CA HW3 report

1. Introduction

Solve inverse kinematics by pseudo inverse Jacobian and least square solution.

2. Fundamentals

i. Jacobian matrix: Each column of is $\frac{\partial \boldsymbol{p}}{\partial \theta_i} = \boldsymbol{a}_i \times (\boldsymbol{p} - \boldsymbol{r}_i)$, where a is a rotation axis, p is the end effector, and r is the position of the joint pivot.

ii. $V = J\dot{\theta} \Leftrightarrow J^{+}V = \dot{\theta}$, where J^{+} is the pseudo inverse of Jacobian. So, we can get the change of each bone by $\theta_{k+1} = \theta_k + \Delta t \times J^{+}V$.

3. Implementation

i. Traverse bones from end to start / end to root and start to root.

```cpp
acclaim::Bone* current = end_bone;
bool stable = false;

while (current != start_bone) {
    boneList.emplace_back(current);
    bone_num++;
    current = current->parent;

    if (current == nullptr) {
        current = start_bone;
        while (current != root_bone) {
            boneList.emplace_back(current);
            bone_num++;
            current = current->parent;
        }
        break;
    }
}
if (current == start_bone)
    boneList.emplace_back(current);
```

ii. Fill each column of Jacobian by $\boldsymbol{a}_i \times (\boldsymbol{p} - \boldsymbol{r}_i)$ where the bone has DOF.

```cpp
for (long long i = 0; i < bone_num; i++) {
    auto b = boneList[i];
    auto rotation = b->rotation.matrix();
    auto rot_ = rotation.block<3, 3>(0, 0);
    Eigen::Vector4d r = end_bone->end_position - b->start_position;
    Eigen::Vector3d r_;
    r_ << r[0], r[1], r[2];
    if (b->dofrx) {
        Eigen::Vector3d v = rot_.col(0);
        auto w = v.cross(r_);
        Jacobian.col(3 * i) << w, 0;
    }
    if (b->dofry) {
        Eigen::Vector3d v = rot_.col(1);
        auto w = v.cross(r_);
        Jacobian.col(3 * i + 1) << w, 0;
    }
    if (b->dofrz) {
        Eigen::Vector3d v = rot_.col(2);
        auto w = v.cross(r_);
        Jacobian.col(3 * i + 2) << w, 0;
    }
}
```

iii. Calculate the pseudo inverse of Jacobian and solve least square.

```cpp
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());
    deltatheta.setZero();

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

    return deltatheta;
}
```

iv. Convert theta from radian to degree, update each bone’s rotation, and clamp by its limits.

```cpp
        for (long long i = 0; i < bone_num; i++) {
            for (int j = 0; j < 3; j++) {
                posture.bone_rotations[boneList[i]->idx][j] += deltatheta[3 * i + j] * 180 / util::PI;
            }
            posture.bone_rotations[boneList[i]->idx][0] = std::clamp(
                posture.bone_rotations[boneList[i]->idx][0], (double)boneList[i]->rxmin, (double)boneList[i]->rxmax);
            posture.bone_rotations[boneList[i]->idx][1] = std::clamp(
                posture.bone_rotations[boneList[i]->idx][1], (double)boneList[i]->rymin, (double)boneList[i]->rymax);
            posture.bone_rotations[boneList[i]->idx][2] = std::clamp(
                posture.bone_rotations[boneList[i]->idx][2], (double)boneList[i]->rzmin, (double)boneList[i]->rzmax);
        }

    }
```

4. Result and Discussion

   i. Smaller step size can produce a smoother animation, but it may need longer to touch the moving target.

   ii. Epsilon does not affect the result much, but larger epsilon can larger the tolerance of the inaccuracy. Somehow lower the calculation time, and creates a rougher animation.

   iii. I tried bdcSVD and jacobiSvd as the least square solver, and the results seems very similar.

5. Bonus

   i. Limit bones’ rotation by its limit.

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

6. Conclusion

This project has few codes to write, but understanding the background knowledge really cost me a lot of time. This is a challenging project, and I learned a lot.