Computer Animation and Special Effect HW3 Report

102062209 邱政凱

這次的作業要求我們實作的是 Inverse Kinematics。相較於上一次作業的 Forward Kinematics 是由給定的 pose 和 motion 檔來把角色的動作從檔案中的 Local Joint Space 映射到 Global 的 Cartecian Space，Inverse Kinematics 是藉由指定在場景中角色的某個點的新的 Cartecian Space 的位置，要我們算出相關的所有關節和骨頭在各自的 Joint Space 的變化。

Inverse Kinematics 實作的方式很多，一般可以分為 analytic method、Inverse-Jacobian、Optimization-based method、Example-based method。Analytic method 通常只適用於關節的結構非常簡單的時候，而 Optimization-based method 設計 Objective Function 可能會很有難度、Example-based method 需要大量的樣本資料，因此實際上我們要實作最好的選擇還是 Inverse-Jacobian 了。

Jacobian 的功用就是可以把所有關節的角速度映射到空間中的線性速度，然而如果用 analytical 的方式來算 Jacobian，關節一複雜就會很難計算，因此我們通常都採用 Geometric Approach 來計算 Jacobian。

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

算出 Jacobian 之後就可以用以上的式子算出每個 bone 在 Joint-space 的全新位置。Jacobian 裡面放的就是每個相關關節在 Joint-space 旋轉時映射到 End-Effector 的 Cartecian space 變量。而 Inverse Jacobian 就可以把 End-Effector 的 Cartecian space 變量映射到每個關節中的 Joint-space 變量。其中 V 就是你希望 End-Effector 在 Cartecian space 中變化的方向。(不過因為通常 Jacobian 不會是正方形矩陣，而且有可能有無法找到對應的映射關係，所以我們通常是用 Psuedo Inverse Jacobian 來做)。

我的實作方式就是先算出 motion 的 FK-Solution。一開始先從 End-Bone 開始，找出這個 Bone 上面 dof==1 的所有旋轉軸，用對應的 rotation matrix 把單位旋轉軸((1,0,0)、(0,1,0)、(0,0,1))從 local space 轉換到 global space，接著算出這個 Axis 和這個 bone 的 start_position 的位置和 end-effector 之間距離的 Cross-Product，並且將它放入 Jacobian 之中。對這個 Bone，把每個 dof==1 的旋轉軸都做一次之後，把當前的 Bone 設成原本 Bone 的 Parent Bone，然後重複以上過程直到做到 Start_bone 為止。

算出 Jcobian 之後，照助教提供的 Appendix 的方法使用 Eigen 就可以算出 pseudo Inverse- jacobian。接著我們把 destination(球球)的位置和 end-effector 的位置相減，可以算出我們希望 Cartecian space 的變量(V)，把這個差丟進 jacobiansvd.solve()就可以算出每個對應關節的所有旋轉軸的新的 Joint space 的

值。接著再次遍歷所有相關關節，然後把對應旋轉軸的新的 Joint space 位置帶入。最後就用 SetJointSpatialPos()把 motion 檔用新算出來的值更新就完成了。

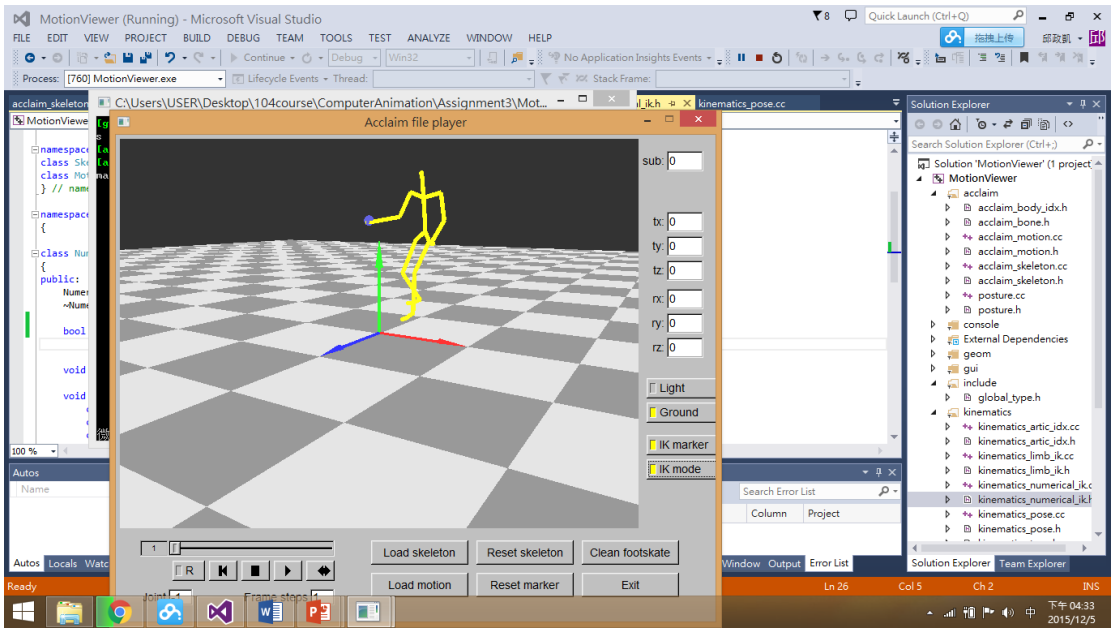
接著還有一些 RealTime 的 issue。因為如果只做到上面這樣的話執行速度其實不太理想，每次改變球的位置會有一點延遲。所以我用了一些 Trick 希望可以改善這樣的狀況。

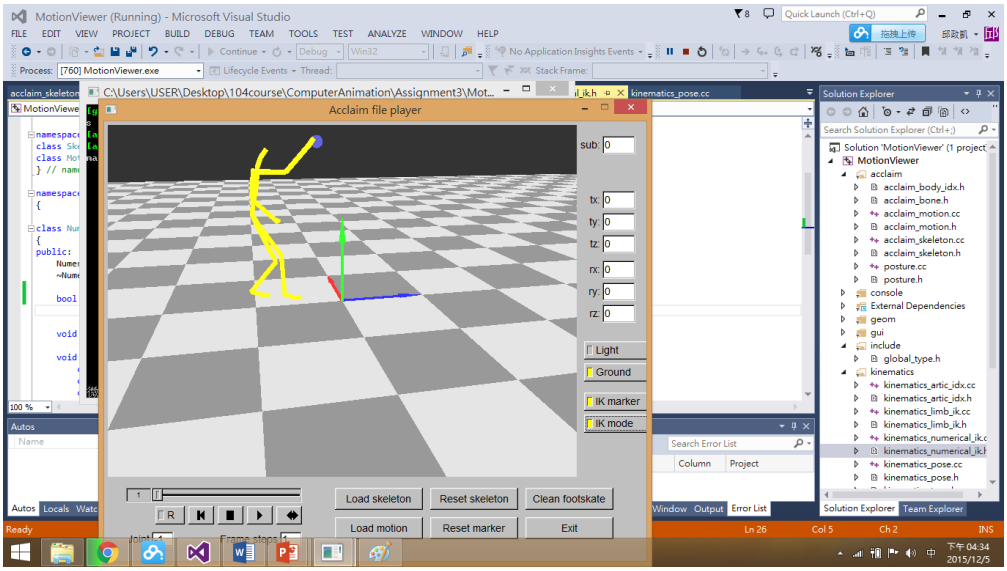
我的 RealTime IK 是可以開啟跟關閉的，助教只需要進到 kinematics_numerial_ik.h 裡面把 isRealTime 設成 TRUE 就可以看作加速的結果。一開始我是把 RealTime IK 關掉的。

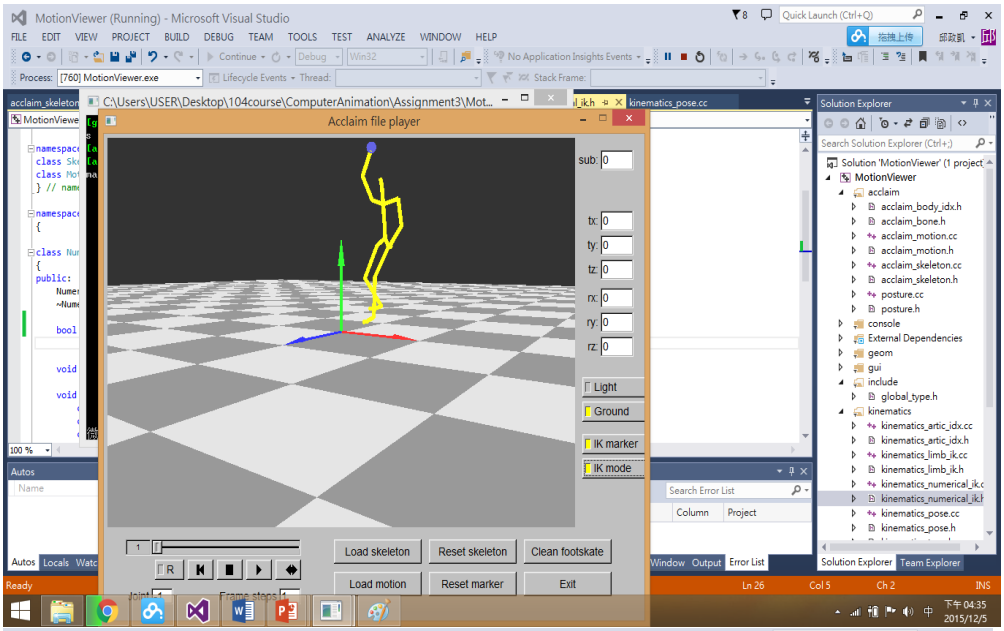
我的加速方法是每次計算出新的角度變化時，去算新的 end-effector 位置和 destination 的位置間的距離 A，以及新的 end-effector 的位置和舊的 end-effector 的位置間的距離 B。直接取用 A/B 當作一個 ratio，然後把一開始算出來的每個關節的 Joint Space 變化量乘上這個 ratio 後重新帶入，再次算出新的 motion。這樣子的加速方法有些利弊。利的地方在於速度會獲得很大的提升，弊的地方在於，這畢竟只是一種近似的方式，這樣算出來的結果通常不會是一次就讓 end effector 直接跑到 destination 的位置而有時候會跑超過，原因是因為都是用同樣的角速度去幫關節做旋轉，無法在 end-effector 靠近 destination 的過程中藉由不斷重複計算 psuedo inverse jacobian 來微調旋轉的過程，所以有時這個加速的 real time ik 的過程關節會在 destination 附近飛來飛去好幾次才穩定。(如果把 Real Time IK 關掉就不會有這種問題，旋轉的過程都會很滑順自然)

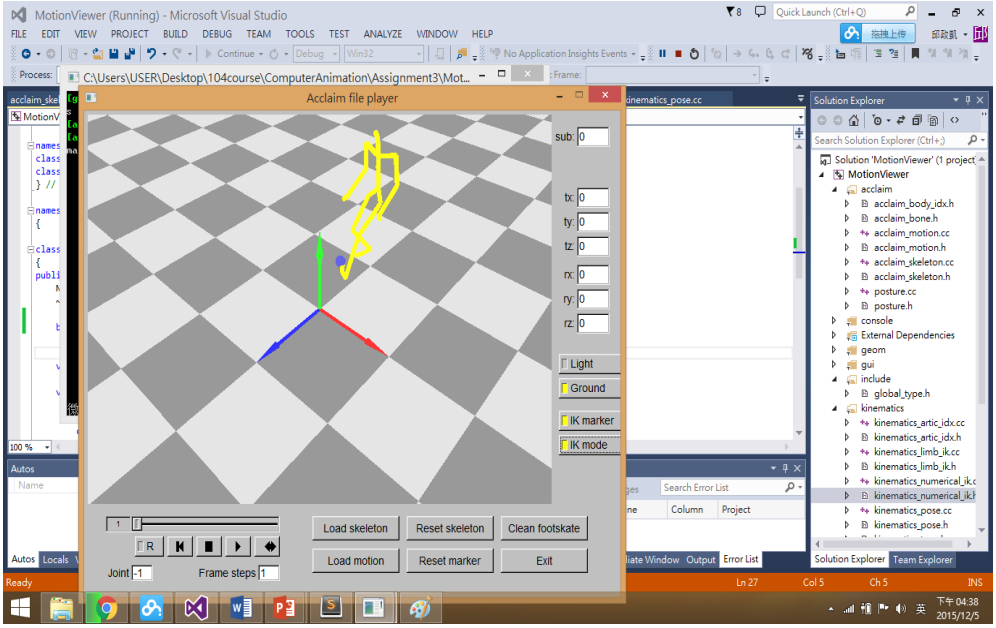
最後就是一些保護措施。每次計算 IK 時我都會先算 start_bone 的 start_position 的距離跟球的距離，如果此距離大於 start_bone 到 end_bone 之間全部 bone 加起來的長度的話就直接不去計算 ik 以免系統爛掉。

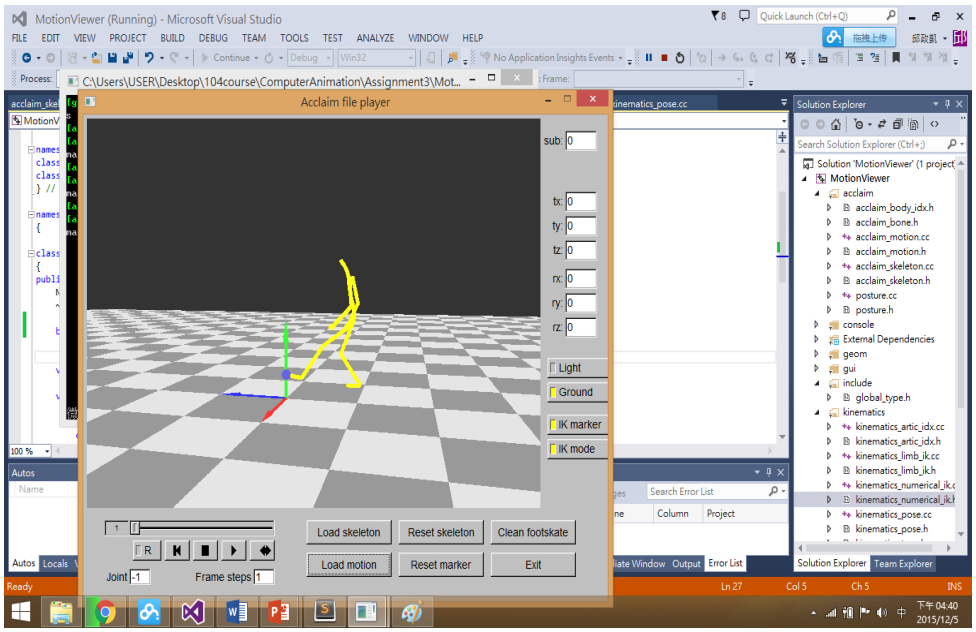
以下是一些結果的圖片：




MotionViewer (Running) - Microsoft Visual Studio
Acclaim file player
sub:
tx:
ty:
tz:
rx:
ry:
rz:
Light
Ground
IK marker
IK mode
Load skeleton
Reset skeleton
Clean footskate
Load motion
Reset marker
Exit
Solution Explorer
MotionViewer
acclaim
console
External Dependencies
geom
gui
include
kinematics


MotionViewer (Running) - Microsoft Visual Studio
Acclaim file player
sub:
tx:
ty:
tz:
rx:
ry:
rz:
Light
Ground
IK marker
IK mode
Load skeleton
Reset skeleton
Clean footskate
Load motion
Reset marker
Exit
Solution Explorer
MotionViewer
acclaim
console
External Dependencies
geom
gui
include
kinematics


MotionViewer (Running) - Microsoft Visual Studio
Acclaim file player
sub:
tx:
ty:
tz:
rx:
ry:
rz:
Light
Ground
IK marker
IK mode
Load skeleton
Reset skeleton
Clean footskate
Load motion
Reset marker
Exit
Joint
Frame steps
Solution Explorer
MotionViewer
acclaim
console
External Dependencies
geom
gui
include
kinematics



結論而言，雖然整個 IK 過程中最麻煩的 Pseudo Inverse Jacobian 的計算可以直接用 Eigen 來完成，但是在這次的作業中還是體會到了處理 IK 會面臨到的許多問題。而畢竟 Psudo_Inverse_Jacobian 的解法算出來的只是可以達到指定 endEffector 線性速度的對應關節最小的角速度，而非”自然”的關節運動，所以如果在沒有加入 Constraint Term 的狀況下只能期望 User Input 不要太過奇怪，要不然 Inverse-Jacobian 不保證能夠算出自然的動作啊。