Cinématique Inverse

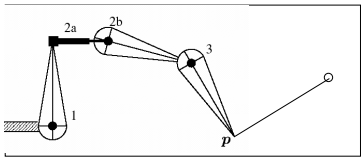
Nicolas Holzschuch
Cours d'Option Majeure 2
Nicolas.Holzschuch@imag.fr

Plan du cours

- Cinématique inverse :
  - Pourquoi faire ?
  - Animation d'un modèle
- Manipulation directe du modèle :
  - Sélection
  - Tirer une partie du modèle

Cinématique inverse

- Objet articulé
  - Liens, articulations
  - Objectif à atteindre



Cinématique inverse

- Donnée : position à atteindre (M)
- Sortie : valeurs des paramètres des articulations
- θ=vecteur des paramètres du modèle
  - $\theta = (\theta_1, \theta_2, \theta_3, \ldots, t_1, t_2, \ldots)$
- Trouver $\theta = g(M)$
- Deux rotations: calcul direct
- Trois rotations : calcul direct
- N articulations : ?

Deux rotations

- Solution directe :

$$d = \sqrt{x^2 + y^2}$$

$$d^2 = (L_1 + L_2 \cos\theta_2)^2 + (L_2 \sin\theta_2)^2$$

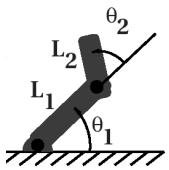
$$d^2 = L_1^2 + L_2^2 + 2L_1L_2 \cos\theta_2$$

$$\cos\theta_2 = \frac{d^2 - L_1^2 - L_2^2}{2L_1L_2}$$

$$\tan(\alpha) = \frac{L_2 \sin\theta_2}{L_1 + L_2 \cos\theta_2}$$

$$\tan(\theta_1 + \alpha) = \frac{y}{x}$$

$$\theta_1 = \arctan\left(\frac{y}{x}\right) - \arctan\left(\frac{L_2 \sin\theta_2}{L_1 + L_2 \cos\theta_2}\right)$$

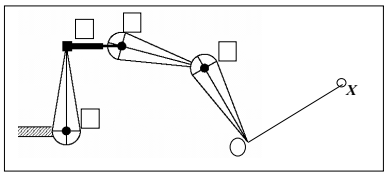


+démo

Trois rotations

- Encore une solution directe
  - trigonométrie
- Paramètre supplémentaire
  - Choix de la solution
- Limites des articulations

Cas général : n articulations



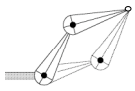
- On veut que $f(\theta)=M$
- θ=vecteur des paramètres du modèle
  - $\theta = (\theta_1, \theta_2, \theta_3, \ldots, t_1, t_2, \ldots)$
- $f(\theta)$ position de l'extrémité du modèle (coord. 2D)
- M position de la cible (coordonnées 2D)
- Connaissant M, trouver θ

Pourquoi c'est difficile ?

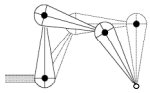
- Problème non-linéaire
- Plusieurs solutions
- Pas toujours bien conditionné
- Limites des articulations

Non-unicité

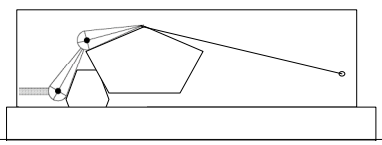
- Deux solutions :



- Intervalle de solutions :

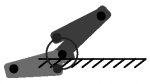


- Pas de solutions :

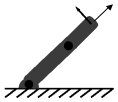


Pas toujours bien conditionné

- Petite différence sur M, grande différence sur θ :



- Changer θ ne rapproche pas de M :



Au fait, que vaut f ?

- Matrice de transformation
- Concaténation des matrices
- $f(\theta) = \mathbf{R}_1(\theta_1)\mathbf{T}_1\mathbf{R}_2(\theta_2)\mathbf{T}_2\mathbf{R}_3(\theta_3)\mathbf{T}_3 \ldots M_0$
  - M_0 position extrémité du bras avant rotations
- Non-linéaire à cause des rotations
- Calcul de f : cinématique directe

Racines d'une fonction non-linéaire

- On veut trouver θ tel que : $f(\theta)$-M = 0
- Linéarisation du problème :

$$f(\theta + h) = f(\theta) + f'(\theta)h + f''(\theta)h^2 + \ldots$$

- On part d'une valeur de θ et de $f(\theta)$
- On ne connaît pas θ tel que $f(\theta+\Delta)=\theta$
- On peut trouver Δ' qui s'en rapproche
- $\theta \leftarrow \theta+\Delta$' et on itère

Linéarisation

- Séries de Taylor :

$$f(\theta + h) = f(\theta) + f'(\theta)h + f''(\theta)h^2 + \ldots$$

- Cas des fonctions à plusieurs variables :

$$f(\theta + h) = f(\theta) + \mathbf{J}(\theta)h + {}^t h\mathbf{H}(\theta)h + \ldots$$

- **J** Jacobien de *f*, forme linéaire
- **H** Hessien de *f*, forme quadratique

Jacobien

- Matrices des dérivées d'une fonction à plusieurs variables :

$$J_{ij} = \frac{\partial f_i}{\partial x_j}$$

$$\mathbf{J}(\theta) = \begin{bmatrix} \frac{\partial f_x}{\partial x_0}(\theta) & \frac{\partial f_x}{\partial x_1}(\theta) & \ldots \\ \frac{\partial f_y}{\partial x_0}(\theta) & \frac{\partial f_y}{\partial x_1}(\theta) & \ldots \\ \frac{\partial f_z}{\partial x_0}(\theta) & \frac{\partial f_z}{\partial x_1}(\theta) & \ldots \end{bmatrix}$$

Jacobien

- Variation de $f(\theta)$ au premier ordre
- Approximation linéaire de *f*
- Matrice 3*n (ou 2*n en 2D)
- Calcul de **J**:
  - $f(\theta) = \mathbf{R}_1(\theta_1)\mathbf{T}_1\mathbf{R}_2(\theta_2)\mathbf{T}_2\mathbf{R}_3(\theta_3)\mathbf{T}_3 \ldots M_0$

$$\frac{\partial f}{\partial \theta_1} = \frac{\partial R_1}{\partial \theta_1} T_1 R_2 T_2 R_3 T_3 \ldots M_0$$

Linéarisation du problème

- $f(\theta)$-M = E, erreur actuelle
- Trouver $\Delta\theta$, tel que $\mathbf{J}(\theta)\Delta\theta$ = E
- $\Delta\theta$: résolution système linéaire
- Approximation linéaire : petits déplacements
  - Petits déplacements dans la direction de $\Delta\theta$
- Série de petits déplacements
- Recalculer **J** et E à chaque étape

Résolution itérative

- Petits pas par petits pas
- À chaque étape :
  - Choix entre plusieurs solutions
  - Prendre solution proche position actuelle
- Taille des pas :
  - Chercher taille optimale ?
  - Rotations : pour x < 2 degrés:
    - sin(x) ≈ x
    - cos(x) ≈ 1
  - Garder pas < 2 degrés

Algorithme

```
inverseKinematics()
  {
    start with previous θ;
    E = target - computeEndPoint();
    for(k=0; k<k_max && |E| > eps; k++){
      J = computeJacobian();
      solve J Δθ = E;
      if (max(Δθ)>2) Δθ = 2Δθ/max(Δθ);
      θ = θ+Δθ;
      E = target - computeEndPoint();
    }
  }
```

La bonne question

- **`solve J Δ = E;`**
- **J** n'est pas inversible
  - En fait, **J** n'est même pas carrée
  - **J** matrice 2*n :

$$\begin{bmatrix} \frac{\partial f_x}{\partial x_0}(\Delta) & \frac{\partial f_x}{\partial x_1}(\Delta) \cdots \\ \frac{\partial f_y}{\partial x_0}(\Delta) & \frac{\partial f_y}{\partial x_1}(\Delta) \cdots \end{bmatrix} \begin{bmatrix} \Delta_0 \\ \Delta_1 \\ \Delta_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} E_x \\ E_y \end{bmatrix}$$

Pseudo-Inverse

- $\mathbf{J}^T\mathbf{J}$ est carrée (n*n). Donc :
  $\mathbf{J}\Delta = E$
  $\mathbf{J}^T\mathbf{J}\Delta = \mathbf{J}^T E$
  $\Delta = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T E$
  $\Delta = \mathbf{J}^+ E$
- $\mathbf{J}^+ = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T$ *pseudo-inverse* de **J**
  - Pareil que l'inverse si **J** est carrée et inversible
  - Propriétés : $\mathbf{J}\mathbf{J}^+\mathbf{J} = \mathbf{J}$, $\mathbf{J}^+\mathbf{J}\mathbf{J}^+ = \mathbf{J}^+$
  - **J** est *m*n* ⇒ $\mathbf{J}^+$ est *n*m*
- Comment calculer $\mathbf{J}^+$?
  - Surtout si $\mathbf{J}^T\mathbf{J}$ n'est pas inversible

Singular Values Decomposition

- Toute matrice m*n peut s'exprimer par SVD:
  - $\mathbf{A}=\mathbf{U}\mathbf{S}\mathbf{V}^T$
    - **U, V** : matrices rectangulaires, colonnes orthogonales
    - S matrice diagonale, *singular values*

$$\mathbf{A} = \mathbf{U} \begin{bmatrix} s_1 & 0 & \cdots & 0 \\ 0 & s_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & s_n \end{bmatrix} \mathbf{V}^T$$

Singular Values Decomposition

- **S** unique à l'ordre et au signe des valeurs près
  - Ordre canonique : s_i positifs, ordre croissant
  - Rang de **A** : nombre de valeurs non-nulles
  - Déterminant : produit des valeurs
- **U, V** : colonnes orthogonales

$$\mathbf{U} = \begin{pmatrix} \vec{h}_1 & \vec{h}_2 & \cdots & \vec{h}_n \end{pmatrix}$$

Pseudo-Inverse avec SVD

- Calculer SVD: $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$
- Pseudo-inverse : $\mathbf{A}^+ = \mathbf{V}\mathbf{S}^{-1}\mathbf{U}^T$

$$\begin{bmatrix} s_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & s_n \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{s_1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \frac{1}{s_n} \end{bmatrix}$$

- Singulière : $s_i = 0$
- Mal conditionnée $s_i << s_0$
  - Prendre 0 au lieu de $1/s_i$ pour ces valeurs
  - Test : $s_i < \varepsilon s_0$

Résoudre **A***X*=**B** avec SVD

- $\mathbf{A}^+$ tronquée
- $\mathbf{A}^+\mathbf{B}$ donne la solution des *moindres carrés* :
  - Pas de solutions : X qui minimise $||\mathbf{A}X-\mathbf{B}||^2$
  - Plusieurs solutions : $||X||^2$ minimal tel que $\mathbf{A}X=\mathbf{B}$
  - Stable numériquement, même pour matrices mal-conditionnées
- SVD : marteau-pilon
  - $O(n^3)$ (lent)
  - Marche toujours, et assez rapide pour nous.
  - Difficile à implémenter, cf. *Numerical Recipes in C*
- Autres méthodes pour IK: CCD, $\mathbf{J}^T$,…

Et la cinématique inverse ?

- On veut résoudre $X=f(\theta)+\mathbf{J}(\theta)\Delta\theta$
  - $f(\theta)$ position de l'extrémité du bras
  - i^e colonne de $\mathbf{J}$ vient de l'articulation i
  - $f(\theta)= \mathbf{R}_1(\theta_1)\mathbf{T}_1\mathbf{R}_2(\theta_2)\mathbf{T}_2\mathbf{R}_3(\theta_3)\mathbf{T}_3 \ldots M_0$

$$\frac{\partial f}{\partial \theta_1} = \frac{\partial R_1}{\partial \theta_1} T_1 R_2 T_2 R_3 T_3 \ldots M_0$$

$$\frac{\partial f}{\partial \theta_1} = R_1(\theta_1 + \frac{\pi}{2}) T_1 R_2 T_2 R_3 T_3 \ldots M_0$$

Calcul du Jacobien

- Jacobien d'une rotation :

$$\vec{r} = f(\theta) - \text{pivot}_i = \begin{pmatrix} r_x \\ r_y \end{pmatrix}$$

$$\frac{\partial f}{\partial \theta_i}(\theta) = \begin{pmatrix} -r_y \\ r_x \end{pmatrix}$$

- Jacobien d'une translation
  - Vecteur dans la direction de translation
- Remarques :
  - Calcul en coordonnées du monde, pas du modèle
  - Degrés/radians !!! (dérivée *=π/180 ?)
  - Un degré de liberté par articulation

Algorithme

```
inverseKinematics()
  {
    Vector θ = getLinkParameters();
    Vector E = target - computeEndPoint();
    for(k=0; k<k_max && E.norm() > eps; k++){
      Matrix J = computeJacobian();
      Matrix J⁺ = pseudoInverse(J);
      Vector Δθ = J⁺E ;
      if (max(Δθ)>2) Δθ *= 2/max(Δθ);
      θ = θ+Δθ;
      putLinkParameters();
      E = target - computeEndPoint();
    }
  }
```

Inverse Kinematics, niveau II

- Limites aux articulations
- Choix de la configuration

Limites aux articulations

- Chaque articulation a un intervalle limité
  - Par ex. le coude : varie sur [0,□]
  - Élément important du réalisme
- Pour forcer les limites :
  - Tester si dépassement
  - Annuler paramètre *i*
  - Recalculer sans *i*
  - Vérifier les autres paramètres

Limites aux articulations

- Algorithme modifié :
  - Après avoir calculé □, test pour ch. articulation:

$$\theta_i^{min} < \theta_i + \Delta\theta_i < \theta_i^{max}$$

  - Si ça sort de l'intervalle :
    - Annuler colonne *i* de **J**
    - Revient à annuler paramètre *i*
  - Recalculer $\mathbf{J}^+$
    - Moindres carrés : □□0
    - Pour plus de robustesse, forcer □=0
  - Trouver □, itérer

Choix de la configuration

- Si on a une solution homogène θ :
  - $\mathbf{J}\theta = 0$
- Si θ solution de $\mathbf{J}\theta = E$, alors $\theta+\theta$ aussi :
  - $\mathbf{J}(\theta+\theta) = \mathbf{J}\theta + \mathbf{J}\theta = E + 0 = E$
- Si on veut modifier θ, de C:
  - On projette C sur le noyau de **J**:

$$C_{proj} = \left(\mathbf{J}^{+}\mathbf{J} - \mathbf{I}\right)C$$

$$\mathbf{J}\left[\left(\mathbf{J}^{+}\mathbf{J} - \mathbf{I}\right)C\right] = \left[\mathbf{J}\mathbf{J}^{+}\mathbf{J} - \mathbf{J}\right]C = (\mathbf{J} - \mathbf{J})C = 0$$

Choix de la configuration

- Valeurs souhaitée : θ_{pref}
- Changement voulu C :
  - $C_i = w_i(\theta_i - \theta_{pref})_i$
  - Poids w_i donne importance relative
- Algorithme modifié :
  - Construire C
  - Utiliser $\theta = \mathbf{J}^{+}E + (\mathbf{J}^{+}\mathbf{J} - \mathbf{I})C$
  - La projection de C sur le noyau ne nuit pas à la convergence
  - La solution penche vers θ_{pref}

Algorithmes numériques

- Beaucoup d'algorithmes for la recherche de racines de systèmes non-linéaires
  - Celui-ci marche, il y en a d'autres
- Recherche de racines lié à l'*optimisation*
  - $F(\theta) = X$ minimise $\|F(\theta) - X\|^2$
- Nombreux problèmes d'optimisation en animation
- Nombreux algorithmes doivent résoudre $\mathbf{A}X = B$

Plan du cours

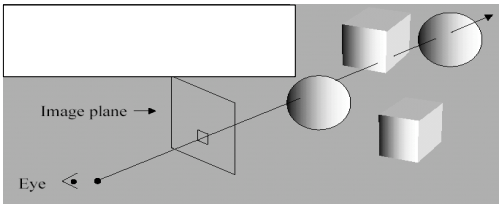
- Cinématique inverse :
  - Pourquoi faire ?
  - Animation d'un modèle
- Manipulation directe du modèle :
  - Sélection
  - Tirer une partie du modèle

Manipulation du modèle

- L'utilisateur clique à la souris
  - Trouver quel objet il a sélectionné
- 2D :
  - Conversion coord. écran vers coord. monde
- 3D :
  - Droite reliant l'œil au pixel
  - Trouver 1er objet intersecté

Sélection

- Intersection droite/modèle
  - Primitive par primitive



Sélection : modèle hiérarchique

- Descente dans la hiérarchie,
  - test à chaque niveau
- Transformation de la droite en coords. locale
- Accélération possible avec boite englobante
- Trouver le point d'intersection le plus proche

Sélection : outils OpenGL

- `glGetDoublev(GL_PROJECTION_MATRIX, pmatrix);`
  - Renvoie la matrice de projection
- `glGetDoublev(GL_MODELVIEW_MATRIX, mmatrix);`
  - Renvoie la matrice du modèle
- `glGetIntegerv(GL_VIEWPORT, viewport);`
  - Renvoie le viewport (x_{min}, y_{min}, width, height)
- Point cliqué *(x,y)*
  - En pixels
  - Convertir en coordonnées du monde :

`gluUnProject(`x_{win}`,` y_{win}`,` z_{win}`, mmatrix, pmatrix, viewport,` $*x_{obj}$`,` $*y_{obj}$`,` $*z_{obj}$`);`

Sélection : outils OpenGL

- Problème : trouver z_{win}
  - Inconnu
  - Point cliqué sur l'écran
    - $z_{win} = 0$
- Droite œil/M_{obj} (donné par `gluUnProject`)
- Intersection avec les objets de la scène

Sélection : outils OpenGL (avancé)

- Mode *sélection* :
- L'utilisateur clique sur l'écran
- On restreint à une zone d'intérêt précise
  - *e.g.* 5*5 pixels autour du point cliqué
- On retrace la scène
  - En mode *sélection*
  - Chaque primitive est nommée
- OpenGL renvoie les noms des primitives

OpenGL : mode sélection

- Restreindre à une zone d'intérêt précise :

```
gluPickMatrix(x, y, delX, delY, viewport);
```

- Appeler *avant* `gluPerspective(…);`
- Mode sélection :
  - Créer un buffer pour stocker les noms des objets:

    ```
    glSelectBuffer(size, buffer);
    ```
  - Passer en mode sélection :

    ```
    glRenderMode(GL_SELECT);
    ```

OpenGL : mode sélection

- En mode sélection :
  - Nommer les primitives :

    ```
    glInitNames();
    glLoadName(int);
    ```
  - Tracer les primitives
  - Également : `glPushName(int); glPopName();`
  - Repasser en mode normal :

    ```
    numHits = glRenderMode(GL_RENDER);
    ```
- `buffer` contient les noms des primitives tracées

OpenGL : mode sélection

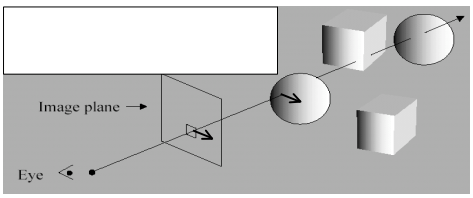
```
glGetIntegerv(GL_VIEWPORT, viewport);
gluPickMatrix(x, y, 5, 5, viewport);
gluPerspective(…);
glSelectBuffer(100, buffer);
glRenderMode(GL_SELECT);
glInitNames();
drawNamedObjects();
numHits = glRenderMode(GL_RENDER);
if (numHits > 1) {
   glGetDoublev(GL_PROJECTION_MATRIX, pmatrix);
   glGetDoublev(GL_MODELVIEW_MATRIX, mmatrix);
   gluUnProject(x,y, 0, mmatrix, pmatrix, viewport,
                objx, objy, objz);
   /* calculer points d'intersection pour ch. objet */
   /* garder objet avec intersection la plus proche */
}
```

Mode sélection

- Facile à implémenter
- Pas forcément le plus rapide
- Peut servir à d'autres choses :
  - Objets visibles d'un point
  - Pré-sélection d'objets dans l'espace
  - Prévision de collisions

Faire glisser

- Vecteur donnée en coordonnées pixel
  - Traduire en coordonnées monde
- En 3D, déplacement parallèle au plan écran
- Mettre à jour position objet sélectionné



Plan du cours

- Cinématique inverse :
  - Pourquoi faire ?
  - Animation d'un modèle
- Manipulation directe du modèle :
  - Sélection
  - Tirer une partie du modèle

Inverse Kinematics, niveau III

- Autres méthodes :
  - Transposée du Jacobien
  - Cyclic Coordinate Descent

Transposée du Jacobien

- Au lieu du pseudo-inverse, utiliser le Jacobien :
  - Au lieu de : θ += $\mathbf{J}^{+}(\Delta)dx$
  - On prend : θ += $\mathbf{J}^{T}(\Delta)dx$
- Pratique :
  - Pas d'inversion
  - Pas de singularités
- Mais pourquoi ça marche ?

Travaux virtuels

- Déplacement infinitésimaux
  - W= force*distance
  - W=moment*angle

$$F \cdot \Delta x = M \cdot \Delta\theta$$
$$F^T \Delta x = M^T \Delta\theta$$
$$\Delta x = \mathbf{J}\Delta\theta$$
$$F^T \mathbf{J}\Delta\theta = M^T \Delta\theta$$
$$F^T \mathbf{J} = M^T$$
$$M = \mathbf{J}^T F$$

Transposée du Jacobien

- Distance à l'objectif = force qui tire l'extrémité
- Remplacer système non-linéaire par système dynamique
  - Lois de la dynamique
- Équivalent à *steepest descent algorithm*

Transposée du Jacobien

- Avantages :
  - Pas d'inversion (numériquement moins cher)
  - Pas de singularités
- Inconvénients :
  - Convergence plus lente
  - $\mathbf{J}^+$ donnait solution avec norme minimale
  - Ici pas le cas :
    - Éléments éloignés ont influence plus grande
    - Problèmes d'échelle

Cyclic Coordinate Descent

- Problème multi-dimensionnel compliqué
- Résoudre une série de problèmes 1D
- Pour chaque articulation :
  - Trouver valeur du paramètre qui se rapproche le plus de la solution
  - Solution analytique
- Itérer

Cyclic Coordinate Descent

- Avantages :
  - Facile à implémenter
  - Efficace (souvent)
  - Numériquement pas cher
- Inconvénients :
  - Mouvements peu naturels
  - Ne trouve pas forcément la solution optimale
    - Limiter les pas à chaque étape
    - Mais convergence plus lente