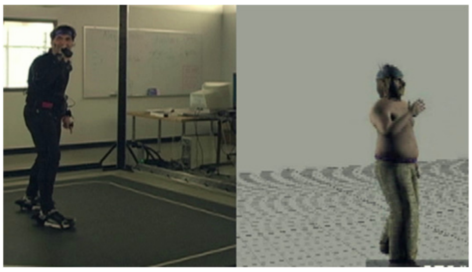
CINÉMATIQUE INVERSE

Alexandre Meyer
Equipe SAARA, Laboratoire LIRIS
M2 pro Image

Retargeting : rappel

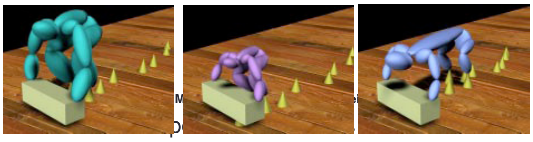
Besoin d'adapter une animation

Problème de retargeting : ici la main



Problem of Hand or foot position!

- Often hand or foot positions do not match



feet in concordance with skeleton morphology

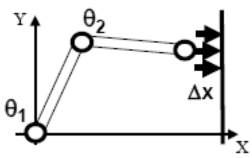
➔Quick overview of inverse kinematic

CINÉMATIQUE INVERSE

INTRODUCTION DU PROBLÈME
MÉTHODES ANALYTIQUES
MÉTHODES ITÉRATIVES
LES CONTRAINTES

Cinématique inverse

- Cinématique inverse
  - Etant donné les positions des extrémités, trouver la pose (=angles)
- Problème non-linéaire (position vs. angles)
  - Possiblement aucun ou plusieurs solutions



Cinématique directe

- Soit le vecteur représentant les M degrés de liberté (DDL) des articulations

$$\Phi = [\varphi_1 \quad \varphi_2 \quad \dots \quad \varphi_M]$$

Par ex., avec 3 rotations pour A articulations : M=3xA

- Et le vecteur représentant les extrémités

$$\mathbf{e} = [e_1 \quad e_2 \quad \dots \quad e_N]$$

Par exemple, si les E extrémités sont des articulations avec positions et orientations, **e** va contenir 6 DDL : 3 translations et 3 rotations et donc N=6xE

Si juste la position, 3 translations et N=3xE

Cinématique directe

- La fonction de cinématique directe f() calcule la position dans le monde des extrémités à partir des DDL des articulations
- La cinématique directe est souvent facile à calculer

$$\mathbf{e} = f(\mathbf{\Phi})$$

Cinématique inverse

- Le but de la cinématique inverse est de calculer le vecteur des DDL des articulations qui positionne chaque extrémité à son but.

Remarque : une cible peut aussi être donnée à une articulation "milieu" du squelette donc le terme extrémité est à prendre au sens large.

- En d'autre terme, il s'agit de l'inverse de la cinématique directe
- $f^{-1}()$ n'est généralement pas simple à calculer

$$\mathbf{\Phi} = f^{-1}(\mathbf{e})$$

Cinématique inverse

De nombreuses approches

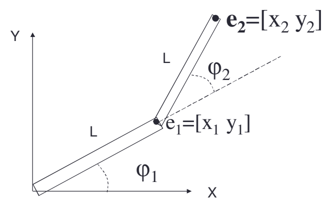
- Méthodes analytiques
  - Géométrique ➔ rapide
  - Ok pour peu de DDL, peu d'articulations
- Méthodes numériques
  - Basées sur des processus itératifs
  - Peu nécessiter de nombreuses itérations
  - Flexible (possibilité d'inclure des contraintes)
  - Se ramène à un problème de minimisation

CINÉMATIQUE INVERSE

MÉTHODES ANALYTIQUES

Configurations de base

- Supposons un squelette 2D avec 2 articulations de longueur L, comportant chacune 1 DDL (rotation)
- Remarque : modèle plus simple que celui présenté dans le cours sur la cinématique directe



Configuration 1 DDL

- Coordonnées de e_1
  - $x_1 = L.\cos(\varphi_1)$
  - $y_1 = L.\sin(\varphi_1)$

- Cinématique inverse=on veut (x_1=X, y_1=Y), trouver φ_1

➔Y/X = $\tan(\varphi_1)$ ➔$\varphi_1=\tan^{-1}(Y/X)$

Attention ne marche que si X,Y est sur le cercle

Configuration 2 DDL dans le plan 2D

- Coordonnées de e_1
  - $x_1 = L.\cos(\varphi_1)$
  - $y_1 = L.\sin(\varphi_1)$
- Coordonnées de e_2
  - $x_2 = x_1+L.\cos(\varphi_1+\varphi_2)$
  - $y_2 = y_1+L.\sin(\varphi_1+\varphi_2)$

2 équations à 2 inconnues

➔à priori pas de solution, 1 ou 2 solutions (solution d'une équation du second degré)

Configuration 3 DDL dans le plan 2D

- Coordonnées de e=e_3
  - $x_3 = L.\cos(\varphi_1)+L.\cos(\varphi_1+\varphi_2)+L.\cos(\varphi_1+\varphi_2+\varphi_3)$
  - $y_3 = L.\sin(\varphi_1)+L.\sin(\varphi_1+\varphi_2)+L.\sin(\varphi_1+\varphi_2+\varphi_3)$

on veut x_3=X et y_3=Y

- 2 équations à 3 inconnus
- Problème sous contraint

→il faut ajouter des contraintes : fixer l'angle de certaines articulations, des positions, etc.

Méthodes analytiques

- Nous n'irons pas plus loin dans les méthodes analytiques qui
  - En pratique sont difficiles à mettre en œuvre
  - L'introduction d'une nouvelle contraintes demande de recalculer toutes les équations
  - N'utilise pas la position précédente comme support et peut donc donner des discontinuités dans le cas de mouvement

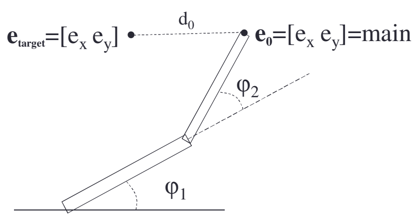
Si besoin voir [ABDEL-RAHMAN 1991]

CINÉMATIQUE INVERSE
MÉTHODES ITÉRATIVES

- Introduction des méthodes itératives
- Descente de gradient
- Utilisant le Jacobien
- A base d'heuristiques

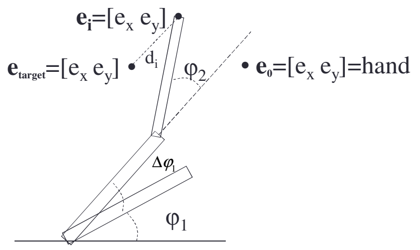
Méthodes itératives : introduction

- Supposons un squelette 2D avec 2 articulations comportant chacune 1 DDL (rotation)
- Une position cible pour l'extrémité main e_{target}
- d = distance entre l'extrémité et la cible



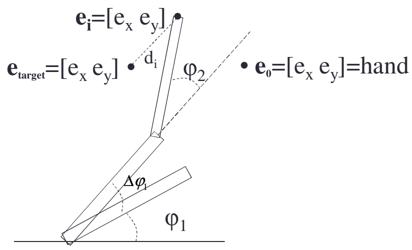
Itérations

1) Bouge un peu φ_1 → φ_1 + delta
→choisir le delta avec d décroissant
2) Idem pour φ_2
3) Itérer jusqu'à ce que d=0



Méthodes itératives : introduction

- Tout le problème : Comment choisir un bon delta?

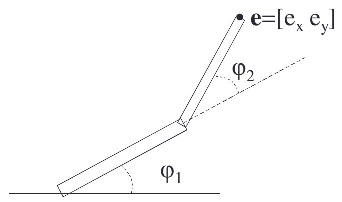


CINÉMATIQUE INVERSE
MÉTHODES ITÉRATIVES

- …
- Descente de gradient
- …

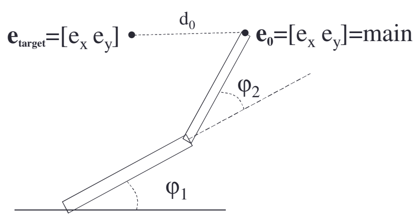
Notre configuration de base

- Supposons un squelette 2D avec 2 articulations
  - de longueur L
  - comportant chacune 1 DDL (rotation)



Descente de gradient

- La fonction f à minimiser est la distance entre e et e_{target}
- $f(\varphi_0, \dots, \varphi_i, \dots, \varphi_n) = |e - e_{target}|$
- f(N dimensions) -> 1D (un scalaire, la distance)



Descente de gradient

- Angles de départ
- Position initiale

$$\begin{pmatrix}\varphi_1^0 & \varphi_2^0 & \dots & \varphi_n^0\end{pmatrix}$$

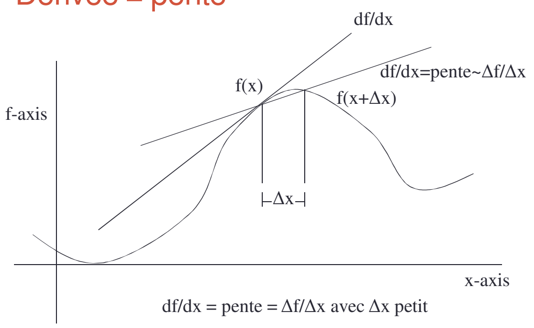
$$\mathbf{e}^0 = f\left(\varphi_1^0, \dots, \varphi_n^0\right)$$

- La dérivée partielle de f est définie comme ceci

$$\frac{df}{d\varphi_1} = \lim_{\Delta\varphi_1 \to 0} \frac{\Delta f}{\Delta\varphi_1} = \lim_{\Delta\varphi_1 \to 0} \frac{f(\varphi_1 + \Delta\varphi_1, \dots, \varphi_n) - f(\varphi_1, \dots, \varphi_n)}{\Delta\varphi_1}$$

$$\approx \frac{f(\varphi_1 + \Delta\varphi_1, \dots, \varphi_n) - f(\varphi_1, \dots, \varphi_n)}{\Delta\varphi_1}$$

Dérivée = pente



df/dx = pente = Δf/Δx avec Δx petit

Descente de gradient

- Comment choisir un bon delta?

utiliser la direction opposée au gradient de f → algorithme de descente du gradient

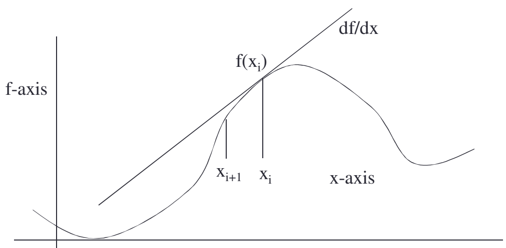
$$\nabla f = \begin{pmatrix} \frac{df}{d\varphi_1} & \frac{df}{d\varphi_2} & \cdots & \frac{df}{d\varphi_n} \end{pmatrix}$$

Avec la notion de dérivée partielle

$$\frac{df}{d\varphi_1} = \lim_{\Delta\varphi_1 \to 0} \frac{\Delta f}{\Delta \varphi_1} = \lim_{\Delta\varphi_1 \to 0} \frac{f(\varphi_1 + \Delta\varphi_1, \dots, \varphi_n) - f(\varphi_1, \dots, \varphi_n)}{\Delta\varphi_1}$$

$$\approx \frac{f(\varphi_1 + \Delta\varphi_1, \dots, \varphi_n) - f(\varphi_1, \dots, \varphi_n)}{\Delta\varphi_1}$$

Descente de gradient



$$x_{i+1} = x_i - df/dx$$

Descente de gradient

Repeat
// Calcul du gradient
ForEach joint i Do
grad$_i$ = (df / dφ_i)
= (f(φ_0, …, φ_i+delta , …, φ_n)- f(φ_0, …, φ_i, …, φ_n))/delta
EndForEach

// Mise à jour des angles avec la direction opposée au gradient
ForEach joint i do $\varphi_i \rightarrow \varphi_i - \beta$*grad$_i$

distance = f(φ_0, …, φ_i, …, φ_n) // = |e- $\mathbf{e}_{target}$|
Adapt(delta);
Adapt(β);
Until distance < 0.01 // signifiant tant que e est proche de $\mathbf{e}_{target}$

Descente de gradient

- Finalement assez facile à coder
- Mais
  - la convergence dépend de
    Adapt(delta);
    Adapt(β);
    qui sont au finale assez pénible à mettre au point
  - Demande de nombreuses itérations pour converger surtout dans le cas de plusieurs extrémités
  - Peut produire des configurations "non naturelles"

MÉTHODES ITÉRATIVES

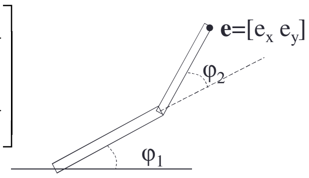
- …
- Utilisant le Jacobien
  - Généralité sur la méthode
  - L'algorithme
  - Transposée, pseudo-inverse, Moindre carrés amortis
- …

Matrice Jacobienne

- La matrice Jacobienne **J(e, φ)** indique comment chaque élément de **e** varient en fonction de chaque variation d'angles

Remarque : Matrice Jacobienne est souvent abrégé en Jacobien

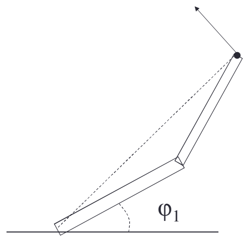
$$J(\mathbf{e},\varphi)=\begin{bmatrix}\dfrac{\partial e_x}{\partial \varphi_1} & \dfrac{\partial e_x}{\partial \varphi_2}\\ \dfrac{\partial e_y}{\partial \varphi_1} & \dfrac{\partial e_y}{\partial \varphi_2}\end{bmatrix}$$



Jacobien

- Considérons ce qui arrive si on incrémente φ_1 d'une petite quantité . Que se passe-t-il pour **e** ?

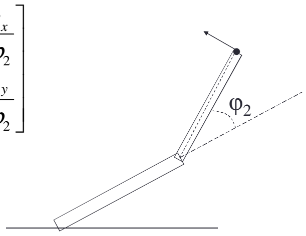
$$\frac{\partial \mathbf{e}}{\partial \varphi_1}=\begin{bmatrix}\dfrac{\partial e_x}{\partial \varphi_1}\\ \dfrac{\partial e_y}{\partial \varphi_1}\end{bmatrix}$$



Jacobien

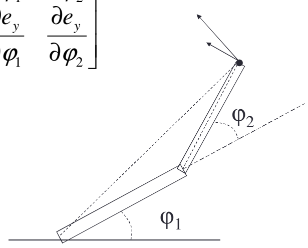
- Et si on incrémente φ_2 un petit peu ?

$$\frac{\partial \mathbf{e}}{\partial \varphi_2}=\begin{bmatrix}\dfrac{\partial e_x}{\partial \varphi_2}\\ \dfrac{\partial e_y}{\partial \varphi_2}\end{bmatrix}$$



Matrice Jacobienne

$$J(\mathbf{e},\varphi)=\begin{bmatrix}\dfrac{\partial e_x}{\partial \varphi_1} & \dfrac{\partial e_x}{\partial \varphi_2}\\ \dfrac{\partial e_y}{\partial \varphi_1} & \dfrac{\partial e_y}{\partial \varphi_2}\end{bmatrix}$$



Matrice Jacobienne

- Le gradient est la "pente" d'une fonction F allant de dimensions N vers un scalaire (dimension 1)
  F : $R^N \rightarrow R$

- Voir la matrice Jacobienne comme une généralisation de la pente d'une fonction allant de dimensions N vers dimensions M
  F : $R^N \rightarrow R^M$
  F : N angles → M coordonnées des extrémités

- Dans notre cas de base (robot plan 2D à 2 bras), F va de 2D (les 2 angles) vers 2D (les 2 coordonnées X et Y de la main en 2D). Matrice Jacobienne est donc 2x2

Matrice Jacobienne

$$J(\mathbf{e},\varphi)=\frac{d\mathbf{e}}{d\varphi}$$

- J(**e**, **φ**) contient toutes les informations nécessaire sur comment une variation des angles **φ** entraine une variation sur chaque composante de **e.**
- Une matrice est aussi une fonction linéaire

Remarque : J est une approximation linéaire (cad suppose que la fonction est un hyper-plan)

Changement incrémental dans la pose

- Supposons un vecteur **Δφ** représentant des petits changements dans les angles (cad dans les DDL des articulations)
- Avec le Jacobien on peut approximer comment **Δφ** va produire des changements sur **e**

$$\Delta\mathbf{e} \approx \frac{d\mathbf{e}}{d\varphi} \cdot \Delta\varphi = J(\mathbf{e},\varphi) \cdot \Delta\varphi = \mathbf{J} \cdot \Delta\varphi$$

- Encore une fois, il s'agit d'une approximation linéaire qui ne reste valide que pour des petits déplacements. La méthode itérative devra donc bouger par petits pas.

Changement incrémental des extrémités

- Si on veut bouger l'extrémité d'un petit déplacement **Δe**, quels petits changements **Δφ** doit-on faire sur les angles?

$$\Delta\mathbf{e} \approx \mathbf{J} \cdot \Delta\varphi$$

$$so:$$

$$\Delta\varphi \approx \mathbf{J}^{-1} \cdot \Delta\mathbf{e}$$

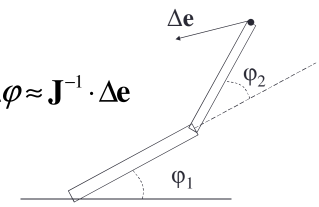
- Remarque : J^{-1} peut-être approximé (voir plus loin)

http://math.ucsd.edu/~sbuss/ResearchWeb/ikmethods/iksurvey.pdf

Changement incrémental de l'extrémité **e**

- Si on veut bouger l'extrémité **e** d'un petit déplacement **Δe**, on peut approximer un changement incrémentale **Δφ à** appliquer aux DDL (angles) des articulations

$$\Delta\varphi \approx \mathbf{J}^{-1} \cdot \Delta\mathbf{e}$$



Changement incrémental de l'extrémité **e**

- Rappelez vous que la cinématique directe est non linéaire (car sin et cos sur les variables d'entrées)
- Ceci implique que l'on ne peut utiliser le Jacobien (qui est une approximation linéaire) seulement autour de la configuration courante.
- Il faut répéter le processus de calculer le Jacobien et de se déplacer jusqu'à arriver au but.

Cible de l'extrémité

- Si **φ** représente les DDL (angles) courants des articulations et **e** représente la position courante de l'extrémité, $\mathbf{e}_{\mathbf{target}}$ la cible que doit attendre l'extrémité

Choisir **Δe**

- On veut choisir des valeurs pour **Δe** qui rapprochent **e** de $\mathbf{e}_{\mathbf{target}}$. On considère donc

$$\Delta\mathbf{e} = \mathbf{e}_{\mathbf{target}} - \mathbf{e}$$

- On espérerait que les variations **Δφ** correspondantes vont amener directement **e** sur sa cible. Seulement la non-linéarité va faire que non. Il est donc plus sage de procéder par petits pas :

$$\Delta\mathbf{e} = \beta(\mathbf{e}_{\mathbf{target}} - \mathbf{e})$$

avec $0 \le \beta \le 1$

L'algorithme à base de Jacobien

while (**e** est loin de $\mathbf{e}_{target}$)
{
Compute J(**e**,**φ**) // pour la pose courante **Φ**
Compute J^{-1} // inversion du Jacobien
$\Delta\mathbf{e} = \beta(\mathbf{e}_{target} - \mathbf{e})$ // un petit pas vers la cible
$\Delta\boldsymbol{\varphi} = J^{-1} \cdot \Delta\mathbf{e}$ // calcule changement des DDL
$\boldsymbol{\varphi} = \boldsymbol{\varphi} + \Delta\boldsymbol{\varphi}$ // applique ces changements
Compute new **e** vector // cinématique directe
}

+ CODE

Les dimensions

- Dans notre cas de base (robot plan 2D à 2 bras), F va de 2D (les 2 angles) vers 2D (les 2 coordonnées X et Y de la main en 2D).

→ Matrice Jacobienne est donc 2x2
  - Inverser une matrice 2x2 : OK

- Un squelette en 3D avec A articulations comportant 3 DDL chacun (3 angles) : N=3A
  On veut contrôler l'unique extrémité en 3D : M=3

→ Matrice Jacobienne est de dimension MxN donc 3 x 3A et donc J^{-1} de dimension 3A x 3
  - Inverse ?

Calculer J^{-1}

Un gros bout du problème est là!!!

- J^{-1} dans le cas d'une matrice carrée
  - Problème de singularité : configuration bras tendu, la matrice n'est plus inversible et le bras va osciller entre les différentes solutions

  → pose donc des problèmes même pour le cas simple des matrices carrées

- J^{-1} peut s'approximer par J^T
  - Simple à coder
  - Converge un peu moins rapide mais évite les instabilités aux singularités
  - (-) Convergence lente dans le cas de plusieurs extrémités
  - Codons J^T pour la compréhension des méthodes Jacobiennes

Calculer J^{-1} : Damped Least Squared Method

- Par la méthode des moindres carrés amortis
  DLSM=Damped Least Squared Method
  - Minimisation par une approche des moindres carrés de

$$\left\|\Delta e - J.\Delta\varphi\right\|^2 + \lambda^2\left\|\Delta\varphi\right\|^2$$

  - Le 2e terme ajoute des contraintes
  - Avec J décomposé par Singular Value Decomposition (SVD)
    - Pour s'affranchir du problème de matrice non carré
    - $J = UDV^T$ avec J de dimension MxN
    - D une matrice diagonale NxN (0 partout sauf diagonale où il y a les valeurs singulières)

Calculer J^{-1} : Damped Least Squared Method

Selectively Damped Least Squares for Inverse Kinematics.
Samuel R. Buss and Jin-Su Kim.
Journal of Graphics Tools, vol. 10, no. 3 (2005) 37-49.
http://math.ucsd.edu/~sbuss/ResearchWeb/ikmethods/

- Adaptation du λ durant le calcul

- Code disponible (avec dépendances) + une version sur mon SVN qui compile (avec GL et GLUI intégré)

VIDEO

Bilan des méthodes basées Jacobien

- Il y a peu de temps j'aurais conseillé cette famille
- En particulier DLSM car
  - (+) converge vite
  - (+) stable, donne de bons résultats même pour plusieurs extrémités
  - (+) code disponible sur le web et facilement intégrable

- Mais (-) difficile d'ajouter les contraintes sans bien comprendre la méthode

MÉTHODES ITÉRATIVES

- …
- A base d'heuristiques
  - Introduction
  - CCD
  - FABRIK

A base d'heuristiques

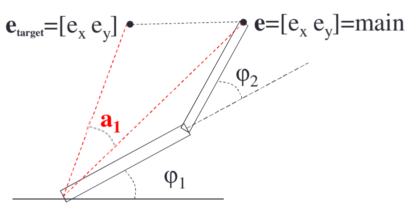
- Approches algorithmiques et géométriques
- Avantages
  → très facile à comprendre, donc à modifier (ajout des contraintes, changement de longueur des articulations, etc.)

- Il a peu de temps cette famille aurait été présenté avant celle à base de Jacobien car elle donnait de moins bons résultats en terme de stabilité, nombre d'itérations, …
  CCD en était la plus connue (utilisée dans les jeux vidéo)

- Mais [FABRIK2011] propose des résultats aussi bon que DLSM avec une approche bien plus abordable

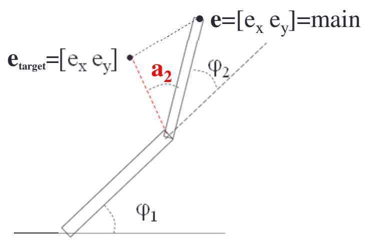
Cyclic Coordinate Descent (CCD)

- Itère sur toutes les articulations
  - Calcule et applique la rotation d'un angle
    a_i = angle entre **e et e$_{target}$**



Cyclic Coordinate Descent (CCD)

- Itère sur toutes les articulations
  - Calcule et applique la rotation d'un angle
    a_i = angle entre **e et e$_{target}$**



Cyclic Coordinate Descent (CCD)

Repeat
  ForEach joint i Do
    a_i = compute angle between **e** and **e$_{target}$**
    Rotate joint i
  EndForEach
Until d<0.01 (meaning while **e** is too far from **e$_{target}$**)

Cyclic Coordinate Descent (CCD)

- (+) facile à coder
- (+) converge rapidement
- (+) ajout du respect des contraintes facilement

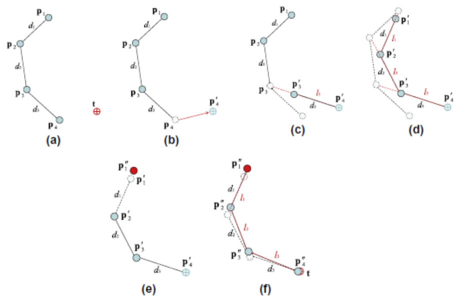
- (-) peut conduire à des poses non réalistes (même avec des contraintes)
- (-) peut provoquer des animations discontinues

FABRIK

FABRIK: a fast, iterative solver for the Inverse Kinematics problem
Andreas Aristidou, Joan Lasenby
Graphical Models, 73(5):243-260, Elsevier 2011

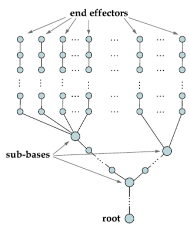
http://www.andreasaristidou.com/FABRIK.html

FABRIK : avec 1 extrémité



FABRIK : avec plusieurs extrémités

- Applique l'algo précédent sur toutes les branches indépendamment mais seulement de l'extrémité vers le nœud de séparation
- Moyenne toutes les positions du nœud de séparation ➔ nouvelle position du nœud
- Applique l'algo précédent du nœud de séparation vers les extrémités

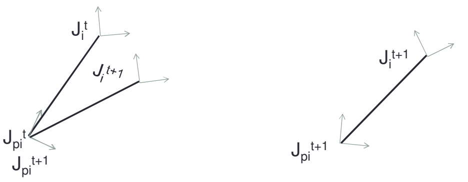


FABRIK : retrouver les angles

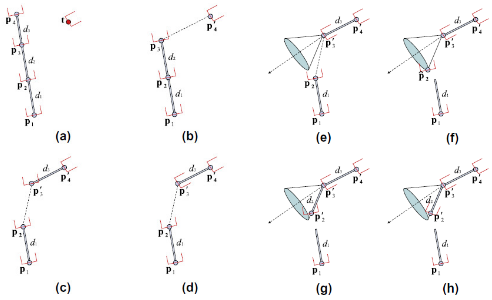
- Cherche la matrice allant de J_i^{t+1} vers son père J_{pi}^{t+1} à l'itération t+1
  - Avec FABRIK on connait la position de J_i et de J_{pi} dans le monde
- On connait les repères à l'itération précédente t

FABRIK : retrouver les angles

- Cherche la matrice allant de J_i^{t+1} vers son père J_{pi}^{t+1}
  - A=Position de J_i^{t+1} dans le repère de son père
  - B=Position de J_i^t dans le repère de son père
- A vers B définit une rotation R d'axe A×B et d'angle $\cos^{-1}$(A.B)
- On applique R à la matrice allant de J_i^t à J_{pi}^t
  ➔on obtient la matrice allant de J_i^{t+1} vers son père J_{pi}^{t+1}



FABRIK : avec des contraintes



FABRIK : la vidéo

http://www.andreasaristidou.com/FABRIK.html

Cinématique inverse : conclusion

- FABRIK semble réunir les avantages

(+) facile à comprendre et à coder

(+) convergence rapide

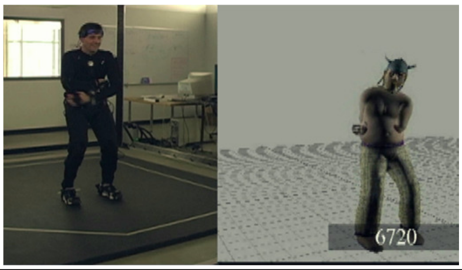
(+) modifications/ajouts très abordables

(+) bons résultats en terme de réalisme

- Alternative possible : SDLSM car code disponible

Et après …

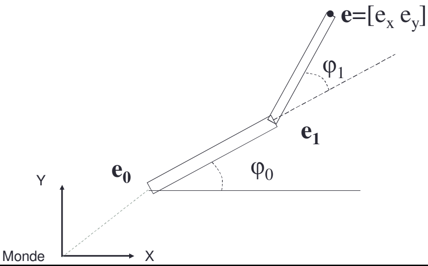
- Ces algorithmes de cinématique inverse ne résolvent pas tout …



BROUILLONS

Notre configuration de base

- Supposons un squelette 2D avec 2 articulations comportant chacune 1 DDL (rotation)



Notre configuration de base

- Articulation 0 : translation T_0 , rotation φ_0 (Racine, Root)
- Articulation 1 : translation T_1 , rotation φ_1
- Articulation 2 : translation T_1 , rotation φ_1 (Extrémité)

Cinématique directe donne e