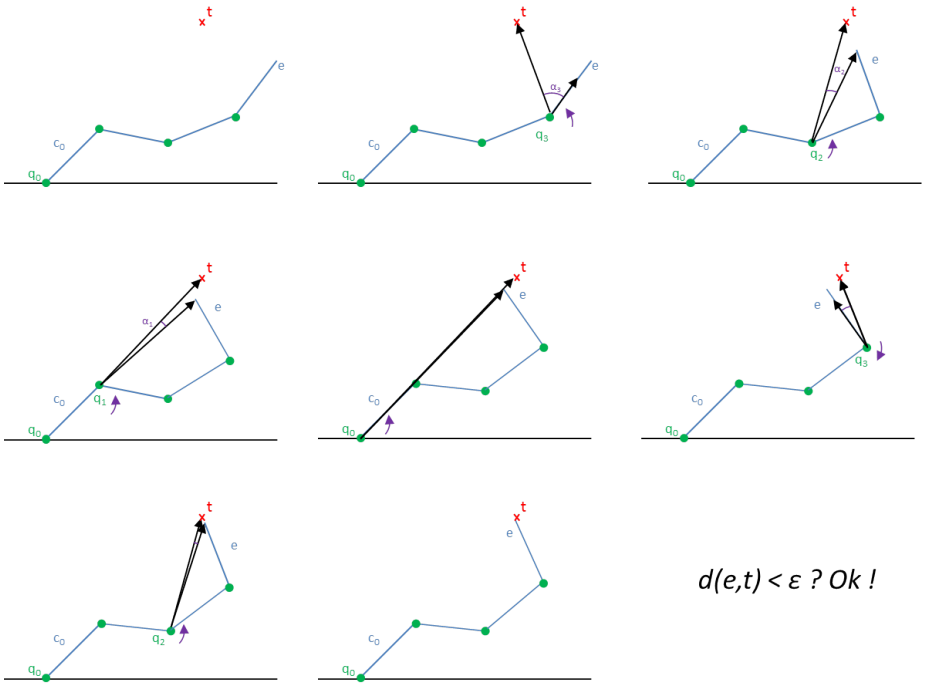

FIGURE 9 – *Un exemple d'application du CCD en 2 dimenstions. 6 rotations sont nécessaires pour atteindre la cible t.*

des paramètres du script, ainsi que ceux passés à la méthode `CCDStep2D`.

2. Ecrire le code de la méthode `ComputeAngle2D` qui calcule un angle signé entre deux `Vector2`. Au besoin se servir de la documentation de `Vector2` et du namespace `Mathf`.

3. La méthode `Update` va être appelée une fois par frame. Elle appelle à son tour la méthode `CCDStep2D`. C'est une méthode récursive dans le sens où elle se rappelle elle-même avec le parent de l'articulation courante (`joint` en anglais). En déduire ce que représente `gameObject`, et attacher le script à cet objet dans la scène depuis la fenêtre `inspector` de `Unity`.
Une fois ceci fait, assigner l'objet `target` en tant que paramètre du script.

4. Ecrire le code de la méthode `CCDStep2D`. S'aider de la documentation de la classe `Transform` afin de trouver la méthode qui permet d'effectuer une rotation autour d'un axe.
Dans la description de l'algorithme il est dit que l'on arrête l'algorithme si la distance entre l'effecteur et la cible $p_e\vec{p}_t$ est infèrieure à un nombre $\epsilon \in R$. Il n'y a pas besoin de tester cette condition ici. Pourquoi supprimer cette condition

d'arrêt n'est pas dangereux dans notre cas ?

5. Lancer l'application. Déplacer le cube cible avec les flèches directionelles. Vérifier que votre implémentation fonctionne.

6. Depuis l'`inspector` désactiver le script `CCD2d.cs`.
Ajouter le script `CCD3d.cs` au bon objet de la scène.
Ecrire le code manquant pour faire fonctionner le script `CCD3d.cs` en s'aidant de la documentation de la classe `Vector3`. Constater que sous `Unity` il est encore plus facile de travailler en 3d qu'en 2d :).

7. Lancer l'application. Les touches `V` et `B` permettent de déplacer le cube le long de l'axe $\vec{Oz}$, vous permettant de tester votre algorithme en 3 dimensions.