

Computer Animation and Special Effects

Hw3 Inverse Kinematics

310551067 吳岱容

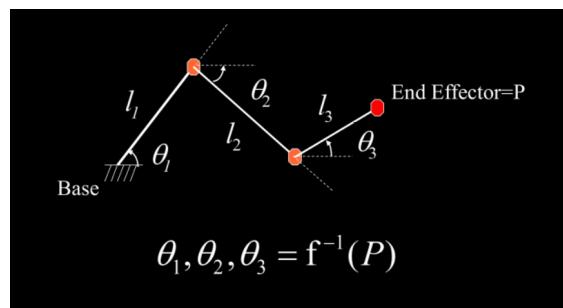
- **Intoduciton**

這次的作業主要實作的是Inverse Kinematics, 在上次的作業我們已經實作了Forward Kinematics, 不同於Forward Kinematics, Inverse Kinematics是給定最終目標的位置, 反推出所有骨頭需要的旋轉角度, Inverse Kinematics有很多不同的計算方式, 例如: Jacobian Pseudoinverse、Jacobian Transpose、Levenberg-Marquardt Damp Least Squares...等方法、這是作業主要時做的是Jacobian Pseudoinverse、利用bdcsvd解linear least squares system等方法。

- **Fundamental**

- Inverse Kinematics

在Inverse Kinematics中, 我們可以決定最終目標的位置, 然後給予其位置反推出每個骨頭應該有的旋轉角度。



能夠自由決定位置很方便, 但隨之而來的問題是計算方面的問題, 比起Forward Kinematics, Inverse Kinematics的計算要複雜很多。

這邊我們主要介紹的是Jacobian Pseudoinverse的方法。

- Jacobian

Jacobian 代表的是函數f的線性近似值, 所以我們可以用e (end-effect)去回推出旋轉角度dθ。

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_1}{\partial \theta_2} & \dots & \frac{\partial f_1}{\partial \theta_n} \\ \frac{\partial f_2}{\partial \theta_1} & \frac{\partial f_2}{\partial \theta_2} & \dots & \frac{\partial f_2}{\partial \theta_n} \\ \vdots & \vdots & \dots & \vdots \\ \frac{\partial f_k}{\partial \theta_1} & \frac{\partial f_k}{\partial \theta_2} & \dots & \frac{\partial f_k}{\partial \theta_n} \end{pmatrix}$$

$$\mathbf{J} = \frac{d\mathbf{e}}{d\theta}$$

$$d\mathbf{e} = \mathbf{J}d\theta$$

$$d\theta = \mathbf{J}^{-1}d\mathbf{e}$$

- Jacobian的近似值

將Jacobian每個column拆開來看並對θ和e做偏微分可以得到：

$$\mathbf{J}_i = \frac{\partial \mathbf{e}}{\partial \theta_i} = \frac{\Delta \mathbf{e}}{\Delta \theta_i} = \begin{bmatrix} \frac{\Delta e_x}{\Delta \theta_i} \\ \frac{\Delta e_y}{\Delta \theta_i} \\ \frac{\Delta e_z}{\Delta \theta_i} \end{bmatrix}$$

針對每一個關節其其中一個維度的旋轉, e的線性變化量de可以看做是在此維度下的旋轉變化量(ωrdt)：

$$\frac{d\mathbf{e}}{dt} = |\omega| \frac{\omega}{|\omega|} \times \mathbf{r} = \frac{d\theta}{dt} \mathbf{a} \times \mathbf{r}$$

其中a維單位長度的旋轉向量(ω/abs(ω)), 透過移項可以得到：

$$\frac{d\mathbf{e}}{d\theta} = \mathbf{a} \times \mathbf{r}$$

$$\mathbf{J}_1 = \frac{\Delta \mathbf{e}}{\Delta \theta_1} = \begin{bmatrix} \frac{\Delta e_x}{\Delta \theta_1} \\ \frac{\Delta e_y}{\Delta \theta_1} \\ \frac{\Delta e_z}{\Delta \theta_1} \end{bmatrix} = \mathbf{a}_i \times (\mathbf{e} - \mathbf{r}_i)$$

- Inverse Jacobian

我們主要的目的是要解出dθ, 因此根據上面的推導會需要Jacobian的反矩陣, 但是Jacobian並不是方陣, 所以我們透過Pseudo Inverse的方式(J+)。

$$\mathbf{J}^+ = (\mathbf{J}^T \mathbf{J})^{-1} \mathbf{J}^T$$

J+可以透過SVD的方式解得：

$$\begin{aligned} \mathbf{J}^+ &= (\mathbf{J}^* \mathbf{J})^{-1} \mathbf{J}^* \\ &= (V \Sigma U^* U \Sigma V^*)^{-1} V \Sigma U^* \\ &= (V \Sigma^2 V^*)^{-1} V \Sigma U^* \\ &= (V^*)^{-1} \Sigma^{-2} V^{-1} V \Sigma U^* \\ &= V \Sigma^{-2} \Sigma U^* \\ &= V \Sigma^{-1} U^* \end{aligned}$$

● Implementation

- Forward Kinematics
基本上和上次作業一模一樣。
- leastSquareSolver

- bdcsvd

透過Eigen module下可以直接使用bdcsvd來求得線性系統解

```
solution = jacobian.bdcSvd(Eigen::ComputeThinU | Eigen::ComputeThinV).solve(target);
return solution;
```

- Pseudo Inverse

利用SVD獲得J+的方式

```

    auto svd = jacobian.jacobiSvd(Eigen::ComputeFullU | Eigen::ComputeFullV);
    const auto& singularVal = svd.singularValues();
    Eigen::Matrix<float, Eigen::Dynamic, Eigen::Dynamic> singularValuesInv(jacobian.cols(), jacobian.rows());
    singularValuesInv.setZero();
    //singularValuesInv = singularVal.inverse();
    double pinvToler = 1.e-6;
    for (int i = 0; i < singularVal.size(); i++) {
        if (singularVal(i) > pinvToler) {
            singularValuesInv(i, i) = 1.0f / singularVal(i);
        } else {
            singularValuesInv(i, i) = 0.0f;
        }
    }/*
    auto pJacobian = svd.matrixV() * singularValuesInv * svd.matrixU();
    solution = pJacobian * target;

```

- Path建立

由end point traverse parent一直到start point, 將每個點放入list中, 在list中代表的是動作中會動到的每個bones。

```

bool reachable = true;
Bone* cur = end;
while (true) {
    if (cur->idx == start->idx) {
        boneList.insert(boneList.begin(), cur);
        break;
    } else if (cur->idx == root->idx) {
        boneList.insert(boneList.begin(), cur);
        reachable = false;
        break;
    } else {
        boneList.insert(boneList.begin(), cur);
        cur = cur->parent;
    }
}

```

考慮有可能start會跟end在root node的兩側(但就這次的作業來說沒有這種情況), 所以需要從start traverse回root, 並將path加入回list中。

```

if (!reachable) {
    std::stack<Bone*> q;
    cur = start;
    while (true) {
        if (cur->idx == root->idx) {
            break;
        } else {
            q.push(cur);
            cur = cur->parent;
        }
    }
    while (!q.empty()) {
        boneList.insert(boneList.begin(), q.top());
        q.pop();
    }
}

```

- Jacobian Matrix

先針對bone檢查其是否有該旋轉自由度，接著根據我們上面implementation中

提到的填入 a_i 和 $(e - r_i)$ 的的cross product。

```
for (int i = 0; i < boneNum; i++) {
    Eigen::Vector3f deltaPos = boneList[i]->endPosition - boneList[i]->startPosition;
    if (boneList[i]->dofrx) {
        auto ai = boneList[i]->rotation.toRotationMatrix().col(0);
        jacobian.col(3 * i) = ai.normalized().cross(deltaPos);
    }
    if (boneList[i]->dofry) {
        auto ai = boneList[i]->rotation.toRotationMatrix().col(1);
        jacobian.col(1 + 3 * i) = ai.normalized().cross(deltaPos);
    }
    if (boneList[i]->dofrz) {
        auto ai = boneList[i]->rotation.toRotationMatrix().col(2);
        jacobian.col(2 + 3 * i) = ai.normalized().cross(deltaPos);
    }
}
```

- θ 的更新

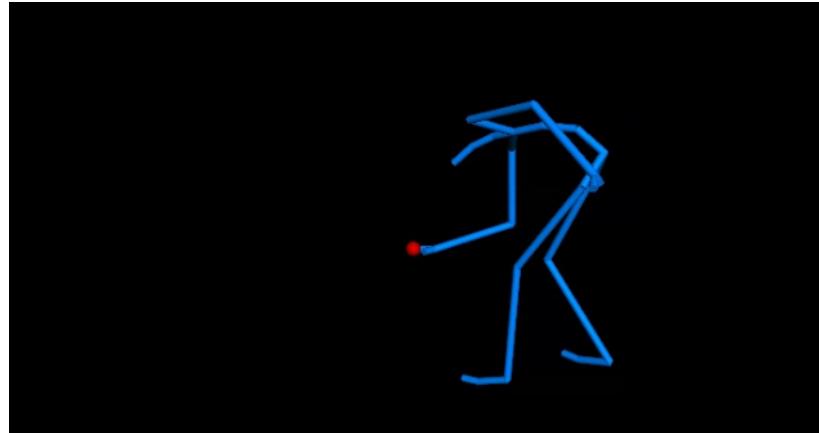
得到Jacobian之後就可以帶入我們先前寫的leastSquareSolver, 由此得到 $d\theta$, 即可對posture的旋轉角度做更新。

```
auto dTheta = leastSquareSolver(jacobian, target - end->endPosition);
```

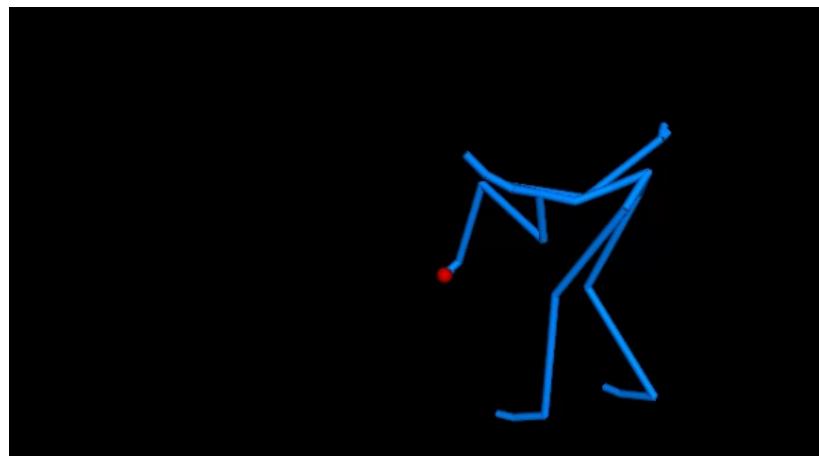
```
posture.eulerAngle[bone.idx] += Eigen::Vector3f(dTheta[0+j*3],dTheta[1+j*3],dTheta[2+j*3]) * step;
```

- Result and discussion

- 不同的least square處理方式
 - bdcsvd



- self-implemented pseudo inverse



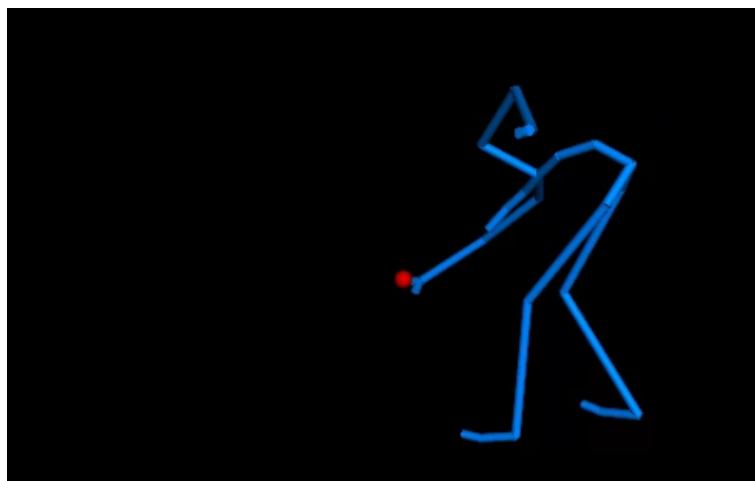
- Discussion

不同的計算方式會有不同的結果, 另外self-implemented pseudo

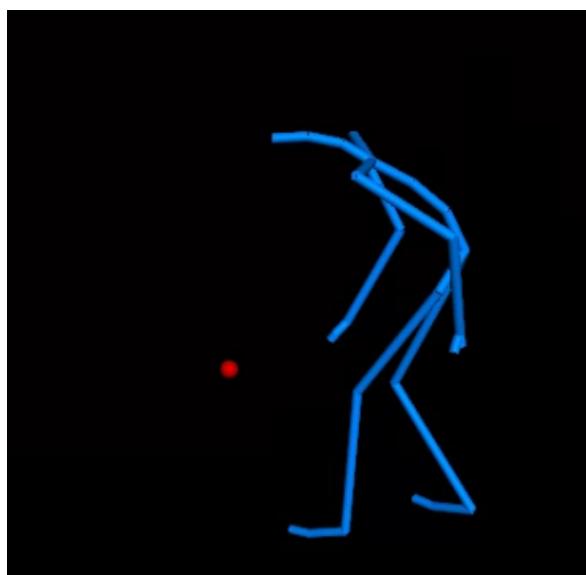
inverse有比較不穩定的結果有可能是因為SVD後的Singular inverse採用的是估算所以可能造成了偏差。

- Step大小的不同

- 0.5f



- 0.0001f



- Discussion

Step值代表的是控制每一次更新的大小，如果過大有可能造成骨架扭曲不穩定的狀況，但如果過小就又會造成骨架無法碰觸到我們設定的點。

- Conclusion

在這次的作業中我們實作了透過Inverse Kinematics，將骨架移動碰觸到我們設定的點，透過這一次作業也對上課所學有更深刻的理解。

- Reference

- Neutrino's Blog: <https://tigercosmos.xyz/post/2020/06/ca/inverse-kinematics/>
 - Samuel R. Buss. 2009. Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods.: <http://graphics.cs.cmu.edu/nsp/course/15-464/Spring11/handouts/iksurvey.pdf>
 - 上課簡報
 - https://eigen.tuxfamily.org/dox/classEigen_1_1BDCSVD.html

