UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Implementation eines Modells zur Simulation von Muskeln

Bachelorarbeit

zur Erlangung des Grades Bachelor of Science (B.Sc.)
im Studiengang Informatik

vorgelegt von
Daniel Remagen

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Dr. Robert Rockenfeller (Mathematisches Institut)

Koblenz, im Mai 2019

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Zusammenfassung

Um realistische Bewegungsabläufe zu simulieren, müssen Muskeln anatomisch korrekt modelliert werden können. Bisher ist es in SimPack nur möglich, Muskeln als gerade Linie zwischen zwei Punkten zu definieren. In dieser Arbeit wird ein Ansatz vorgestellt, bei dem Ellipsen definiert werden können, durch die ein Muskel laufen muss. Dabei entsteht vor allem das Problem, die Länge dieses Muskels durch die Ellipsen zu berechnen. Es wird ein Algorithmus vorgestellt, der den kürzesten Weg eines Muskelpfades durch diese Ellipsen berechnet. Dieser Algorithmus wird anschließend in `Fortran 90` umgesetzt und in ein bestehendes Muskelmodell in SimPack integriert.

abstract

In order to simulate realistic motion sequences, muscles must be able to be modelled anatomically correct. Yet it is only possible in SimPack to define muscles as a straight line between two points. This thesis presents an approach where ellipses can be defined through which a muscle must pass. The main problem is to calculate the length of this muscle through the ellipses. An algorithm is presented that calculates the shortest path of a muscle path through this ellipses. This algorithm is then implemented in `Fortran 90` and integrated into an existing muscle model in SimPack.

Inhaltsverzeichnis

1 Einleitung

Um komplexe Bewegungsabläufe simulieren zu können, werden sogenannte Mehrkörpersimulationen verwendet. Hierbei werden Muskeln als ein Kraftelement zwischen zwei Punkten dargestellt. Die Kraft, die ein Muskel aufbringt, ist von vielen Faktoren, z.B. der Länge des Muskels, abhängig. Diese Länge wird aktuell einfach als euklidischer Abstand zwischen zwei Punkten im Raum berechnet (vgl. [Klo18]). Dies entspricht jedoch nur in seltenen Ausnahmen einem anatomisch korrekten Muskel. Im menschlichen Körper werden Muskeln oft von anatomischen Strukturen, wie z.B. Sehnen, Knochen, Knorpel oder anderen Muskeln, in ihrem Verlauf beeinflusst. Hierfür soll eine Lösung nach [Ham19] vorgestellt werden, mit der der Verlauf eines Muskels über Ellipsen modelliert wird und die Länge des Muskels in einem verhältnismäßigen Aufwand näher am anatomischen Vorbild des Menschen berechnet werden kann.

2 Problemstellung

Ausgangslage ist die Implementation eines Hill-Muskel-Modells in SimPack, einem Mehrkörpersimulationensystem (vgl. [Klo18]). Innerhalb dieses Modells wird die Kraft eines Muskels, unter anderem abhängig von dessen Länge, berechnet. Wenn durch einen Muskel eine Streckung an einem Gelenk erzeugen werden soll, zeigt sich folgendes Problem:
Abbildung 1 zeigt eine stilisierte Bewegung eines Knochens K_1. Durch einen Muskel L soll eine Streckung um das Gelenk G erreicht werden. Nach aktuellem Stand ist es jedoch nur möglich, den Muskel als direkte Verbindung L_S zwischen zwei Punkten zu definieren und diese zwei Punkte durch die berechnete Kraft zusammenzuziehen. Das Abstoßen zweier Punkte ist nicht möglich, da ein Muskel sich nur zusammenziehen kann. Eine Streckung von L_S kann nur durch die Kontraktion des Antagonisten von L_S erfolgen. In Abbildung 1 wird die Bewegung des Knochens K_1 durch Kontraktion des Muskels L_S gezeigt, wobei K_1' den Knochen und L_S' den Muskel nach der ausgeführten Bewegung zeigt. Hieraus ergeben sich zwei Probleme: Zum Einen wird durch eine Kontraktion des Muskels L_S keine Streckung, sondern nur eine Beugung erfolgen. Somit wird nicht die gewünschte, sondern die gegenteilige Bewegung ausgeführt. Zum Anderen ist die Länge des Muskels L_S

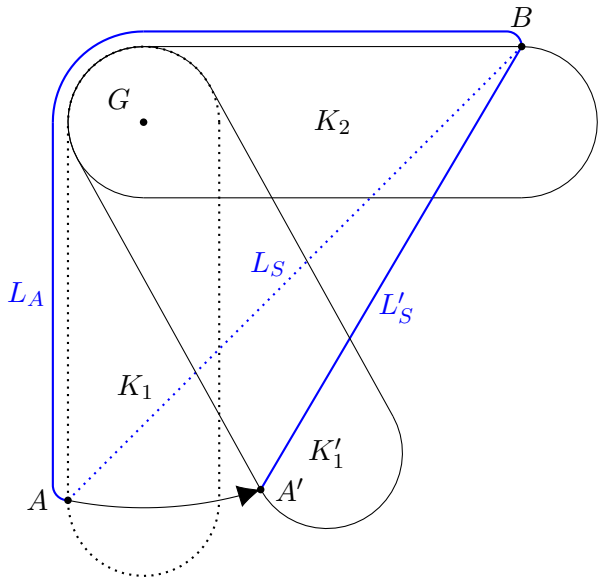


Abbildung 1: Bewegung eines Knochen K_1 durch den Muskel L_S um das Gelenk G in SimPack. K'_1 und L'_1 zeigen den Zustand nach der Bewegung. L_A ist der anatomisch korrekte Verlauf eines Muskel für die Streckung. K_2 ist in diesem Beispiel fixiert und kann sich nicht bewegen.

im Verhältnis zum anatomisch korrekten Muskel L_A zu kurz, und somit die erzeugte Kraft nicht korrekt, da diese unter anderem von der Muskellänge abhängt. Ein anatomisch korrekter Muskel L_A läuft um das Gelenk G bzw. die Knochen K_1 und K_2 herum, damit bei der Kontraktion von L_A eine Streckung im Gelenk erfolgt. Nachfolgend soll in dieser Arbeit besprochen werden, wie sich zum Einen das Problem der inkorrekten Muskellänge und zum Anderen das Problem der korrekten Kraftanwendung in SimPack lösen lässt.
Die Idee zu Abschnitt 3.2 bis Abschnitt 3.5 sowie Abschnitt 4.1 bis Abschnitt 4.6 stammen von [Ham19].

3 Mathematische Grundlagen

Zunächst werden in diesem Kapitel einige notwendige, mathematische Grundlagen besprochen, die in den Berechnungen innerhalb dieser Arbeit benötigt werden.

3.1 Transformation von Koordinaten und Vektoren im $\mathbb{R}^3$

Da im späteren Verlauf sowohl Punkte als auch Vektoren im $\mathbb{R}^3$ in ein neues Koordinatensystem transformiert werden müssen, wird hier ein besonderes Verfahren genutzt: Es werden homogene Koordinaten verwendet um beide Transformationen mit ein und derselben 4×4 Matrix durchführen zu können. Hierfür wird durch die Abbildung $\mathcal{T}:\mathbb{R}^3\to\mathbb{R}^4$ ein Punkt $P\,(p_X|p_Y|p_Z)$ und ein Vektor $\vec{v}=(v_X,v_Y,v_Z)^T$ wie folgt abgebildet:

$$\mathcal{T}:\begin{pmatrix}p_X\\p_Y\\p_Z\end{pmatrix}\longmapsto\begin{pmatrix}p_X\\p_Y\\p_Z\\1\end{pmatrix}\qquad\mathcal{T}:\begin{pmatrix}v_X\\v_Y\\v_Z\end{pmatrix}\longmapsto\begin{pmatrix}v_X\\v_Y\\v_Z\\0\end{pmatrix}$$

Dem entsprechend wird die Umkehrabbildung $\mathcal{T}^{-1}:\mathbb{R}^4\to\mathbb{R}^3$ wie folgt definiert:

$$\mathcal{T}^{-1}:\begin{pmatrix}p_X\\p_Y\\p_Z\\1\end{pmatrix}\longmapsto\begin{pmatrix}p_X\\p_Y\\p_Z\end{pmatrix}\qquad\mathcal{T}^{-1}:\begin{pmatrix}v_X\\v_Y\\v_Z\\0\end{pmatrix}\longmapsto\begin{pmatrix}v_X\\v_Y\\v_Z\end{pmatrix}$$

Wenn nun ein Punkt P oder ein Vektor $\vec{v}$ von einem Koordinatensystem in ein anderes transformiert werden soll, wird wie folgt vorgegangen:
Gegeben sind zwei Koordinatensysteme im $\mathbb{R}^3$, ein globales Koordinatensystem X,Y,Z und ein Koordinatensystem $\mathcal{X},\mathcal{Y},\mathcal{Z}$, das durch seinen Ursprung $U\,(u_X|u_Y|u_Z)$ und Koordinatenachsen $\mathcal{X}=(\mathcal{X}_X,\mathcal{X}_Y,\mathcal{X}_Z)^T$, $\mathcal{Y}=(\mathcal{Y}_X,\mathcal{Y}_Y,\mathcal{Y}_Z)^T$ und $\mathcal{Z}=(\mathcal{Z}_X,\mathcal{Z}_Y,\mathcal{Z}_Z)^T$ im globalen Koordinatensystem definiert ist. Die Transformation von dem Koordinatensystem X,Y,Z in das Koordinatensystem $\mathcal{X},\mathcal{Y},\mathcal{Z}$ erfolgt, indem zuerst mittels einer Translationsmatrix in den Ursprung von X,Y,Z verschoben und als Nächstes mit einer Rotationsmatrix rotiert wird:

$$\begin{pmatrix}\mathcal{X}_X&\mathcal{X}_Y&\mathcal{X}_Z&0\\\mathcal{Y}_X&\mathcal{Y}_Y&\mathcal{Y}_Z&0\\\mathcal{Z}_X&\mathcal{Z}_Y&\mathcal{Z}_Z&0\\0&0&0&1\end{pmatrix}\cdot\begin{pmatrix}1&0&0&-u_X\\0&1&0&-u_Y\\0&0&1&-u_Z\\0&0&0&1\end{pmatrix}=:T$$

Die Matrix T beinhaltet sowohl die Translation als auch die Rotation. Mit $\mathcal{T}$ wird ein Punkt $P_X\,(p_X|p_Y|p_Z)$ zuerst in den $\mathbb{R}^4$ abgebildet. Mit Multiplikation der Matrix T kann nun die Transformation vom Koordinatensystem X, Y, Z in das Koordinatensystem $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ erfolgen. Zuletzt muss P mit $\mathcal{T}^{-1}$ in den $\mathbb{R}^3$ zurück überführt werden:

$$\mathcal{T}: \begin{pmatrix} p_X \\ p_Y \\ p_Z \end{pmatrix} \mapsto \begin{pmatrix} p_X \\ p_Y \\ p_Z \\ 0 \end{pmatrix} \quad , \quad T \cdot \begin{pmatrix} p_X \\ p_Y \\ p_Z \\ 0 \end{pmatrix} = \begin{pmatrix} p_\mathcal{X} \\ p_\mathcal{Y} \\ p_\mathcal{Z} \\ 0 \end{pmatrix} \quad , \quad \mathcal{T}^{-1}: \begin{pmatrix} p_\mathcal{X} \\ p_\mathcal{Y} \\ p_\mathcal{Z} \\ 0 \end{pmatrix} \mapsto \begin{pmatrix} p_\mathcal{X} \\ p_\mathcal{Y} \\ p_\mathcal{Z} \end{pmatrix}$$

Als Ergebnis erhält man den transformierten Punkt $P_\mathcal{X}\,(p_\mathcal{X}, p_\mathcal{Y}, p_\mathcal{Z})$ im Koordinatensystem $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$. Analog erfolgt die Transformation eines Vektors $\vec{v}_X$ vom Koordinatensystem X, Y, Z in das Koordinatensystem $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ erst mit der Abbildung von $\vec{v}_X$ durch $\mathcal{T}$ in den $\mathbb{R}^4$, dann wird mit T multipliziert und anschließend mit $\mathcal{T}^{-1}$ in den $\mathbb{R}^3$ zurücküberführt:

$$\mathcal{T}: \begin{pmatrix} v_X \\ v_Y \\ v_Z \end{pmatrix} \mapsto \begin{pmatrix} v_X \\ v_Y \\ v_Z \\ 0 \end{pmatrix} \quad , \quad T \cdot \begin{pmatrix} v_X \\ v_Y \\ v_Z \\ 0 \end{pmatrix} = \begin{pmatrix} v_\mathcal{X} \\ v_\mathcal{Y} \\ v_\mathcal{Z} \\ 0 \end{pmatrix} \quad , \quad \mathcal{T}^{-1}: \begin{pmatrix} v_\mathcal{X} \\ v_\mathcal{Y} \\ v_\mathcal{Z} \\ 0 \end{pmatrix} \mapsto \begin{pmatrix} v_\mathcal{X} \\ v_\mathcal{Y} \\ v_\mathcal{Z} \end{pmatrix}$$

Das Ergebnis ist der transformierte Vektor $\vec{v}_\mathcal{X} = (v_\mathcal{X}, v_\mathcal{Y}, v_\mathcal{Z})^T$ im Koordinatensystem $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$. Dadurch, dass die vierte Komponente der Abbildung $\mathcal{T}$ von einem Vektor $\vec{v} \in \mathbb{R}^3$ Null ist, wird dieser nicht von der Translation beeinflusst. Ein Punkt $P \in \mathbb{R}^3$, dessen Abbildung durch $\mathcal{T}$ in der vierten Komponente eins ist, jedoch schon. Gleichzeitig ist es so möglich die Transformation von Punkten und Vektoren mit einer einzelnen Matrix T durchzuführen, was eine Implementation vereinfacht. Für eine bessere Übersicht werden nachfolgend transformierte Punkte als $P\,(p_\mathcal{X}, p_\mathcal{Y}, p_\mathcal{Z})$ als $P' = (p'_X, p'_Y, p'_Z)$ und transformierte Vektoren $\vec{v} = (v_\mathcal{X}, v_\mathcal{Y}, v_\mathcal{Z})^T$ als $\vec{v}\,' = (v'_X, v'_Y, v'_Z)^T$ bezeichnet.

3.2 Ellipsen im Raum

Eine Ellipse E kann durch einen Mittelpunkt und zwei Halbachsen definiert werden. Nachfolgend sei M der Mittelpunkt und die Vektoren $\vec{g}$ und $\vec{h}$ im $\mathbb{R}^3$ die Halbachsen der Ellipse (vgl. Abbildung 2). Ein Punkt P auf dem Rand

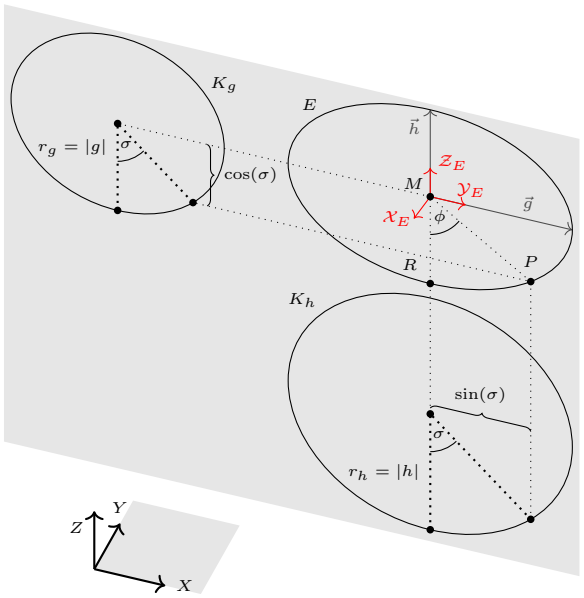


Abbildung 2: Ellipse im Raum mit lokalem Koordinatensystem $\mathcal{X}_E, \mathcal{Y}_E, \mathcal{Z}_E$ und Konstruktion nach La-Hire

der Ellipse kann über folgende Gleichungen beschrieben werden:

$$P(\sigma) = M + \vec{g} \cdot \sin(\sigma) - \vec{h} \cdot \cos(\sigma) \qquad \text{mit } \sigma \in [0, 2\pi) \tag{1}$$

$$\tan(\phi) = \frac{|\vec{g}|}{|\vec{h}|} \cdot \tan(\sigma) \tag{2}$$

Die Gleichung (1) ergibt sich durch die Konstruktion einer Ellipse nach La-Hire. Der Punkt P auf der Ellipse E wird über zwei Kreise K_g, mit Radius $r_g = |\vec{g}|$, und K_h, mit Radius $r_h = |\vec{h}|$ berechnet. Dabei ist zu beachten, dass der Winkel σ in den Kreisen K_g und K_h mit der Gleichung (2) zu berechnen ist und nicht dem Winkel ϕ in der Ellipse entspricht. Das Koordinatensystem $\mathcal{X}_E, \mathcal{Y}_E, \mathcal{Z}_E$ ist das lokale Koordinatensystem der Ellipse. Hierbei ist es für weitere Berechnungen wichtig, dass der Vektor $\vec{g}$ und die Achse $\mathcal{Y}_E$, bzw. der Vektor $\vec{h}$ und die Achse $\mathcal{Z}_E$ in die selbe Richtung zeigen und das $M = R + \vec{h}$ gilt.

3.3 Koordinatensystem zur Umlenkung an einer Ellipse

Um die nachfolgenden Berechnungen zur Lagebeziehung bzw. dem minimalen Abstand zwischen einer Ellipse E und Geraden g zu vereinfachen, wird

zunächst ein spezielles Koordinatensystem definiert. Gegeben sind der Mittelpunkt M der Ellipse, sowie zwei Punkte A und B, die auf der Geraden g liegen. Die $\mathcal{X}$-Achse $\vec{e}_{\mathcal{X}}$ zeigt von Punkt A zu Punkt B, die $\mathcal{Z}$-Achse $\vec{e}_{\mathcal{Z}}$ ist orthogonal zu der Ebene, die durch die drei Punkte A, B und M aufgespannt wird, und die $\mathcal{Y}$-Achse $\vec{e}_{\mathcal{Y}}$ ist orthogonal zu $\vec{e}_{\mathcal{X}}$ und $\vec{e}_{\mathcal{Z}}$ (vgl. Abbildung 4).

$$\begin{aligned}\vec{e}_{\mathcal{X}} &= \frac{B-A}{|B-A|}\\ \vec{e}_{\mathcal{Z}} &= \frac{\vec{e}_{\mathcal{X}} \times (M-A)}{|\vec{e}_{\mathcal{X}} \times (M-A)\,|}\\ \vec{e}_{\mathcal{Y}} &= \vec{e}_{\mathcal{Z}} \times \vec{e}_{\mathcal{X}}\end{aligned}$$

Das durch die Achsen $\mathcal{X}$, $\mathcal{Y}$ und $\mathcal{Z}$ aufgespannte Koordinatensystem wird im Nachfolgenden als Umlenkungskoordinatensystem, kurz $\mathcal{K}_U$, bezeichnet. Eine Transformation in $\mathcal{K}_U$ kann, wie in Abschnitt 3.1 beschrieben, erfolgen.

3.4 Lagebeziehung einer Ellipse zu einer Geraden

Im Rahmen dieser Arbeit werden nur zwei Arten von Lagebeziehungen zwischen einer Ellipse E, wie in Gleichung (1) definiert, und einer Geraden g durch die Punkte A und B betrachtet: Entweder g läuft durch E oder g läuft an E vorbei (vgl. Abbildung 3). Zuerst wird ein Punkt C als $C(\sigma_C) := M + c \cdot \vec{g} \cdot \sin(\sigma_C) - c \cdot \vec{h} \cdot \cos(\sigma_C)$ definiert, der in der $\mathcal{Y}_E$-$\mathcal{Z}_E$-Ebene der Ellipse liegt. Mit Hilfe des Parameters c kann entschieden werden, ob g innerhalb oder außerhalb der Ellipse E liegt. Als Nächstes werden der Mittelpunkt M und die beiden Vektoren $\vec{g}$ und $\vec{h}$ in das Koordinatensystem $\mathcal{K}_U$ (vgl. Abschnitt 3.3) transformiert. Das Ergebnis ist der transformierte Punkt $M'(m'_X|m'_Y|0)$ und die Vektoren $\vec{g}\,' = (g'_X, g'_Y, g'_Z)^T$ und $\vec{h}' = (h'_X, h'_Y, h'_Z)^T$. Durch die Transformation in $\mathcal{K}_U$ wird der Punkt C zu C', allerdings werden beide Punkte durch den selben Winkel θ_C bzw. σ_C auf der Ellipse beschrieben, weswegen $\sigma_C = \sigma_{C'}$ gilt. Der transformierte Punkt C' hat die Form $C'(\sigma_C) = M' + c \cdot \vec{g}\,' \cdot \sin(\sigma_C) - c \cdot \vec{h}' \cdot \cos(\sigma_C)$. Um mit C' die Lage von E und g zu bestimmen soll dieser Punkt auf der Geraden g liegen. Nach der Transformation in $\mathcal{K}_U$ zeigt die $\mathcal{X}$-Achse von A nach B. Daher muss für C' gleichzeitig $C'(\sigma_C) = (c'_X, 0, 0)^T$ gelten. Daraus ergibt sich folgende

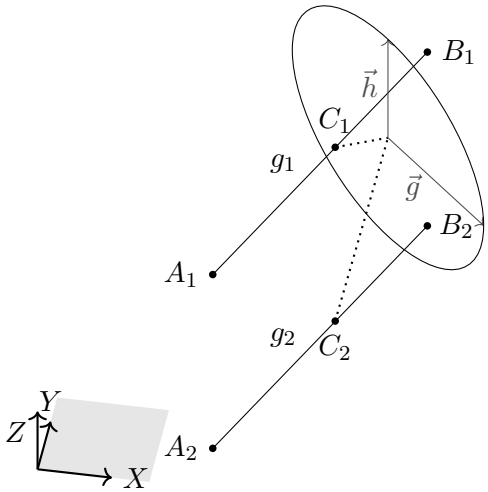


Abbildung 3: Eine Ellipse E, sowie eine Gerade g_1, die durch die Ellipse läuft, und eine Gerade g_2, die an der Ellipse vorbei läuft.

Gleichung:

$$\begin{pmatrix} c'_X \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} m'_X \\ m'_Y \\ 0 \end{pmatrix} + c \cdot \begin{pmatrix} g'_X \\ g'_Y \\ g'_Z \end{pmatrix} \cdot \sin(\sigma_C) - c \cdot \begin{pmatrix} h'_X \\ h'_Y \\ h'_Z \end{pmatrix} \cdot \cos(\sigma_C)$$

$$c'_X = m'_X + c\left(g'_X \sin(\sigma_C) - h'_X \cos(\sigma_C)\right) \tag{3}$$

$$0 = m'_Y + c\left(g'_Y \sin(\sigma_C) - h'_Y \cos(\sigma_C)\right) \tag{4}$$

$$0 = c\left(g'_Z \sin(\sigma_C) - h'_Z \cos(\sigma_C)\right) \tag{5}$$

M_Z ist Null da, per Definition der Achse $e_\mathcal{Y}$, M immer innerhalb der $\mathcal{X}$-$\mathcal{Y}$-Ebene liegt. Das Lösen des Gleichungssystems, bestehend aus Gleichungen (3), (4) und (5), führt zu einer Lösung für c, c'_X und σ_c. Wenn $|c| \leq 1$ ist, liegt C innerhalb oder auf der Ellipse E und g läuft durch die Ellipse, falls $|c| > 1$ ist, liegt C außerhalb der Ellipse und g läuft an der Ellipse vorbei.

3.5 Minimaler Abstand einer Ellipse zu einer Geraden

Gegeben sind eine Ellipse E, wie in Gleichung (1) definiert, sowie ein Startpunkt A und Endpunkt B der Strecke $\overline{AB}$. Nun soll der Abstand d zwischen der Ellipse und der Strecke $\overline{AB}$ minimiert werden. Dafür wird ein Punkt $P(\sigma_P)$ auf der Ellipse berechnet, sodass der Abstand von P zu $\overline{AB}$ minimal ist. Zunächst werden alle benötigten Koordinaten, A, B und M, sowie

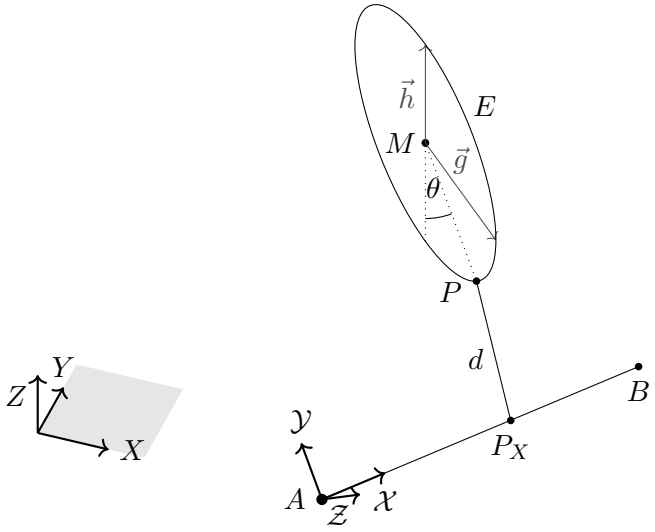


Abbildung 4: Eine Ellipse E mit Startpunkt A, Endpunkt B und Abstand d zur Strecke $\overline{AB}$. $\mathcal{X}$, $\mathcal{Y}$ und $\mathcal{Z}$ bilden das Umlenkungskoordinatensystem $\mathcal{K}_U$.

die Vektoren $\vec{g}$ und $\vec{h}$ in das Koordinatensystem $\mathcal{K}_U$ (vgl. Abschnitt 3.3) transformiert. Daraus ergeben sich die transformierten Punkte $A'\,(0|0|0)$, $B'\,(b'_X|0|0)$ und $M'\,(m'_X|m'_Y|0)$, sowie die Halbachsen $\vec{g}\,' = (g'_X, g'_Y, g'_Z)^T$ und $\vec{h}' = (h'_X, h'_Y, h'_Z)^T$. Als nächstes wird ein Distanzvektor $\vec{d}\,' := (d'_X, d'_Y, d'_Z)^T$ eingeführt, der von $\overline{A'B'}$ nach P' zeigt. Damit $\vec{d}\,'$ den Abstand zwischen $\overline{A'B'}$ und E minimiert, muss $\vec{d}\,'$ sowohl orthogonal zu $\overline{A'B'}$ als auch orthogonal zur Tangente an der Ellipse im Punkt P' sein. $\vec{d}\,'$ lässt sich als $\vec{d}\,' = P' - P'_X \cdot e_{\mathcal{X}}$ definieren, wodurch $d'_X = 0$ gilt und $\vec{d}\,'$ somit orthogonal zu $e_{\mathcal{X}}$ bzw. $\overline{A'B'}$ ist. Damit $\vec{d}\,'$ auch orthogonal zu der Tangente in P' ist, muss $\vec{d}\,'$ folgende Gleichung erfüllen:

$$
\begin{aligned}
0 &= d'^{T} \cdot \frac{\mathrm{d}P'}{\mathrm{d}\sigma_P}\,(\sigma_P) && (6)\\
&= \begin{pmatrix} 0 \\ d'_Y \\ d'_Z \end{pmatrix}^{T} \cdot \begin{pmatrix} g'_X \cdot \cos(\sigma_P) + h'_X \cdot \sin(\sigma_P) \\ g'_Y \cdot \cos(\sigma_P) + h'_Y \cdot \sin(\sigma_P) \\ g'_Z \cdot \cos(\sigma_P) + h'_Z \cdot \sin(\sigma_P) \end{pmatrix} \\
&= d'_Y \cdot \big(g'_Y \cos(\sigma_P) + h'_Y \sin(\sigma_P)\big) + d'_Z \cdot \big(g'_Z \cos(\sigma_P) + h'_Z \sin(\sigma_P)\big) && (7)
\end{aligned}
$$

$\frac{\mathrm{d}P'}{\mathrm{d}\sigma_P}\,(\sigma_P)$ ist die Ableitung von P' nach σ_P. Durch die Transformation in $\mathcal{K}_U$ wird der Punkt P zu P', allerdings werden beide Punkte durch den selben

Winkel θ_P bzw. σ_P auf der Ellipse beschrieben, weswegen $\sigma_P = \sigma_{P'}$ gilt. Durch die Definition $\vec{d}' = P' - P'_X \cdot \vec{e}_{\mathcal{X}}$ ergibt sich:

$$\begin{pmatrix} 0 \\ d'_Y \\ d'_Z \end{pmatrix} = \begin{pmatrix} m'_X \\ m'_Y \\ 0 \end{pmatrix} + \begin{pmatrix} g'_X \\ g'_Y \\ g'_Z \end{pmatrix} \cdot \sin(\sigma_P) - \begin{pmatrix} h'_X \\ h'_Y \\ h'_Z \end{pmatrix} \cdot \cos(\sigma_P)$$

$$0 = m'_X + g'_X \cdot \sin(\sigma_P) - h'_X \cdot \cos(\sigma_P)$$

$$d'_Y = m'_Y + g'_Y \cdot \sin(\sigma_P) - h'_Y \cdot \cos(\sigma_P) \tag{8}$$

$$d'_Z = m'_Z + g'_Z \cdot \sin(\sigma_P) - h'_Z \cdot \cos(\sigma_P) \tag{9}$$

Um σ_P zu bestimmen werden Gleichungen (8) und (9) in Gleichung (7) eingesetzt:

$$\begin{aligned} 0 = &\left(m'_Y + g'_Y \cdot \sin(\sigma_P) - h'_Y \cdot \cos(\sigma_P)\right) \cdot \left(g'_Y \cos(\sigma_P) + h'_Y \sin(\sigma_P)\right) + \\ &\left(m'_Z + g'_Z \cdot \sin(\sigma_P) - h'_Z \cdot \cos(\sigma_P)\right) \cdot \left(g'_Z \cos(\sigma_P) + h'_Z \sin(\sigma_P)\right) \end{aligned}$$

Durch Multiplizieren der Klammern, ersetzen von $\cos^2(\sigma_P) = 1 - \sin^2(\sigma_P)$ und $K = |\vec{g}|^2 - |\vec{h}|^2 - {g'_X}^2 + {h'_X}^2$, sowie Ausnutzen der Tatsache, dass $\vec{g'}^T \cdot \vec{h'} = 0$ gilt, ergibt sich folgende, vereinfachte Gleichung:

$$\begin{aligned} 0 = &\sin^4(\sigma_P)\left(K^2 + 4{g'_X}^2{h'_X}^2\right) + \sin^3(\sigma_P)\left(2m'_Y g'_Y K - 4g'_X h'_X m'_Y h'_Y\right) \\ &+ \sin^2(\sigma_P)\left(-K^2 + {m'_Y}^2{h'_Y}^2 - 4{g'_X}^2{h'_X}^2 + {m'_Y}^2{g'_Y}^2\right) \\ &+ \sin(\sigma_P)\left(2g'_X h'_X m'_Y h'_Y - 2m'_Y g'_Y K\right) + {g'_X}^2{h'_X}^2 - {m'_Y}^2{g'_Y}^2 \end{aligned} \tag{10}$$

Für eine ausführliche Erläuterung der einzelnen Umformungsschritte siehe [Car19]. Durch eine Substitution von $x = \sin(\sigma_P)$ ergibt sich eine quartische Gleichung, die z.B. nach Cardano [Car68] gelöst werden kann. Falls es mehr als eine reelle Lösung für σ_P gibt, wird diejenige als Lösung gewählt, für die der Abstand d am kürzesten ist, da der Abstand minimiert werden soll.

4 Problemlösung

Um den Verlauf eines Muskels besser darstellen zu können, werden Ellipsen definiert, durch die ein Muskel verlaufen muss (vgl. Abbildung 6). Dadurch soll erreicht werden, dass der modellierte Muskel in seinem Verlauf dem anatomischen Muskel entspricht. Die Länge des Muskels im Modell kann anhand

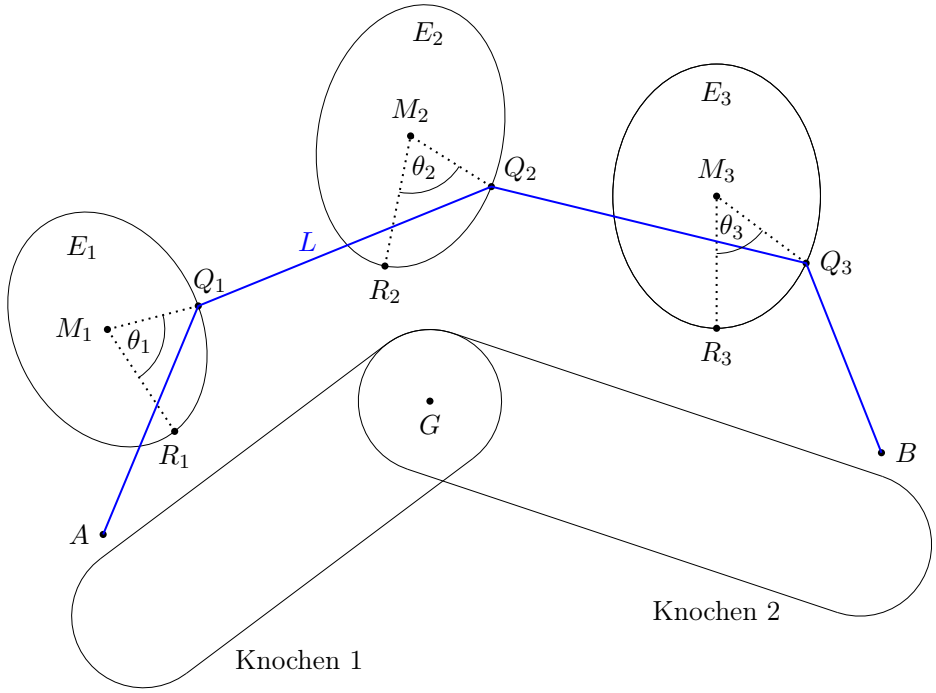


Abbildung 5: Ellipsen E_1, E_2 und E_3 mit Umlenkpunkten Q_1, Q_2 und Q_3, sowie Startpunkt A und Endpunkt B. L ist der der kürzeste Pfad von A nach B über alle Q_i. Die Knochen 1 und 2 sind über ein Gelenk in G verbunden.

der Ellipsen und Muskelansatzpunkte am Knochen berechnet werden.
Die Problemlösung teilt sich in mehrere Schritte: Zunächst muss erkannt werden, ob eine Ellipse die Strecke zwischen zwei Punkten A und B umlenkt. Eine solche Ellipse wird aktive Ellipse mit Umlenkpunkt Q genannt. Danach wird der Weg von A nach B über Q minimiert. Zuletzt wird ein Algorithmus vorgestellt, der den Weg über n Ellipsen minimiert. Hierbei werden auch Grenzen und Probleme aufgezeigt.

4.1 Modellierung von Muskeln über Ellipsen

Die Modellierung von Muskelpfaden über Ellipsen lässt sich wie folgt beschreiben: Ein Muskel L zwischen zwei Knochen ist die kürzeste Verbindung zwischen seinem Ansatzpunkt A und seinem Endpunkt B der über die Umlenkpunkte Q_i mit $i = 1, .., n$ verläuft. In Abbildung 5 wird beispielhaft die Modellierung eines Muskels über $n = 3$ Ellipsen gezeigt. Jeder Punkt Q_i liegt auf einer zugehörigen Ellipse E_i, welche über einen Befestigungspunkt

R_i, einen Mittelpunkt M_i, sowie zwei Halbachsen $\vec{g}_i$ und $\vec{h}_i$ definiert wird. Der Winkel θ_i bestimmt die Lage von Q_i eindeutig auf E_i. Der Befestigungspunkt R_i beschreibt die Lage einer Ellipse relativ zu einem Bezugssystem, meist der Knochen, an dem der Muskel befestigt ist. Alternativ kann die relative Lage von E auch über M und ein Bezugssystem definiert werden. Der Befestigungspunkt R_i kann dann mit $R_i = M_i - \vec{h}_i$ berechnet werden. Dies bedeutet, dass sich eine Ellipse E, die an einem Knochen befestigt ist, immer mit dem Knochen bewegt. Die Position und Orientierung von E kann sich also im Raum verändern, allerdings bleibt die Lage in Relation zum Bezugssystem immer gleich. Es wird davon ausgegangen, dass die Ellipsen die Länge des Muskels L beeinflussen. Ungeachtet davon wirken die Ellipsen sonst nicht auf den Muskel. Insbesondere ist L in Umlenkpunkt Q_i ohne Reibung, d.h. es entsteht kein Widerstand.

4.2 (In)aktive Ellipse

Als Erstes muss entschieden werden, ob eine Ellipse E aktiv oder inaktiv ist. Aktiv bedeutet, dass der Pfad zwischen einem Startpunkt A und einem Endpunkt B von E umgelenkt wird. Inaktiv bedeutet, dass die Strecke $\overline{AB}$ innerhalb der Ellipse liegt. Um zu Entscheiden, ob E aktiv oder inaktiv ist, wird das Verfahren zur Berechnung der Lagebeziehung zwischen einer Ellipse und einer Geraden, wie in Abschnitt 3.4 vorgestellt, verwendet. Wenn der Schnittpunkt C innerhalb oder auf E liegt, wird E aktiv genannt, andernfalls inaktiv. Abbildung 3 zeigt eine Ellipse E, die in Bezug auf g_1 aktiv und in Bezug auf g_2 inaktiv ist. Für eine aktive Ellipse muss im nächsten Schritt Q berechnet werden, damit die Länge von A über Q nach B ermittelt werden kann. Wenn E inaktiv ist, ist die Muskellänge gleich der Länge von $\overline{AB}$.

4.3 Wegminimierung über eine Ellipse

Um die Länge des Pfades L vom Punkt A über eine aktive Ellipse E zum Punkt B zu minimieren (vgl. Abbildung 6), wird wie folgt vorgegangen: Es wird der Punkt Q auf dem Rand der Ellipse gesucht, der den Pfad L von A über Q, und damit über die Ellipse, zum Punkt B minimiert. Der Punkt Q ist dabei nur abhängig von dem Winkel ϕ, der die Lage von Q auf der Ellipse beschreibt, bzw. analog dem Winkel σ, der der Kreisrepräsentation

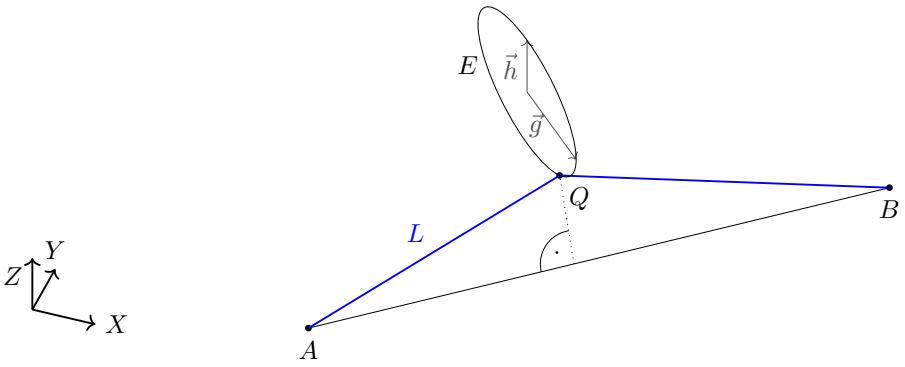


Abbildung 6: L ist der Pfad von A über Q nach B. Q liegt auf der Ellipse.

der Ellipse entspricht (vgl. Abbildung 2):

$$Q(\sigma) = M + \vec{g} \cdot \sin(\sigma_Q) - \vec{h} \cos(\sigma_Q)$$

Somit lässt sich die Suche nach Q als Problem formulieren, für das folgende Bedingung erfüllt sein muss:

$$\frac{dL}{d\sigma}(\sigma = \sigma_Q) = 0 \tag{11}$$

Gesucht wird also das σ_Q, welches den Pfad L minimiert. Würde die Gleichung (11) in Abhängigkeit von σ und allen Ellipsenparametern umformuliert, würde eine Differentialgleichung 6. Ordnung (vgl. [Ham19]) entstehen, die numerisch zu lösen ist. Aus Effizienzgründen wird ein anderer Ansatz verfolgt: Statt Gleichung (11) zu lösen, wird Q so gewählt, dass der Abstand von E zu $\overline{AB}$ minimiert wird. Die Berechnung von Q erfolgt wie in Abschnitt 3.5 erläutert. In fast allen Fällen minimiert Q dann auch den Pfad L. Probleme entstehen dann, wenn die Ellipse in der $\mathcal{X}$-$\mathcal{Y}$-Ebene liegt (vgl. Abbildung 7). In solch einem Fall ist eine numerische Korrektur notwendig.

4.4 Numerische Korrektur

Gegeben sind die Ellipse E, ein Startpunkt A', ein Endpunkt B' und der Punkt Q' auf E in Abhängigkeit von σ_Q, der den geringsten Abstand von E zu $\overline{A'B'}$ hat, sowie der zu Q' zugehörige Distanzvektor $\vec{d}' = (0, d'_Y, d'_Z)^T$ im Koordinatensystem $\mathcal{K}_U$. Um sicherzugehen, dass Q' den Pfad $L = |Q' - A'| + |B' - Q'|$ minimiert, wird ein Rotationsellipsoid $\mathcal{R}$, mit Mittelpunkt

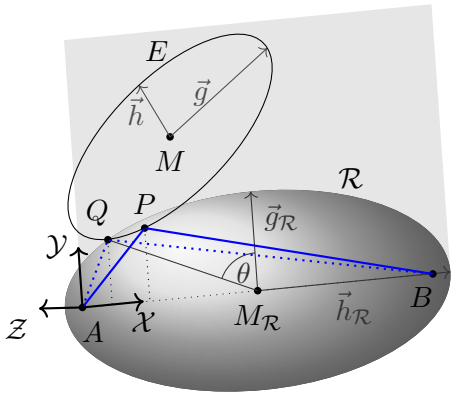


Abbildung 7: Ellipse E mit Rotationsellipsoid $\mathcal{R}$ zur numerischen Korrektur. E liegt in der $\mathcal{X}$-$\mathcal{Y}$-Ebene. Q hat den kleinsten Abstand zu $\overline{AB}$, aber P minimiert den Pfad zwischen den Punkten A und B. Der Ursprung von $\mathcal{K}_U$ liegt im Punkt A.

$M_\mathcal{R}$, sowie den Halbachsen $\vec{g}_\mathcal{R}$ und $\vec{h}_\mathcal{R}$, so konstruiert (vgl. Abbildung 7), dass A' und B' die Brennpunkte sind und Q' auf der Oberfläche liegt.

$$\vec{h}_\mathcal{R} = \frac{L}{2} \cdot e_\mathcal{X} \tag{12}$$
$$\vec{g}_\mathcal{R} = \frac{\sqrt{L^2 - {b'_X}^2}}{2|\vec{d}|} \cdot \vec{d}$$

Der Rotationsellipsoid wird durch die Rotation der Ellipse $\mathcal{E} : P_\mathcal{E}(\sigma) = M_\mathcal{R} + \vec{g}_\mathcal{R} \cdot \sin(\sigma) - \vec{h}_\mathcal{R} \cdot \cos(\sigma)$ um die $\mathcal{X}$-Achse erzeugt. Als Konsequenz dieser Konstruktion ist $L = |P_\mathcal{R} - A'| + |B' - P_\mathcal{R}|$ über alle Punkte $P_\mathcal{R}$ auf der Oberfläche von $\mathcal{R}$ gleich. Falls sich E und $\mathcal{R}$ in mehr als einem Punkt schneiden, heißt das, dass es einen Punkt $P' \neq Q'$ auf E gibt, der einen kürzeren Pfad $L = |P' - A'| + |B' - P'|$ hat als Q'. Um zu entscheiden, ob nach einem solchen Punkt P' gesucht werden muss, wird die Steigung α_E von E in Q' mit der Steigung $\alpha_\mathcal{R}$ von $\mathcal{R}$ in Q' verglichen. Beide Steigungen beziehen sich auf die $\mathcal{X}$-Achse. Hierfür wird zunächst θ_Q bestimmt, welches die Lage von Q' auf $\mathcal{R}$ beschreibt:

$$Q' = A' + \frac{b'_X}{2} \cdot \vec{e}_\mathcal{X} - \vec{h}_\mathcal{R} \cdot \sin(\theta_Q) + \vec{g}_\mathcal{R} \cdot \cos(\theta_Q) \tag{13}$$
$$\vec{d}' = \cos(\theta_Q) \cdot \vec{g}_\mathcal{R} \tag{14}$$

Insbesondere lässt sich Gleichung (13) für Q'_X im Koordinatensystem $\mathcal{K}_U$ durch zwei Eigenschaften vereinfachen: Zum einen ist $A' = (0, 0, 0)$, da A' der Ursprung von $\mathcal{K}_U$ ist. Zum anderen ist die x-Komponente von $\vec{g}_\mathcal{R}$ null, da $d'_X = 0$ ist. Durch Verwendung der Definition von $\vec{h}_\mathcal{R}$ in Gleichung (12) ergibt sich somit für Q'_X folgende Gleichung:

$$Q'_X = \frac{b'_X}{2} - \vec{h}_\mathcal{R} \cdot \vec{e}_\mathcal{X} \cdot \sin(\theta_Q) = \frac{b'_X}{2} - \frac{L}{2} \cdot \sin(\theta_Q) \tag{15}$$

Mit Gleichungen (14) und (15) kann α_E wie folgt berechnet werden:

$$\alpha_\mathcal{R} = \frac{|\vec{g}_\mathcal{R} \cdot \sin(\theta_Q)|}{|\vec{h}_\mathcal{R} \cdot \cos(\theta_Q)|} \tag{16}$$

Durch die Gleichung (6) ist die Ableitung von Q' orthogonal zu $\vec{d}'$. Gleichzeitig ist $\vec{d}'$ orthogonal zur $\mathcal{X}$-Achse. Deswegen ist die Steigung der Tangenten in Q' an E bezogen auf die $\mathcal{X}$-Achse Null. Deswegen wird mit Hilfe des Differenzenquotienten von $Q'(\sigma_Q)$ und $Q^- := Q'(\sigma_Q - 1°)$ bzw. $Q'(\sigma_Q)$ und $Q^+ := Q'(\sigma_Q + 1°)$ die Steigung α_E^- bzw. α_E^+ berechnet:

$$\alpha_E^- = \frac{|\vec{d}'(\sigma_Q)| - |\vec{d}'(\sigma_Q - 1°)|}{|g'_X(\sin(\sigma_Q - 1°) - \sin(\sigma_Q)) - h'_X(\cos(\sigma_Q - 1°) - \cos(\sigma_Q))|}$$
$$\alpha_E^+ = \frac{|\vec{d}'(\sigma_Q)| - |\vec{d}'(\sigma_Q + 1°)|}{|g'_X(\sin(\sigma_Q + 1°) - \sin(\sigma_Q)) - h'_X(\cos(\sigma_Q + 1°) - \cos(\sigma_Q))|}$$

Hier ist zu beachten, dass g'_X und h'_X die X-Koordinaten der Halbachsen $\vec{g}'$ und $\vec{h}'$ der Ellipse E und nicht des Rotationsellipsoid $\mathcal{R}$ sind. Wenn $\alpha_E^- \leq \alpha_\mathcal{R}$ oder $\alpha_E^+ \leq \alpha_\mathcal{R}$ ist, schneidet die Ellipse E den Rotationsellipsoid $\mathcal{R}$ in mehr als einem Punkt. Somit gibt es einen Punkt P' auf E, der innerhalb von $\mathcal{R}$ liegt und somit einen kürzeren Pfad L hat. Der Parameter σ_P, der die Lage von P' auf E definiert, liegt im Intervall $I = \left(\alpha_\mathcal{R}/\alpha_E^- \cdot 1°, \alpha_\mathcal{R}/\alpha_E^+ \cdot 1°\right)$. Mit einem numerischen Verfahren kann nun nach dem $\sigma_P \in I$ gesucht werden, welches den Pfad $L = |P'(\sigma_P) - A'| + |B' - P'(\sigma_P)|$ minimiert.

4.5 Rotation der Ellipse

Alternativ zur numerischen Korrektur aus Abschnitt 4.4 kann auch ein anderer Ansatz verfolgt werden: Eine Ellipse $E : P' = M' + \vec{g}' \cdot \sin(\sigma) - \vec{h}' \cdot \cos(\sigma)$ wird vor der Berechnung des Umlenkpunktes Q' so um ihren Mittelpunkt

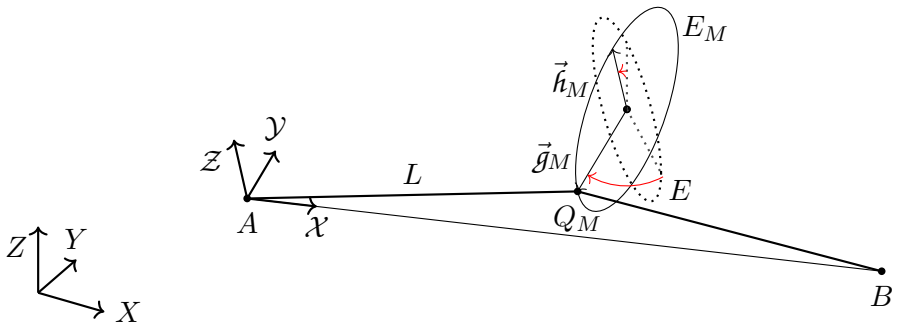


Abbildung 8: Die Rotation der Ellipse E um M ergibt die Ellipse E_M, die parallel zur $\mathcal{Y}$-$\mathcal{Z}$-Ebene ist.

M' rotiert, dass E orthogonal zur Strecke $\overline{A'B'}$ bzw. parallel zur $\mathcal{Y}$-$\mathcal{Z}$-Ebene ist. Durch diese Rotation ist Q', welcher den Abstand d von E zu $\overline{A'B'}$ minimiert (vgl. Abschnitt 3.5), auch gleichzeitig der Punkt, der den Pfad $L = |Q' - A'| + |B' - Q'|$ minimiert. Allerdings geschieht dies unter Verlust von Freiheitsgraden bei der Modellierung der Muskelstränge, da die Ausrichtung der Ellipse nicht mehr frei bestimmt werden kann. Die Rotation um M' kann nach [Ham19] durch eine Projektion der Halbachsen $\vec{g}'$ und $\vec{h}'$ auf $\mathcal{Y}$-$\mathcal{Z}$-Ebene erreicht werden:

$$\vec{g}_M = \left(g'_Y \cdot \vec{e}_{\mathcal{Y}} + g'_Z \cdot \vec{e}_{\mathcal{Z}}\right) \cdot \frac{|\vec{g}'|}{\sqrt{{g'_Y}^2 + {g'_Z}^2}}$$

$$\vec{h}_M = \left(h'_Y \cdot \vec{e}_{\mathcal{Y}} + h'_Z \cdot \vec{e}_{\mathcal{Z}}\right) \cdot \frac{|\vec{h}'|}{\sqrt{{h'_Y}^2 + {h'_Z}^2}}$$

Mit $\vec{g}_M = \left(g_X, g_Y, g_Z\right)^T$ und $\vec{h}_M = (h_X, h_Y, h_Z)^T$ als neue Halbachsen ergibt sich nun die Ellipse $E_M : P'_M = M' + \vec{g}_M \cdot \sin(\sigma) - \vec{h}_M \cdot \cos(\sigma)$ die parallel zur $\mathcal{Y}$-$\mathcal{Z}$-Ebene ist (vgl. Abbildung 8). Der Punkt Q'_M auf E_M wird wie in Abschnitt 3.5 berechnet, allerdings mit vereinfachter Gleichung (10), da $g_X = 0$ und $h_X = 0$ gilt, mit $K = |\vec{g}_M|^2 - |\vec{h}_M|^2$:

$$\begin{aligned} 0 = &\sin^4(\sigma_Q) \cdot K^2 + \sin^3(\sigma_Q) \cdot 2m'_Y g_Y K \\ &+ \sin^2(\sigma_Q) \cdot (-K^2 + {m'_Y}^2 {h_Y}^2 + {m'_Y}^2 {g_Y}^2) \\ &+ \sin(\sigma_Q) \cdot (-2m'_Y g_Y K) - {m'_Y}^2 {g_Y}^2 \end{aligned} \tag{17}$$

Q'_M hat minimalen Abstand zu $\overline{A'B'}$ und minimiert nun den Pfad $L = |Q'_M - A'| + |B' - Q'_M|$, da E_M und ein Rotationsellipsoid, konstruiert wie in Abschnitt 4.4, nur einen gemeinsamen Punkt Q' haben.
Hier ist allerdings zu erwähnen, dass die Projektion der Halbachsen $\vec{g}'$ und $\vec{h}'$ auf die $\mathcal{Y}$-$\mathcal{Z}$-Ebene wie sie in [Ham19] vorgestellt wird, nur dann funktioniert, wenn E nicht orthogonal zur $\mathcal{Y}$-$\mathcal{Z}$-Ebene ist. Wenn E orthogonal zur $\mathcal{Y}$-$\mathcal{Z}$-Ebene ist, dann gilt $g'_Z = 0$ und $h'_Z = 0$:

$$\vec{g}_M = \left(g'_Y \cdot \vec{e}_\mathcal{Y} + 0 \cdot \vec{e}_\mathcal{Z}\right) \cdot \frac{|\vec{g}'|}{\sqrt{{g'_Y}^2 + 0}} = g'_Y \cdot \vec{e}_\mathcal{Y} \cdot \frac{|\vec{g}'|}{\sqrt{{g'_Y}^2}} = |\vec{g}'| \cdot \vec{e}_\mathcal{Y} =: \vec{g}_O$$

$$\vec{h}_M = \left(h'_Y \cdot \vec{e}_\mathcal{Y} + 0 \cdot \vec{e}_\mathcal{Z}\right) \cdot \frac{|\vec{h}'|}{\sqrt{{h'_Y}^2 + 0}} = h'_Y \cdot \vec{e}_\mathcal{Y} \cdot \frac{|\vec{h}'|}{\sqrt{{h'_Y}^2}} = |\vec{h}'| \cdot \vec{e}_\mathcal{Y} =: \vec{h}_O$$

Somit werden durch die Projektion $\vec{g}'$ und $\vec{h}'$ nur auf die $\mathcal{Y}$-Achse abgebildet. Da $\vec{g}_M$ und $\vec{h}_m$ linear abhängig sind, wird durch die Gleichung $P'_O = M' + \vec{g}_O \cdot \sin(\sigma) - \vec{h}_O \cos(\sigma)$ keine Ellipse erzeugt:

$$P'_O = M' + \vec{g}_O \cdot \sin(\sigma) - \vec{h}_O \cos(\sigma)$$
$$P'_O = M' + \left(|\vec{g}'| \cdot \sin(\sigma) - |\vec{h}'| \cdot \cos(\sigma)\right) \cdot \vec{e}_\mathcal{Y}$$

Da $\sin(\sigma)$ und $\cos(\sigma)$ beschränkte Funktionen sind, ist auch der Term $\left(|\vec{g}'| \cdot \sin(\sigma) - |\vec{h}'| \cdot \cos(\sigma)\right)$ beschränkt. Demnach liegen die Punkte P'_O auf einer Strecke.

4.6 Wegminimierung über n Ellipsen

Bisher kann nur der Pfad über eine Ellipse minimiert werden. In diesem Abschnitt wird ein Verfahren vorgestellt um den Weg über n Ellipsen zu minimieren.
Zu Beginn sind ein Startpunkt A und ein Endpunkt B des Muskels, meist die Ansatzpunkte am Knochen, sowie die Ellipsen E_1 bis E_n, mit den zugehörigen Parametern R_i, M_i, $\vec{g}_i$ und $\vec{h}_i$, durch die der Muskel laufen soll, gegeben (vgl. Abbildung 5 für $n = 3$). Der Algorithmus, um die Länge des Pfades L, der den Muskelpfad modelliert, über alle Ellipsen E_i, bzw. über die Umlenkpunkte Q_i, zu minimieren besteht aus vier Schritten:

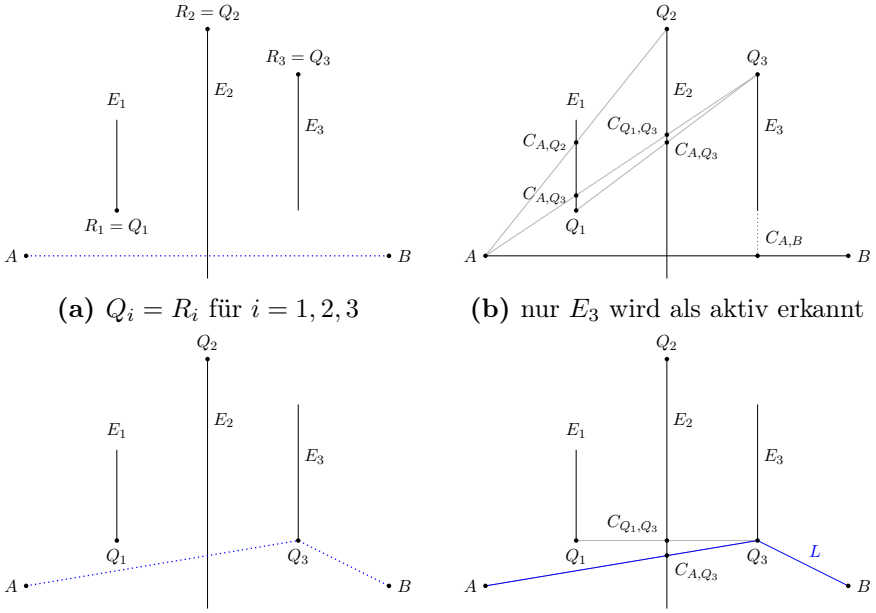
(i) Start Konfiguration
Zunächst werden alle Koordinaten A, B, R_i und M_i, sowie die Vektoren $\vec{g}_i$ und $\vec{h}_i$ für $i = 1, .., n$ in ein globales Koordinatensystem transformiert. Die Umlenkpunkte Q_i werden zu Beginn als die Punkte R_i definiert.

(ii) (In)aktive Ellipse
Zunächst muss für jede Ellipse E_i entschieden werden, ob diese aktiv oder inaktiv ist (vgl. Abschnitt 4.2). Für diese Unterscheidung werden die benachbarten Umlenkpunkte bzw. die Start- und Endpunkte verwendet, d.h. für jede Ellipse muss ein eigenes Umlenkungskoordinatensystem berechnet werden. Falls zwei oder mehr Ellipsen hintereinander inaktiv sind, muss die Berechnung erneut erfolgen, da es Fälle gibt, in denen eine korrekte Erkennung sonst nicht möglich wäre. Um aktive Ellipsen richtig zu erkennen, werden hierfür die Umlenkpunkte von aktiven, benachbarten Ellipsen bzw. die Ansatzpunkte verwendet.

(iii) Berechnung des stückweise kürzesten Weges
Wenn keine Ellipse E aktiv ist, ist die Muskellänge die direkte Verbindung zwischen A und B. Falls nur eine Ellipse E_i aktiv ist, muss nach der Berechnung von Q_i überprüft werden, ob benachbarte inaktive Ellipsen aktiv geworden sind. Falls mehr als zwei Ellipsen aktiv sind, wird stückweise für jede Ellipse E_i mit ihren benachbarten, aktiven Umlenkpunkten bzw. Start- und Endpunkt der kürzeste Pfad berechnet. Hierbei werden Schritt (ii) und Schritt (iii) so lange wiederholt, bis die gewünschte Genauigkeit für alle Q_i's erreicht ist und die Anzahl der Umlenkpunkte konvergiert.

(iv) Endergebnis
Das Ergebnis ist die Position aller Q_i's, für die die Länge des Pfad L minimal wird, sowie die Länge des Pfades L.

Aus diesen Positionen und der Muskellänge L kann mit dem Hill-Model aus [Klo18] die Kraft und Richtung des Muskels berechnet werden.
Hier ist allerdings zu erwähnen, dass der vorgestellte Algorithmus nach [Ham19] nicht alle aktiven Ellipsen immer korrekt erkennen kann. Abbildung 9 zeigt einen Fall, in dem Ellipse E_1 als inaktiv erkannt wird, obwohl sie den Muskel L umlenken müsste. Zu Beginn des Algorithmus werden,



(a) $Q_i = R_i$ für $i = 1, 2, 3$ **(b)** nur E_3 wird als aktiv erkannt

(c) Q_3 wird neu berechnet **(d)** E_1 wird nicht als aktiv erkannt

Abbildung 9: Drei Ellipsen, bei denen Ellipse E_1 nicht korrekt als aktiv erkannt wird, obwohl sie den Muskelpfad L umlenken müsste. Zur besseren Übersicht sind alle Ellipsen orthogonal zu L und werden deswegen nur als Linie dargestellt.

wie in Schritt (i) beschrieben, alle Umlenkpunkte Q_1 bis Q_3 auf die Befestigungspunkte R_1 bis R_3 gesetzt (vgl. Abbildung 9a). In Schritt (ii) wird über alle Ellipsen iteriert und nur E_3 wird als aktiv erkannt (vgl. Abbildung 9d). C_{A,Q_1} ist der Schnittpunkt von der Ellipse, auf der C liegt, und der Strecke $\overline{AQ_1}$. Da $C_{A,B}$ nicht innerhalb der Ellipse E_3 liegt, wird E_3 aktiv. Daraufhin wird wie in Schritt (iii) der Umlenkpunkt Q_3 berechnet (vgl. Abbildung 9c). Da E_3 die einzige aktive Ellipse ist, müssen die benachbarten inaktiven Ellipsen, hier nur E_2, erneut überprüft werden (vgl. Abbildung 9d). Da sowohl C_{Q_1,Q_3}, als auch C_{A,Q_3}, innerhalb von E_2 liegen, bleibt E_2 inaktiv und L ist der Pfad des Muskels. Somit wird nicht erkannt, dass E_1 den Pfad des Muskels umlenkt.

Damit der Algorithmus solche Ellipsen ebenfalls erkennen kann, müssen nach der Berechnung von einem oder mehreren Umlenkpunkten in Schritt (iii) nicht nur die benachbarten Ellipsen, sondern alle Ellipsen erneut auf Aktivität überprüft werden.

5 Implementation

Nachfolgend wird erläutert, wie die vorgestellte Modellierung eines Muskelpfades, sowie die Problemlösungen zur Berechnung seiner Länge, aus Abschnitt 4 in `Fortran 90` und SimPack umgesetzt wurden.

5.1 SimPack

SimPack ist eine Software um mechanische Mehrkörpersysteme zu modellieren, die vom Institut für Medizintechnik und Informationsverarbeitung (MTI) am Campus Koblenz verwendet wird. Dem Nutzer oder der Nutzerin wird eine graphische Oberfläche zur Verfügung gestellt, mit der Objekte im Raum platziert und Bewegungen, z.B. durch Festlegung von Kraft und Richtung an einem Objekt, ausgeführt werden. Punkte im Raum werden dabei als sogenannte Marker dargestellt. Jedes Objekt wird immer relativ zu einem Marker definiert. Bewegungen von Objekten werden durch Integration aus Kraftgleichungen berechnet. Innerhalb der von SimPack zur Verfügung gestellten Methoden können auch vom Nutzer selbst geschriebene Routinen zur Berechnung von Kräften, sogenannte *uforce*'s, verwendet werden. Innerhalb einer *uforce* wird die Kraft berechnet und anschließend auf die zwei Marker, den From-Marker und den To-Marker, angewandt, zwischen denen sie definiert ist. Die *uforce* kann wahlweise in `C#` oder `Fortran 90`, bzw. genauer im fixed-Format, geschrieben werden. Aufgrund der Tatsache, dass die bestehenden Erweiterungen am MTI in `Fortran 90` geschrieben sind, wurde der Algorithmus aus Abschnitt 4.6 ebenfalls in `Fortran 90` umgesetzt. Hierbei bietet SimPack die Möglichkeit, bestehende *uforce*'s mit externen Dateien zu erweitern. Diese müssen, da SimPack die Dateien selbst kompiliert und einbindet, allerdings wie die *uforce* im fixed-Format geschrieben sein. Um die Übersicht in der *uforce* zu gewährleisten, wurde der Algorithmus daher innerhalb eines eigenen `MODULE`'s umgesetzt.

5.2 Umsetzung in `Fortran 90`

Da SimPack nur das fixed-Format erlaubt, sind bei der Implementierung einige Besonderheiten zu beachten: Die ersten sechs Zeichen einer Zeile müssen frei bleiben, da diese Stellen mit bestimmten Funktionen belegt sind. Dabei handelt es sich um ein Relikt aus der Zeit, als `Fortran`-Programme auf Lochkarten geschrieben wurden. Des Weiteren ist die Länge einer Zeile

Mathematisches Definition	Variable in `Fortran 90`
Startpunkt A	`global_a`
Endpunkt B	`global_b`
Mittelpunkt M	`global_m`
$\vec{h}$	`vec_g`
$\vec{g}$	`vec_h`
Umlenkpunkt Q	`q`

Tabelle 1: Übersicht über mathematische Definitionen und entsprechende Variablen in `Fortran 90` für die Minimierung eines Pfades über eine Ellipse

auf 72 Zeichen beschränkt. Falls ein Statement länger ist, kann durch ein Zeichen an der sechsten Stelle in einer Zeile die vorherige Zeile um diese erweitert werden. In `Fortran 90` gibt es die Möglichkeit, Unterprogramme, entweder als `FUNCTION`, die nur genau einen Rückgabewert hat, oder als `SUBROUTINE`, die keine oder mehrere Rückgabewerte haben kann, umzusetzen. Beim Aufruf einer `FUNCTION` wird der Rückgabewert immer direkt einer Variable zugewiesen, der Aufruf einer `SUBROUTINE` erfolgt über einen `call` Befehl. Wenn eine `SUBROUTINE` Werte zurückgeben soll, müssen diese also beim Aufruf als Argumente mit übergeben werden. Zur Lösung der quartischen Gleichung (10) bzw. Gleichung (17) wird ein `MODULE` mit dem Namen `quartic.f90` von Ralph L. Carmichael [Car99] verwendet. Es berechnet analytisch sowohl reelle als auch komplexe Nullstellen einer quartischen Gleichung. Das Original `quartic.f90` ist im free-Format geschrieben und wurde daher, damit es in SimPack integriert werden kann, in das fixed-Format übersetzt. Dabei wurden nur Änderungen an der Formatierung des Quellcodes vorgenommen. Das in das fixed-Format übersetzte `MODULE` hat den Namen `PolynomialRoots`.

Zuerst wurde die Minimierung eines Pfades L über eine Ellipse E als `SUBROUTINE` umgesetzt. Tabelle 1 zeigt die Varaibalen, die als Argumente für die `SUBROUTINE` verwendet wurden. Der Umlenkpunkt Q in globalen Koordinaten wird innerhalb der `SUBROUTINE` berechnet und in `q` gespeichert. Dabei wurde sowohl das einfache Verfahren aus Abschnitt 4.3 als auch die numerische Korrektur aus Abschnitt 4.4 und die Rotation der Ellipse aus Abschnitt 4.5 implementiert, um den Nutzer oder der Nutzerin später mehr Möglichkeiten zur Modellierung zu geben. Somit gibt es insgesamt drei `SUBROUTINE`'s um einen Umlenkpunkt zu berechnen:

- `single_ellipse_without_correction` entspricht Abschnitt 4.3
- `single_ellipse_with_correction` entspricht Abschnitt 4.4
- `single_ellipse_co_rotating` entspricht Abschnitt 4.5

Zuletzt wurde in `full_via_ellipse` das Verfahren aus Abschnitt 4.6 als `SUBROUTINE` umgesetzt, um die Länge eines Muskelpfades über n Ellipsen zu minimieren. Nachfolgend wird die Implementierung dieser vier `SUBROUTINE`'s genauer erläutert.

5.3 Implementation: `single_ellipse_without_correction`

Entsprechend Abschnitt 4.3 erfolgt erst die Transformation in das Koordinatensystem $\mathcal{K}_U$. Die transformierten Punkte A', B' und M', sowie die transformierten Vektoren, $\vec{h}'$ und $\vec{g}'$, werden in den Variablen `a`,`b und` `m`, sowie `h` und `g`, zwischengespeichert. Mit `is_active(m,g,h)` wird, wie in Abschnitt 4.2 beschrieben, berechnet, ob die Ellipse aktiv oder inaktiv ist. Falls E aktiv ist, wird `true`, bzw. wenn E inaktiv ist, `false`, zurückgegeben. Da nur ein `boolean` zurückgegeben wird, handelt es sich bei `is_active` um eine `FUNCTION`. Nur wenn die E aktiv ist, wird der Umlenkpunkt Q berechnet. Dafür werden gemäß Gleichung (10) die Koeffizienten der quartischen Gleichung berechnet. Nach einer Substitution von $\sin(\sigma) = x$ werden mit `PolynomialRoots` die vier Nullstellen, x_1 bis x_4, berechnet. Da Gleichung (10) komplexe Nullstellen haben kann, werden diese, falls vorhanden, als Nächstes verworfen. Durch die Substitution $\sin(\sigma) = x$ gilt $\sigma_i = \arcsin(x_i)$. Weil $\arcsin(x)$ nur in den Wertebereich $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ abbildet, σ aber im Bereich $[0, 2\pi)$ liegen kann, soll hier die Verwendung von arcsin vermieden werden. Stattdessen wird wie folgt vorgegangen: Zur Berechnung von Q muss σ_i in die Gleichung (1) im Koordinatensystem $\mathcal{K}_U$, $P'(\sigma_i) = M' + \vec{g}' \cdot \sin(\sigma_i) - \vec{h}' \cdot \cos(\sigma_i)$, eingesetzt werden. Mit $\sigma_i = \arcsin(x_i)$ ergibt sich $\sin(\sigma_i) = \sin(\arcsin(x_i)) = x_i$. Um $\cos(\sigma_i)$ zu berechnen, werden die Gleichungen $\sin^2(\sigma_i) + \sin^2(\sigma_i) = 1$ und $\sin(\sigma_i) = x_i$ verwendet. Daraus ergibt sich $\cos(\sigma_i) = \pm\sqrt{1 - x_i^2}$. Somit kann Q innerhalb von `Fortran 90` durch $Q' = M' + \vec{g}' \cdot x_i - \vec{h}' \cdot \left(\pm\sqrt{1 - x_i^2}\right)$ berechnet werden. Im ungünstigen Fall gibt es vier Nullstellen und somit, wegen $\pm\sqrt{1 - x_i^2}$, acht Kombinationen, die überprüft werden müssen. Nach der Iteration über alle

acht Kombinationen wird mit der Kombination x_i und $\pm\sqrt{1-x_i^2}$, für welche der Abstand d' minimal ist, als letztes Q im globalen Koordinatensystem berechnet und in `q` gespeichert.

5.4 Implementation: `single_ellipse_with_correction`

In der `SUBROUTINE single_ellipse_with_correction` wird das Verfahren aus Abschnitt 4.4 umgesetzt: Zunächst wird wie in Abschnitt 5.3 der Q' mit minimalem Abstand zu Strecke $\overline{AB}$ berechnet. Mit dem Rotationsellipsoid $\mathcal{R}$ bzw. dessen Steigung in Q' wird entschieden, ob eine Korrektur notwendig ist und falls ja, wo der Punkt P' auf der Ellipse liegt, der einen kürzeren Pfad $L = |P' - A'| + |B' - P'|$ hat, als Q'. Um $\alpha_{\mathcal{R}}$ zu berechnen werden $\sin(\theta_Q)$ und $\cos(\theta_Q)$ benötigt. Da $\cos(\theta_Q)$ aus Gleichung (14) und $\sin(\theta_Q)$ aus Gleichung (15) berechnet werden können, wird in `Fortran 90` der Winkel θ_Q selbst nicht berechnet. Stattdessen wird $\sin(\theta_Q)$ in der Variable `sin_theta` bzw. $\cos(\theta_Q)$ in der Variable `cos_theta` gespeichert. Diese Variablen können direkt in Gleichung (16) für $\alpha_{\mathcal{R}}$ eingesetzt werden. So kann wieder vermieden werden, $\arcsin(x)$ oder $\arccos(x)$ verwenden zu müssen. Um α_E^- und α_E^+ zu berechnen, wird mit der Formel

$$\sigma_Q = \arctan\left(\frac{\sin(\sigma_Q)}{\sin(\sigma_Q)}\right)$$

σ_Q berechnet und in der Variable `sigma` gespeichert. Dabei wird die in `Fortran 90` vordefinierte `FUNCTION atan2(a,b)` verwendet, da diese, im Gegensatz zu arctan, in den Wertebereich $(-\pi, \pi]$ abbildet. Mit `sigma` können α_E^- und α_E^+ berechnet werden, um zu entscheiden, ob eine Korrektur notwendig ist. Falls keine Korrektur notwendig ist, minimiert der Punkt Q' den Pfad über die Ellipse, falls eine Korrektur notwendig ist, wird mit Bisektion im Intervall $I = \left(\alpha_{\mathcal{R}}/\alpha_E^- \cdot 1°, \alpha_{\mathcal{R}}/\alpha_E^+ \cdot 1°\right)$ nach dem σ_P gesucht, welches die Länge des Pfades $L = |P' - A'| + |B' - P'|$ minimiert. Dabei wird P' mit Bisektion bis auf eine Genauigkeit von $4 \cdot \epsilon$ berechnet, wobei ϵ die Maschinengenauigkeit für `double precision` Variablen von `Fortran 90` ist.

5.5 Implementation: `single_ellipse_co_rotating`

Für die Rotation einer Ellipse E durch `single_ellipse_co_rotating` werden die Achsen $\vec{g}$ und $\vec{h}$ zuerst, wie in Abschnitt 4.5 vorgestellt, im globalen

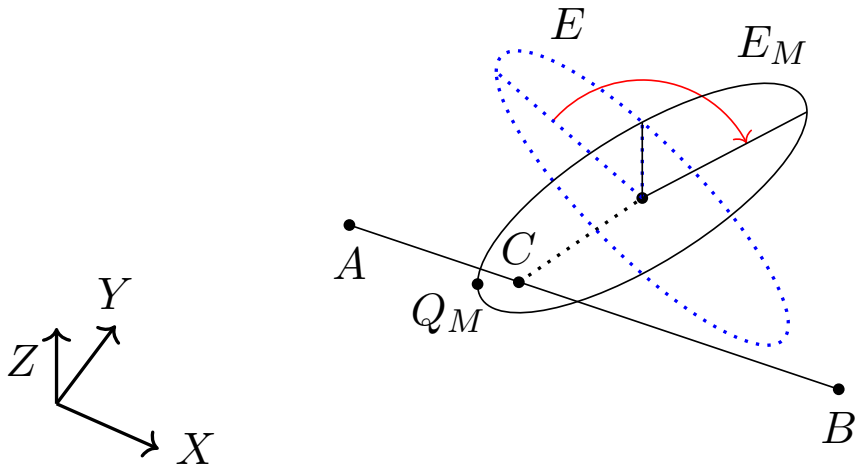


Abbildung 10: Ellipse E ist vor der Rotation um M aktiv. Die rotierte Ellipse E_M ist allerdings inaktiv.

Koordinatensystem rotiert, sodass sie parallel zur $\mathcal{Y}$-$\mathcal{Z}$-Ebene sind. Erst danach erfolgt die Entscheidung, ob E aktiv oder inaktiv ist. Diese Reihenfolge ist wichtig, da sonst folgendes Problem (vgl. Abbildung 10) entsteht: Die Ellipse E ist vor der Rotation inaktiv und somit müsste ein Umlenkpunkt Q berechnet werden. Daraufhin würde E rotiert und auf der rotierten Ellipse E_M der Umlenkpunkt Q_M als aktiver Umlenkpunkt berechnet. Q_M verlängert allerdings den Pfad $L = |Q_M - A| + |B - Q_M|$. Abbildung 10 zeigt, dass der Punkt C auf der Strecke $\overline{AB}$ innerhalb der Ellipse E_M liegt. Somit ist E_M inaktiv. Nur wenn E zuerst zu E_M rotiert wird, kann mit E_M korrekt entschieden werden, ob E_M aktiv oder inaktiv ist. Nach der Rotation wird wie in Abschnitt 5.3 verfahren, allerdings mit der vereinfachter Gleichung (17), wie in Abschnitt 4.5 erklärt. Zuletzt wird Q mit den rotierten Achsen im globalen Koordiatensystem berechnet.

5.6 Implementation: `full_via_ellipse`

Als Letztes wurde der in Abschnitt 4.6 vorgestellte Algorithmus zu Wegminimierung über n Ellipsen in `full_via_ellipse` als `SUBROUTINE` implementiert. Tabelle 2 zeigt die Eingabeparameter, die alle vom Typ `double` sind. Die Variablen `global_m`, `global_h`, `global_g` und `q` sind Array's der Länge n, in der an der Stelle i ein Vektor mit drei Einträgen steht. `muscle_length` ist ein Array der Länge eins, also nur eine einfache Zahl. Zusätzlich zu diesen

Definition in Abschnitt 4.6	`Fortran 90`	Dimension
Startpunkt A	`origin`	`(3)`
Endpunkt B	`insertion`	`(3)`
Mittelpunkte M_1 bis M_n	`global_m`	`(n,3)`
Vektoren $\vec{h}_1$ bis $\vec{h}_n$	`global_h`	`(n,3)`
Vektoren $\vec{g}_1$ bis $\vec{g}_n$	`global_g`	`(n,3)`
Umlenkpunkte Q_1 bis Q_n	`q`	`(n,3)`
Länge des Muskelpfads L	`muscle_length`	`(1)`

Tabelle 2: Übersicht über mathematische Definitionen und entsprechende Variablen und deren Dimension in `Fortran 90` für die Minimierung eines Pfades über n Ellipsen.

Eingabeparametern gibt es noch eine weitere Variable `algorithm`, mit der einer der drei Algorithmen aus Abschnitt 5.3 bis Abschnitt 5.5 ausgewählt werden kann. Zusätzlich zeigt Tabelle 3 eine Übersicht wichtiger Variablen, die innerhalb von `full_via_ellipse` verwendet werden, um die Länge des Pfads L zu minimieren.

`Fortran 90`	Dimension	type
`via_points`	`(n+2,3)`	`double`
`active_points`	`(n,3)`	`integer`
`transform_matrices`	`(n,4,4)`	`double`
`g_r`	`(n,3)`	`double`
`h_r`	`(n,3)`	`double`
`m`	`(n,3)`	`double`
`h`	`(n,3)`	`double`
`g`	`(n,3)`	`double`
`active`	`(n)`	`boolean`
`converging`	`(n)`	`boolean`

Tabelle 3: Übersicht über wichtige, interne Variablen von `full_via_ellipse`

Nachfolgend werden diese Variablen und deren Funktion erläutert:

- `via_points`
  Das Array speichert alle Punkte, durch die der Muskelpfad L läuft, inklusive dem Startpunkt A, aller Umlenkpunkte Q_i und dem Endpunkt B, daher die Länge $n + 2$. Der Umlenkpunkt Q_i befindet sich an Stelle $(i + 1)$ in `via_points`, da `via_points(1)` dem Startpunkt A

entspricht.

- `active_points`
  In diesem Array werden an der Stelle i zu jeder Ellipse E_i die benachbarten, aktiven Punkte durch die der Muskel läuft, also die Umlenkpunkte, bzw. der Start- oder Endpunkt des Muskels, zu E_i gespeichert. Genauer gesagt werden hier nur die Indizes der Punkte innerhalb von `via_points` verwaltet (vgl. Tabelle 4).

- `transform_matrices`
  Hier werden in einem Array der Länge n, an der Stelle i, die Transformationsmatrizen $T_i \in \mathbb{R}^4$ gespeichert. So ist es im späteren Verlauf möglich, für die Prüfung auf Aktivität und die Transformation in das Koordinatensystem $\mathcal{K}_U$ auf die gespeicherte Matrizen T_i zuzugreifen und den Rechenaufwand zu minimieren.

- `g_r` und `h_r`
  Hier werden, falls die Ellipse rotiert werden soll, die rotierten Halbachsen $\vec{g}$ und $\vec{h}$ gespeichert.

- `m`, `g` und `h`
  In diesen Arrays werden die transformierten Punkte M', sowie die transformierten Halbachsen $\vec{g}'$ und $\vec{h}'$ gespeichert.

- `active`
  In diesem Array der Länge n wird für jede Ellipse E_i gespeichert, ob diese aktiv oder inaktiv ist.

- `converging`
  Hier wird gespeichert, ob der Umlenkpunkt Q_i konvergiert. Dies wird für die Abbruchbedingung des Algorithmus `full_via_ellipse` benötigt.

Der Programmablauf von `full_via_ellipse` orientiert sich am vorgestelltem Algorithmus aus Abschnitt 4.6:
Alle Punkte und Vektoren sind in einem globalen Koordinatensystem gegeben, die Ellipsen werden als aktiv definiert und die Umlenkpunkte Q_i sind auf die Befestigungspunkte R_i gesetzt.
Da es schwierig ist, aktive Ellipsen korrekt zu erkennen (vgl. Schritt (ii) in

i	`active_points(i)`	aktiv
1	(1,3)	Nein
2	(2,4)	Nein
3	(3,5)	Nein

(a) Initalisierung

i	`active_points(i)`	aktiv
1	(1,4)	Nein
2	(1,4)	Nein
3	(1,5)	Ja

(b) E_3 ist aktiv und Q_3 wird berechnet

i	`active_points(i)`	aktiv
1	(1,4)	Nein
2	(1,4)	Nein
3	(1,5)	Ja

(c) Zweiter Durchlauf mit neuem Q_3

i	`active_points(i)`	aktiv
1	(1,4)	Ja
2	(2,4)	Nein
3	(2,5)	Ja

(d) E_1 wird ebenfalls als aktiv erkannt.

Tabelle 4: Übersicht üder den Inhalt von `active_points` für die Ellipsen E_i aus Abbildung 9. `active_points(i)` speichert die Indizes der benachbarten, aktiven Umlenkpunte von E_i.

Abschnitt 4.6 und Abbildung 9), wurde das Array `active_points` eingeführt. Zu jeder Ellipse werden hier die benachbarten, aktiven Umlenkpunkte gespeichert. Damit können auch Ellipsen, wie Ellipse E_1 in Abbildung 9, korrekt als aktiverkannt werden. Tabelle 4 zeigt wie in `active_points` die benachbarten, aktiven Umlenkpunkte für Abbildung 9 verwaltet werden.
Diese Umlenkpunkte in `active_points` werden für alle Berechnungen, $\mathcal{K}_U$, Aktivität und Umlenkpunkt Q_i der Ellipse E_i, verwendet. Für alle aktiven Ellipsen E_i wird, gemäß dem gewählten Algorithmus aus Abschnitt 5.3 bis Abschnitt 5.5, der Umlenkpunkt Q_i berechnet. Da die Transformation in das Koordinatensystem $\mathcal{K}_U$ schon erfolgt ist, werden minimal abgeänderte Varianten der Algorithmen aus Abschnitt 5.3 bis Abschnitt 5.5 verwendet. Diese bekommen die bereits transformierten Punkte aus `m` bzw. die Vektoren aus `g` und `h` als Argument, sodass die Transformationen nur einmal berechnet werden müssen, sowie die transformierten, aktiven, benachbarten Umlenkpunkte, die mit `transform_matrices` und `active_points` ermittelt werden. Hier kann die zuvor in `transform_matrices` gespeicherte Matrix T_i wieder verwendet werden.
Nachdem ein Umlenkpunkt Q_i berechnet wurde, wird seine Position mit dem Umlenkpunkt aus der vorherigen Iteration verglichen. Falls die Abweichung weniger als $1 \cdot 10^{-6}$ (dies entspricht in SimPack einer Abweichung von einem Mikrometer) beträgt, wird angenommen, dass die Position von Q_i

konvergiert. Falls die Abweichung niedriger gewählt wird, kann es sein, dass bei Verwendung von `ellipse_with_correction` die Umlenkpunkte nicht konvergieren, da durch die numerische Annäherung auf E_i an Q_i sich die Punkte Q_i minimal bewegen.
Wenn alle Umlenkpunkte Q_i konvergieren und keine neuen aktiven Ellipsen gefunden werden, wird die Länge des Pfades L vom Startpunkt A über alle aktiven Umlenkpunkte Q_i bis zum Endpunkt B berechnet und in `muscle_length` gespeichert. Falls einzelne Schritte von `full_via_ellipse` genauer nachvollzogen werden wollen, befindet sich im Anhang die kommentierte Implementation.

5.7 Integration in SimPack

Ausgangslage für die Integration in SimPack ist die *uforce* von [Klo18], in der die Kraft eines Muskels, unter anderem abhängig von dessen Länge L, berechnet wird. Bis jetzt wurde in der *uforce* die Länge des Muskels als euklidischer Abstand zwischen dem From-Marker und dem To-Marker berechnet. Diese Berechnung wird nun durch die Länge des Musklepfades L über Ellipsen ersetzt. SimPack bietet nicht direkt die Möglichkeit, Ellipsen im $\mathbb{R}^3$ zu definieren, allerdings können Ellipsoide definiert werden. Eine Ellipse E kann also als Ellipsoid, bei dem eine Achse null ist, definiert werden. Ein Ellipsoid in SimPack wird durch seine drei Halbachsen, a,b und c, aufgespannt. Die *uforce* geht davon aus, dass die a-Achse des Ellipsoid null ist und die Ellipse somit durch b und c aufgespannt wird. Ein Objekt wird immer an einem Marker platziert. Dieser Marker ist für die Ellipse E der Mittelpunkt M, der relativ zu dem Knochen, an dem die Ellipse befestigt sein soll, platziert wird. Die Ellipsen werden vor dem Kraftelement modelliert.
Die Benutzeroberfläche der *uforce* wurde wie folgt angepasst: Innerhalb der Eigenschaften der *uforce* werden zuerst der Startpunkt A und der Endpunkt B des Muskelpfades angegeben. Diese können, müssen aber nicht mit dem From-Marker oder To-Marker übereinstimmen. Danach kann über ein Drop-Down-Menü der gewünschte Algorithmus, `without_correction`, `with_correction` oder `co_rotating`, zur Berechnung der Umlenkpunkte ausgewählt werden. Falls keine Auswahl getroffen wird, ist der Algorithmus `without_correction` das Standardverfahren. Dann wird die Anzahl der Ellipsen festgelegt. Momentan sind maximal zehn Ellipsen möglich, da SimPack es zwar erlaubt zur Ausführungszeit der *uforce* die Anzahl dynamisch

anzupassen, dennoch muss zum Start des Programms einen maximale Anzahl festgelegt sein. Als letztes muss jede Ellipse E_i angegeben werden. Diese können, sofern sie bereits modelliert sind, über Navigationsmenü ausgewählt werden.
Innerhalb der *uforce* kann man auf alle ihre Eigenschaften zugreifen. Unter anderem können aus den angegeben Ellipsen alle Mittelpunkte M_i, sowie Halbachsen $\vec{g}_i$ und $\vec{h}_i$ abgefragt werden. Diese werden in Array's, wie in Tabelle 2, gespeichert, damit mit einem einfachen `call`-Befehl `full_via_ellipse` aufgerufen werden kann. Als Ergebnis ergibt sich die Länge des Muskelpfades L in `muscle_length`. Diese wird dann im vorhandenen Hill-Muskel-Modell innerhalb der *uforce* verwendet um die Kraft zu berechnen.
Bei der Anwendung der berechneten Kraft, die zwischen zwei Umlenkpunkten, oder einem Umlenkpunkt und einem Muskelansatzpunkt wirken müsste, ergibt sich folgendes Problem: SimPack erlaubt es nicht, die Kraft einer *uforce* zwischen zwei Punkten anzuwenden, deren Positionen erst innerhalb der *uforce* berechnet werden. Es ist zwar möglich einen Marker zu erstellen, der frei durch eine *uforce* bewegt werden kann, dies führt allerdings zu zwei Problemen. Zum einen befindet dieser sich zum Start einer Simulation immer im Ursprung des globalen Koordinatensystems. Der Umlenkpunkt kann also zum Start nicht auf der Ellipse definiert werden. Zum anderen müsste der Umlenkpunkt Q_i als From-Marker oder To-Marker an die *uforce* übergeben werden und gleichzeitig seine Position von selbiger berechnet werden. Diese Verschachtelung ist in SimPack nicht zulässig. Das heißt, es ist nicht möglich berechnete Umlenkpunkte Q_i als From-Marker oder To-Marker zu definieren. Zusätzlich kann die Kraft nur zwischen genau zwei Punkten, dem From-Marker und dem To-Marker, angewandt werden. Es ist also nicht möglich die Kraft auf jedes Teilstück des Muskelpfades wirken zu lassen. Daher wurde zur Kraftanwendung wie folgt verfahren: Als From-Marker und To-Marker werden zwei Punkte gewählt, die nicht im selben Referenzsystem, also nicht dem selben Knochen angehören, liegen. Wenn es z.B. nur eine Ellipse gibt, die im selben Referenzsystem wie B definiert ist, wird als From-Marker der Startpunkt A und als To-Marker der Mittelpunkt M gewählt. Die Wahl des Mittelpunkt ist sinnvoll, da dieser zumindest in der Nähe von Q liegt. Bei mehreren Ellipsen kann man den From-Marker und den To-Marker z.B. als Mittelpunkte von zwei, in verschiedene Referenz-

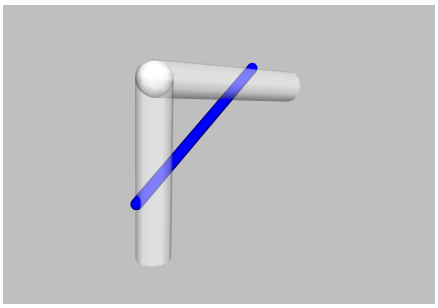
systemen liegenden, Ellipsen setzen. Prinzipiell ist es auch möglich, beliebig definierte Punkte als From-Marker und To-Marker, also als Marker zwischen denen die Kraft wirkt, zu definieren.

6 Auswertung

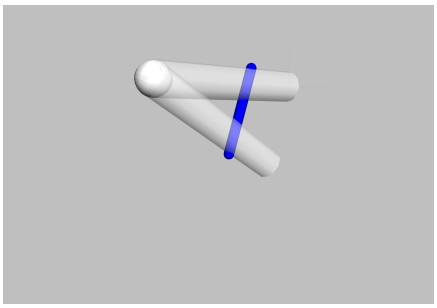
Eine erste Bewertung der neuen *uforce* kann durch eine optische Bestätigung einer korrekt ausgeführten Muskelbewegung vorgenommen werden. Abbildung 11 zeigt einen direkten Vergleich zwischen der alten und der neuen *uforce* bei der Modellierung eines Muskels. Beide Muskeln verwenden die selben Parameter, wie z.B. die Ansatzpunkte am Knochen. Wie in Abbildung 11a zu sehen, konnte in der alten Modellierung der Muskelpfad nur durch die zwei Knochen hindurch dargestellt werden. Durch eine Kontraktion des Muskels wurde zudem eine Beugung, wie in Abbildung 11b zu erkennen, ausgeführt, da die Muskelansatzpunkte zusammengezogen werden. Abbildung 11c zeigt die neue *uforce*, in der der Muskelpfad mit Hilfe einer Ellipse modelliert werden kann. Wie in Abschnitt 5.7 erklärt, wirkt die berechnete Kraft hier zwischen dem Mittelpunkt der Ellipse und dem unteren Ansatzpunkt des Muskels. Eine Kontraktion des Muskels bewirkt, wie in Abbildung 11d dargestellt, nun eine Streckung. In Abbildung 11e wurde der Verlauf des Muskelpfades mit zwei Ellipsen modelliert. Hier wirkt die berechnete Kraft zwischen den beiden Mittelpunkten der Ellipsen. In Abbildung 11f ist zu erkennen, dass die untere Ellipse am, zu Beginn nach unten hängenden, Knochen befestigt ist und sich mit dem Knochen bewegt.

7 Fazit

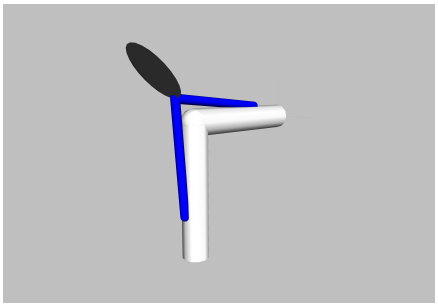
Mit der neuen *uforce* ist es in SimPack möglich, den Verlauf eines Muskelpfades über Ellipsen zu modellieren. Dies ermöglicht dem Nutzer oder der Nutzerin mehr Freiheit den Muskelpfad näher am anatomischen Vorbild darzustellen. Das zugrundeliegende mathematische Modell wurde ausführlich besprochen, der Algorithmus zur Wegminierung vorgestellt und dessen Implementation in `Fortran 90` bzw. Integration in SimPack erläutert. Hierbei sind auch die Möglichkeiten und Grenzen aufgezeigt worden. Innerhalb der Implementation wurde darauf geachtet, Berechnungen möglichst effizient umzusetzen. Gleichzeitig sollte der Quellcode nachvollziehbar bleiben, um



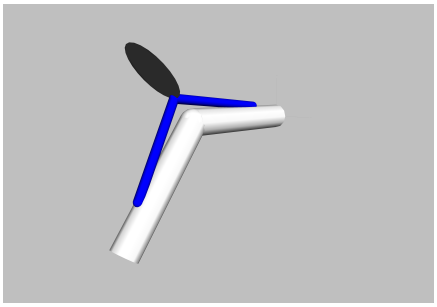
(a) Startposition



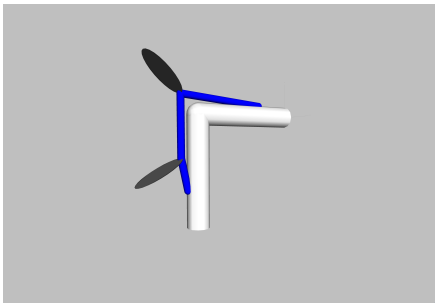
(b) Ende der Bewegung nach einer Sekunde



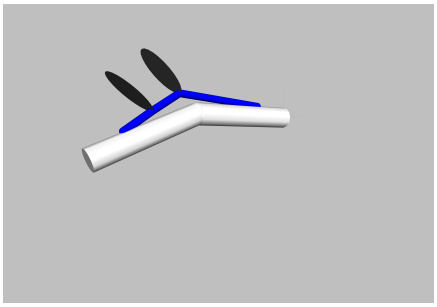
(c) Startposition, eine Ellipse



(d) Ende der Bewegung nach einer Sekunde



(e) Startposition, zwei Ellipsen



(f) Ende der Bewegung nach einer Sekunde

Abbildung 11: Eine einfach modellierte Bewegung zweier Knochen um ein Gelenk. a und b verwenden die alte *uforce*. c bis f verwenden die neue *uforce*. Der Muskelpfad wird hier blau dargestellt, die Knochen sind weiß. Der obere Knochen ist fixiert und nur der zu Beginn nach unten gerichtete Knochen kann sich bewegen. Die Abbildungen wurden mit SimPack erstellt.

nachträglich Anpassungen oder Verbesserungen vorzunehmen. Daher gibt es an manchen Stelle noch Optimierungspotenzial.

8 Ausblick

Die Bewertung in Abschnitt 6 zeigt, dass mit der Darstellung eines Muskelspfades über Ellipsen näher am anatomischen Muskelpfad modelliert werden kann. Es ist möglich, mit einem anatomisch modellierten Muskelverlauf eine anatomisch korrekte Bewegung auszuführen (vgl. Abbildung 11). Es fehlt eine Validierung der genauen, berechneten Kräfte in der *uforce* durch die Modellierung über Ellipsen bzw. der Berechnung der Länge des Muskelpfades in `via_ellipse`. Um dies zu erreichen, ist es nötig, ein anatomisches Modell, z.B. den modellierten Klimmzug eines Menschen in [Klo18], mit über Ellipsen modellierten Muskelpfaden zu versehen. Aus der Simulation und den daraus gewonnenen Daten kann eine genauere Bewertung der Implementation erfolgen. In [Ham19] wurde an drei Beispielen, dem kurzen Großzehenstrecker, dem Schneidermuskel und dem großen Gesäßmuskel, gezeigt, dass der Ansatz der Modellierung eines Muskelspfades über Ellipsen generell nahe an dem anatomischen Vorbild ist. Berechnete Kräfte in der Simulation von [Ham19] decken sich mit experimentellen Daten. Somit ist grundsätzlich anzunehmen, dass eine Modellierung eines Muskelpfades über Ellipsen eine Verbesserung darstellt.

9 Anhang

Implementation von `full_via_ellipse` in `Fortran 90`. Kommentare werden durch ein `!` markiert. Ein `&` zu Beginn der Zeile heißt, dass die vorherige Zeile um die aktuelle Zeile verlängert wird, da im fixed-Format nur 72 Zeichen pro Zeile erlaubt sind.

```
   SUBROUTINE full_via_ellipse(origin,insertion,global_m, global_h,
&  global_g, q ,muscle_length, algorithm)
   IMPLICIT NONE

   ! coordinates in global system
   integer, INTENT(IN) :: algorithm ! 1 = without correction,
   ! 2 = with correction, 3 = co-rotating, default = without
       correction
   real(dp), DIMENSION(3), INTENT(IN) :: origin, insertion
   real(dp), DIMENSION(:,:), INTENT(in) :: global_m
   real(dp), DIMENSION(:,:), INTENT(in) :: global_h, global_g
   real(dp), DIMENSION(:,:), INTENT(OUT) :: q

   ! local varaibles
   logical, Dimension(:), allocatable :: active, converging
   real(dp), DIMENSION(:,:),allocatable::via_points,m,g,h,g_r,h_r
   real(dp), DIMEnsION(3) :: temp_m, temp_g, temp_h, e_x,e_y,e_z
   real(dp), DIMENSION(4,4) :: temp_transform
   integer, DIMENSION(:,:), allocatable  :: active_points
   real(dp), DIMENSION(:,:,:), allocatable ::  transform_matrices
   real(dp) :: muscle_length
   logical :: all_converging, new_active
   integer :: numberofellipses, i, j, counter

   ! Initilaization
   numberofellipses = size(global_m,1)
   allocate(active(numberofellipses))
   allocate(m(numberofellipses,3))
   allocate(g(numberofellipses,3))
   allocate(h(numberofellipses,3))
   allocate(via_points(numberofellipses+2,3))
   allocate(transform_matrices(numberofellipses,4,4))
   allocate(converging(numberofellipses))
   allocate(active_points(numberofellipses,2))
   allocate(g_r(numberofellipses,3))
   allocate(h_r(numberofellipses,3))
   active = .true.
   converging = .false.
   all_converging = .false.
   via_points(1,:) = origin
   via_points(numberofellipses+2,:) = insertion
   muscle_length = ZERO
   new_active = .FALSE.
```

```
   ! via_points: a, q1, ..., qn, b -> for active condition check
   ! via_pints(i+1) = q of ellipse i
   via_points(2:numberofellipses-1,:) = q

   ! set all ellipses active => via points set
   do i=1,numberofellipses
     active_points(i,:) = (/i,i+2/)
   end do

   ! Start of algorithm
   do while(.not. all_converging)

   ! loop to dertemiante which ellipse is active
   do i=1,numberofellipses

   ! save transform matrices (so it needed to be calcualted just once)
   transform_matrices(i,:,:) = get_transformation_matrix(
&  via_points(active_points(i,1),:),
&  via_points(active_points(i,2),:),global_m(i,:))

   ! rotate g and h for algorithm = co rotating
   if (algorithm .EQ. 3) then
     ! roatating ellipse's semi-axes
     e_x = normalize(via_points(active_points(i,2),:) -
&    via_points(active_points(i,1),:))
     e_z = normalize(cross(e_x,(global_m(i,:) -
&    via_points(active_points(i,1),:))))
     e_y = cross(e_z,e_x)
     g_r(i,:) =
&    (global_g(i,2)*e_y+global_g(i,3)*e_z)*norm(global_g)/
&    (sqrt(global_g(i,2)*global_g(i,2)+global_g(i,3)*global_g(i,3)))
     h_r(i,:) =
&    (global_h(i,2)*e_y+global_h(i,3)*e_z)*norm(global_h)/
&    (sqrt(global_h(i,2)*global_h(i,2)+global_h(i,3)*global_h(i,3)))
     ! m,g,h to save in EDF (so they needed to be calcualted just once
        )
     ! for rotating semi-axes
     m(i,:)=transform_point(global_m(i,:),transform_matrices(i,:,:))
     g(i,:)=transform_vector(g_r(i,:),transform_matrices(i,:,:))
     h(i,:)=transform_vector(h_r(i,:),transform_matrices(i,:,:))

   else
     ! m,g,h to save in EDF (so they needed to be calcualted just once
        )
     ! for with and without correction
     m(i,:)=transform_point(global_m(i,:),transform_matrices(i,:,:))
     g(i,:)=transform_vector(global_g(i,:),transform_matrices(i,:,:))
     h(i,:)=transform_vector(global_h(i,:),transform_matrices(i,:,:))
     end if
```

```
    ! check if ellipses are activ
    if (is_active(m(i,:),g(i,:),h(i,:))) then
     active(i) = .true.
     ! save via-points for which ellipse i is active as active via
         points
     active_points(i,:) = (/active_points(i,1),active_points(i,2)/)
     ! tell next ellipse the via point on ellipse i is active
     if (i.LT.(numberofellipses-1)) active_points(i+1,1) = i+1
    else
     active(i) = .false.
     ! next ellipse actve point = previous active point from i-1
     if(i.LT.numberofellipses) then
     active_points(i+1,1) = active_points(i,1)
     end if
    end if
    end do

    ! tell every inactive ellipse which is the next active via point
    do i=1,numberofellipses-1
    ! j = current ellipse, counting backwards, start numberofellipses-1
    j = numberofellipses-i
      if (.NOT.active(j)) then
        if (active(j+1)) then
           active_points(j,2) = j+2
        else
           active_points(j,2) = active_points(j+1,2)
        end if
      end if
    end do

    ! assuming there is no new active point
    new_active = .False.

    ! serach for wrong inactiv detectet ellipses
    do i=1,numberofellipses
    ! only nessesary for inactive ellipses
    if (.not.active(i)) then
      ! check if ellipse is active with adjacent active via-points
      temp_transform = get_transformation_matrix(
&     via_points(active_points(i,1),:),
&     via_points(active_points(i,2),:),global_m(i,:))

    if (algorithm .EQ. 3) then
      ! roatating ellipse's semi-axes
      e_x = normalize(via_points(active_points(i,2),:) -
&     via_points(active_points(i,1),:))
      e_z = normalize(cross(e_x,(global_m(i,:) -
&     via_points(active_points(i,1),:))))
      e_y = cross(e_z,e_x)
      g_r(i,:) =
&     (global_g(i,2)*e_y+global_g(i,3)*e_z)*norm(global_g)/
```

```
&     (sqrt(global_g(i,2)*global_g(i,2)+global_g(i,3)*global_g(i,3)))
      h_r(i,:) =
&     (global_h(i,2)*e_y+global_h(i,3)*e_z)*norm(global_h)/
&     (sqrt(global_h(i,2)*global_h(i,2)+global_h(i,3)*global_h(i,3)))
      ! m,g,h to save in EDF (so they needed to be calcualted just once
         )
      ! for rotating semi-axes
      temp_m=transform_point(global_m(i,:),temp_transform)
      temp_g=transform_vector(g_r(i,:),temp_transform)
      temp_h=transform_vector(h_r(i,:),temp_transform)

    else
      ! m,g,h to save in EDF (so they needed to be calcualted just once
         )
      ! for with and without correction
      temp_m=transform_point(global_m(i,:),temp_transform)
      temp_g=transform_vector(global_g(i,:),temp_transform)
      temp_h=transform_vector(global_h(i,:),temp_transform)
    end if
      if (is_active(temp_m, temp_g, temp_h)) then
        active(i) = .true.
        ! new_active = true => at least on more iteration
        new_active = .true.
        ! tell next ellipse, that i is active
        active_points(i+1,1) = i+1
        m(i,:) = temp_m
        g(i,:) = temp_g
        h(i,:) = temp_h
        else
        ! pass previous active via-point to follwing ellipse
        ! in case a previos point is added
        if (i .LT. numberofellipses) then
          active_points(i+1,1) = active_points(i,1)
        end if
      end if
    end if
    end do

    ! calculation of via-points
    do i=1,numberofellipses

    ! find new via-point if ellipse is active
    if (active(i)) then
      select case(algorithm)

      case (1)
      call full_single_ellipse_without_correction( m(i,:), h(i,:),
      ! via_points(i+1,:) <=> via-point of ellipse i
&       g(i,:), via_points(i+1,:), global_m(i,:),
&       global_g(i,:), global_h(i,:))
```

```
      case (2)
      call full_single_ellipse_with_correction( m(i,:), h(i,:),
      ! via_points(i+1,:) <=> via-point of ellipse i
&       g(i,:), via_points(i+1,:), global_m(i,:),
&       global_g(i,:), global_h(i,:),
&      ! a and b in edf needed for correction
&       transform_point(via_points(active_points(i,1),:),
&       transform_matrices(i,:,:)),
&       transform_point(via_points(active_points(i,2),:),
&       transform_matrices(i,:,:)))

      case (3)
      call full_single_ellipse_co_rotating( m(i,:), h(i,:),
      ! via_points(i+1,:) <=> via-point of ellipse i
&       g(i,:), via_points(i+1,:), global_m(i,:),
&       h_r(i,:),g_r(i,:))

      case default
      call full_single_ellipse_without_correction( m(i,:), h(i,:),
      ! via_points(i+1,:) <=> via-point of ellipse i
&       g(i,:), via_points(i+1,:), global_m(i,:),
&       global_h(i,:), global_g(i,:))
      end select

      ! check if via_point is converging compared to last iteration
      ! 0.000001 = 10^-6 meter (simpack) => precision to 1 Mikrometer
      if (norm(q(i,:) - via_points(i+1,:)) .LE. 0.000001) then
        converging(i) = .true.
      end if

    else
      ! case for ellipse i is inactive, via-point of i is converging
      converging(i) = .true.
    end if
    end do

    ! assume alle via-ponits are converging
    all_converging = .true.

    ! check if all via-points are converging
    do i=1,numberofellipses
      ! q saves the new via-points for comparison wether the
      ! via-points are converging or not
      q(i,:) = via_points(i+1,:)
      ! if one via-point is not converging "all_converging" is set
          false
        if (converging(i) .eqv. .false.) all_converging = .false.
    end do

    ! at least one more iteration if a new ellipse is active
      if (new_active) all_converging = .false.
```

```
    end do

    ! calcualting muscle length
    ! j = start, i+1 = via-point if active allipse i
    j = 1
    do i=1,numberofellipses+1
    if ((i .EQ. numberofellipses+1) .OR. active(i)) then
      muscle_length = muscle_length +
&     norm(via_points(i+1,:)-via_points(j,:))
      j = i+1
    end if
    end do
    END SUBROUTINE full_via_ellipse
```

Literaturverzeichnis

[Klo18] Sascha Kloft. „Mukuloskelettale Modellierung des menschlichen Klimmzuges“. Masterarbeit. Universität Koblenz-Landau, 2018.

[Ham19] M. Hammer u. a. „Tailoring anatomical muscle paths: a sheath-like solution for muscle routing in musculoskeletal computer models“. In: *Mathematical Biosciences* 311 (2019), S. 68–81.

[Car19] Marc Carletto. „Mathematische Grundlagen zur Modellierung von Kraftumlenkung in Mehrkörpersimulationen“. Bachelorarbeit. Universität Koblenz-Landau, 2019.

[Car68] G. Cardano und T.R. Witmer. *Ars Magna Or The Rules of Algebra*. Dover Books on Advanced Mathematics. Dover, 1968.

[Car99] Ralph Carmichael. *quartic.f90*. Version 1.3. 1999. URL: `http://www.pdas.com/quarticdownload.html` (besucht am 27. 04. 2019).