

CASE HW3

109550135 范恩宇

1. Introduction :

這次作業是要模擬「人體用某個身體部位碰到球」所產生的動作型態。由於須根據 bone 改變的 target position 逆推回各自的 start position，過程要以 Inverse Kinematics 實作並瞭解如何使用 Jacobian Matrix。

2. Fundamentals :

- Iterative Inverse Kinematics

不同於上次的 Forward Kinematics 是藉著給 bone 的改變量(平移、旋轉等)後才算 bone 的 end point，這次的 Inverse Kinematics 則是先給 end point 再算過程的變量。此外，這裡利用 step 來一步步將 bone 的 end position 修正至接近 target，因此為 iterative。

- Pseudo Inverse of the Jacobian

Jacobian Matrix 是用來表現一多變數向量函數的最佳線性逼近，而在這次作業負責表示 end effector 的瞬間變化量，我們算出

Jacobian 之後要將它轉成 inverse 並乘上「end effector 到 target 的

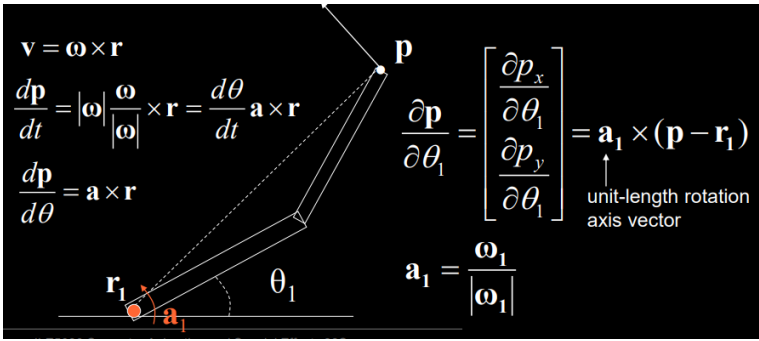
向量」來得到 bone 的旋轉角度變化。但因為並非所有 Jacobian

Matrix 都可逆，這裡才使用 Jacobian 的 pseudo inverse。

過程要從 target 回溯到 start bone 的，每個 bone 的 rotation 都

要算出一個 Jacobian Matrix 的 column，並且如同講義這處給的公

式得到每次 rotation 之 xyz 三維各自的偏微分。


$$\mathbf{v} = \boldsymbol{\omega} \times \mathbf{r}$$
$$\frac{d\mathbf{p}}{dt} = |\boldsymbol{\omega}| \frac{\boldsymbol{\omega}}{|\boldsymbol{\omega}|} \times \mathbf{r} = \frac{d\theta}{dt} \mathbf{a} \times \mathbf{r}$$
$$\frac{d\mathbf{p}}{d\theta} = \mathbf{a} \times \mathbf{r}$$
$$\frac{\partial \mathbf{p}}{\partial \theta_1} = \begin{bmatrix} \frac{\partial p_x}{\partial \theta_1} \\ \frac{\partial p_y}{\partial \theta_1} \end{bmatrix} = \mathbf{a}_1 \times (\mathbf{p} - \mathbf{r}_1)$$
$$\mathbf{a}_1 = \frac{\boldsymbol{\omega}_1}{|\boldsymbol{\omega}_1|}$$

unit-length rotation axis vector

3. Implementation :

- Forward kinematics

作業二的 forward solver 沒有使用 hint 處提到的 axis，因此直接

把那次作業的搬來，沒改動任何東西。簡單來說就是用 BFS 來

traverse 所有 bone，過程中若 Bone 的 sibling 尚未被標記為 visited

則向那 traverse。最後也檢查 Bone 的 child 是否有漏跑，有則一樣

開跑 BFS。

● Least Square Solver

看到 hint 處說可以用線性代數的 SVD，我就直接套用 Eigen 中的 JacobiSVD，並將得到的「Jacobian 的 pseudo inverse」轉為 minimum norm solution 後 return。

```
Eigen::VectorXd deltatheta(Jacobian.cols());
// get pseudo inverse of the Jacobian with SVD
Eigen::JacobiSVD<Eigen::Matrix4Xd> svd(Jacobian, Eigen::ComputeThinU | Eigen::ComputeThinV);
// minimum-norm solution
deltatheta = svd.solve(target);
return deltatheta;
```

● Inverse Kinematics

首先 bone num 我是從 end bone 開始一直往其 parent 搜尋，如果算到 start bone 或是 root bone 就停下。

```
size_t bone_num = 1; // 0
std::vector<acclaim::Bone*> boneList;

acclaim::Bone* current = end_bone;
while (current != start_bone && current != root_bone) {
    boneList.push_back(current);
    bone_num++;
    current = current->parent;
}
```

接著建立 Jacobian。過程先將 rotation theta 的 x,y,z 三個方向各別算出並依照 $a_i \times (p - r_i)$ 的公式求出偏微分後的結果，再依據當前跑的 xyz 維度，將前面得到的偏微分結果存進對應的 Jacobian column。

```
current = end_bone;
for (long long i = 0; i < bone_num; i++) {

    Eigen::Matrix3d rot_mat = current->rotation.linear();
    Eigen::Vector4d delta_pos = end_bone->end_position - current->start_position;

    for (int j = 0; j < 3; j++) {
        Eigen::Vector3d ai = Eigen::Vector3d::Zero();
        if (j == 0) { // dofrx == true
            ai = (rot_mat * Eigen::Vector3d(1, 0, 0)).normalized();
        }
        else if (j == 1) { // dofry == true
            ai = (rot_mat * Eigen::Vector3d(0, 1, 0)).normalized();
        }
        else { // dofrz == true
            ai = (rot_mat * Eigen::Vector3d(0, 0, 1)).normalized();
        }
        Eigen::Vector3d radius = Eigen::Vector3d::Zero();
        radius << delta_pos.head(3);
        Eigen::Vector3d p_dif = ai.cross(radius);
        //cout<<j<<": "<<par_dif.xO<<" "<<par_dif.yO<<" "<<par_dif.zO<<"\n";
        Jacobian.col(i * 3 + j) << p_dif, 1;
    }
    current = current->parent;
}
```

最後將剛得到的 Jacobian Matrix 從 end bone 開始往其 parent 回溯，藉前面做好的 Least Square Solver 與 bone 應該旋轉的方向，從而依據當前的 xyz 維度更新 bone rotation。

```
current = end_bone;
for (long long i = 0; i < bone_num; i++) {
    for (int j = 0; j < 3; j++) { // x, y, z
        posture.bone_rotations[current->idx][j] += deltatheta[i * 3 + j];
    }
    current = current->parent;
}
```

4. Result and Discussion :

- How different step and epsilon affect the result

我認為 step 有點類似 machine learning 學到的 gradient descent 之 learning rate，值越大則收斂越快但易偏離理想結果，值越小則收斂較慢但貼近目標。Epsilon 則是 bone end 碰到目標的誤差，值越小則越讓 end effector 和 target 能在距離較近時結束運算，反之同理。像這次作業中，就是讓 target 和 end bone 之 end position 的距離小於 epsilon 時停止。

- Touch the target or not

試了多種 xyz 三維度拉桿的組合，發現 bone 基本上會試圖用 end 端來碰球，當然能彎曲的程度或 bone 本身長度不夠時就會無法碰觸，在沒做 bonus 的情況下就會開始亂甩，跟讓筋很硬的人做坐姿

體前彎會有的表現蠻類似的。尤其 bone 的 start position 到 end position 的路徑中有跨過 root 時，bone 的實際長度會更短，畢竟 root 固定，使得能動的限於 root position 到 end position。

● Least square solver

除了現在 code 用的 SVD(使用 Eigen 的 JacobiSVD)，我也試過用 $J^T * (J * J^T)^{-1} * \text{target}$ 的公式(註解在 SVD 的 code 的下方)。兩者能讓人形穩定拿球的範圍目測差距不遠，但也可能是因為我沒做 bonus。或許加了判斷是否 stable 與讓 bone 在 unstable 時不會亂甩的條件後會有所變動。

5. Conclusion :

一開始看到作業投影片的量 and 相關公式時有點嚇到，不太確定從何下手。不過硬著頭皮把 bone 等等的.h 檔結構看清楚並弄懂 E3 討論區同學的問題與助教的解答後，自己有的問題基本上都解決的差不多，可惜最近時間真的不夠做 bonus。實在希望某些課也能開和這堂課一樣的作業討論區，能確定自己實際在做什麼的同時，學到一些沒想過可以成功的做法，真的不錯。