•Introduction/Motivation

用很多關節和骨頭組成一個骨架，模擬人的行為。每個關節有不同的DOF，有不同的可活動的角度。可設定一個target的位置，我們想要讓人的某個部位去觸碰target，且可以設定從哪個關節開始可以活動，我們要盡量讓他的姿勢自然一點。

目的是熟悉inverse kinematics，給定可活動的關節(start bone)、要去觸碰target的關節(end bone)、要觸碰的目標位置(target)，然後使用IK去計算每個關節應有的角度，我們會使用iterative Jacobian的方法計算角度並更新到關節上。

• Fundamentals

1.線性代數，cartesian space轉換到joint space中間需要很多矩陣的運算。

2.數值方法，iterative Jacobian去不斷計算角度並更新到關節上就是一種數值方法。

3.物理，有很多骨頭的旋轉、歐拉角與四元數的轉換等等。

4.C++能力，要會寫C++、懂得使用物件、了解大型專案如何運作才能順利完成程式，並要熟悉新的Library。

• Implementation

1. void forwardKinematics(...)：將當前frame的關節與骨頭的資料從joint space map到cartesian space。可直接使用HW2的code。
2. Eigen::VectorXf leastSquareSolver(...)：解jacobian和target vector的leastsquare問題，也就是求出 x 令(| jacobian * x - target |)最小。我google後找到Eigen中有內建的函式可以解這個問題，所以我就直接套用，但我發現套用後編譯的時間從7秒變成30秒。
3. void inverseKinematics(...)：根據設定的start bone、end bone、target位置，利用iterative Jacobian的方式求出應該要令每個bone旋轉多少角度來讓end bone碰到target。
    1. 首先從end到start把中間經過的bone放到boneList中，且要注意不是start ~ end中間所有編號的bone都可以活動，而是end到start中有parent-child關係的才可以活動，且有兩種可能 1.start是end的ancestor node，直接end到start即可 2.start不是end的ancestor node，變成end到root，root到start。
    實作時1.end to start即可 2.traverse時因為一個node的parent只會有一個，所以用end to root再從start to root比較好，要注意不要讓root被放進去兩次，且要注意取到root的parent(是nullptr)導致後面操作時整個程式crash
    2. 計算target到end bone的endposition的向量，如果該向量的norm小於epsilon則可提早結束迭代。接著計算jacobian，有rx, ry, rz，每個軸分別對應到一個jacobian column，所以一個bone會產生出三個jacobian column。且要注意bone如果有ri的自由度的話才需要計算它對應的jacobian column，不然就讓它維持0, 0, 0即可。計算jacobian column的方式是：
    bone_ri = bone.rotation * Eigen::Vector3f::UnitI()，然後bone_ri X (p - ri)即為答案。
    bone_ri的部分我本來用(bone.rotation.vec()(0), 0, 0)當作bone_rx，且yz同樣方法去算的話，最後結果會錯，應該是因為rotation不是這樣用。最後再把jacobian和target到end bone的endposition的向量送去解leastSquare，解出的答案即為dTheta，而我在這邊讓它順便*

step，因為step代表每次迭代的delta t，而等等在更新角度時每次都要*step，所以這邊在設定時就直接先 *step來節省運算。

3. 對每個bone的eulerangle更新對應的dTheta * step上去，再將eulerangle轉成四元數，也就是rotation。

• Result and Discussion

How different step and epsilon affect the result

step代表每次迭代時更新角度公式中的的delta t，因為更新時用的是歐拉法，我們是用線性去local的模擬角度變化，而如果step太大會讓線性的逼近不準，導致最後的結果有問題。例如：在程式中除了step的其他參數、start、end、target都不動的情況下。step 0.1時可以碰到target；1.0時可以碰到target，且姿勢和0.1的差不多；2.0時可以碰到target，但姿勢和之前差滿多的了；2.5時就不能碰到target了，且還差很遠。

epsilon代表target和end effector的距離可以接受的誤差範圍，距離在誤差範圍內時即可提早停止迭代了，就相當於如果end effector伸進target為球心形成的半徑為epsilon的球中時即可停止迭代。

Touch the target or not

成功碰觸到target這件事的定義會根據我們設定的epsilon的值不同而不同。在不同的起始姿勢會影響最後能不能成功碰觸到target，有可能從碰一個target後改變起始姿勢再去碰另一個target可以碰到，而從預設的起始姿勢去碰另一個target則碰不到。例如：在程式中的參數、start、end都不動的情況下，target的第二個值先改成14可以碰到，再改成1.6也可以碰到；而如果直接用一開始的起始姿勢去碰第二個值被改成1.6的target則會碰不到。

碰不碰的到這件事也取決於IK的計算，若step設太大導致IK模擬誤差太大也會造成產生的結果沒辦法成功碰到target。例如：上面step提到的step為2.5的情況。

Least square solver

助教建議用SVD的方式去求偽逆矩陣，再去解least square problem，而我發現Eigen中就有內建的函式可以使用，且還有另一個函式可以在解出svd後解least square的問題，所以我直接使用Eigen內建的函式，但我發現套用後編譯的時間從7秒變成30秒，助教說是因為template有可能會編譯比較久。

• Bonus (Optional)

在更新每個bone的rotations前、更新bone的eulerangle後，檢查bone的eulerangle的x, y, z是不是在[bone.rimin, bone.rimax]之間(ri = rx, ry, rz)，若在範圍之間的話就不用動；若不在範圍內的話就把對應eulerangle的ri設為邊界值。

如此即可達成限制關節活動角度在範圍內的效果，可以讓模擬出的姿勢更加自然，但可以觸碰到的target位置的範圍也會變小，因為不能用一些違反人體工學的姿勢去觸碰target了，但這樣的結果就是我們想要的，我們要讓模擬的結果更合理。

• Conclusion

我對旋轉的部分滿不熟悉的，在算bone_ri一開始用錯誤的方式計算，而且boneList的部分也誤會助教的意思，取到錯誤的bone，但結果卻仍能碰到target，只是姿勢會變得很奇怪，而且加上bonus的程式碼之後反而完全碰不到target且姿勢變很奇怪。還以為是bonus的問題，花了很多時間去google卻找不到結果，詢問助教之後助教才給了我提示。

經過這次作業後，因為本來以為bonus那邊有問題，花了很多時間搜尋資料，所以有對旋轉更熟悉一些了。