3 Inverse Kinematik

In Aufgabe 2 wurde ein modifizierbares Skelett implementiert. Die Modifikation erfolgte dabei durch Einstellen der Gelenkwinkel. In dieser Aufgabe soll eine weitere Interaktionsmöglichkeit implementiert werden. Durch Vorgabe einer Zielposition für einen gewählten Knochen sollen die Gelenkwinkel so optimiert werden, dass der Knochen die Zielposition bestmöglich erreicht. Die Optimierung soll durch den CCD-Algorithmus erfolgen.

3.1 Berechnen der kinematischen Kette (4 Punkte)

Aktivieren Sie in `main.cpp` die Registrierung des `IKViewer`. Danach können Sie in der Baumansicht des Skeletts (SkeletonViewer) den zu positionierenden Endeffektor auswählen (Spitze des ausgewählten Knochens). Mit Strg+Mausklick in der Szene wird die Zielposition des Endeffektors bestimmt. Diese wird mit einem weißen Kreuz dargestellt nachdem sie die erste Aufgabe erfolgreich implementiert haben.

Zur Vorbereitung des CCD-Algorithmus muss die relevante kinematische Kette berechnet werden. Implementieren Sie dies in `calculate_kinematic_chain()` in `IKViewer.cpp`. Gehen Sie davon aus, dass die Skelettwurzel den Beginn der kinematischen Kette darstellt. Das Ende soll die Spitze des Endeffektorknochens sein. Als Ergebnis soll die kinematische Kette im Feld `kinematic_chain` stehen (2 Punkte). Dabei handelt es sich um eine Liste von Transformationen. Folgende Klassenhierarchie wird für Transformationen verwendet:

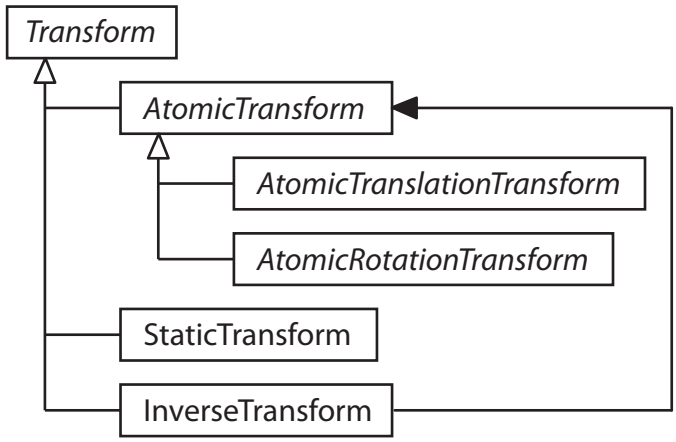

Abbildung 1: Klassenhierarchie für Transformationen

Die Klasse `AtomicTransform` beschreibt dabei eine Transformation, die durch einen skalaren Parameter parametrisiert wird. Sie wird hauptsächlich zur Definition von Freiheitsgraden genutzt. Die Klasse `StaticTransform` beschreibt eine konstante Transformation, die nicht verändert werden kann. `InverseTransform` repräsentiert die Inverse einer atomaren Transformation und wird in dieser Teilaufgabe nicht verwendet.

Berechnen Sie außerdem eine Transformationsmatrix vom Wurzelknoten zum Endeffektor und speichern Sie diese in `current_endeffector_matrix` ab. Außerdem soll beim Auswählen eines neuen Knochens dessen Endeffektorposition initial als Zielposition verwendet werden. Berechnen Sie diese Position im globalen Koordinatensystem (`target_position`) (2 Punkt).

3.2 Optimierung eines Einzelgelenks (4 Punkte)

Der CCD-Algorithmus baut auf der schrittweisen Optimierung von einzelnen Gelenken auf. Implementieren Sie `AtomicRotationTransform::optimize_value()` in `AtomicTransform.cpp` (2 Punkte). Diese Funktion optimiert den aktuellen Gelenkwinkel eines Rotationsgelenks. Dabei werden die Parameter `local_vector` und `target` übergeben. `local_vector` stellt dabei einen beliebigen Punkt im lokalen Koordinatensystem der Transformation dar. `target` stellt die Zielposition dieses Punkts dar. Die Funktion soll nun den Gelenkwinkel α so optimieren, dass $R_{axis,\alpha} \cdot local\_vector = target$ in einem Least-Squares-Sinn erfüllt ist. Nutzen Sie zum Setzen des Winkels die Funktion `set_value()`, die den Winkel in Grad entgegennimmt. Nutzen Sie den negativen Winkel, falls `inverse true` ist.

Gewährleisten Sie die Einhaltung der Winkelbegrenzungen, auf die Sie über `lower_limit` und `upper_limit` Zugriff haben (2 Punkte). Beachten Sie, dass die Rotationsgruppe zyklisch ist. Das heißt, dass eine arithmetische Begrenzung zwischen den Grenzen nicht immer zum gewünschten Ergebnis führt (Bsp.: Optimaler Winkel: 350°, Grenzen: $[10°, 50°]$).

3.3 CCD Implementierung (3 Punkte)

Nutzen Sie nun die zuvor implementierten Hilfsfunktionen, um den CCD-Algorithmus in `optimize()` des `IKViewer` zu implementieren. Führen Sie `max_iterations` Iterationen des Algorithmus aus oder bis der Abstand des Endeffektors zum Ziel `distance_threshold` unterschreitet. Die Optimierung soll dabei stets vom Endeffektor aus beginnen, um die notwendigen Anpassungen so gering wie möglich zu halten. Gehen Sie weiterhin davon aus, dass der Wurzelknoten des Skeletts statisch bleibt. Beachten Sie, dass das Skelett eine globale Positionierungsmatrix (`get_origin()`) hat.

3.4 Variable Basis (7 Punkte)

Im Gegensatz zur vorherigen Teilaufgabe soll die Basis der kinematischen Kette nun nicht mehr der statische Wurzelknoten sein, sondern dynamisch festgelegt werden können. Das heißt, dass während der Optimierung der Basisknochen an seiner Position bleibt. Sie können im SkeletonViewer den Basisknochen mit dem Button „Choose IK Base“ wählen, nachdem Sie einen Knochen in der Baumansicht gewählt haben.

Erweitern Sie `calculate_kinematic_chain` um die variable Basis. Suchen Sie dafür als erstes den tiefsten gemeinsamen Vorfahren des Basisknochens und des Endeffektors (2 Punkte). Passen Sie die kinematische Kette entsprechend an. Hierzu benötigen Sie die Klasse `InverseTransform`. Die `current_endeffector_matrix` soll so angepasst werden, dass sie eine Transformation von der Basis zum Endeffektor darstellt. Berechnen Sie außerdem eine Transformation aus dem globalen Koordinatensystem zum Basisknochen und speichern Sie diese in `current_base_matrix` ab. Sollten Sie weitere Klassenvariablen benötigen, können Sie diese anlegen (3 Punkte).

Zum Schluss muss noch die globale Skelettpositionierung (`set_origin`) angepasst werden. Berechnen Sie diese so, dass die Transformation aus dem globalen Koordinatensystem in das lokale System des Basisknochens über die Optimierung hinweg konstant bleibt (2 Punkte).

Abgabe

Verpacken Sie alle Quelldateien des `src`-Ordners in ein ZIP-Archiv und laden Sie sie in Opal hoch. Achten Sie darauf, dass der Quellcode auf den zur Abnahme verwendeten Rechnern lauffähig ist (Windows, Visual Studio $\geq$ 2015). Zum Testen stehen Ihnen die Rechnerpools der Fakultät zur Verfügung. Sprechen Sie ggfs. mit Ihrem Tutor.