REPORT – HW3

Introduction/Motivation:

這次作業是對所提供 skeleton 實作 Inverse Kinematics。這次的目的是去更新 posture.bone_rotations 讓 skeleton 在自己的範圍內的 end_bone 可以碰到球。

Fundamentals:

Jacobian Matrix: 求解 IK 問題的一種工具，能夠描述末端執行器在關節空間中的運動對關節角度的變化的影響。在 IK，用於計算末端執行器在一定目標位置下的關節角度變化量，以使其到達目標位置。
Pseudo Inverse: 為了求處方向的變化量，計算 Jacobian Matrix 的 pseudo inverse 並將其應用於目標向量，解決計算 Jacobian 的 inverse 的困難。
Orientation: 為了 end_bone 到達 target_pos 需要更新 posture.bone_rotation，使用小的 step 計算出最後的 posture.bone_rotation。

Implementation:

Pseudo Inverse:
對 Jacobian Matrix 使用 SVD 應用在 target_pos 找到 min(| jacobian * deltatheta - target |)的 deltatheta。

```
deltatheta = Jacobian.bdcSvd(Eigen::ComputeThinU | Eigen::ComputeThinV).solve(target);
```

Inverse Kinematics:
存 end_bone 到 start_bone 全部的 bone，已 child 到 parent 的方式去找到 root，如果沒見到 start_bone 就需要從 start_bone 到 root 全部儲存。計算總共 bones 需要用來做 IK。

```
bool visited = false;
while (current != root_bone) {
    boneList.push_back(current);
    if (current == start_bone) {
        visited = true;
        break;
    }
    current = current->parent;
}
if (!visited) {
    while (current != root_bone) {
        boneList.push_back(current);
        current = current->parent;
    }
}
bone_num = boneList.size();
```

計算 arm vectors，檢查現在所處理的 bone 的有沒有 DoF 分別為 x, y, z 並去計算去計算 Jacobian columns，使用$\mathbf{J_1} = \boldsymbol{a_1} \times (\boldsymbol{p} - \boldsymbol{r_1})$公式去計算對應的 x, y, z，$(\boldsymbol{p} - \boldsymbol{r_1})$是計算算 arm 得公式。

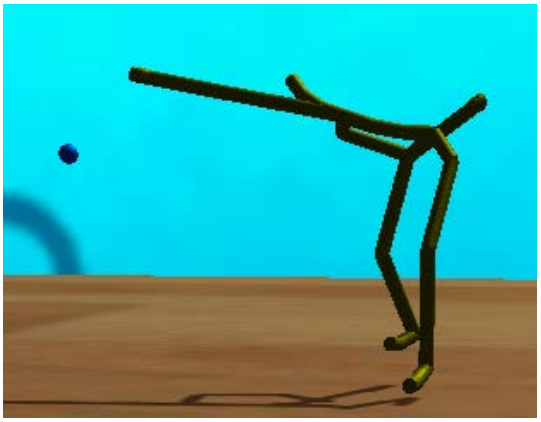
```
Jacobian.setZero();
for (long long i = 0; i < bone_num; i++) {
    Eigen::Vector3d arm = end_bone->end_position.head<3>() - boneList[i]->start_position.head<3>();
    Eigen::Affine3d rotation = boneList[i]->rotation;
    if (boneList[i]->dofrx) {
        Eigen::Vector3d unit_rotation = rotation.matrix().col(0).head<3>();  // x
        Eigen::Vector3d J = unit_rotation.cross(arm);
        Jacobian.col(3 * i) = Eigen::Vector4d(J[0], J[1], J[2], 0);
    }
    if (boneList[i]->dofry) {
        Eigen::Vector3d unit_rotation = rotation.matrix().col(1).head<3>();  // y
        Eigen::Vector3d J = unit_rotation.cross(arm);
        Jacobian.col(3 * i + 1) = Eigen::Vector4d(J[0], J[1], J[2], 0);
    }
    if (boneList[i]->dofrz) {
        Eigen::Vector3d unit_rotation = rotation.matrix().col(2).head<3>();  // z
        Eigen::Vector3d J = unit_rotation.cross(arm);
        Jacobian.col(3 * i + 2) = Eigen::Vector4d(J[0], J[1], J[2], 0);
    }
}
```

使用所算出來的 deltatheta 轉成 degree 去更新 posture.bone_rotation

```
acclaim::Bone curr = *boneList[i];
Eigen::Vector3d delta = deltatheta.segment(i * 3, 3);
posture.bone_rotations[curr.idx] += util::toDegree(Eigen::Vector4d(delta[0], delta[1], delta[2], 0));
```

Result and Discussion:



對於 step 的影響，使用太大的 step 會收斂快，但結果相對不准，使用小的 step 收斂比較慢，結果會比較准。
對於 epsilon 的影響，如果設太大的 epsilon 會導致最後 end_bone 跟 target_pos 距離比較遠，如果設太小的 epsilon 會導致一直要去計算，需要比較多的時間和成本。所以選擇適當的 epsilon。
結果可以看到 end_bone 有碰到球，但因為沒有限制 bone rotation 導致 target 超出能碰的範圍內 skeleton 的身體會有亂動的情形。
Least square solver 有很多種方法，比如 Gaussian Elimination、Cholesky decomposition、QR method、Singular value decomposition。前兩種比較適合給系統矩陣式正定的，如果對精度和穩定性要求較高適合使用後兩種。
在作業中需要計算 Matrix 的 Pseudo Inverse 所以 SVD 更適合。
Deltatheta 是一個 dynamic vector，在實作上很多問題。
計算 Jacobian columns 的部分，因為要使用 cross 所以計算過程都要以 Vector3d 來計算，使用 Vector4d 會出錯。
實作 bonus 時設定 bool stable 的位置很重要，和還原 posture 的位置。
有嘗試修改一些 constant 變數，發現 max_iteration 也不能設太小。

Bonus (Optional):

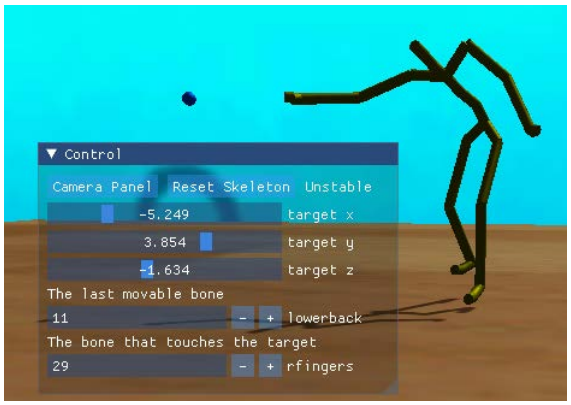
增加分為 x, y, z rotation 的限制，增加 bool stable 來檢查超出範圍的情況且利用 stable 來還原 posture，解決 skeleton 亂動的情形。

```
bool stable;
for (int iter = 0; iter < max_iteration; ++iter) {
    stable = true;

if (posture.bone_rotations[curr.idx][0] < curr.rxmin) {
    stable = false;
    posture.bone_rotations[curr.idx][0] = curr.rxmin;
} else if (posture.bone_rotations[curr.idx][0] > curr.rxmax) {
    stable = false;
    posture.bone_rotations[curr.idx][0] = curr.rxmax;
}
if (posture.bone_rotations[curr.idx][1] < curr.rymin) {
    stable = false;
    posture.bone_rotations[curr.idx][1] = curr.rymin;
} else if (posture.bone_rotations[curr.idx][1] > curr.rymax) {
    stable = false;
    posture.bone_rotations[curr.idx][1] = curr.rymax;
}

if (posture.bone_rotations[curr.idx][2] < curr.rzmin) {
    stable = false;
    posture.bone_rotations[curr.idx][2] = curr.rzmin;
} else if (posture.bone_rotations[curr.idx][2] > curr.rzmax) {
    stable = false;
    posture.bone_rotations[curr.idx][2] = curr.rzmax;
}
```

```
if (!stable) {
    posture = original_posture;
    forwardSolver(posture, root_bone);
}
return stable;
```



Conclusion:

這次作業主要目的是實作 Inverse Kinematics。在每一步計算都需要細節，max_iteration、epsilon、step 要設適當的變數。在計算 deltatheta 的過程，Jacobian Matrix 使用 Pseudo Inverse 來取代真正的 Inverse。另外，在全部計算過程需要注意是使用 Vector3d 或是 Vector4d 避免發生錯誤。