2020 CA HW3 Report

一、 作業目標

本次作業目標在於讓同學們在了解 Forward kinematics 的情況下，再學會 Inverse Kinematics 的實作方法與其計算方式。

二、 環境

IDE: Visual studio 2017 / 2019
Platform: Windows
Graphics API: OpenGL
OpenGL Loading Library: glad2
OpenGL Toolkit: glfw
UI Library: dear imgui
Math Library: Eigen

三、 作業內容

1. Inverse Kinematics using Pseudo Inverse
   使用 Pseudo Inverse 的方法，實作出整個 Inverse Kinematics
2. 實作 Pseudoinverse

四、 實作

3. Inverse Kinematics using Pseudo Inverse

Inverse Kinematics 有相當多種計算方式，本次使用的是 Pseudo Inverse。
首先需要找到當我每個關節旋轉後，我的末端（例如手指末端）會移動到哪個位置。
因此需要找到該 Jacobian matrix：

$$J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial p_x}{\partial \theta_1} & \frac{\partial p_x}{\partial \theta_2} \\ \frac{\partial p_y}{\partial \theta_1} & \frac{\partial p_y}{\partial \theta_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathbf{p}}{\partial \theta_1} & \frac{\partial \mathbf{p}}{\partial \theta_2} \end{bmatrix}$$

又因為每一個關節的角度都會影響到下一個關節的起始點，因此連帶影響了後面的計算，因此若是 Jacobian matrix 直接用角度計算，當關節過多，整個 Jacobian matrix 的計算量也會過大，因此這邊改用另一種方向來計算：

$$\frac{\partial \mathbf{p}}{\partial \theta_i} = \mathbf{a}_i \times (\mathbf{p} - \mathbf{r}_i)$$

a 代表轉軸在世界座標中的單位向量，p 代表終點在世界座標中的位置向量，r 則代表關節在世界座標中的位置向量。
a 的計算，只需要取原本關節的 local 轉軸，也就是很普通的 xyz 軸，透過關節本身的 rotation 旋轉矩陣，就可以得到該關節在世界座標中的三個轉軸了，並且要記得取單位向量。
p 是想要抵達的座標點。
r 是關節的位置向量，而此次的實作中，我們都只有儲存每一根骨頭的資訊，因此要從骨頭的起點來取得關節的位置。
由於在三維空間中有三個轉軸，因此在 Jacobian matrix 中，每一個轉軸都需要占用一整個 column 來計算，因此每個關節都有三個 column。

```
Eigen::Vector4d rotationAxisX = (temp_bone->rotation * Eigen::Vector4d(1, 0, 0, 0)).normalized();
Eigen::Vector4d rotationAxisY = (temp_bone->rotation * Eigen::Vector4d(0, 1, 0, 0)).normalized();
Eigen::Vector4d rotationAxisZ = (temp_bone->rotation * Eigen::Vector4d(0, 0, 1, 0)).normalized();

Jacobian.col(jacobianCount + 0) = rotationAxisX.cross3(target_pos - temp_bone->start_position);
Jacobian.col(jacobianCount + 1) = rotationAxisY.cross3(target_pos - temp_bone->start_position);
Jacobian.col(jacobianCount + 2) = rotationAxisZ.cross3(target_pos - temp_bone->start_position);
```

需要注意的是，若該關節有些角度不能轉動，必須將該角度在 Jacobian matrix 中的那個 column 全部設為 0。
接下來則是從末端往前遞迴到最後可動骨頭，把所有經過的骨頭都算入 Jacobian matrix 即可。

```
for (int i = 0; i < bone_num; i++) {

    int jacobianCount = i * 3;

    if (temp_bone->dofrx) {
        Eigen::Vector4d rotationAxisX = (temp_bone->rotation * Eigen::Vector4d(1, 0, 0, 0)).normalized();
        Jacobian.col(jacobianCount + 0) = rotationAxisX.cross3(target_pos - temp_bone->start_position);
    }
    if (temp_bone->dofry) {
        Eigen::Vector4d rotationAxisY = (temp_bone->rotation * Eigen::Vector4d(0, 1, 0, 0)).normalized();
        Jacobian.col(jacobianCount + 1) = rotationAxisY.cross3(target_pos - temp_bone->start_position);
    }
    if (temp_bone->dofrz) {
        Eigen::Vector4d rotationAxisZ = (temp_bone->rotation * Eigen::Vector4d(0, 0, 1, 0)).normalized();
        Jacobian.col(jacobianCount + 2) = rotationAxisZ.cross3(target_pos - temp_bone->start_position);
    }

    temp_bone = temp_bone->parent;
}
```

下一個步驟則是取得新的角度，新的角度來自於一步步逼近終點，逼近的方式則是使用 Pseudo Inverse 的方式，得到所有關節所有角度的 partial 值，並且將這些 partial 都乘上一個小小的步階，就可以得到我們要的 delta theta：

```
Eigen::VectorXd deltatheta = step * pseudoInverseLinearSolver(Jacobian, desiredVector);
```

將此 delta theta 加到每個關節上，並且再更新一次全新的 bone，就可以得到一個新的姿勢。更新姿勢的部分在下一次 iteration 剛開始才會計算。

```
for (int i = 0; i < bone_num; i++) {
    int deltaThetaNum = i * 3;
    int bone_idx = temp_bone->idx;

   Eigen::Vector4d deltaRotation(deltatheta[deltaThetaNum + 0] * (180 / M_PI),
                                 deltatheta[deltaThetaNum + 1] * (180 / M_PI),
                                 deltatheta[deltaThetaNum + 2] * (180 / M_PI), 0);

   posture.bone_rotations[bone_idx] += deltaRotation;
   temp_bone = temp_bone->parent;
}
```

若得到的新的角度所算出來的新的末端，與終點的距離並未在誤差值內，則繼續重複以上步驟，持續逼近。

1. 實作 Pseudo Inverse

Inverse Jacobian 不能直接計算是礙於 Jacobian 的特性，row 跟 column 數量不一定一樣，因此才使用 Pseudo Inverse。而 Pseudo Inverse 只是眾多算出 Inverse Jacobian 方式中的一種而已。

$$V = J\dot{\theta}$$

$$J^T V = J^T J\dot{\theta}$$

$$(J^T J)^{-1} J^T V = (J^T J)^{-1} J^T J\dot{\theta}$$

$$\Big\downarrow \quad J^{+} = (J^T J)^{-1} J^T$$

$$J^{+} V = \dot{\theta}$$

要注意此方法的每個 column 都必須是 linearly independent 才能使用，在本次實作中，我們的角度是一個 column 一個 column 存的，因此每個 column 不可能都獨立。但是每個 Row 由於分別代表 xyz，三者確實無任何相關性，因此我們需要改良一下 Pseudo Inverse 的算法：

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

在實作中如果知道此公式的話，剩下的就相當簡單了：

```
Eigen::MatrixX4d pseudoInverseJacobian = Jacobian.transpose() * ((Jacobian * Jacobian.transpose())).inverse();
Eigen::VectorXd solution = pseudoInverseJacobian * target;

return solution;
```

但是在我們的實作中，由於 vector 都是以四個 element 的形式儲存，因此在 code 中的 Jacobian matrix 中也會有四個 row，並且最後一個 row 的所有值都是 0。如此一來便不符合每一個 row 都要 linearly independent（有任何一個 row 全部都是 0 就代表不 independent），在計算時必須把最後一行拿掉：

```
Eigen::Matrix3Xd newJacobian = Jacobian.topRows(3);
Eigen::Vector3d newTarget = target.head<3>();

Eigen::MatrixX3d pseudoInverseJacobian =
    newJacobian.transpose() * ((newJacobian * newJacobian.transpose())).inverse();

Eigen::VectorXd solution = pseudoInverseJacobian * newTarget;

return solution;
```

五、 Bonus

Stable 問題：

雖然不知道用意是什麼，但是要知道會不會 stable，只需要在末端與終點位置的誤差值小於 epsilon 時 return true，反之則是在 iterate 次數抵達 max_iteration 前都沒有碰到終點時 return false。

六、 難點（疑問）與解法

本次實作中，最大難點：

Debug 過程相當不順利。

在實作其他類型的作業時，只需要一步一步將過程全部印出來，就能很快找到從哪個步驟開始出錯，但這兩次的作業，還有以後如果還有相似的 code 要打的話，這種問題只會愈來愈嚴重。唯一能做的就只有不斷重複檢查，物理方程式有沒有打錯，除此之外就沒有更好的辦法了。

七、 結語

這幾次的作業至少在網路上都還找得到答案，實在很擔心未來繼續選擇這方面的道路，在碰壁時沒有任何人可以告訴我正確答案，會讓一直無法成功的絕望感愈來愈強烈，只能更加注意自己的細心程度，不要像這次一樣 Matrix3Xd 打成 MatrixX3d，所謂魔鬼藏在細節哩，只有減少這種糟糕的失誤，才能讓未來的自己更輕鬆。