電腦動畫與特效 HW2 110612117 張仲瑜

I. Introduction/Motivation

在這個作業中，完成了正向運動學與反向運動學的核心程式，並透過兩種方式計算骨架間的相對位移及旋轉，完成跑步、跳躍等骨架動畫。

II. Fundamental

A. 正向動力學

從根節點開始，透過深度優先或廣度優先的方式逐一計算整個骨架中每個節點繪製時的絕對座標。計算公式為 $\boldsymbol{T_{i+1} = {}_{0}^{i}RV_i + T_i}$，其中 $\boldsymbol{V_i}$ 為骨頭在本地座標下的向量；$\boldsymbol{{}_{0}^{i}R}$ 為自根節點累積的旋轉矩陣，而 $\boldsymbol{T_i}$ 為 frame i 原點在全域座標下的座標。該公式的意義為將本地座標的向量，乘上前面骨頭累積的旋轉矩陣後，再加上先前父節點的運算結果，即可更新子節點的全域座標。

B. 反向動力學

與正向動力學相對，先給定終點的目標位置，反推骨架應該怎麼移動。概念為給定目標位置後，回推每個關節自由度需要移動多少角度以逼近該結果($\boldsymbol{d\theta = J^{-1}de}$)。實作上先計算 Jacobian 矩陣去尋找函數的線性逼近值，再透過 SVD 去解出反矩陣，並迭代進行產生使誤差越來越小的$\Delta\theta$直到誤差可以接受。Jacobian 可以透過以下公式計算: $\boldsymbol{J_i = \frac{\Delta e}{\Delta \theta i}} =$

$\begin{bmatrix} \frac{\Delta ex}{\Delta \theta i} \\ \frac{\Delta ey}{\Delta \theta i} \\ \frac{\Delta ez}{\Delta \theta i} \end{bmatrix} = \boldsymbol{a_j}\ \boldsymbol{cross}\ (\boldsymbol{e} - \boldsymbol{r_j})$ ，其中a_j為單位長度的旋轉軸向量，$\boldsymbol{e}$ 為目標終點，$\boldsymbol{r_j}$ 為第 j 個關節的座標，相減即為臂長。

C. 本地座標軸

以每根骨頭起始點為原點的座標軸系，骨頭本身的向量即以此座標軸儲存，因此需要在計算完旋轉即平移後轉回 Global 座標軸來與其他骨頭在同個座標下繪製。

D. 全域座標軸

以(0, 0, 0)為原點的座標軸系，紀錄繪製骨架時採用的全域座標，表示每根骨頭的絕對位置。

III. Implementation

A. 正向動力學

```
26    if (bone->parent) {
27        bone->start_position = bone->parent->end_position;
28        bone->rotation = bone->parent->rotation * bone->rot_parent_current *
29                         util::rotateDegreeZYX(posture.bone_rotations[bone->idx]);
30    }
31    else {
32        bone->start_position = posture.bone_translations[bone->idx];
33        bone->rotation = util::rotateDegreeZYX(posture.bone_rotations[bone->idx]);
34    }
35
36    bone->end_position = bone->start_position + (bone->rotation * bone->dir * bone->length);
37    if (bone->child) {
38        forwardSolver(posture, bone->child);
39    }
40
41    if (bone->sibling) {
42        forwardSolver(posture, bone->sibling);
43    }
44 }
```

26~30

如果為子節點，則將骨頭起始位置設為父節點的結束位置，

並將全域的父節點旋轉矩陣乘上座標轉換(全域–>本地)的矩陣，再乘上自己在 posture 中定義的旋轉矩陣(本地座標軸)。

31~34

如果為起始節點，則直接存取 posture 中對應的值。

36

$\boldsymbol{T_{i+1}} = {}_{\boldsymbol{0}}^{\boldsymbol{i}}\boldsymbol{R}\boldsymbol{V_i} + \boldsymbol{T_i}$ 依序分別為 end_pos, rotation, dir*len, start_pos

37~43 透過 BFS 迭代整顆骨架樹

B. 反向動力學

```
54      Eigen::VectorXd deltatheta(Jacobian.cols());
55      deltatheta.setZero();
56      Eigen::JacobiSVD<Eigen::MatrixXd> svd(Jacobian, Eigen::ComputeThinU | Eigen::ComputeThinV);
57      deltatheta = svd.solve(target);
58      return deltatheta;
```

使用 Eigen 函式庫進行 SVD 分解

並使用 solve 成員函式解 pseudoInverse.

```
112         for (long long i = 0; i < bone_num; i++) {
113             Eigen::Vector3d arm = (target_pos[chainIdx] - *jointChains[chainIdx][i + 1]).head<3>();
114             Eigen::Matrix3d rotMatrix = boneChains[chainIdx][i]->rotation.matrix().block<3, 3>(0, 0);
115             if (boneChains[chainIdx][i]->dofrx) {
116                 Jacobian.col(3 * i) << rotMatrix.col(0).cross(arm), 0.0;
117             }
118             else {
119                 Jacobian.col(3 * i).setZero();
120             }
121
122             if (boneChains[chainIdx][i]->dofry) {
123                 Jacobian.col(3 * i + 1) << rotMatrix.col(1).cross(arm), 0.0;
124             }
125             else {
126                 Jacobian.col(3 * i + 1).setZero();
127             }
128
129             if (boneChains[chainIdx][i]->dofrz) {
130                 Jacobian.col(3 * i + 2) << rotMatrix.col(2).cross(arm), 0.0;
131             } else {
132                 Jacobian.col(3 * i + 2).setZero();
133             }
134         }
```

113~114

計算求得 Jacobian matrix 矩陣需要的臂長與單位旋轉矩陣

皆取三維即可

115~133

如果該方向有自由度，則設單位旋轉矩陣外積臂長，

沒有的話則將該行設為 0

```
145     for (long long i = 0; i < bone_num; i++) {
146         posture.bone_rotations[boneChains[chainIdx][i]->idx] += util::toDegree(deltatheta.segment(3 * i, 3));
147         //bonus
148         if (posture.bone_rotations[boneChains[chainIdx][i]->idx][0] > boneChains[chainIdx][i]->rxmax) {
149             posture.bone_rotations[boneChains[chainIdx][i]->idx][0] = boneChains[chainIdx][i]->rxmax;
150         }
151         else if (posture.bone_rotations[boneChains[chainIdx][i]->idx][0] < boneChains[chainIdx][i]->rxmin) {
152             posture.bone_rotations[boneChains[chainIdx][i]->idx][0] = boneChains[chainIdx][i]->rxmin;
153         }
154
155         if (posture.bone_rotations[boneChains[chainIdx][i]->idx][1] > boneChains[chainIdx][i]->rymax) {
156             posture.bone_rotations[boneChains[chainIdx][i]->idx][1] = boneChains[chainIdx][i]->rymax;
157         }
158         else if (posture.bone_rotations[boneChains[chainIdx][i]->idx][1] < boneChains[chainIdx][i]->rymin) {
159             posture.bone_rotations[boneChains[chainIdx][i]->idx][1] = boneChains[chainIdx][i]->rymin;
160         }
161
162         if (posture.bone_rotations[boneChains[chainIdx][i]->idx][2] > boneChains[chainIdx][i]->rzmax) {
163             posture.bone_rotations[boneChains[chainIdx][i]->idx][2] = boneChains[chainIdx][i]->rzmax;
164         }
165         else if (posture.bone_rotations[boneChains[chainIdx][i]->idx][2] < boneChains[chainIdx][i]->rzmin) {
166             posture.bone_rotations[boneChains[chainIdx][i]->idx][2] = boneChains[chainIdx][i]->rzmin;
167         }
168     }
```

146

將須改變的旋轉矩陣切出來，轉換成角度後再存回

posture.bone_rotations

148~167 (Bonus)

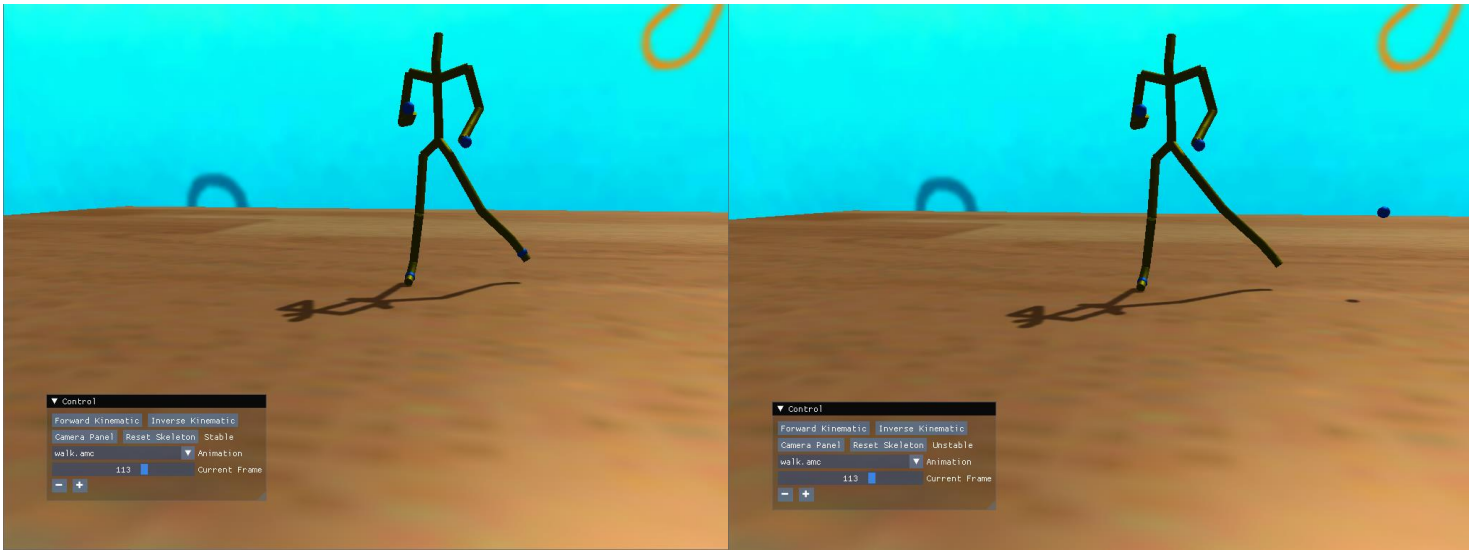
根據 bone 三個維度的旋轉限制做調整，將所有低於或超過限制的值調回規定上下限

IV. Result and Discussion

A. 正向動力學(略，見 Demo 影片)

B. 反向動力學

i. 手腳皆可在 stable 條件下任意拉動(左)，若拉到抵達不了的區域，則會順利停在最後一個可行的點(右)



ii. 有(左)無(右)關節旋轉上下限限制

iii. 不同的 Step、Epsilon、迭代次數的影響

1. Step: 影響到收斂的速度，過小的 step 會導致迭代速度過慢，超過次數了都還沒有使誤差小於 epsilon 的範圍；過大的則可能使誤差持續震盪，無法收斂。
2. Epsilon: 可容忍的誤差，過小的容忍誤差需要搭配很小的 step 與大量的迭代次數才可能完成，但計算速度也會變慢許多；過大的 epsilon 則會降低成的精

準度。

3. 迭代次數: 需要與前兩者搭配設定，在精準度與效率間取得平衡。

V. Conclusion

這次的實作可說是收穫良多，在上課時滿滿抽象的概念都很清楚的地在以上幾個程式碼中實作出來，儘管過程中遇到許多錯誤如 Eigen matrix 的維度沒有對齊、反向運動學中的參數錯誤導致動手腳會抬起來等，但在細心除錯後都順利解決。運動學的應用在機械手臂機台、機器人領域等非常廣泛，希望下次有機會運用深度學習相關的模型來處理類似問題，學習更進階的模擬方式。

VI. Demo link:

https://drive.google.com/drive/folders/1Frg9a6_plTunAQTJi0M_haolUplEKGWk?usp=drive_link