# 1. Introduction

這份報告的引言部分介紹了前向運動學和逆向運動學的基本原理。前向運動學負責將關節運動數據轉換成實際的空間坐標，而逆向運動學則用於根據目標位置推算關節角度。這兩種技術對於計算機動畫和特效創作至關重要，它們能有效模擬和控制多關節物體的動作，例如人類和機器人。

# 2. Fundamentals:

- Forward Kinematics:

  為更新所有 bone 的 translation and Rotation，大概原理是讀出 root 的 translation and Rotation 把他的末端接到下一個 B 的初始值，在算下一個的 translation and Rotation，直到 traverse 每個 Bone。

- Local Coordinate:

  每個 bone 都有自己的 Local Coordinate，此為座標戲中相對於某個特定對象的座標系統，相較於 Global Coordinate 更方便使用者來操作不同的物件，不需要考慮其他物體對當前操作對象的影響。

- Global Coordinate:

  整個場景的坐標系，他提供一個統一參考的框架，可以對對象進行位置上的比對和定位，通常為 Local Coordinate 處理完後，把處理完的結果丟到 Global Coordinate 中，確保所有動作都能在場景中表現出來。

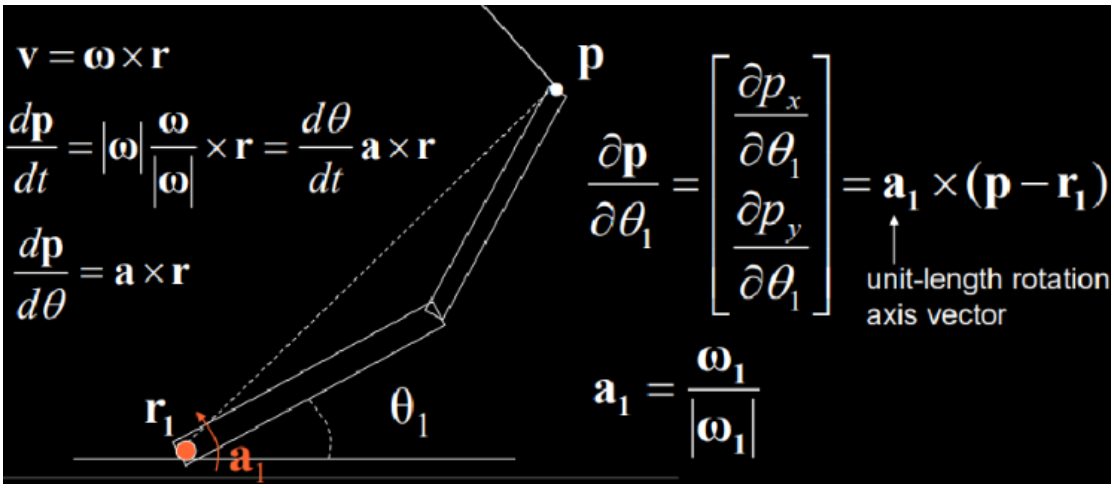
- Iterative Inverse Kinematics:

相較於 Forward Kinematics 他是 bone 改變後，依據其改變量去算 bone 的 end point ,而 code 中是利用 step 來一步步將 bone 的 end point 修正至接近 target 為 iterative。

- Pseudo Inverse of the Jacobian:

Jacobian Matrix 是表示多變數向量函數的最佳線性逼近，我們算出 Jacobian 後，要將他轉成 inverse 並乘上向量來得到 bone 的旋轉角度的變化，這邊用 jacabian 的 psedo inverse 因為不是每個都是可逆的。

以下為參考的課本講義



# 3. Implementation:

- Forward Kinematics:

將一開始從 root 開始，設置骨骼的頭尾端和 Rotation 值，接下來透過 DFS 去 traverse 剩下的 Bone。

```
//      then x degrees along x - axis
bool visited[31] = {};
bone->start_position = posture.bone_translations[0];
bone->rotation = bone->rot_parent_current;
bone->rotation = util::rotateDegreeZYX(posture.bone_rotations[bone->idx]);
bone->end_position = bone->start_position;
bone->end_position += bone->rotation * (bone->dir.normalized() * bone->length);
visited[0] = true;

DFS(posture, bone->child, visited);
```

Traverse 的過程中，標記是否訪問過，更新整個的骨骼結構和全局位置和旋轉狀態，其中跟根結點不一樣的是 rotaion 要多乘上 parent Rotation，骨骼的選轉是由父骨骼的旋轉與當前相對於父骨骼的選轉的成績決定的，末端位置根據起始位置，旋轉和方向向量計算得出。

會先遍歷子骨骼再去遍歷兄弟骨骼，因為是 DFS

```
void DFS(const acclaim::Posture& posture, acclaim::Bone* bone, bool visited[]) {
    visited[bone->idx] = true;

    if (bone->parent != nullptr) {
        bone->start_position = bone->parent->end_position;
        bone->rotation = bone->parent->rotation * bone->rot_parent_current;
        bone->rotation *= util::rotateDegreeZYX(posture.bone_rotations[bone->idx]);
        bone->end_position = bone->start_position + bone->rotation * (bone->dir.normalized() * bone->length);
    }

    if (bone->child != nullptr && !visited[bone->child->idx]) {
        DFS(posture, bone->child, visited);
    }

    acclaim::Bone* tmp = bone->sibling;
    while (tmp != nullptr) {
        if (!visited[tmp->idx]) {
            DFS(posture, tmp, visited);
        }
        tmp = tmp->sibling;
    }
```

- PseudoInverseLinearSolver:

他說可以用線代的 SVD，我就查一下 Eigen 裡面的功能，發現可以直接套用 JacobiSVD，將結果的 Jacovian pseudo inverse 轉成 minimum norm solution，將其 return。

```
Eigen::VectorXd pseudoInverseLinearSolver(const Eigen::Matrix4Xd& Jacobian, const Eigen::Vector4d& target) {
    // TODO (find x which min(| jacobian * x - target |))
    // Hint:
    //   1. Linear algebra - least squares solution
    //   2. https://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_inverse#Construction
    // Note:
    //   1. SVD or other pseudo-inverse method is useful
    //   2. Some of them have some limitation, if you use that method you should check it.
    Eigen::VectorXd deltatheta(Jacobian.cols());
    Eigen::JacobiSVD<Eigen::Matrix4Xd> svdSolver(Jacobian, Eigen::ComputeThinU | Eigen::ComputeThinV);
    deltatheta = svdSolver.solve(target);



    return deltatheta;
}
```

- Inverse Kinematics:

Armvector 是當前關節到末端的向量，計算兩者的位置差，用於後續的計算，rotationMatrix 為當前骨骼的選轉矩陣，初始劃一個三維向量 axis，看 dofr?來決定能不能處理當前的 bone。

再來計算旋轉後的軸向量，再將 3D 向量轉為 4D 並 normalize，計算 Jacobomatrix 的 column，賦值到 Jacobi matrix。

```
for (long long i = 0; i < bone_num; i++) {
    Eigen::Vector4d armVector = *jointChains[chainIdx][0] - *jointChains[chainIdx][i + 1];
    auto rotationMatrix = boneChains[chainIdx][i]->rotation;
    for (int dofIndex = 0; dofIndex < 3; dofIndex++) {
        if ((dofIndex == 0 && boneChains[chainIdx][i]->dofrx) ||
            (dofIndex == 1 && boneChains[chainIdx][i]->dofry) ||
            (dofIndex == 2 && boneChains[chainIdx][i]->dofrz)) {
            Eigen::Vector3d axis = Eigen::Vector3d::Zero();
            axis[dofIndex] = 1;

            Eigen::Vector3d rotatedAxis = rotationMatrix.rotation() * axis;
            Eigen::Vector4d dofCol(rotatedAxis.x(), rotatedAxis.y(), rotatedAxis.z(), 0.0);
            dofCol.normalize();
            Eigen::Vector3d jacobianCol = rotatedAxis.cross(armVector.head<3>());
            Jacobian.block<3, 1>(0, 3 * i + dofIndex) = jacobianCol;
        }
    }
}
```

遍歷所有骨骼，檢查可不可以旋轉，如果可以用 deltatheta 來更新角度，更新的公式為(deltatheta[i * 3 + j] * 180 / PI)，要把弧度換成角度。

BONUS:

用 std::clamp 來對更新的角度進行限制，其中 currentBone->r?min/r?max 為其邊界值，這會導致其更穩定。

```
//   1. You cannot ignore rotation limit of the bone.
double PI = 3.14159265358979323846;
for (long long i = 0; i < bone_num; i++) {
    acclaim::Bone* currentBone = boneChains[chainIdx][i];
    for (int j = 0; j < 3; j++) {
        bool isDofActive = false;
    if ((j == 0 && currentBone->dofrx) ||
        (j == 1 && currentBone->dofry) ||
        (j == 2 && currentBone->dofrz)) {
        isDofActive = true;
    }

    if (isDofActive) {
        posture.bone_rotations[currentBone->idx][j] += ((deltatheta[i * 3 + j]*180)/PI);
    }


    }
    posture.bone_rotations[currentBone->idx][0] =
        std::clamp(posture.bone_rotations[currentBone->idx][0], (double)currentBone->rxmin,
                   (double)currentBone->rxmax);
    posture.bone_rotations[currentBone->idx][1] =
        std::clamp(posture.bone_rotations[currentBone->idx][1], (double)currentBone->rymin,
                   (double)currentBone->rymax);
    posture.bone_rotations[currentBone->idx][2] =
        std::clamp(posture.bone_rotations[currentBone->idx][2], (double)currentBone->rzmin,
                   (double)currentBone->rzmax);


}
```

# 4. Result and discussion:

- How different step and epsilon affect the result:

實作可發現，epsilon 越小那個點點越接近 target，因為在預設的程式中是看 target 和 endbone 的距離小於 epsilon 時停止，而 step 不管太大 or 太小都會導致其 unstable 不能移動，較小的 step 表示更穩定更精細的挑整但是收斂數度慢，較大的 step 可以加快收斂數度但是會不太穩。

- 算法的改進和優化:

雖然現在的算法表現穩定，但是遇到複雜的場景或者更動態的目標，挑整 step 值以及其他應用可以提高 IL 的實際應用價值和精確度。

# 5. Bonus:

在 IK 中，有考慮關節的限制，保證其關節旋轉的角度不超出預設的最大最小限制，保證算法實際應用的安全性和可行性。

```
}
posture.bone_rotations[currentBone->idx][0] =
    std::clamp(posture.bone_rotations[currentBone->idx][0], (double)currentBone->rxmin,
               (double)currentBone->rxmax);
posture.bone_rotations[currentBone->idx][1] =
    std::clamp(posture.bone_rotations[currentBone->idx][1], (double)currentBone->rymin,
               (double)currentBone->rymax);
posture.bone_rotations[currentBone->idx][2] =
    std::clamp(posture.bone_rotations[currentBone->idx][2], (double)currentBone->rzmin,
               (double)currentBone->rzmax);
```

# 6. Conclusion:

其實這次的作業量極大，加上現在大二修的課不夠多，對資料結構的運用還沒有到那麼熟，加上 IK 的概念有點難理解，最後就是問同學和上網找資料來慢慢解決，bonus 其實也只是優化 rotation 而已，但是，經過這次作業真的學到了很多東西。

# 7. DEMO link:

https://youtu.be/U_4G9irwLkk