HW2 REPORT

110511010 楊育陞

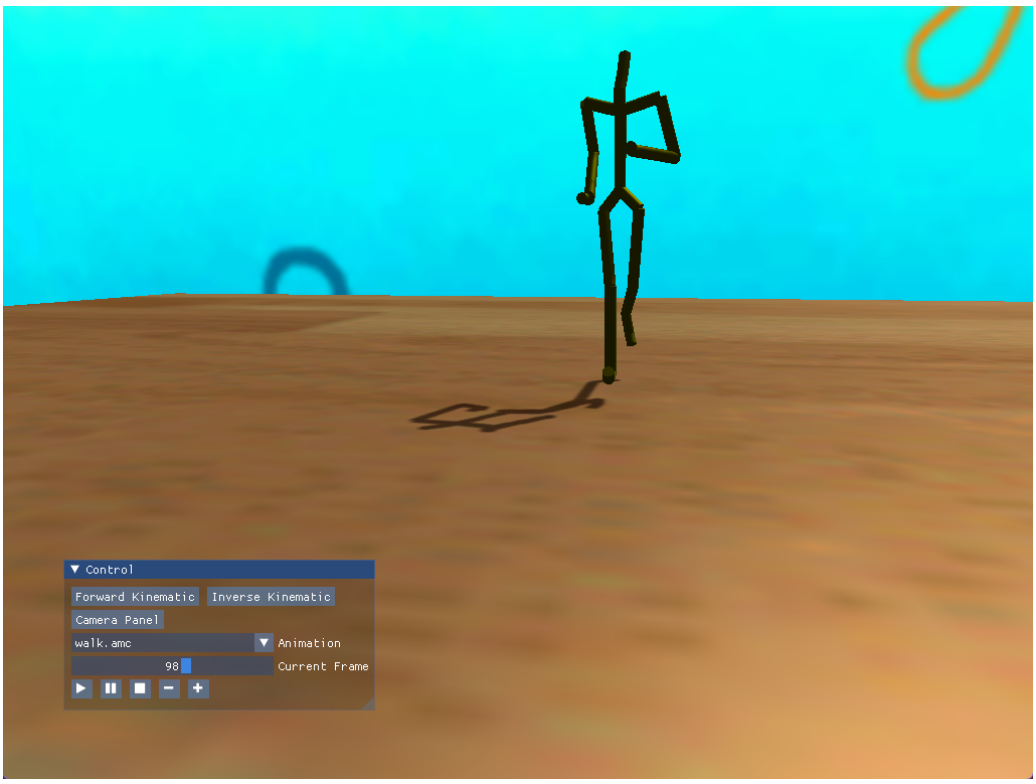
1 Introduction

In this homework, we implemented the forward kinematics (FK) and inverse kinematics (IK). Given a skeleton and the motion data, we calculated the global coordinates of the joints in the skeleton using forward kinematics. For the inverse kinematics, we implemented the inverse Jacobian method to calculate the joint angles of the skeleton given the target position of the end effectors. If the target position is not reachable, the skeleton will move back to the previous position. Also, I modified the step and epsilon values to see how they affect the results of IK.

2 Fundamentals

2.1 Structure of the Skeleton

The skeleton is a tree structure of bones. Each bone has its start position, end position, and local coordinate system. To render the skeleton, we need to get the global coordinates of the bones.



2.2 Local and Global Coordinates

Each bone has its local coordinate system. In the motion data, the rotation angles are given in the local coordinate system of the bones. So the bones rotate around their local axes based on the motion data. The global coordinate is the world coordinate system. To render the skeleton, we need to calculate the global coordinates of the bones, namely, to know the position and orientation of the bones in the world.

2.3 Forward Kinematics

The forward kinematics is to calculate the global coordinates of the bones given the rotation angles of the bones. We first calculate the global coordinates of the root bone, then calculate the global coordinates of the child bones based on the global coordinates of the parent bones. Using the notation in the slides, the global coordinates of bone i can be calculated as follows:

$$_iT =_0^{i-1} RV_{i-1} +_{i-1} T$$

Where $_0^{i-1}R$ is the rotation matrix that transform local coordinates of bone $i-1$ to the global coordinate. V_{i-1} is the local vector of bone i. These are calculated as follows:

$$_0^{i-1}R =_0^1 R \cdot_1^2 R \cdots_{i-2}^{i-1} R$$

$$V_{i-1} = \hat{V}_{i-1} l_{i-1}$$

2.4 Inverse Kinematics

The inverse kinematics is to calculate the rotation angles of the bones given the target position of the end effectors. We use the inverse Jacobian method to calculate the rotation angles. The Jacobian matrix is in the form of:

$$J = \begin{bmatrix} \frac{\partial p}{\partial \theta_1} & \frac{\partial p}{\partial \theta_2} & \cdots & \frac{\partial p}{\partial \theta_n} \end{bmatrix}$$

Each column of the Jacobian matrix is the partial derivative of the position vector p with respect to the rotation angle θ. These columns are calculated as follows:

$$\frac{\partial p}{\partial \theta_i} = a_i \times (p - r_i)$$

Where a_i is the axis of rotation of bone i, 'p' is the target position of the end effector, and r_i is the start position of the bone i.

In the inverse Jacobian method, we ratively update the rotation angles to minimize the error between the target position and the current position of the end effector. The update is as follows:

$$\theta_{i+1} = \theta_i + \Delta t J^{-1}(\theta_i) V$$

Where Δt is the step size, V is the desired vector between the target position and the current position of the end effector.

3 Implementation

3.1 Forward Kinematics

For the forward kinematics, I recursively call the ‘forwardSolver‘ function to calculate the global coordinates of the bones. The function will first call by the root bone, then call by the child bones using BFS. In each call, the function will set the start position as the end position of the parent bone. Then calculate the rotation matrix, which is the product of the parent bone's rotation matrix, the transfrom matrix from parent bone to current bone, and the rotation matrix from the motion data. Then calculate the end position of the bone by adding start position and the rotation matrix times the local vector of the bone. The code is as follows:

```
void forwardSolver(const acclaim::Posture& posture, acclaim::Bone* bone) {
    if (bone->idx == 0) {
        bone->start_position = posture.bone_translations[bone->idx];
        bone->rotation = bone->rot_parent_current * util::rotateDegreeZYX(posture.bone_rotations[bone->idx]);
        bone->end_position = posture.bone_translations[bone->idx];
    } else {
        bone->start_position = bone->parent->end_position;
        bone->rotation = bone->parent->rotation * (bone->rot_parent_current * util::rotateDegreeZYX(posture.bone_
        bone->end_position = bone->rotation * (bone->dir * bone->length) + bone->start_position;
    }
    for (acclaim::Bone* child = bone->child; child != nullptr; child = child->sibling) {
        forwardSolver(posture, child);
    }
}
```

3.2 Pseudo Inverse of Jacobian

For the pseudo inverse needed in the inverse kinematics, I implemented the ‘pseudoInverseLinearSolver‘ function using the SVD method as follows:

```
Eigen::VectorXd pseudoInverseLinearSolver(const Eigen::Matrix4Xd& Jacobian, const Eigen::Vector4d& target) {
    Eigen::VectorXd deltatheta(Jacobian.cols());
\    deltatheta.setZero();
    deltatheta = Jacobian.jacobiSvd(Eigen::ComputeThinU | Eigen::ComputeThinV).solve(target);
    return deltatheta;
}
```

3.3 Inverse Kinematics

For the inverse kinematics, I implemented the computation of the Jacobian matrix and the update of the rotation angles.

3.3.1 Jacobian Matrix

Each column of the Jacobian matrix is the partial derivative of the position vector with respect to the rotation angle. These columns are the cross product of the axis of rotation and the vector from the start position of the bone to the target position. The code is as follows:

```
for (long long i = 0; i < bone_num; i++) {
    Eigen::Matrix4d a = boneChains[chainIdx][i]->rotation.matrix();
    Eigen::Vector4d p_r = *jointChains[chainIdx][0] - *jointChains[chainIdx][i+1];
    if (boneChains[chainIdx][i]->dofrx) {
        Jacobian.col(3 * i) = a.col(0).normalized().cross3(p_r);
    }
    if (boneChains[chainIdx][i]->dofry) {
        Jacobian.col(3 * i + 1) = a.col(1).normalized().cross3(p_r);
    }
    if (boneChains[chainIdx][i]->dofrz) {
        Jacobian.col(3 * i + 2) = a.col(2).normalized().cross3(p_r);
    }
}
```

3.3.2 Update of Rotation Angles

The update of the rotation angles is done by adding the product of the step size, the pseudo inverse of the Jacobian matrix, and the desired vector on the current rotation angles. The code is as follows:

```
Eigen::VectorXd deltatheta = step * pseudoInverseLinearSolver(Jacobian, desiredVector);
for (long long i = 0; i < bone_num; i++) {
    posture.bone_rotations[boneChains[chainIdx][i]->idx] += deltatheta.segment(3 * i, 3);
    if (boneChains[chainIdx][i]->rxmax < posture.bone_rotations[boneChains[chainIdx][i]->idx](0)) {
        posture.bone_rotations[boneChains[chainIdx][i]->idx](0) = boneChains[chainIdx][i]->rxmax;
    }
    ...
}
```

Also, I check if the rotation angles exceed the limits of the bones. If so, I set the rotation angles to the limits.

4 Result and Discussion

4.1 Result

demo link: `https://youtu.be/6iYpeph2l3I`

The FK part works well. In the IK part, by adjusting the step, epsilon, and max iteration, I made the skeleton reach the target position smoothly and normally.

4.2 Effect of parameters

In the IK part, I modified the step and epsilon values to see how they affect the results of IK.

4.2.1 Step

The step is the size of the update of the rotation angles, which affects the speed of the skeleton reaching the target position. If the step is too small, the skeleton will update too slowly to reach the target position. If the step is too large, the skeleton may oscillate around the target position.

4.2.2 Epsilon

The epsilon is the threshold of the error between the target position and the current position of the end effector. If the error is less than epsilon, the skeleton will stop updating the rotation angles. The larger epsilon is, the less accurate the skeleton will reach the target position. But if epsilon is too small, it would be hard for the skeleton to fulfill the condition.

4.2.3 Max Iteration

The max iteration is the maximum number of iterations of the update of the rotation angles. The larger max iteration is, the more time the skeleton will take to reach the target position, which will make the program delayed. But if max iteration is too small, the skeleton may not reach the target position.

5 Conclusion

In this homework, I implemented the main functions of forward kinematics and inverse kinematics. I think this homework really helps me make sure I understand the concepts of these topics. Especially, the calculation of the rotation matrix and the Jacobian matrix is hard but interesting.