Introduction to Computer Animation HW3 Report

資工系 0416303 楊博凱

- Introduction/Motivation

這次作業我們要實作的也是模擬角色的動作，與第二次作業不同的地方在於第二次作業採用的方法是 Forward Kinematics，而這次我們使用的是 Inverse Kinematics。作業中我們會透過已經有的 Helper Function 幫助我們實作 Forward Kinematics 以算出 Skeleton Pose，並運用 Rotation matrix, Axis, Euler angle 算出我們所需要的 Jacobian Matrix。透過 Pseudo Inverse 我們可以求出 orientation 的變化量，進而更新每個 Bone 的旋轉角度，以模擬出整個人型，使手臂的 End position 更加接近 Target Position。

- Fundamentals
  - Iterative IK

Inverse Kinematics 的目標是要計算出會影響 end effector 的 joint DOFs，使得 end effector 可以達到我們所設定的 target position。換句話說，Forward Kinematics 是給定角度之後，我們要算出到達的點；反之，Inverse Kinematics 則是先給定我們要到達的點，在算出應該要變化的角度。由於我們希望能一步一步到達 target position，因此需要透過不斷計算與修正，調整 orientation 使得 end effector 接近 target position。所以我們採用 iterative 的方式計算，透過 step_每次小小的修正，最後到達目標。

- effects of step:

若 step 太大收斂快，但是相對不准；而 step 小，則收斂慢，然而相對比較準

- distance_epsilon:

此距離是 target position 與 end effector 的距離的誤差可以接受的範圍。若設的太大，兩者距離會比較遠，若設的太小，容易使 end effector 一直要計算，因此比較慎選適合的 epsilon。

  - Jacobian Matrix

Jacobian Matrix 可以表現為一個多變數 vector function 的最佳線性逼近。在我們的作業中，他代表了我們要計算的系統的 partial derivatives。換句話說，他定義了 end effector 如何相對的改變 instantaneous changes。

- Pseudo Inverse

  在算出上面的 Jacobian Matrix 之後，我們知道 end effector 如何影響各個角度在 x axis, y axis, z axis 的關係，而為了求出 orientation 的變化量，我們需要計算出 Jacobian Matrix 的 inverse matrix 並將其乘上 V (target 與 end effector 的距離)。然而計算 Jacobian 的 inverse 非常困難，雖然我們可以用 tranpose 代替，然而我們選擇使用的方法是 pseudo inverse，計算出 J+來取代比較難算的 inverse。

- Orientation

  我們可以得知每個 bone 的旋轉角度 (以 Euler angle 的形式表示)，然而為了使 end effector 到達 target position，我們需要改變這些 orientation，藉由小的 step 一步步算出我們最後的 orientation。

Implementation

- Iterative IK

  我們要藉由 Iterative IK 計算出 orientation 的改變量，使得 end effector 可以一步步改變 original_whole_body_joint_pos6d 裡面的 angular vector 而達到我們所設定的 target position。而為了得到每個 bone 的 start position 以及 end position 我們必須運用 fk_solver_->ComputeSkeletonPose( ) 算出 skeleton pose。

  - Jacobian Matrix

  為了求出 Jacobian Matrix，我們需要從 skeleton pose 的 bone 中取的該 bone 的 rotation matrix，將其轉換為 x axis, y axis, z axis 的 orientation，並個別 normalize 後放到 Jacobian Matrix 相對應的 column 中。

```
//  **********************************
//      Initialize Jacobian Matrix
//  **********************************

math::MatrixN_t JacobianMat(3, 3 * (end_bone_idx - (start_bone_idx - 4) + 1));
JacobianMat.setConstant(0);

//  **********************************
//      Calculate Jacobian Matrix
//  **********************************

for (int j = start_bone_idx - 4; j <= end_bone_idx; ++j){
    int colIndex = (j - start_bone_idx + 4) * 3;
    int index = j;
    if (j < start_bone_idx - 1) //  lower back to start bone
        index = j - 10;

    math::Vector3d_t e = pose[end_bone_idx].end_pos();
    math::Vector3d_t ri = pose[index].start_pos();
    math::Vector3d_t dist = e - ri;

    //  Calculate X Axis
    JacobianMat.col(colIndex) = pose[index].rotation().col(0).normalized().cross(dist);
    //  Calculate Y Axis
    JacobianMat.col(colIndex + 1) = pose[index].rotation().col(1).normalized().cross(dist);
    //  Calculate Z Axis
    JacobianMat.col(colIndex + 2) = pose[index].rotation().col(2).normalized().cross(dist);
}
```

- Pseudo Inverse

在算出上面的 Jacobian Matrix 之後，我們需要求出 orientation 的變化量。而我們做的事情就是運用 pseudo inverse，計算出 J+並將其乘上 target position 減掉 end effector 的值。由於我自己發現投影片上的計算 J+的方法 (J+ = (JT * J).inverse * JT) 總是會使 inverse 算出來的值非常小，而讓手臂不見。與很多同學討論後他們也有這種情況，因此我決定使用在網路上查到別人計算 J+的方法: J+ = JT * (J * JT).inverse

```
math::VectorNd_t thetaDot = linear_system_solver_.get()->Solve(JacobianMat, target_pos - pose[end_bone_idx].end_pos());
math::VectorNd_t returnVec(Coef_mat.transpose() * (Coef_mat * Coef_mat.transpose()).inverse() * desired_vector);
```

- Orientation

在得到 orientation 的變化量之後，我們可以將 delta x, delta y, delta z 轉換為 Euler angle 加上原來的 angular vector，得到新的 orientation。

```
math::VectorNd_t thetaDot = linear_system_solver_.get()->Solve(JacobianMat, target_pos - pose[end_bone_idx].end_pos());
for (int i = start_bone_idx; i <= end_bone_idx; i++){
    Eigen::Matrix3d m;
    m = Eigen::AngleAxisd(thetaDot[(i - start_bone_idx + 4) * 3 + 0] * M_PI, Eigen::Vector3d::UnitX())
        * Eigen::AngleAxisd(thetaDot[(i - start_bone_idx + 4) * 3 + 1] * M_PI, Eigen::Vector3d::UnitY())
        * Eigen::AngleAxisd(thetaDot[(i - start_bone_idx + 4) * 3 + 2] * M_PI, Eigen::Vector3d::UnitZ());

    math::Vector3d_t delta;
    delta = math::ComputeEulerAngleXyz(m);
    ans[i].set_angular_vector(ans[i].angular_vector() + step_ * delta);
}
```

• Result & Discussion

○ Problems Encountered:

1. 一開始在計算 $\mathbf{a}_i$ (unit length rotation axis in world space)的時候，不知道要使用的是哪一個 vector。自己嘗試過 pose 裡面的 rotation matrix 也嘗試過 skeleton 裡面的 axis()，更嘗試過 angular vector。最後發現是 rotation matrix，然而需要將 x axis, y axis, z axis 的角度分別求出來。
2. 因為 1 的問題，造成自己一直以為 Jacobian Matrix 中一個骨頭的角度對應的是一個 column，後來在問過教授之後才知道，原來是決定於該骨頭的自由度。因此 Jacobian Matrix 的很多 column 很可能是對應同一個骨頭的 rotation matrix。
3. 因為問題 1 與 2 的問題，因此自己 return 回來的 orientation 的變化量一直是使用錯誤的方式在更新 angular vector。
4. 計算 pseudo inverse 的時候發現只要遇到 inverse()就會產生負的無窮大，導致手會一直不見，在更新傳入的 Jacobian Matrix 之後，也更新 J+的算法才的使得 end effector 能成功靠近 target。

- Iterative Inverse Kinematics:
  1. 對於 step 的影響，我發現若 step 太大收斂快，但是相對不准；而 step 小，則收斂慢，然而相對比較準。
  2. 對於 distance epsilon，則是會因設的太大，而使 target position 最後與 end effector 的距離會比較遠。若設的太小，反而容易使 end effector 一直要計算，因此比較慎選適合的 epsilon。
- Jacobian Matrix:
  1. 由於 Jacobian 的性質，我們可以知道每個角度都會受到其 parent 角度的影響，若我們從 arm (id = 25)開始計算 Jacobian matrix，算出的角度會比較奇怪，而從比較前面的骨頭 lowerback (id = 11)開始計算 Jacobian Matrix 雖然計算量比較大，然而得到的角度會比較正確，也比較符合正確的轉動方式。

Conclusion

- 我們要拿做完 FK 之後的 pose 的 rotation matrix，將其乘以(1,0,0), (0,1,0), (0,0,1)得到相對於 x, y, z 軸的旋轉角度，以得到轉軸。
- 若 step 太大收斂快，但是相對不准；而 step 小，則收斂慢，然而相對比較準。所以藉由 Small increments 得到的答案會比較正確。
- distance epsilon 不可以設的太大造成結果偏差太大，也不可以設的太小造成計算太繁複。
- 由於 Jacobian 不一定是方陣，而計算 inverse 會變得值很醜，而 Jacobian Transpose 方法又太簡單，因此我們採用 Pseudo Inverse 來取代真正的 inverse。而我自己發現 J+ = JT * (J * JT).inverse 的計算方法比 J+ = (JT * J).inverse * JT 更正確。